Bachelor's thesis

Information Technology

Digital Media

2016

Johannes Ylönen

MULTIPLAYER NETWORKING WITH AN OPEN SOURCE LIBRARY

– Implementation plan for HactEngine

TURUN AMMATTIKORKEAKOULU

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Digital Media

2016 | 40

Instructors: Reetta Raitoharju;Principal Lecturer, Ph. D. Mika Luimula

Johannes Ylönen

MULTIPLAYER NETWORKING WITH AN OPEN SOURCE LIBRARY

The main purpose of this thesis was to compare open source networking libraries and plan how to integrate the selected library into HactEngine. HactEngine is a cross-platform game engine developed by a start-up company called Indium Games. This thesis was written as research for an actual implementation, which is a part of the Tekes-funded “HactEngine 1.0” project.

The thesis describes some of the most fundamental technologies, which are important to understand when developing multiplayer networking. Internet protocols, such as TCP and UDP, are introduced first and then two network topologies, client-server and peer-to-peer, from the perspective of multiplayer game development.

The thesis next introduces HactEngine and the technologies used in it; this section covers programming languages and a package system to integrate third party libraries. The most important part of the integration is the interface which is generated from C++ code with the SWIG software development tool.

The library to integrate was selected by comparing four different open source libraries. The first two of them, ENet and RakNet, are intended for game development. Boost.Asio and POCO were selected to find a different angle to development, because they were designed for the development of any kind of networked application. RakNet was selected, because it had the most features for multiplayer development and there would have been too much work to use any of the others. The last section of the thesis describes how to integrate RakNet. It demonstrates what the interface could look like and what features are required for minimal functionality of server and client demos.

This thesis provides a good list of references for anyone who is interested in multiplayer development. The thesis provides different perspectives to the integration and information, how to add ready-made features to ease the game development. Multiplayer networking is an important part of HactEngine to attract new developers.

KEYWORDS:

multiplayer, game, engine, SWIG, HactEngine, RakNet

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Mediatekniikka

2016 | 40

Ohjaajat: Reetta RaitoHarju; Yliopettaja, FT Mika Luimula

Johannes Ylönen

MONINPELIN VERKKO-OHJELMOINTI AVOIMEN LÄHDEKOODIN KIRJASTOLLA

Opinnäytetyön päätavoite oli vertailla avoimen lähdekoodin verkko-ohjelmointikirjastoja ja suunnitella kirjaston integraatio HactEngineen. HactEngine on Indium Games -yrityksen kehittämä, monella eri alustalla toimiva pelimoottori. Työ kirjoitettiin tutkimuksena varsinaista toetutusta varten, joka on osa Tekesin rahoittamaa HactEngine 1.0 -projektia.

Työssä esitellään tärkeimpiä teknologioita, jotka ovat tärkeä perusta verkko-ohjelmoinnille. Niitä tutkittiin erityisesti moninpelikehityksen näkökulmasta. Tärkeimpiä ovat asiakas-palvelin- ja vertaisverkkotopologiat, ne määrittelevät, miten viestejä lähetetään pelaajien välillä.

HactEnginen näkökulmasta integrointiin liittyy ohjelmointikieli C++, jolla pelimoottori on kirjoitettu, sekä skirptikieli Lua, jota käytetään pelien ohjelmointiin. Pakettijärjestelmän avulla voidaan lisätä tuki eri kirjastoille. Yksittäinen paketti tarvitsee SWIG-ohjelmistokehitystyökalulla C++-kielellä kirjoitetusta koodista generoidun rajapinnan, jota käytetään skriptikielen avulla.

Integroitava kirjasto valittiin vertailemalla neljää erilaista avoimen lähdekoodin kirjastoa, joista kaksi ensimmäistä, ENet ja RakNet, on suunniteltu moninpelien tarpeisiin. Toiset kaksi, Boost.Asio ja POCO, on tarkoitettu yleisesti kaikkeen verkko-ohjelmointiin. Ne valittiin, jotta saataisiin erilainen näkökulma kehitystyöhön. Kirjastoksi valittiin RakNet, koska sillä on eniten ominaisuuksia moninpelien kehitykseen ja muilla kirjastoilla kehittäminen olisi ollut liian työlästä.

Tärkeimpänä tuloksena saatiin suunnitelma integraatiosta. Asiakas- ja palvelin-esimerkkikoodeissa esitellään, miltä asiakasrajapinta voisi näyttää ja mitä ominaisuuksia palvelin tarvitsee. Opinnäytetyöhön löydettiin hyviä kirjalähteitä moninpelien kehitykseen, sekä se antaa erilaisia näkökulmia, miten kirjaston integraatio kannattaa tehdä ja minkälaisilla valmiilla ominaisuuksilla voi helpottaa pelinkehittäjien työtä.

ASIASANAT:

moninpelit, pelimoottori, SWIG, HactEngine, RakNet

CONTENT

APPENDICES

PICTURES

1 INTRODUCTION

A nerd playing games alone has become an ancient stereotype, when playing games online with friends or strangers is more popular than ever and is one of the fastest growing areas of the gaming industry. MMO (Massively Multiplayer Online Game), one category of multiplayer gaming, takes alone 21% of all revenue in global games market (Newzoo 2013). Growth of social mobile games and popularity of electronic sport competitions is another reason for rapid growing.

The popularity of online multiplayer games started as a new competitive game mode by games like Quake and Duke Nukem 3D. It has since expanded into all categories of gaming, like MMORPG (Massively multiplayer online role-playing game) game World of Warcraft, FPS (First-person Shooter) video game franchise Call of Duty and sports simulation series NHL. Multiplayer brings engaging and addictive features that leads to longer game sessions and potentially longer lifespan for a game. (App Annie 2015)

HactEngine is an open source and cross-platform game engine developed by Indium Games. It is built modular, so that it is easy for the developer to add or ignore any package. The core of HactEngine is developed with C++, but it uses Lua as its language for developing the game. An advanced developer can add their own bindings for any language or add new packages either with Lua, C++ or both. Indium Games has got funding from Tekes, the Finnish Funding Agency for Innovation to develop the game engine to the point, where it is ready to be published as open source, as it will have the needed basic features and documentation to build a game with.

The objective of this thesis is to find the best open source networking library for HactEngine, by comparing libraries which are designed to ease the development of networking functionalities in applications. The first part of this thesis describes the theory for the fundamental needs of multiplayer networking. Where the first area is protocols and second is multiplayer networking topologies that are the most common. These give a developer a way to communicate between the players, whether it is real time or turn-based with four or 1000 simultaneous connections.

In the second part there is a comparison of the libraries. The first two libraries, RakNet and ENet, are intended specially for multiplayer game development, RakNet has complete support for lobby systems and other common multiplayer features. The other two, Boost.Asio and POCO, are designed for the development of all kinds of network- and internet-based applications. As a result of the research and comparison RakNet was chosen as the most suitable for the needs of HactEngine. It has a stable code base with a lot of ready features and support for a variety of platforms. The last parts describe the plan of implementing RakNet support into HactEngine. Which features are needed and what needs to be taken into account from the perspective of HactEngine.

The need of a networking library for HactEngine was not only important because of the popularity driven needs of developers, but also by the internal needs of Indium Games.

2 MULTIPLAYER NETWORKING TECHNOLOGIES

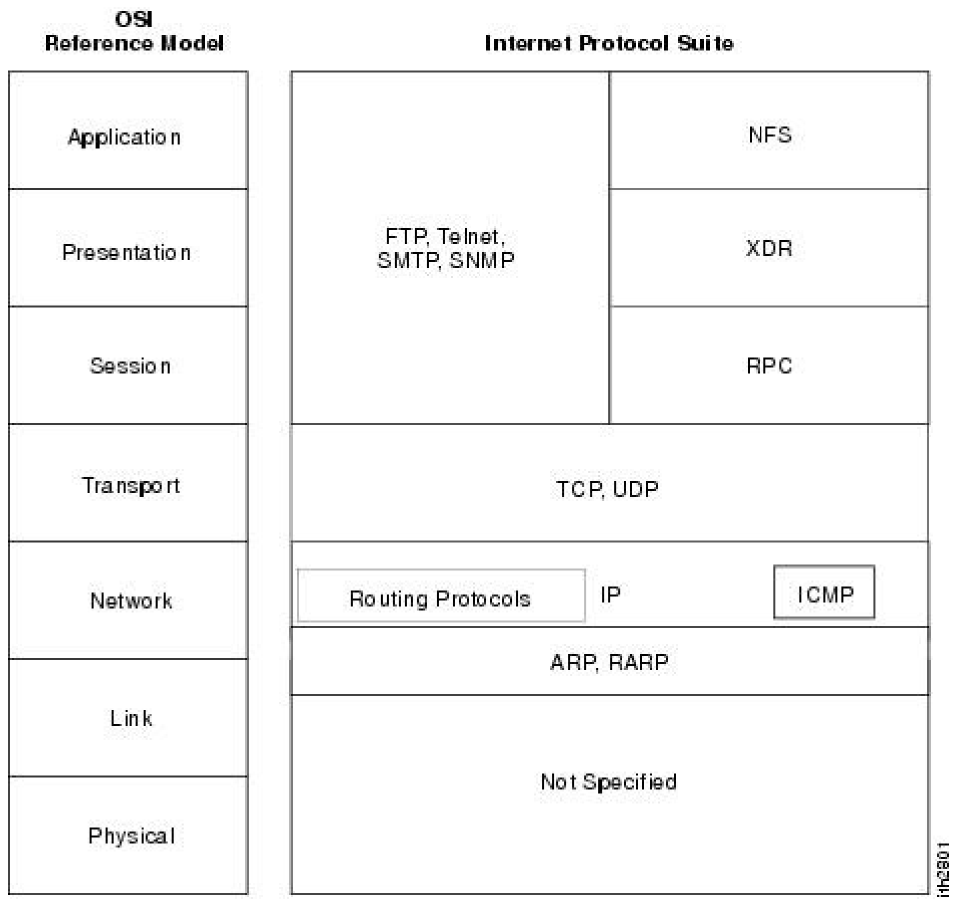
This part of the thesis describes the technologies, which are used in multiplayer networking. There are several protocols that are used in networked applications, but without exception Internet Protocol (IP), Transmission Protocol (TCP) and User Datagram Protocol (UDP) are the most important. This section tries to open up how these work and why they are the foundation of game networking. Finally, after the basics are understood, it is time to explain the common multiplayer game networking topologies: client-server and peer-to-peer.

2.1 Internet Protocols

Internet protocols are the very first thing one should understand, when starting to develop a networked application. The idea of the protocols is to define the fundamental architecture for communication between two devices. When developing a multiplayer game, the developer will very likely face these protocols at some level. At least learning the basics will greatly increase understanding and prevent bugs. (Oki et al. 2012, 1-5)

There are two guidelines, the Open Systems Interconnection (OSI) reference model and the Internet Protocol Suite (TCP/IP). The OSI reference model is an abstract description for layered communications and computer network protocol design. The Internet Protocol Suite defines some major protocols, which are used in different layers of the OSI model. (Oki et al. 2012, 1-5)

The two guidelines are described best with the Picture 1; the left side of the figure describes the seven layers of the OSI reference model and the right side has some popular protocols from the Internet Protocol Suite. These protocols are arranged next to corresponding layer of the OSI model. (Cisco Systems 2014)



Picture 1. OSI Reference Model and Internet Protocol Suite (Cisco Systems 2014)

While all the layers are important in the OSI model, the network and transport layers are the most topical for this thesis. The session, presentation and application layers are not described in detail. They are tied to services that a library may offer and are not usually standardized in game networking. The bottom two layers, physical and link, are not important for this thesis either. They define the physical devices and links like routers, switches and cables.

2.1.1 Internet Protocol (IP)

The Internet Protocol is the primary protocol of the network layer. According to the protocol specification (RFC 791): “The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks." (Postel, J. 1981).

It uses IPv4 or IPv6 addresses to route the datagrams. A datagram, in turn, is an IP packet which contains information, such as the addresses and some control information for routing. Although the Internet Protocol has enough information to deliver packages in a connected network, it does not verify the delivery. It needs a protocol from the transport layer to carry packets.

IPv4 and IPv6 are different versions of Internet Protocol. These use addresses which define a logical address of a system, so that it is publicly reachable. An IPv4 address is a 32-bit number, which means it can have 4 294 967 296 different addresses. The address range is already too small and that is the reason why IPv6 was developed. An IPv6 address is a 128-bit number, therefore in theory it allows 2^{128} (3.4×10^{38}) addresses. (Glazer, J. & Madhav, S. 2015, 24-38)

2.1.2 Transmission Control Protocol (TCP)

Transmission Control Protocol is a reliable protocol from the transport layer of the OSI model. It includes many features to control the transmission, for example it has a feature to send lost packets again with a retransmission mechanism. At first, TCP uses a three-way handshake to establish the connection and then, by flow control, it makes sure the packets are transmitted to the receiver in the right order. Protocols like HTTP, POP and FTP are dependent on TCP. (Oki et al. 2012, 19-21)

From the perspective of game networking, Transmission Control Protocol may be used where transmission must be reliable. Examples of these kinds of situations are synchronizing player data to the server, downloading of assets and communication in a turn-based game. On the other hand, communication in a trun-based game may have been produced on top of the UDP protocol, but with TCP-like functioning. This is because TCP may add unnecessary features that affects on performance.

2.1.3 User Datagram Protocol (UDP)

User Datagram Protocol is also from the transport layer, but it is called a connectionless protocol. The reason is that UDP does not provide any handshaking before sending packets and neither does it care if the packet was received by the destination. However, User Datagram Protocol does verify that the data is not corrupted at the destination. (Oki et al. 2012, 25-26)

UDP has less data per packet than TCP, and is therefore a lot faster and simpler to use. Therefore, UDP allows the transmission to “drop” packages, meaning that the packages will not be transmitted, but transmitted packages are guaranteed to not be corrupted. The higher performance is the reason why UDP is a better choice when developing real-time applications, such as real-time multiplayer games or media streaming software. The only disadvantage is that the developer may have to add some TCP-like features to enable better control for transmission. (Oki et al. 2012, 267-268)

In real-time multiplayer games there is a need to transmit a lot of data to a client and it is not essential that all data is received by the client, but that most of the data is delivered with high performance. As a conclusion, both TCP and UDP, have their own use cases and both of them can be used in different parts of multiplayer game networking.

2.1.4 Network Address Translation (NAT)

At the moment, IPv4 is the most common protocol, but its disadvantage is that there are not enough public IP addresses for everyone, which is why NAT was developed. NAT enables one IP address to connect a large subnet of hosts to an another wider network. A NAT gateway records IP address and the port of a subnet host which is sending a packet, then it rewrites the packet with its own ip-address and a randomly selected free port. The destination system receives the packet as it was sent originally by the NAT gateway. When the destination system, in turn, sends the packet back to the NAT gateway, the gateway knows which host in the subnet had sent it originally. (OpenBSD)

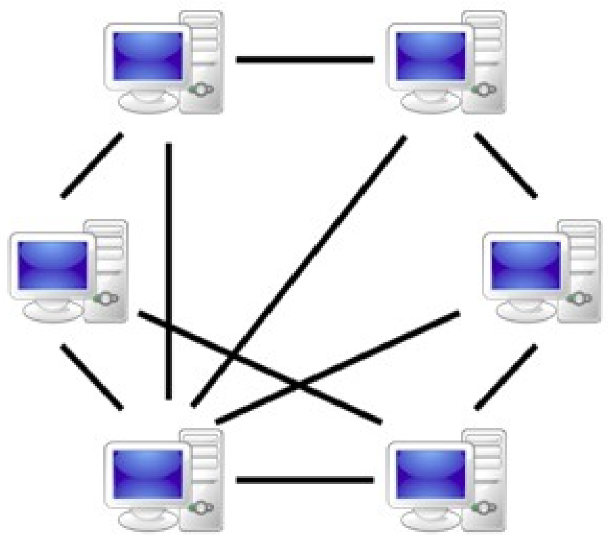
2.2 Multiplayer Networking

The first multiplayer games played with computers, worked through different text-based solutions. One could play turn-based games like chess through e-mail or multi-user dungeon games with a text-based Telnet connection. You could also play turn-based games through websites, where there can be graphics and simple animations. These still exist with different variations and there are many popular games, but most modern multiplayer games require very different technologies (Lecky-Thompson, G.W. 2008).

This thesis is limited to modern multiplayer networking technologies because HactEngine is targeting to them. That said, this section describes the most common modern networking topologies, peer-to-peer and client-server. These are very closely related to game design and understanding them will reduce extra work.

2.2.1 Peer-to-Peer Topology

The very first real-time multiplayer games were connected through a peer-to-peer connection. With peer-to-peer devices are connected with each other in a fully connected mesh (Fiedler, G. 2008). This means, that the devices can send messages straight to each other. The topology is shown in the Picture 2.



Picture 2. A peer-to-peer network (Nutaq Innovations 2015)

For games, peer-to-peer is a little problematic on its own; it is simple, but generates some problems. For example, usually with peer-to-peer players only send commands about what they do and other players do the rendering based on that. It means that the game may render differently or in desynchronization between the players. A way to fix this is to wait that every player has received the command before rendering. But this means that each player has latency equal to the slowest player. (Fiedler, G. 2008)

Peer-to-peer topology is best suited for games where the game world is predictable. This statement has at least two reasons. If all information about the environment is delivered between everyone, there will be a lot of latency and unnecessary traffic, someone must be authoritative about the game world. And secondly, if the world is rendered based on commands, it must behave in the same way for everyone. (Glazer, J. & Madhav, S. 2015, 168-169)

Even if the game world is not predictable, for example, if it is based on physics, peer-to-peer can still be used. One way could be that a certain peer is authoritative for different parts of the game or it could be used in a way that one peer is authoritative for everything. The latter is actually a very common way and usually referred to as the client-server model. The client-server model, in turn, is based on the same named topology described in “2.2.2 Client-Server Topology”. When client-server is used in peer-to-peer, it reduces the costs of dedicated servers, but needs more resources from players. (Glazer, J. & Madhav, S. 2015, 168-169)

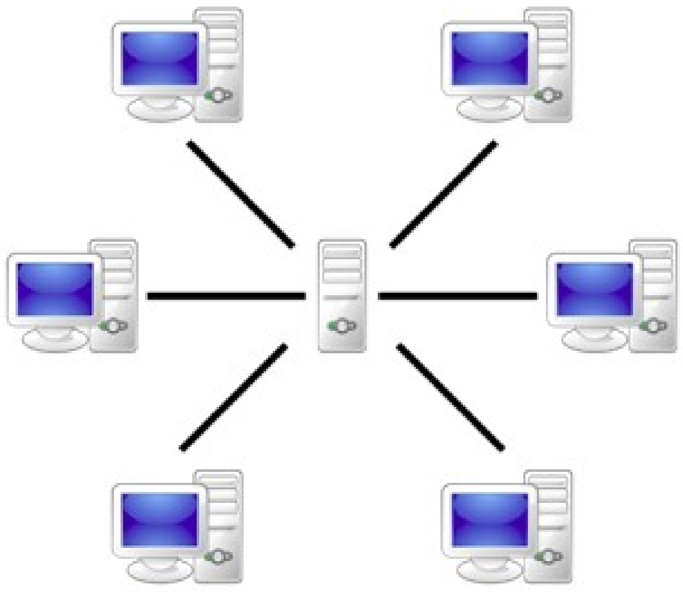
With the client-server model one player needs to act as the server, therefore is the host without usually even knowing it. The same can be achieved without peer-to-peer, but if the host player quits, the game session ends. Peer-to-peer can handle this with host migration and change the host on the fly.

With peer-to-peer there needs to be a way to connect players with each other by matchmaking. This service is provided by game console manufacturers or digital distribution platforms like Steam. In other scenarios the developer may have to provide the solution by itself or use a third-party library.

2.2.2 Client-Server Topology

Another common topology is client-server and as was told previously, it is also used as a model of transferring data between players. Here it is described first as a network topology, but when called as the client-server model, it can be used with both topologies.

As a topology, client-server is a very common way to build network services, one very well-known application is a web-browser. Briefly said, there is a server which contains a service. Then, a client who connects to the server asks for the service, the server returns what the client asked. In the previous example the server does most of the processing and returns only the results to the client. Picture 3 shows the network topology for client-server.



Picture 3. Client-server network (Nutaq Innovations 2015)

A pure client-server model is the base for modern games where there are many players in the same world and the server must run the game code to be up to date about all players. This model can be implemented in many ways: extremely large games, like MMORGPs, will need scalable dedicated servers. On the other hand, games with a smaller amount of players works well on top of peer-to-peer networking, but with the client-server model. Someone could also use the client-server model without any need of dedicated servers, but then the player needs to be able to find out the IP address of the opponent. As was shown, the implementation can vary depending on the resources of the company, the type of the game or by the limitations of the target platform.

A client-server multiplayer game runs the game instance on a server and the server is authoritative. In other words, it simulates everything in the game world and tells a client what to do. If a client and the game server disagree about the state, the server is always correct. The main job of the client is to send user inputs to the server and then render the world based on the response from the server. If there is too much latency, the client may compensate it by interpolating or extrapolating the lost frames. Interpolation is used to calculate frames between two positions, if the two positions should have frames in-between. And extrapolation is used to predict coming frames if they are late. (Glazer, J. & Madhav, S. 2015, 166-167, 236-238)

3 HACTENGINE

HactEngine is a cross-platform game engine developed by Indium Games. Its development was started in 2012. HactEngine is developed for the needs of Indium Games, but will be published as open source when enough basic functionalities are working. Indium Games got funding from Tekes to keep the development of the engine on the forefront. Open sourcing the engine will help to build a community and gain publicity for the company and the engine.

The core of HactEngine is developed with C++ and it uses SDL – a low-level direct media layer library – as its base. SDL is a very popular cross-platform library, which is used by many big and small companies. The user interface for HactEngine is written with Qt – an open source application framework maintained by a Finnish company, The Qt Company.

A game developer will mostly use the Lua scripting language to create games and is therefore able to develop almost everything while the game is running. The engine is built modular and the community can easily add any other programming language to use instead of Lua. They can also create their own packages and easily disable or enable any other package. The engine uses OpenGL for graphics and supports both 2D and 3D graphics. Most of the graphics assets can also be changed while the game is running.

3.1 C++

C++ is a high-performance programming language, it was created by Bjarne Stroustrup in 1979. The initial purpose was to be “a better C” (Standard C++ Foundation 2016). The C programming language is one of the most used programming languages, developed at Bell Laboratories in 1972 by Dennis Ritchie (Fresh2refresh.com 2014). The most important addition of C++ to C is object-oriented programming, which is important when writing complex applications. The newest version of the C++ standard is C++14.

The core of HactEngine is written in C++, to achieve great performance. Performance is very vital for a game engine, when there is a requirement of processing a lot of complex code and graphics in real-time. C++ will also be one of the default languages for writing packages for HactEngine.

3.2 Lua

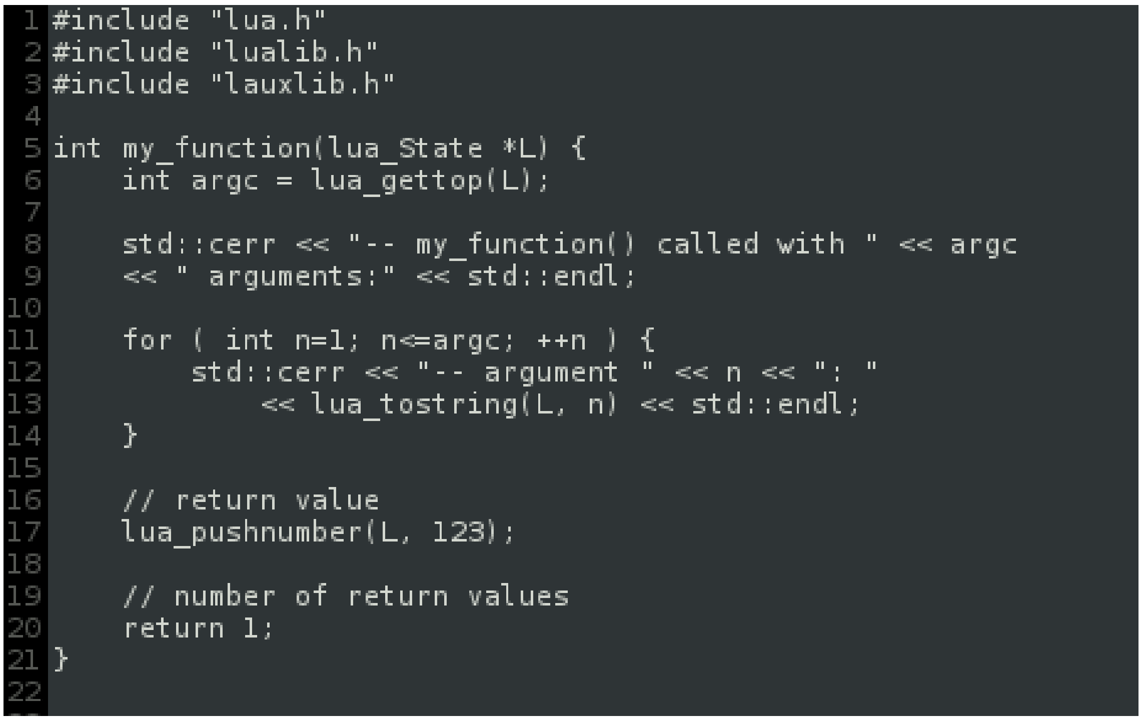
Lua is a fast scripting language; it is an interpreted language, which means it is compiled at run-time. Lua is developed by a team at PUC-Rio, the Pontifical Catholic University of Rio de Janeiro in Brazil. Lua is portable and simple, it doesn't provide large features – like object-oriented features – but does provide mechanisms to allow powerful extending. (Lua.org 2016)

In general, scripting languages, especially Lua, are popular in game scripting, because of the fact that they speed up and ease development. In HactEngine Lua can be edited while the game is running, thereby achieving even faster development.

3.3 SWIG

SWIG is a command line tool to connect C and C++ applications with different high-level programming languages like Lua, Python, Ruby and C#. Development of SWIG was started by Dave Beazley in 1995. SWIG can be used for rapid prototyping and debugging or to add an extension module with a scripting language. The latter is the reason why SWIG was added to HactEngine, it connects Lua to the C++ application. (Swig.org 2015a)

The normal way to combine Lua with C++ is to use the C API for Lua, which is a set of C functions to communicate with Lua (Swig.org 2015a). The first limitation of this way is the C API itself, it does not directly support C++ features. And the second is that it is time consuming to first, write the application in C++ or C, second to write the interface for Lua with the C API and finally use the interface from Lua. The developer will also need to understand an extra layer, the C API.



```
 1 #include "lua.h"
 2 #include "lualib.h"
 3 #include "lauxlib.h"
 4 
 5 int my_function(lua_State *L) {
 6     int argc = lua_gettop(L);
 7 
 8     std::cerr << "-- my_function() called with " << argc
 9     << " arguments:" << std::endl;
10 
11     for ( int n=1; n<=argc; ++n ) {
12         std::cerr << "-- argument " << n << ": "
13             << lua_tostring(L, n) << std::endl;
14     }
15 
16     // return value
17     lua_pushnumber(L, 123);
18 
19     // number of return values
20     return 1;
21 }
22 
```

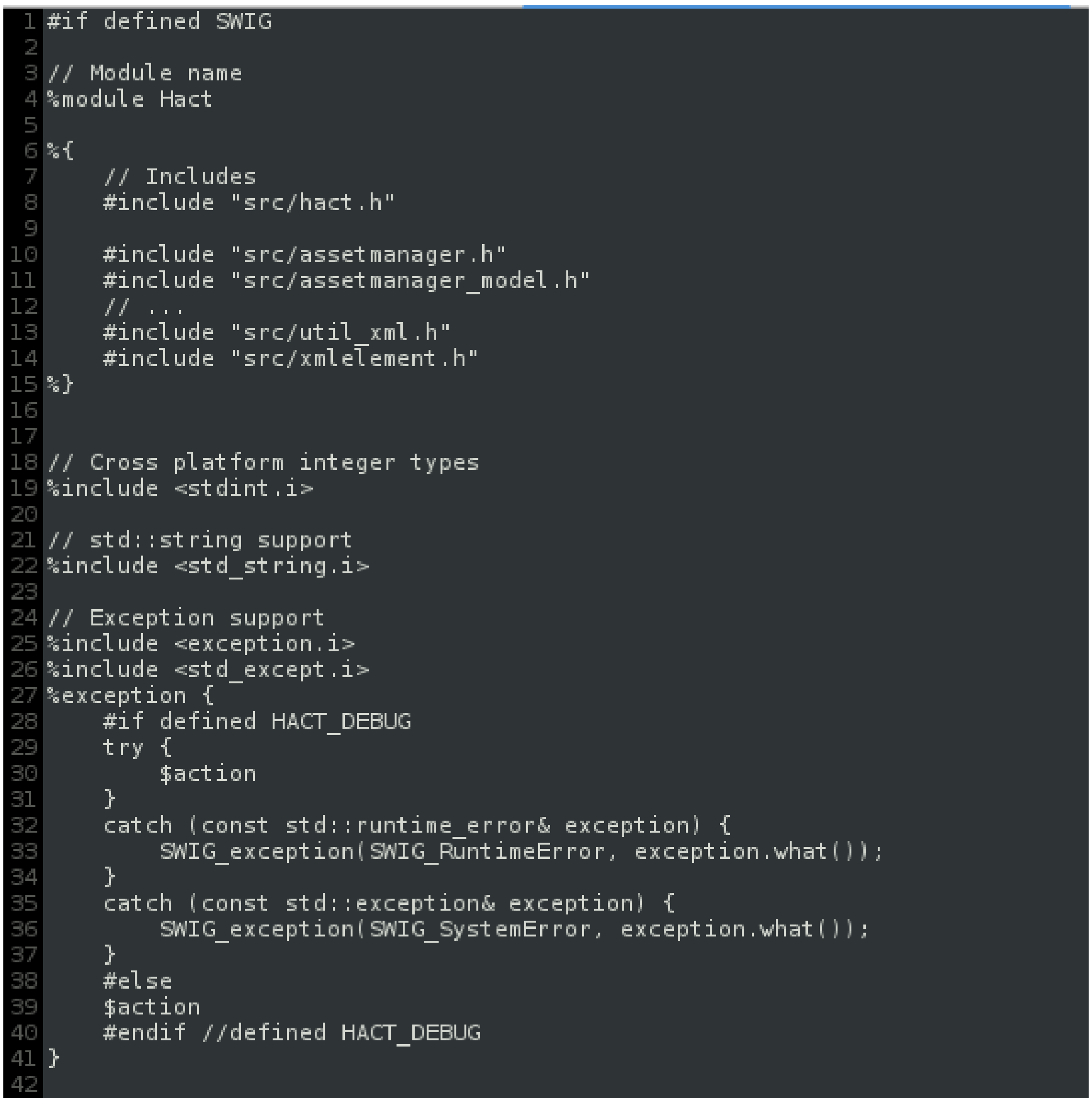
Picture 4. Lua C API example

Picture 4 shows an example of the usage of the C API; every value which must be usable from Lua is held in the stack. At line six, the first line of the function, lua_gettop returns the index of the top element in the stack, therefore it gives a count of the stack elements (Lowe, P.). Then the for loop prints each stack element and returns one number value to Lua. When writing an interface, the function will call an existing C or C++ function. This procedure will have to be repeated for every function in the application.

SWIG is used to generate the code for the interface, it automates most of the work and supports most features of C++. SWIG works by reading specified C++ header files or function declarations for C, which are introduced in an interface file which is used as the input to SWIG. The interface file can also include customizations to extend SWIG. (Swig.org 2015b)

Picture 5 shows an example of an interface file in HactEngine, which is given as input for the SWIG command:

swig -lua -c++ swig_hact.i



```
#if defined SWIG

// Module name
%module Hact

%{
    // Includes
    #include "src/hact.h"

    #include "src/assetmanager.h"
    #include "src/assetmanager_model.h"
    // ...
    #include "src/util_xml.h"
    #include "src/xmlelement.h"
%}


// Cross platform integer types
%include <stdint.i>

// std::string support
%include <std_string.i>

// Exception support
%include <exception.i>
%include <std_except.i>
%exception {
    #if defined HACT_DEBUG
    try {
        $action
    }
    catch (const std::runtime_error& exception) {
        SWIG_exception(SWIG_RuntimeError, exception.what());
    }
    catch (const std::exception& exception) {
        SWIG_exception(SWIG_SystemError, exception.what());
    }
    #else
    $action
    #endif //defined HACT_DEBUG
}

```

Picture 5. A part of the swig_hact.i file in HactEngine

After a successful command, SWIG generates a file named swig_hact_wrap.cxx, which can be compiled and linked with the application. The file contains static functions which use the C API functions like demonstrated in Picture 4. The file is not intended to be easy readable or modifiable, as it is optimized for efficiency and should be modified only through the interface file.

3.4 Packages

HactEngine is built modular, the developer should be able to extend or disable features easily. The idea is not only to keep the code base modular, but also to implement a package system. The implementation of packages is not yet ready, but the design is at the stage that new packages can be designed at a high-level. The reason to build such a system is simple, the easier it is to add new features by the community, the faster the community can grow. It will also limit the overload of the core features and keep the core code clean.

In theory, the packages can be written in any programming language. The only requirement is to have an interface for the scripting language of the game, which at present situation is Lua. SWIG is used to generate bindings for Lua when the package is written in C or C++. The preferred language to write packages is C++, because SWIG can be used to generate wrappers for almost any other scripting language and the performance is probably better with C++ than with a scripting language. Another option is to write it with Lua, as it is the default scripting language.

The best way to integrate a third-party library is by writing a C++ interface for the library and then generate a wrapper for the interface with SWIG. If the library is simple enough, the interface may not be needed. An advantage of the interface is a coherent documentation with HactEngine.

Every package will be compiled in to a dynamic library and loaded at run-time. Some issues to resolve for packages are how to handle the dependencies, test the packages and support automatic documentation. The documentation would be formatted and displayed like the documentation for HactEngine. Packages will have a package index in the internet to ensure good discoverability.

4 COMPARISON OF THE OPEN SOURCE LIBRARIES

There are not too many open source libraries for multiplayer game network development. Most of the libraries one will find are commercial, with different monetization models. The reason most likely is that a multiplayer game will have a need for dedicated servers and it is easy to build monetization around a full service.

The developer will get a ready SDK, tools and infrastructure, with no cost or very cheap cost to start. This ecosystem is either provided by a console manufacturer, digital distribution platform or other third party company. Where the first two will usually provide those free of charge and cover the expenses as a cut from the revenue of the game, since they let developers use a powerful ecosystem to sell games. The third party companies will raise the share based on the usage of their servers. An advantage is that these companies provide support for many platforms and integrations with previously introduced ecosystems.

All of those ready commercial services are great, they can simplify and speed up development. However this all comes with a price, not only money, but with code that the developer cannot modify or easily expand. If there is a bug or an innovative new idea, the developer will have to find a way around it or wait for the service provider to fix it.

Open source libraries will give a great flexibility, with – in many cases – a cost of longer development time. Open source libraries may not have support for all platforms and may have a lack of development resources and money. At the end, which of the two is the best depends on development resources, time, preferences or anything else. But there is no doubt that open source will give a better understanding about the technology and greater freedom of usage. From the perspective of HactEngine, it is preferred to use open source libraries, as it is a continuum for the flexibility of HactEngine.

4.1 Criteria

The purpose was to find versatile open source libraries, with similar features that would cover the most of common needs of multiplayer game development. After research it became evident that there is not much choice to get enough libraries for comparison. Either the libraries were outdated or they were limited by features, hence the criteria were changed.

Two libraries were selected with most of the basic features directed to multiplayer game development. The basic features to cover were client-server and peer-to-peer networking and a layer to ease network programming.

As an alternative, two libraries were selected that are designed for common network application development. These give another view into development and an opportunity to learn more. There were no strict requirements for features, because the most popular libraries will in any case support the common protocols and technologies. The idea was to find libraries to ease and speed up development.

All of the libraries needed to have references and a community. They needed to have support for different platforms, at least for Windows, Linux and Mac. Support for popular mobile platforms, Android and iOS, was good to have, because HactEngine will support these in the future. Development could have been done without any library, but the amount of work would be too big, as the library should support many different platforms.

4.2 Choosing Libraries

It has become clear it was hard to find multiplayer game libraries that have support for most of the common features in multiplayer games. The selection was made by searching the web and identifying which were the most recommended libraries.

The most versatile library, which was found is RakNet; it also had the most references from the game industry. It was found first and quickly became a case study for a very suitable library. Another library designed for game development is ENet, which was commonly compared with RakNet, but being more low-level.

The Libraries Boost.Asio and POCO were chosen as an alternative view angle to development. These were also often noted when searching the internet for multiplayer game development libraries. But they are also modern and well maintained as networking libraries.

4.3 RakNet

4.3.1 History

RakNet is the most high level of the compared libraries, it provides all of the features one will need for multiplayer game development. RakNet was started by Kevin Jenkins in the early 2000s, while he was in college. (Jenkins, K. 2015a)

An employment to nFusion Interactive led to the birth of the early version of RakNet. Kevin Jenkins founded Jenkins Software LLC for RakNet and the licenses for version 1.0 were sold for PC developers. Jenkins also released a restricted GPL licensed version to capture indie game developers. The open source community helped RakNet to grow by adding new features and fixes. (Jenkins, K. 2015a)

After 2009, Jenkins Software LLC began to get bigger customers like Sony Online Entertainment and NetDevil Ltd (Jenkins, K. 2015a). In 2014 RakNet was acquired by Oculus VR LLC and the whole library was published under modified open source BSD license, on GitHub (Oculus VR 2014).

4.3.2 Development and Community

Through time, RakNet has been used by many different game companies, be it a AAA studio, an indie game studio or something else. Unfortunately the development of RakNet seems to have stopped, after the RakNet source code was published on GitHub.

Although the development has stopped after the Oculus acquisition, the library is still maintained by different forks on GitHub. These forks have a lot of bug fixes and some enhancements to the code. The community keeps RakNet as modern and stable as it was before.

4.3.3 Features

RakNet is cross-platform, with support for Windows, PlayStation 3, PlayStation Vita, XBOX 360, Linux, Mac, the iPhone, Android, Google's Native Client, Windows Phone 8, and Windows Store 8. It is integrated into various game engines, like Havok (Vision Game Engine), Unity, Gamebryo, and Terminal Reality (Infernal Engine). RakNet is developed with C++ and it has bindings for C#. (Jenkins Software LLC 2015b)

RakNet has a communication layer which implements reliable and unreliable UDP communication, secure connections and packets with serialization and deserialization. It has support for voice and text-chat communications between players. RakNet also has support for remote procedure calls (RPC), which allows the developer to call native C and C++ procedures with automatically serialized parameter lists. (Jenkins Software LLC 2015a)

Players can be connected either with peer-to-peer or client-server. To connect players there is support for a lobby server, where players can create an account for the game. The lobby server uses a database to enable features like friends, rooms, quick match, ranking, email and multiple game titles with the same account. The lobby server needs a dedicated server, but RakNet also supports lobbies provided by Steam, PlayStation 3, XBOX 360 and Games for Windows Live. (Jenkins Software LLC 2015a)

4.4 ENet

4.4.1 Development and Community

There is no written history about ENet, but according to the MIT license of the library, it has been open source since year 2002. The ENet library is copyrighted by Lee Salzman. On his own homepage he states, that ENet was initially written for a game called Cube. Cube was released in 2001. (Salzman, L.; Salzman, L. 2005; Salzman, L. 2015c)

The source code for ENet can be found on Github. The library is not very actively developed anymore by the maintainer. But there are almost 200 forks and many people showing their interest for the library. ENet was mentioned and recommended frequently when searching the Internet for a game networking library. (Salzman, L. 2012)

There is a community developed wrapper for the Godot game engine and bindings for the programming languages Ruby and Lua (Corcoran, L. 2013, McLean, J. 2015; Vranish, J. 2015;). There are no known popular games which use ENet.

4.4.2 Features

The intention of ENet is to provide a simple communication layer on top of UDP. ENets specialty is an optional reliable UDP implementation. It does not have any high-level features like a lobby, serialization or authentication, which are provided by RakNet. (Salzman, L. 2015a)

ENet has support for Windows and Unix-like operating systems, like Linux, BSD and OS X (Salzman, L. 2015b). It is not officially supported on iOS and Android, but searching the Internet gave few successful attempts.

ENet provides some low-level features to provide reliable networking, these features include connection management, fragmentation and reassembly. It has support for client-server, but does not have an implementation for peer-to-peer.

As a summary, it only has very basic features compared to RakNet. To provide other features, those must be developed either with libraries like Boost.Asio and POCO or with standard tools provided by operating system or programming language.

4.5 Boost.Asio

Boost.Asio is part of the Boost C++ libraries, all Boost libraries are free, high-quality and portable. Most of the libraries are licensed by the open source Boost Software License. Boost libraries are often used as a base for new standard specifications of C++. It was started by members of Standards Committee Library Working Group, but today there are thousands of developers around the world. (Rivera, R; Dawes et al. 2016)

Boost.Asio was started by Christopher M. Kohlhoff in 2003. It is cross-platform, with support for all modern operating systems. It can be used for any kind of network application or asynchronous low-level I/O programming. Actually, the name “Asio” stands for “asynchronous input/output”, which means that operations can be executed concurrently. With Boost.Asio, concurrency is for interaction with outside of the program in some way, for example networks, files or serial ports. It focuses primarly on networking, but has been extended to include other resources also. (Schäling, B.; Kohlhoff, C. 2015)

The library provides support for common network protocols TCP, UDP and ICMP. It also has an interface for a low-level socket API. Boost.Asio provides SSL support to encrypt communication, it is based on the OpenSSL toolkit.

4.6 POCO

POCO is a collection of modern and cross-platform libraries for network- and Internet-based applications. The POCO libraries are written in modern C++ and are licensed under the Boost Software License. The project was started by Günter Obiltschnig in 2004 and has since gathered hundreds of contributors and a wide community of users. (Pocoproject.org 2016a)

POCO is designed to be modular and easy to understand. It features a lot more than just basic networking support. It has zlib-based compression and decompression, cryptography based on OpenSSL and platform-independent filesystem utilities. POCO also includes XML and JSON tools, unified database access and multithreading support. (Pocoproject.org 2016b)

For networking, POCO has support for many different protocols, in addition to usual TCP, UDP and WebSocket, POCO also has HTTP, FTP, SMTP and POP3. From the standpoint of multiplayer game programming, these other protocols are not vital, but useful features for larger implementations. POCO also has a caching framework to make frequently used data easy and fast to access. (Pocoproject.org 2016b)

4.7 The Comparison

All of the libraries are suitable to be used for multiplayer game development, but they all vary with features and have different intentions. RakNet is a full featured high-level library for easy and fast development and gives more time to the developer to concentrate on the game. ENet instead provides basic client-server functionality with robust UPD and optional reliable UDP. The developer can use services provided by Steam, consoles or third party, if there is need for a lobby-system or user authentication.

The choice between these depends basically on the need of the developer. If there is no need for all the features of RakNet, Enet is a perfect choice, as it is small sized and does not have any dependencies.

Boost.Asio and POCO give more freedom to implement multiplayer features, but the features must be written from the ground up by the developer. This could be the best case for in-house implementations, new innovative implementations or just for fun of learning.

It would be very interesting to create one from the ground up, but it would take a lot of time to develop, not to speak of making it stable across all platforms. HactEngine should be able to support many different features and be easy to use, in order to get people interested. While development of the engine itself has enough work, RakNet is the best choice for HactEngine. Also, HactEngine needs to target all games delivered outside of any distribution channel like Steam.

5 INTEGRATING RAKNET

RakNet is integrated as a package into HactEngine. The design of the first version of the package is described in this section. The interface will be designed to be simple and it will group features for more rapid development. The interface can also provide lower level functions, but it is preferred to use grouped features for better performance in the scripting language. The high-level simple interface can sometimes feel limited, so it should be easily extendable.

RakNet will be used for both the server and the client side. The server will have features like a lobby system or a master server to connect the clients. Depending on the game, the clients can be connected to a fully connected peer-to-peer mesh or use the server to run the game and keep the clients only as clients. For now, the peer-to-peer mesh is designed, but client-server can be added later.

5.1 Server

The server application should be so that it could be configured to support only the needed features. It should be installable to any server, be it Windows, Linux or OS X. There will be a need to build an environment with the needed dependencies and the server must be able to scale automatically on the fly. The server will not have an interface to the scripting language, but will have scripts for configuring and deploying.

The server can be used as a service to connect clients to a peer-to-peer mesh or to be an authoritative server for client-server. It would be hard to create a server application which could take into account every different game type, therefore the best that can be done is to support easy extending of the application. The designed version will support connecting clients to a peer-to-peer mesh.

5.1.1 Infrastructure

The deployment of the server could be done with a server container such as Docker. A server container wraps the software and its dependencies in a lightweight container, but shares the operating system kernel with others. Every container is isolated and can be run almost anywhere. A container is similar to a virtual machine, but does not run an entire guest operating system and is much more portable and efficient. (Docker 2016)

One server can contain multiple containers, but one server will not handle many thousands of concurrent users. There is need for a service called a load-balancer, which guides the clients to servers which have free resources. A script running on another server will monitor RakNet servers and a create new server for the load-balancer when all of the others are full. The script would use an API provided by a service provider and remove the unneeded servers automatically. Creating new servers depends on a cloud server provider, the easiest would be to use Google Cloud or Microsoft Azure.

The RakNet server will also need to be able to synchronize data between the servers, so that the player will not notice the scaling. RakNet provides a CloudServer/CloudClient plugin to automate this behavior. Automatically created servers will subscribe to a mesh of servers and synchronize data. An option for synchronizing data is to use a separate scalable database server, which is used by all load-balanced servers.

5.1.2 Features

A basic example of most of the following functionalities is presented as a code listing in Appendix 1. The server is initialized with a RakPeerInterface object, which is the main interface for network connections. The object will specify how many clients can connect to it, what port is used and whether there is need for secure connections. It will take care of communication to clients. Also, all

RakNet plugins used in an application, will be attached to the RakPeerInterface object. Plugins will serve different services for clients.

One common problem with connecting comes from NAT, the clients can not always be routed between each other. This can be fixed with NatPunchthroughServer, a plugin provided by RakNet. It will resolve the public IP address and port for each client. NatPunchthroughServer may have problems with about every five percent of the connections and it can vary a lot depending on the quality of routers. (Jenkins, K. 2015b)

RakNet has a service called NatTypeDetectionServer, which can be used by a client to detect if its router has NAT and which type of NAT it is. This information helps the client to select the connection method. For example, NatPunchthroughServer may not be able to complete, or may not be even needed for connecting to a specific client. The downside of the service is, that the server needs to have four external IP addresses pointing to it. Support for multiple IP addresses is not common from service providers, but may be requested from some providers. It may also in cur additional costs. (Jenkins, K. 2015c)

The UDPProxyServer plugin can be used as a fallback service for connecting. It will route packets transparently between the source and the destination client. The UDP proxy server can be CPU intensive and may need to have more than one dedicated server, as it is used even if the clients are connected through peer-to-peer. When there is need for several UDP proxy servers, they are coordinated with a server running RakNet with UDPProxyCoordinator. UDPProxyServer is almost an 100% guaranteed way to create connections, but should not be the preferred method, as it will increase costs greatly. (Jenkins, K. 2015b)

After a successful connection with a client, the server can start communicating with it. The server can run a RoomsPlugin instace to maintain a list of rooms created by clients and enables clients to join automatically or choose a specific room. The client must be added as logged in with the LoginRoomsParticipant function of the plugin, before they can make use of the service. The plugin supports multiple game titles. The RoomsPlugin is actually part of the Lobby2Server plugin, but can be used alone.

Lobby2Server provides support for database related features, like user accounts, ranking, friends and matchmaking. RakNet has built-in support for PostgreSQL databases with the Lobby2Server_PGSQL class, which is inherited from Lobby2Server. The lobby server can use RoomsPlugin or friend invites to connect clients together. Lobby2Server needs an administration client to configure it, the client will, for example, add the game title to the database. The server runs Lobby2MessageFactory, which creates database functionality for messages coming from a client (Jenkins, K. 2012).

The main loop of the server listens for packets coming from clients and the server functionality can be extended based on these messages. In the example code in Appendix 1, the server will give an anonymous username for a client when it connects to the server.

5.2 Client

On the client side RakNet will be integrated as a package for HactEngine. Like HactEngine, RakNet is written in C++, therefore an obvious way to integrate RakNet is by creating an interface with C++ and generate wrappers for a scripting language. The interface should support all available features, but the first version of the package will support at least the features described below.

5.2.1 Features

Appendix 2 and 3 will show some features needed for basic client to work with server described previously. The features in appendices will be used in finished implementation through the interface. The appendices introduce the Client class which works as an interface, but is not a complete presentation of the interface.

Like the server, a client uses a RakPeerInterface object as a base for all RakNet related operations and all the client side plugins are attached to it. The Start function of the Client class starts the networking interface for RakNet. The prepareConnection function will try to enable easier connectivity.

The NatTypeDetectionClient plugin can be used to detect the NAT type if the server is running NatTypeDetectionServer. The client can use information received by the plugin to find the best way to connect with another client. If there is NAT or NatTypeDetectionClient is not used, the client can try UPnP. UPnP (Universal Plug and Play) is a standard to allow devices to connect seamlessly together (The Open Connectivity Foundation 2016). RakNet can use it to configure a UPnP enabled router with port mapping, which allows clients to connect with each other.

UPnP is not always possible, if it fails there must be another way to configure port mapping. To eliminate a need on manually configuring port forwarding to the router, RakNet can use different plugins to bypass this problem. NatPunchthroughClient uses the server running the NatPunchthroughServer plugin to find a route to the other client. Currently, UPnP and NatPunchthrough cover almost all cases and should be enough for most developers.

There are still two more plugins to use as a fallback, Router2 and UDPProxyClient. Router2 uses a third client to route data between two clients. The downside of the plugin is that it will consume bandwidth and computing resources for the third client and therefore the connection can be slower. UDPProxyClient uses the server running a UDPProxyServer plugin to route its messages to the other client. The downside of this method is that UDPProxyServer will need dedicated servers. But it will give an absolute fallback connection between two clients.

The client will use RoomPlugin and Lobby2Client to find other players. Lobby2Plugin is used when the client wants to login to the game server and enable more services to enhance the game experience. A logged-in client can invite friends to play with and have better matchmaking based on the properties of another client. RoomPlugin can be used to create rooms with different properties or by using quick-join to connect automatically in a room with free slots. If there are no rooms, a new one will be created. One of the clients is a moderator for the room and can manage it. The room can be used for communication before starting the game. The communication can be either hidden from the end-user or a visible chat where the end-user can participate.

If the clients are supposed to connect in a peer-to-peer mesh, they will connect using a FullyConnectedMesh2 plugin. The server will give addresses to other clients that can be used to create the peer-to-peer mesh. The plugin will set one of the clients as the host, which should be responsible for the game session. If the host disconnects, the class will automatically migrate to a new host.

There are still a lot of other plugins available for RakNet, but the ones introduced here are the most vital for a multiplayer game. The design of implementing an interface for any plugin will follow the same style.

5.2.2 Interface

The Client class represents a simple version of an interface, the public functions will be available from the scripting language. The plugins are included inside the same interface, but must be explicitly enabled to be usable. Usage of the interface needs a Client object, which can take basic configuration values into the constructor.

The interface will group functions which are always used together for a feature. Some interface functions can control many plugins if they are enabled. For example the PrepareConnection function in Client will detect the NAT type, if the NatTypeDetectionClient plugin is enabled, otherwise it tries the UPnP approach.

While RakNet has good documentation, the interface will have its own documentation, because the interface will work a little differently and at a higher level. Because the interface is very high level, it is attractive by simplicity, but may negatively affect developers who want to do something different. This is a problem which must be taken into account in design. For example, the first version will only support peer-to-peer, but the developer can use the client-server model on top of it or the developer can be encouraged to add it into the package. Extensions by the community will also help by adding features into the official package. The interface can also include some lower level functions, which could lower the barrier to extend the features without any need for writing C++ code.

The interface does not need to depend on HactEngine in any way, it can be used separately anywhere. The only requirements are either support for C++ or support for a SWIG wrapper, which works with almost any language.

6 CONCLUSION

The theory of networked applications is very broad, therefore the theory in this thesis was limited to the most fundamental technologies which are vital to understand when developing multiplayer games. There would have also been a lot more multiplayer theory, but as the thesis was about the design of the integration of a network library, individual concepts were not so important.

The comparison of networking libraries helped me to learn more about different technologies and concepts there are in use for multiplayer games. There were several libraries like ENet, but no others like RakNet. The first problem was to find competitors for RakNet, because it was selected very early for the comparison.

Most of the comparable libraries were commercial, as was RakNet before 2014. Because there were no straight competitors, I chose to take another angle and compare with libraries for general network programming. Another good idea for the comparison part would have been to select commercial but comparable libraries with RakNet and compare their application program interfaces to find ideas about designing an interface for RakNet.

Open source libraries are widely used in game development, but commercial options for multiplayer games are providing very competitive ready-made solutions, with free-to-start monetization. The comparison helped me to realize this and gave an idea about designing RakNet integration with ready-made aspects, like for example, one-click deploy for the servers and a simple interface. The developer would still need to find servers and maybe preconfigure them, but usage of the package is easy to start.

The planning for this thesis was started a lot before the writing began while I was developing multiplayer for the Moomin Adventures: Jam Run mobile game. The multiplayer was chosen to be developed with a ready-made solution by Google, which was only for mobile development. At the time came the idea for the next game by Indium Games – currently named Infamis: Legends of the Arena – it will also have multiplayer, but is targeted for computer platforms.

The original purpose was to write this thesis about the integration of RakNet, but due the schedule it was not possible. I would have liked to design the interface at a more detailed level, but it was not possible, as the package system for HactEngine was not yet designed. Fortunately, I was able to describe some of the technologies that are important when creating a package for HactEngine and what is needed to get started with RakNet. What the interface could look like was presented in the Client class within the demo code of RakNet's basic functionality.

The thesis was an important part of creating a more deep understanding of multiplayer game developement and helped me to find good book references for an even deeper understanding. The thesis can be used as a reference when a developer is thinking, if they should consider some open source library to develop multiplayer game functionality.

REFERENCES

App Annie 2015. Special E3 Gaming Report: The Growth of Mobile & Online Multiplayer. Referenced: 20.01.2016 http://blog.appannie.com/app-annie-idc-gaming-report-2015-h1-e3-edition/

Cisco Systems 2014. Internet Protocols. Referenced: 08.05.2016 http://docwiki.cisco.com/wiki/Internet_Protocols

Corcoran, L. 2013. Lua bindings for ENet. Referenced: 21.04.2016 http://leafo.net/lua-enet/

Dawes, B.; Abrahams, D. & Rivera, R. 2016. Boost C++ Libraries. Referenced: 22.04.2016 http://www.boost.org/

Docker 2016. What is Docker? Referenced: 07.05.2016 https://www.docker.com/what-docker

Fiedler, G. 2008. What every programmer needs to know about game networking. Referenced: 31.03.2016 http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/

Fresh2refresh.com 2014. What is C language. Referenced: 23.04.2016 http://fresh2refresh.com/c-programming/c-language-history/

Glazer, J. & Madhav, S. 2015. Multiplayer Game Programming: Architecting Networked Games, Addison-Wesley Professional, Crawfordsville, Indiana, USA.

Jenkins Software LLC 2015a, Features. Referenced: 21.04.2016 http://www.raknet.com/features.html

Jenkins Software LLC 2015b, Supported Platforms. Referenced: 21.04.2016 http://www.raknet.com/platforms.html

Jenkins, K. 2015a. The history of RakNet. Referenced: 17.04.2016 http://www.jenkinssoftware.com/index.html

Jenkins, K. 2015b. NAT Punchthrough overview. Referenced: 29.04.2016 http://www.jenkinssoftware.com/raknet/manual/natpunchthrough.html

Jenkins, K. 2015c. NAT Type detection. Referenced: 29.04.2016 http://www.jenkinssoftware.com/raknet/manual/nattypedetection.html

Jenkins, K. 2012. RakNet::Lobby2Plugin Class Reference. Referenced: 07.05.2016 http://www.jenkinssoftware.com/raknet/manual/Doxygen/classRakNet_1_1Lobby2Plugin.html#ea2398d48dc769d0d88b7819601636c4

Kohlhoff, C. 2015. Rationale. Referenced: 22.04.2016 http://www.boost.org/doc/libs/1_60_0/doc/html/boost_asio/overview/rationale.html

Lecky-Thompson, G.W. 2008. Fundamentals of Network Game Development, Course Technology / Cengage Learning, Herndon, VA, USA.

Lowe, P. , Lua: Functions and Types: lua_gettop. Referenced: 23.04.2016 http://pgl.yoyo.org/luai/i/lua_gettop

Lua.org 2016. Lua: about. Referenced: 23.04.2016 https://www.lua.org/about.html

McLean, J. 2015. An ENet wrapper for Godot. Referenced: 21.04.2016 https://github.com/jrimclean/gdnet

Newzoo 2013. Global Games Market Report Infographics. Referenced: 20.01.2016 https://newzoo.com/insights/infographics/global-games-market-report-infographics-2013/

Nutaq Innovations , Mobile ad hoc networks (MANET). Referenced: 10.04.2016 http://www.nutaq.com/blog/mobile-ad-hoc-networks-manet

Oculus VR 2014. Announcing Oculus Connect, RakNet Open Source, and E3 2014 Awards. Referenced: 17.04.2016 https://www.oculus.com/en-us/blog/announcing-oculus-connect-raknet-open-source-and-e3-2014-awards/

Oki, E.; Rojas-Cessa, R. & Tatipamula, M. 2012. Advanced Internet Protocols, Services, and Applications, Wiley, Hoboken, NJ, USA.

OpenBSD , Network Address Translation. Referenced: 07.05.2016 http://www.openbsd.org/faq/pf/nat.html

Pocoproject.org 2016a. Overview. Referenced: 23.04.2016 http://pocoproject.org/index.html

Pocoproject.org 2016b. Features. Referenced: 23.04.2016 http://pocoproject.org/features.html

Postel, J. 1981. INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION. Referenced: 10.03.2016 https://tools.ietf.org/html/rfc791

Rivera, R. Community. Referenced: 22.04.2016 http://www.boost.org/community/

Salzman, L. 2015a. ENet: ENet. Referenced: 21.04.2016 http://enet.bespin.org/index.html

Salzman, L. 2015b. ENet: Installation. Referenced: 21.04.2016 http://enet.bespin.org/Installation.html

Salzman, L. 2015c. ENet: License. Referenced: 21.04.2016 http://enet.bespin.org/License.html

Salzman, L. 2012. ENet reliable UDP networking library. Referenced: 21.04.2016 https://github.com/lsalzman/enet

Salzman, L. 2005. CHANGE HISTORY LOG. Referenced: 21.04.2016 http://cubeengine.com/docs/history.html

Salzman, L. , Lee Salzman's Page of Random Stuff. Referenced: 21.04.2016 http://sauerbraten.org/lee/

Schäling, B. , Chapter 32. Boost.Asio. Referenced: 21.04.2016 http://theboostcpplibraries.com/boost.asio

Standard C++ Foundation 2016. Big Picture Issues, C++ FAQ. Referenced: 23.04.2016 https://isocpp.org/wiki/faq/big-picture

Swig.org 2015a. Executive Summary. Referenced: 23.04.2016 http://www.swig.org/exec.html

Swig.org 2015b. SWIG Tutorial. Referenced: 23.04.2016 http://www.swig.org/tutorial.html

The Open Connectivity Foundation 2016. About UPnP. Referenced: 29.04.2016 http://openconnectivity.org/upnp

Vranish, J. 2015. Ruby library for games networking. Referenced: 21.04.2016 https://github.com/Dahrkael/rENet

RakNet server demo code (raknetserver.cpp)

```
/*
The MIT License (MIT)

Copyright (c) 2016 Indium Games

Permission is hereby granted, free of charge, to any person obtaining a copy…
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to …
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in a…
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

#include "RakPeerInterface.h"

#include "RakNetTypes.h"
#include "BitStream.h"
#include "RakSleep.h"

// NAT punchthrough server and UDP proxy server related includes.
#include "NatPunchthroughServer.h"
#include "NatTypeDetectionServer.h"
#include "SocketLayer.h"
#include "UDPProxyCoordinator.h"
#include "UDPProxyServer.h"

// Lobby related includes.
#include "Lobby2Message.h"
#include "PGSQL/Lobby2Server_PGSQL.h"
#include "PGSQL/Lobby2Message_PGSQL.h"

#include "RoomsPlugin.h"

#include <string>
#include <iostream>
#include <map>
#include <vector>
#include <memory>

#include "commondefines.h"


#define CLIENT_PAIR RakNet::RakNetGUID, RakNet::RakString

int main(void) {
    printf("Starting the server.\n");

    // Used to identify anonymous ids;
    int anonymousIndex = 0;

    // Get peer instance to manage connections.
    RakNet::RakPeerInterface *peer = RakNet::RakPeerInterface::GetInstance();
    // Setting maximum number of clients to allow connect.
    peer->SetMaximumIncomingConnections(MAX_CLIENTS);

    RakNet::SocketDescriptor socketDescriptor(SERVER_PORT, 0);
    // Start a socket for the port.
    if (peer->Startup(MAX_CLIENTS, &socketDescriptor, 1) != RakNet::RAKNET_S…
        printf("Startup call failed\n");
        return 0;
    }
    else {
        printf("Started on port %i\n", socketDescriptor.port);
    }

    printf("Our guid is %s\n", peer->GetGuidFromSystemAddress(
                                RakNet::UNASSIGNED_SYSTEM_ADDRESS).ToStr…
    printf("Started on %s\n", peer->GetMyBoundAddress().ToString(true));



    // Object for NAT punchthrough server.
    RakNet::NatPunchthroughServer natPunchthroughServer;
    peer->AttachPlugin(&natPunchthroughServer);

    // Enable debug messages.
    std::unique_ptr<RakNet::NatPunchthroughServerDebugInterface_Printf>
                debugInterface(
                    new RakNet::NatPunchthroughServerDebugInterface_Printf()…
    natPunchthroughServer.SetDebugInterface(debugInterface.get());

    /*
    // NatTypeDetectionServer needs to have four ip-address pointing to this
    // same server. Because this isn't supported by that many cloud server
    // provider, it is commented out.
    // Object to detect NAT type of client's router.
    RakNet::NatTypeDetectionServer natTypeDetectionServer;
    peer->AttachPlugin(&natTypeDetectionServer);

    // Initialize NAT type detection server.
    char ipListStr[MAXIMUM_NUMBER_OF_INTERNAL_IDS][128];
    RakNet::SystemAddress ipList[MAXIMUM_NUMBER_OF_INTERNAL_IDS];
    for(int i=0; i < MAXIMUM_NUMBER_OF_INTERNAL_IDS; i++) {
        ipList[i].ToString(false, ipListStr[i]);
    }
    RakNet::SocketLayer::GetMyIP(ipList);

    // Start NAT type detection server.
    natTypeDetectionServer.Startup(ipListStr[1], ipListStr[2], ipListStr[3]);
    */



    // Create UDP proxy coordinator server, this will get request from
    // UDPProxyClient and relay it to one of the ProxyServer running servers.
    //
    // Normally coordinator is on an another server, now using in this same …
    // testing purposes.
    RakNet::UDPProxyCoordinator udpProxyCoordinator;
    // Set password for coordinator.
    udpProxyCoordinator.SetRemoteLoginPassword(COORDINATOR_PASSWORD);

    // Create UDP proxy server.
    // Also proxy servers are usually on seperated servers, this will forward
    // datagrams from clients where NAT punchthrough didn't work.
    RakNet::UDPProxyServer udpProxyServer;

    // Attach UDP proxy plugins to interface.
    peer->AttachPlugin(&udpProxyServer);
    peer->AttachPlugin(&udpProxyCoordinator);

    // Login proxy server ip-address to proxy coordinator, so it is available
    // for clients to start forwarding.
    // Using ip-address of this server, because the coordinator is on same s…
    udpProxyServer.LoginToCoordinator(
                            COORDINATOR_PASSWORD, peer->GetMyBoundAddress(…




    // Initialize the RoomsPlugin object.
    RakNet::RoomsPlugin roomsPluginServer;
    peer->AttachPlugin(&roomsPluginServer);

    // Set Game Title to connect clients to room in same game.
    roomsPluginServer.roomsContainer.AddTitle(GAME_TITLE);




    // Initialize Lobby2Server with PostgreSQL support.
    RakNet::Lobby2Server_PGSQL lobby2Server;
    peer->AttachPlugin(&lobby2Server);

    // Initialize message factory, it creates messages from message IDs.
    RakNet::Lobby2MessageFactory_PGSQL messageFactory;
    lobby2Server.SetMessageFactory(&messageFactory);

    // Set room plugin for lobby server.
    lobby2Server.SetRoomsPlugin(&roomsPluginServer);

    printf("Connecting to database\n");
    const char *connectionString = "user=postgres password=postgres \
                                        dbname=raknet hostaddr=127.0.0.1…

    if (lobby2Server.ConnectToDB(connectionString, 4) == false) {
        printf("Database connection failed\n");
        return 0;
    }

    printf("Lobby2Server started and waiting for connections\n");

    RakNet::Lobby2Server::ConfigurationProperties c;
    c.requiresEmailAddressValidationToLogin = false;
    c.requiresTitleToLogin                  = true;
    c.accountRegistrationRequiresCDKey      = false;
    c.accountRegistrationRequiredAgeYears   = 0;
    lobby2Server.SetConfigurationProperties(c);




    RakNet::Packet *packet;

    std::map <CLIENT_PAIR> freeClients;
    printf("\nStarting the main loop.\n");

    // Main loop
    while (1) {

        // Collect received packets.
        for (packet=peer->Receive(); packet ;
                peer->DeallocatePacket(packet), packet=peer->Receive()) {

            std::cout << "\n\nNew Packet from:" << packet->guid.g << std::en…

            switch (packet->data[0]) {
                case ID_REMOTE_DISCONNECTION_NOTIFICATION:
                    printf("A client has disconnected.\n");
                    break;
                case ID_CONNECTION_REQUEST_ACCEPTED:
                    printf("ID_CONNECTION_REQUEST_ACCEPTED\n");
                    break;
                case ID_REMOTE_CONNECTION_LOST:
                    printf("A client has lost the connection.\n");
                    break;
            case ID_REMOTE_NEW_INCOMING_CONNECTION:
                printf("A client has connected.\n");
                break;
            case ID_NEW_INCOMING_CONNECTION: {
                printf("A connection is incoming.\n");
                // Create temporary anonymous username.
                RakNet::RakString username;
                username.Set("AnonymousUser%i", anonymousIndex);
                anonymousIndex++;

                // Add to free clients list.
                freeClients.insert(std::pair<CLIENT_PAIR>(packet->guid, …

                RakNet::BitStream bsOut;
                // Use a BitStream to write a custom user message
                // Bitstreams are easier to use than sending casted stru…
                // and handles endian swapping automatically.
                bsOut.Write((RakNet::MessageID)ID_ANONYMOUS_USERNAME);
                bsOut.Write(username);
                peer->Send(
                    &bsOut,
                    HIGH_PRIORITY,
                    RELIABLE_ORDERED,
                    0,
                    packet->systemAddress,
                    false
                );

                roomsPluginServer.LoginRoomsParticipant(
                                username,
                                packet->systemAddress,
                                packet->guid,
                                RakNet::UNASSIGNED_SYSTEM_ADDRESS
                );
            } break;
            case ID_DISCONNECTION_NOTIFICATION:
                printf("A client has disconnected.\n");
                break;
            case ID_CONNECTION_LOST: {
                printf("A client lost the connection.\n");
                freeClients.erase(packet->guid);
            } break;
            case ID_ALREADY_CONNECTED:
                printf("ID_ALREADY_CONNECTED\n");
                break;
            case ID_INVALID_PASSWORD:
                printf("ID_INVALID_PASSWORD\n");
                break;
            case ID_NO_FREE_INCOMING_CONNECTIONS:
                printf("ID_NO_FREE_INCOMING_CONNECTIONS\n");
                break;
            case ID_CONNECTION_ATTEMPT_FAILED:
                printf("ID_CONNECTION_ATTEMPT_FAILED\n");
                break;
            case ID_CONNECTION_BANNED:
                printf("ID_CONNECTION_BANNED\n");
                break;
            case ID_IP_RECENTLY_CONNECTED:
                printf("ID_IP_RECENTLY_CONNECTED\n");
                break;
            case ID_INCOMPATIBLE_PROTOCOL_VERSION:
                printf("ID_INCOMPATIBLE_PROTOCOL_VERSION\n");
                break;
            default: {
                if (packet->data[0] < MESSAGE_ID_NAMES_SIZE) {
                    printf("Id not implemented: %s\n",
                                        MESSAGE_ID_NAMES[packet->dat…
                }
                else {
                        printf("Message with identifier %i has arrived.\n",
                                                                        packet->data…
                    }
                } break;
            }

            // Show a list of freeClients.
            std::cout << "Client List:\n";
            for (auto &client : freeClients) {
                std::cout << "\t" << client.first.g << " username: "
                                                << client.second << std::end…
            }
        }

        // This sleep keeps RakNet responsive
        RakSleep(30);
    }

    RakNet::RakPeerInterface::DestroyInstance(peer);

    return 0;
}
```

RakNet client demo code (client.h)

```
/*
The MIT License (MIT)

Copyright (c) 2016 Indium Games

Permission is hereby granted, free of charge, to any person obtaining a copy…
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to …
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in a…
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

#include "RakPeerInterface.h"

#include "RakNetTypes.h"
#include "MessageIdentifiers.h"
#include "BitStream.h"
#include "RakSleep.h"

#include "NatPunchthroughClient.h"
#include "NatTypeDetectionClient.h"

#include "FullyConnectedMesh2.h"
#include "TeamManager.h"
#include "HTTPConnection2.h"
#include "TCPInterface.h"
#include "ReadyEvent.h"
#include "RPC4Plugin.h"
#include "ReplicaManager3.h"
#include "NetworkIDManager.h"
#include "RoomsPlugin.h"

#include "upnphelper.h"
#include "commondefines.h"

#include <string>
#include <iostream>
#include <stdint.h>
#include <unistd.h>
#include <thread>
#include <chrono>
#include <mutex>
#include <vector>

#ifndef __CLIENT_H
#define __CLIENT_H

#define PLUGIN_PTR(plugin) std::unique_ptr<plugin, PluginDeleter<plugin>>

enum ClientState {
    CLIENT_CONNECTION_ERROR = -1,
    CLIENT_INITIALIZED = 0,
    CLIENT_STARTED,
    CLIENT_PREPARING_CONNECTION,
    CLIENT_PREPARED,
    CLIENT_CONNECTING_SERVER,
    CLIENT_CONNECTED
```

```
};

enum RoomState {
    ROOM_WAITING_PEERS,
    ROOM_READY,
    ROOM_CONNECTING,
    ROOM_FULLY_CONNECTED,
    ROOM_IN_MATCH
};

template <class T>
struct PluginDeleter {
    PluginDeleter() {}

    void operator()(T *plugin) const {
        printf("Deleted a RakNet plugin.\n");
        T::DestroyInstance(plugin);
    }
};

struct Room {
    Room() : state_(ROOM_WAITING_PEERS) {}
    ~Room() {}

    RakNet::RakNetGUID GetModerator() {
        return moderator_;
    }

    void SetModerator(RakNet::RakNetGUID moderator) {
        moderator_ = moderator;
    }

    std::vector<RakNet::RoomMemberDescriptor> GetMembers() {
        return roomMembers_;
    }

    void SetMembers(
            std::vector<RakNet::RoomMemberDescriptor> roomMembers) {
        roomMembers_ = roomMembers;
    }

    RoomState GetState() {
        return state_;
    }

    void SetState(RoomState state) {
        state_ = state;
    }

    std::vector<RakNet::RoomMemberDescriptor> roomMembers_;
    RakNet::RakNetGUID moderator_;

    // If the game allows spectators, this will wait for very everyone to be
    // ready, then the state is ROOM_READY.
    RoomState state_;
};

class Client : public RakNet::RoomsCallback {
public:

    Client(char const *serverAddress, unsigned short serverPort,
        unsigned short clientPort, char const *gameTitle,
        unsigned char maxPeers = 6);

    ~Client() {}

    void EnableFullyConnectedMesh();

    void EnableNatPunchthroughClient(bool enableDebug = false);

    void EnableNatTypeDetectionClient(char const *serverAddress,
                                      unsigned short serverPor…

    void EnableRoomsPlugin();

    bool Start();

    void PrepareConnection();

    bool Connect();

    void QuickGame(uint32_t timeout = 30000, unsigned short minimumPlayers =…
                                  unsigned short maximumPlayers = …

    void Update();

    int GetState() {
        return (int)state_;
    }

    const char * MyGuid() {
        return peer_->GetMyGUID().ToString();
    }

    bool IsRoomReady();

    void ConnectToModeratorInRoom();

private:

    void ConnectServerThread();

    void OpenUpnp();

    void OpenUpnpThread();

    void DetectNatThread();

    std::unique_ptr<std::recursive_mutex> mutex_;

    // Title of game, must be same as in server.
    char const *gameTitle_;

    // Local port to use for the client.
    unsigned short clientPort_;

    // Maximum count of peers at the same time.
    unsigned char maxPeers_;

    // State of the Client.
    ClientState state_;

    // Structure to keep current room.
    // TODO: Remember to update this on callbacks and notifications.
    std::unique_ptr<Room> room_;

    // My username for game.
    RakNet::RakString username_;

    // Address of the NAT type detection server.
    RakNet::SystemAddress natTypeServerAddress_;

    // Address and GUID for RakNet server.
    RakNet::AddressOrGUID serverAddressOrGuid_;

    // Get peer instance to manage connections.
    PLUGIN_PTR(RakNet::RakPeerInterface) peer_;

    // This is used to execute room commands in RoomsPlugin server.
    std::unique_ptr<RakNet::RoomsPlugin> roomsPlugin_;

    // Used to connect peers in a peer-to-peer mesh.
    PLUGIN_PTR(RakNet::FullyConnectedMesh2) fullyConnectedMesh2_;

    // This must be used for peer-to-peer if client is behind a NAT router.
    PLUGIN_PTR(RakNet::NatPunchthroughClient) natPunchthroughClient_;

    // This can be used to speed up the connecting of peers.
    PLUGIN_PTR(RakNet::NatTypeDetectionClient) natTypeDetectionClient_;

    // Type of NAT if detected.
    RakNet::NATTypeDetectionResult natType_;

    // Interface to print messages from NatPunchthroughClient.
    std::unique_ptr<RakNet::NatPunchthroughDebugInterface_Printf> debugInter…

public:
    // All available callbacks for RoomsPlugin.

    virtual void CreateRoom_Callback(const RakNet::SystemAddress &senderAddr…
                RakNet::CreateRoom_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }

    virtual void EnterRoom_Callback(const RakNet::SystemAddress &senderAddre…
                RakNet::EnterRoom_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }

    virtual void JoinByFilter_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::JoinByFilter_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }

    virtual void LeaveRoom_Callback(const RakNet::SystemAddress &senderAddre…
                RakNet::LeaveRoom_Func *callResult);

    virtual void GetInvitesToParticipant_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::GetInvitesToParticipant_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }

    virtual void SendInvite_Callback(const RakNet::SystemAddress &senderAddr…
                RakNet::SendInvite_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }

    virtual void AcceptInvite_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::AcceptInvite_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }

    virtual void StartSpectating_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::StartSpectating_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }
```

```

	virtual void StopSpectating_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::StopSpectating_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void GrantModerator_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::GrantModerator_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void ChangeSlotCounts_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::ChangeSlotCounts_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetCustomRoomProperties_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetCustomRoomProperties_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void GetRoomProperties_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::GetRoomProperties_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void ChangeRoomName_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::ChangeRoomName_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetHiddenFromSearches_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetHiddenFromSearches_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetDestroyOnModeratorLeave_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetDestroyOnModeratorLeave_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetReadyStatus_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetReadyStatus_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void GetReadyStatus_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::GetReadyStatus_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetRoomLockState_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetRoomLockState_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}


	virtual void GetRoomLockState_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::GetRoomLockState_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}


	virtual void AreAllMembersReady_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::AreAllMembersReady_Func *callResult);


	virtual void KickMember_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::KickMember_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}


	virtual void UnbanMember_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::UnbanMember_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}


	virtual void GetBanReason_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::GetBanReason_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}


	virtual void AddUserToQuickJoin_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::AddUserToQuickJoin_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}


	virtual void RemoveUserFromQuickJoin_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::RemoveUserFromQuickJoin_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}


	virtual void IsInQuickJoin_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::IsInQuickJoin_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}
```

```


    virtual void SearchByFilter_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::SearchByFilter_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }


    virtual void ChangeHandle_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::ChangeHandle_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }


    virtual void Chat_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::Chat_Func *callResult) {
        (void) senderAddress;
        callResult->PrintResult();
    }


    // Notifications due to other room members

    virtual void QuickJoinExpired_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::QuickJoinExpired_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void QuickJoinEnteredRoom_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::QuickJoinEnteredRoom_Notification *notification);

    virtual void RoomMemberStartedSpectating_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomMemberStartedSpectating_Notification *notificati…
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomMemberStoppedSpectating_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomMemberStoppedSpectating_Notification *notificati…
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void ModeratorChanged_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::ModeratorChanged_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void SlotCountsSet_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::SlotCountsSet_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void CustomRoomPropertiesSet_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::CustomRoomPropertiesSet_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomNameSet_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomNameSet_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void HiddenFromSearchesSet_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::HiddenFromSearchesSet_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomMemberReadyStatusSet_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomMemberReadyStatusSet_Notification *notification);

    virtual void RoomLockStateSet_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomLockStateSet_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomMemberKicked_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomMemberKicked_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomMemberHandleSet_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomMemberHandleSet_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomMemberLeftRoom_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomMemberLeftRoom_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomMemberJoinedRoom_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomMemberJoinedRoom_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomInvitationSent_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomInvitationSent_Notification *notification) {
        (void) senderAddress;
        notification->PrintResult();
    }

    virtual void RoomInvitationWithdrawn_Callback(
                const RakNet::SystemAddress &senderAddress,
                RakNet::RoomInvitationWithdrawn_Notification *notification) {
        (void) senderAddress;
```

```
            notification->PrintResult();
        }

        virtual void RoomDestroyedOnModeratorLeft_Callback(
                    const RakNet::SystemAddress &senderAddress,
                    RakNet::RoomDestroyedOnModeratorLeft_Notification *notificat…
            (void) senderAddress;
            notification->PrintResult();
        }

        virtual void Chat_Callback(
                    const RakNet::SystemAddress &senderAddress,
                    RakNet::Chat_Notification *notification) {
            (void) senderAddress;
            notification->PrintResult();
            printf("Chat=%s\nFiltered=%s\n", notification->chatMessage.C_String(…
                                        notification->filteredChatMessage.C_String()…
        }
    };

    #endif
```

RakNet client demo code (client.cpp)

```
/*
The MIT License (MIT)

Copyright (c) 2016 Indium Games

Permission is hereby granted, free of charge, to any person obtaining a copy…
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to …
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in a…
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

#include "client.h"

Client::Client(char const *serverAddress, unsigned short serverPort,
        unsigned short clientPort, char const *gameTitle,
        unsigned char maxPeers)
        : serverAddressOrGuid_(RakNet::SystemAddress(serverAddress, serverPo…
        clientPort_(clientPort), gameTitle_(gameTitle), maxPeers_(maxPeers),
        peer_(RakNet::RakPeerInterface::GetInstance()),
        natPunchthroughClient_(nullptr), natTypeDetectionClient_(nullptr),
        roomsPlugin_(nullptr), fullyConnectedMesh2_(nullptr),
        debugInterface_(nullptr),
        natType_(RakNet::NATTypeDetectionResult::NAT_TYPE_UNKNOWN),
        state_(CLIENT_INITIALIZED),
        mutex_(new std::recursive_mutex()) {
    printf("Constructing the client and configuring RakNet peer interface.\n…
}

/*
 * FullyConnectedMesh2 will connect clients to same peer-to-peer network and
 * manage host migration.
 */
void Client::EnableFullyConnectedMesh() {
    fullyConnectedMesh2_.reset(RakNet::FullyConnectedMesh2::GetInstance());
    peer_->AttachPlugin(fullyConnectedMesh2_.get());

    fullyConnectedMesh2_->SetConnectOnNewRemoteConnection(false, "");
    fullyConnectedMesh2_->SetAutoparticipateConnections(false);
}

/*
 * NAT punchthrough will help to connect clients when both are behind NAT.
 */
// TODO: There could be optional support to use different servers for NAT
// punchthrough.
void Client::EnableNatPunchthroughClient(bool enableDebug) {
    natPunchthroughClient_.reset(RakNet::NatPunchthroughClient::GetInstance(…
    peer_->AttachPlugin(natPunchthroughClient_.get());

    if (enableDebug) {
        debugInterface_.reset(
                            new RakNet::NatPunchthroughDebugInterface_Printf…
        natPunchthroughClient_->SetDebugInterface(debugInterface_.get());
    }
}

```

```
void Client::EnableNatTypeDetectionClient(char const *serverAddress,
                                          unsigned short serverPor…
    natTypeServerAddress_ = RakNet::SystemAddress(serverAddress, serverPort);

    natTypeDetectionClient_.reset(RakNet::NatTypeDetectionClient::GetInstanc…
    peer_->AttachPlugin(natTypeDetectionClient_.get());
}

/*
 * RoomsPlugin is used to connect clients for same match.
 */
void Client::EnableRoomsPlugin() {
    roomsPlugin_.reset(new RakNet::RoomsPlugin());

    // Initialize RoomsPlugin for client.
    roomsPlugin_->SetServerAddress(serverAddressOrGuid_.systemAddress);
    roomsPlugin_->SetRoomsCallback(this);

    peer_->AttachPlugin(roomsPlugin_.get());
}

/*
 * The Start function will start a socket in specified port.
 */
bool Client::Start() {
    RakNet::SocketDescriptor socketDescriptor(clientPort_, 0);

    // First parameter of Startup is allowed connections, when allowed
    // connection is either incoming or outgoing.
    if (peer_->Startup(maxPeers_, &socketDescriptor, 1)
                                                   != RakNet::RAKNET_STARTE…
        printf("Startup call failed.\n");
        return false;
    }

    printf("Your guid is %s\n", peer_->GetMyGUID().ToString());
    printf("Started on %s\n", peer_->GetMyBoundAddress().ToString(true));

    return true;
}

/*
 * The PrepareConnection function will either start NAT type detection thread
 * and try UPnP to open ports in router or jsut try UPnP.
 *
 * This is optional, but actually required if the server is in different LAN
 * than the clients.
 *
 * You will need to wait for this to be ready before continuing
 * (state_ == CLIENT_PREPARED).
 */
void Client::PrepareConnection() {
    state_ = CLIENT_PREPARING_CONNECTION;

    if (natTypeDetectionClient_) {
        printf("Trying to detect NAT, if it is found, open UPnP.\n");
        // Detect NAT type and try UPnP
        natTypeDetectionClient_->DetectNATType(natTypeServerAddress_);
        std::thread workerThread(&Client::DetectNatThread, this);
        workerThread.detach();
    }
    else {
        // Try UPnP.
        OpenUpnp();
    }
}

/*
 * Connect to the server.
 *
 * You will need to wait for this to be ready before continuing
 * (state_ == CLIENT_CONNECTED).
 */
bool Client::Connect() {
    state_ = CLIENT_CONNECTING_SERVER;

    // TODO: There is support for security (password/publicKey).

    // Connect to server running RakNet.
    auto connectAttempt = peer_->Connect(
        serverAddressOrGuid_.systemAddress.ToString(false),
        serverAddressOrGuid_.systemAddress.GetPort(),
        0,0
    );

    if (connectAttempt == RakNet::CONNECTION_ATTEMPT_STARTED) {
        printf("Connection attempt started.\n");
    }
    else {
        printf("Connecting attempt failed, with id: %i.\n", connectAttempt);
        return false;
    }

    serverAddressOrGuid_.rakNetGuid =
        peer_->GetGuidFromSystemAddress(serverAddressOrGuid_.systemAddress);


    std::thread workerThread(&Client::ConnectServerThread, this);
    workerThread.detach();

    return true;
}

/*
 * Try to quick join in a room. This will create a new room, if rooms not
 * availabe.
 */
void Client::QuickGame(uint32_t timeout, unsigned short minimumPlayers,
                                              unsigned short maximumPlayers) {
    if (!maximumPlayers) {
        maximumPlayers = minimumPlayers;
    }

    if (roomsPlugin_) {
        // Connect to quick game
        printf("\nAdd client to QuickJoin\n");

        RakNet::AddUserToQuickJoin_Func quickJoinFunction;
        quickJoinFunction.userName = username_.C_String();
        quickJoinFunction.networkedQuickJoinUser.timeout = timeout;
        quickJoinFunction.networkedQuickJoinUser.minimumPlayers = minimumPla…

        // This is used to set maximum count for QuickJoin rooms.
        quickJoinFunction.networkedQuickJoinUser.query.AddQuery_NUMERIC(
                                DefaultRoomColumns::GetColumnName(
                                    DefaultRoomColumns::TC_USED_PUBLIC_SLOTS…
                                maximumPlayers,
                                DataStructures::Table::QF_LESS_THAN
        );

        quickJoinFunction.gameIdentifier = gameTitle_;

        // Send command to the server.
        roomsPlugin_->ExecuteFunc(&quickJoinFunction);
    }
    else {
        printf("The RoomPlugin is not initialized.\n");
    }
```

```
}

/*
 * This receives messages from other peers and the server.
 *
 * You must run this once in a frame (e.g. main loop).
 */
void Client::Update() {
    RakNet::Packet *packet;

    for (packet=peer_->Receive(); packet;
            peer_->DeallocatePacket(packet), packet=peer_->Receive()) {

        std::cout << "New Packet from:" << packet->guid.g << std::endl;

        switch (packet->data[0]) {
            case ID_REMOTE_DISCONNECTION_NOTIFICATION:
                printf("A client has disconnected.\n");
                break;
            case ID_CONNECTION_REQUEST_ACCEPTED: {
                // We have successfully connected to another client.
                printf("ID_CONNECTION_REQUEST_ACCEPTED\n");
                if (fullyConnectedMesh2_) {
                    // TODO: Maybe should check if there is a room connecting
                    // and the sender is in members of the room.
                    fullyConnectedMesh2_->StartVerifiedJoin(packet->guid);
                }
            } break;
            case ID_REMOTE_CONNECTION_LOST:
                printf("A client has lost the connection.\n");
                break;
            case ID_REMOTE_NEW_INCOMING_CONNECTION:
                printf("A client has connected.\n");
                break;
            case ID_NEW_INCOMING_CONNECTION:
                printf("A connection is incoming.\n");
                break;
            case ID_DISCONNECTION_NOTIFICATION:
                printf("ID_NEW_INCOMING_CONNECTION\n");
                break;
            case ID_CONNECTION_LOST:
                // Couldn't deliver a reliable packet - i.e. the other
                // system was abnormally terminated.
                printf("ID_CONNECTION_LOST\n");
                break;
            case ID_ALREADY_CONNECTED:
                printf("ID_ALREADY_CONNECTED\n");
                break;
            case ID_INVALID_PASSWORD:
                printf("ID_INVALID_PASSWORD\n");
                break;
            case ID_NO_FREE_INCOMING_CONNECTIONS:
                printf("ID_NO_FREE_INCOMING_CONNECTIONS\n");
                break;
            case ID_CONNECTION_ATTEMPT_FAILED:
                printf("ID_CONNECTION_ATTEMPT_FAILED\n");
                break;
            case ID_CONNECTION_BANNED:
                printf("ID_CONNECTION_BANNED\n");
                break;
            case ID_IP_RECENTLY_CONNECTED:
                printf("ID_IP_RECENTLY_CONNECTED\n");
                break;
            case ID_INCOMPATIBLE_PROTOCOL_VERSION:
                printf("ID_INCOMPATIBLE_PROTOCOL_VERSION\n");
                break;
            case ID_NAT_TYPE_DETECTION_RESULT:
                printf("ID_NAT_TYPE_DETECTION_RESULT\n");
                break;

	virtual void StopSpectating_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::StopSpectating_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void GrantModerator_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::GrantModerator_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void ChangeSlotCounts_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::ChangeSlotCounts_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetCustomRoomProperties_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetCustomRoomProperties_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void GetRoomProperties_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::GetRoomProperties_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void ChangeRoomName_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::ChangeRoomName_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetHiddenFromSearches_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetHiddenFromSearches_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetDestroyOnModeratorLeave_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetDestroyOnModeratorLeave_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void SetReadyStatus_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::SetReadyStatus_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
	}

	virtual void GetReadyStatus_Callback(
			const RakNet::SystemAddress &senderAddress,
			RakNet::GetReadyStatus_Func *callResult) {
		(void) senderAddress;
		callResult->PrintResult();
```

```
        printf("\n\n");
    }
    peer_->DeallocatePacket(packet);
}

/*
 * Function that is used in a thread to establish connection to the server.
 */
void Client::ConnectServerThread() {
    RakNet::Packet *packet;

    bool connecting = true;
    while (connecting || username_.IsEmpty()) {

        if (state_ == CLIENT_CONNECTION_ERROR) {
            break;
        }

        // This sleep keeps RakNet responsive
        std::this_thread::sleep_for(std::chrono::milliseconds(30));

        if (connecting) {
            switch (peer_->GetConnectionState(serverAddressOrGuid_)) {
                case RakNet::ConnectionState::IS_CONNECTED: {
                    printf("RakNet::ConnectionState::IS_CONNECTED\n");
                    connecting = false;
                } break;
                case RakNet::ConnectionState::IS_PENDING:
                    //printf("RakNet::ConnectionState::IS_PENDING\n");
                    break;
                case RakNet::ConnectionState::IS_CONNECTING:
                    //printf("RakNet::ConnectionState::IS_CONNECTING\n");
                    break;
                case RakNet::ConnectionState::IS_DISCONNECTING: {
                    printf("RakNet::ConnectionState::IS_DISCONNECTING\n");
                    std::lock_guard<std::recursive_mutex> lock(*mutex_);
                    state_ = CLIENT_CONNECTION_ERROR;
                } break;
                case RakNet::ConnectionState::IS_SILENTLY_DISCONNECTING: {
                    printf("RakNet::ConnectionState::IS_SILENTLY_DISCONNECTI…
                    std::lock_guard<std::recursive_mutex> lock(*mutex_);
                    state_ = CLIENT_CONNECTION_ERROR;
                } break;
                case RakNet::ConnectionState::IS_DISCONNECTED: {
                    printf("RakNet::ConnectionState::IS_DISCONNECTED \n");
                    std::lock_guard<std::recursive_mutex> lock(*mutex_);
                    state_ = CLIENT_CONNECTION_ERROR;
                    } break;
                case RakNet::ConnectionState::IS_NOT_CONNECTED: {
                    printf("RakNet::ConnectionState::IS_NOT_CONNECTED \n");
                    std::lock_guard<std::recursive_mutex> lock(*mutex_);
                    state_ = CLIENT_CONNECTION_ERROR;
                } break;
            }
        }

        // Collect received packets.
        for (packet=peer_->Receive(); packet;
                    peer_->DeallocatePacket(packet), packet=peer_->Receive()…

            std::cout << "New Packet from:" << packet->guid.g << std::endl;

            switch (packet->data[0]) {
                case ID_ANONYMOUS_USERNAME: {
                    printf("ID_ANONYMOUS_USERNAME ");
                    RakNet::BitStream bitStream(
                                        packet->data, packet->length, false);

                    // Ignore message id.
                    bitStream.IgnoreBytes(1);

                    std::lock_guard<std::recursive_mutex> lock(*mutex_);
                    bitStream.Read(username_);

                    printf("  %s\n", username_.C_String());
                } break;
            }
            printf("\n\n");
        }
    }

    if (state_ != CLIENT_CONNECTION_ERROR) {
        std::lock_guard<std::recursive_mutex> lock(*mutex_);

        printf("Connected to the server:\n");
        printf("\tSystemAddress: %s\n",
                        serverAddressOrGuid_.systemAddress.ToString(…

        serverAddressOrGuid_.rakNetGuid =
            peer_->GetGuidFromSystemAddress(serverAddressOrGuid_.systemAddre…
        printf("\tRakNetGUID: %s\n",
                            serverAddressOrGuid_.rakNetGuid.ToString…
        state_ = CLIENT_CONNECTED;
    }
    peer_->DeallocatePacket(packet);
}

/*
 * This starts a thread for UPnP opening.
 *
 * If UPnP does not work, you will have to use either NAT punchthrough,
 * Router2 or UDP Proxy when connecting to other clients. It is preferred to
 * use at least NAT punchthrough, because it works in most situations. If th…
 * is no fallback with Router2 or UDP Proxy, you will have to tell the end u…
 * to manually configure and open ports in router.
 */
void Client::OpenUpnp() {
    printf("Discovering UPnP.\n");
    std::thread workerThread(&Client::OpenUpnpThread, this);
    workerThread.detach();
}

/*
 * Function used in a thread to open ports from the router with UPnP.
 */
void Client::OpenUpnpThread() {
    DataStructures::List<RakNet::RakNetSocket2 *> sockets;
    peer_->GetSockets(sockets);

    bool success = UPNPOpen(sockets[0]->GetBoundAddress().GetPort(), 2000);

    std::lock_guard<std::recursive_mutex> lock(*mutex_);
    if (success) {
        printf("UPnP is supported, created port mapping.\n");
        natType_ = RakNet::NATTypeDetectionResult::NAT_TYPE_SUPPORTS_UPNP;
    }
    state_ = CLIENT_PREPARED;
}

/*
 * Function for a thread to detect existence or the type of NAT.
 */
void Client::DetectNatThread() {
    RakNet::Packet *packet;

    bool waitingResult = true;

    while (state_ == CLIENT_PREPARING_CONNECTION && waitingResult) {
        // Collect received packets from the server.
        for (packet=peer_->Receive(); packet;
                peer_->DeallocatePacket(packet), packet=peer_->Receive()…

            std::cout << "New Packet from:" << packet->guid.g << std::endl;

            switch (packet->data[0]) {
                case ID_NAT_TYPE_DETECTION_RESULT: {
                    // This will be received only, if natTypeDetectionClient…
                    // is used.
                    printf("DetectNat   ID_NAT_TYPE_DETECTION_RESULT\n");
                    natType_ = (RakNet::NATTypeDetectionResult)packet->data[…
                    printf("NAT Type is %s (%s)\n",
                        RakNet::NATTypeDetectionResultToString(natType_),
                        RakNet::NATTypeDetectionResultToStringFriendly(natTy…

                    if (natType_ != RakNet::NAT_TYPE_NONE) {
                        printf("You are behind a NAT router.\n");
                        OpenUpnp();
                    }
                    else {
                        printf("There is no NAT between the server and clien…
                        std::lock_guard<std::recursive_mutex> lock(*mutex_);
                        state_ = CLIENT_PREPARED;
                    }
                    waitingResult = false;
                    break;
                }
            }
            printf("\n\n");
        }

        // This sleep keeps RakNet responsive
        std::this_thread::sleep_for(std::chrono::milliseconds(30));
    }
    peer_->DeallocatePacket(packet);
}

/*
 * Check if there is a room ready.
 */
bool Client::IsRoomReady() {
    if (room_ && room_->GetState() == ROOM_READY) {
        return true;
    }

    return false;
}

/*
 * If this client isn't the moderator, it will try to connect to the moderat…
 */
void Client::ConnectToModeratorInRoom() {
    if (room_ && room_->GetState() == ROOM_READY && fullyConnectedMesh2_) {
        mutex_->lock();
        room_->SetState(ROOM_CONNECTING);
        mutex_->unlock();

        peer_->SetMaximumIncomingConnections(maxPeers_);

        // The host in peer-to-peer isn't the moderator of a room, it is the…
        // with the smallest time counting from this reset.
        fullyConnectedMesh2_->ResetHostCalculation();

        if (peer_->GetMyGUID() != room_->GetModerator()) {
            auto roomMembers = room_->GetMembers();
            for (auto &member : roomMembers) {
                if (member.guid == room_->GetModerator()) {
                    printf("Connecting to the moderator:\n");
                    printf("\t\tGUID: %s\n", member.guid.ToString());
                    printf("\t\tAddress: %s\n", member.systemAddress.ToStrin…

                    // Connect to the moderator.
                    if (natPunchthroughClient_) {
                        bool openNat = natPunchthroughClient_->OpenNAT(
                            member.guid, serverAddressOrGuid_.systemAddress);
                        if (openNat) {
                            printf("OpenNAT started.\n");
                        }
                        else {
                            printf("Failed to start OpneNAT.\n");
                        }
                    }
                    else {
                        auto connectAttempt = peer_->Connect(
                            member.systemAddress.ToString(false),
                            member.systemAddress.GetPort(),
                            0, 0
                        );
                        if (connectAttempt
                                        == RakNet::CONNECTION_ATTEMPT_STARTE…
                            printf("Connection attempt started.\n");
                        }
                        else {
                            printf("Connecting attempt failed, with id: %i.\…
                                                                    connectAttem…
                        }
                    }
                }
            }
        }
    }
    else if (!room_) {
        printf("There is now room to connect to.\n");
    }
    else if (!fullyConnectedMesh2_) {
        printf("FullyConnectedMesh2 is not enabled.\n");
    }
    else {
        printf("The room participants are not ready.\n");
    }
}


/*
 * Callback when leaving a room, will reset the member variable containing
 * a room.
 */
void Client::LeaveRoom_Callback(const RakNet::SystemAddress &senderAddress,
                                        RakNet::LeaveRoom_Func *callResult) {
    (void) senderAddress;
    callResult->PrintResult();
    std::lock_guard<std::recursive_mutex> lock(*mutex_);
    room_.reset(nullptr);
}

/*
 * When entered in a room after calling QuickJoin function with the RoomPlug…
 */
void Client::QuickJoinEnteredRoom_Callback(
            const RakNet::SystemAddress &senderAddress,
            RakNet::QuickJoinEnteredRoom_Notification *notification) {
    (void) senderAddress;
    notification->PrintResult();

    mutex_->lock();
    room_.reset(new Room());

    DataStructures::List<RakNet::RoomMemberDescriptor> roomMembers =
            notification->joinedRoomResult.roomDescriptor.roomMemberList;

    std::vector<RakNet::RoomMemberDescriptor> roomMembersVector;

    if (roomMembers.Size() > 0) {
        printf("Room members: \n");
        for (size_t i = 0; i < roomMembers.Size(); i++) {
            printf("\n\t%s\n", roomMembers[i].name.C_String());
            roomMembersVector.push_back(roomMembers[i]);

            switch (roomMembers[i].roomMemberMode) {
                case RMM_MODERATOR: {
                    printf("\tRMM_MODERATOR\n");
                    room_->SetModerator(roomMembers[i].guid);
                } break;
                case RMM_PUBLIC:
                    printf("\tRMM_PUBLIC\n");
                    break;
                case RMM_RESERVED:
                    printf("\tRMM_RESERVED\n");
                    break;
                case RMM_SPECTATOR_PUBLIC:
                    printf("\tRMM_SPECTATOR_PUBLIC\n");
                    break;
                case RMM_SPECTATOR_RESERVED:
                    printf("\tRMM_SPECTATOR_RESERVED\n");
                    break;
                case RMM_ANY_PLAYABLE:
                    printf("\tRMM_ANY_PLAYABLE\n");
                    break;
                case RMM_ANY_SPECTATOR:
                    printf("\tRMM_ANY_SPECTATOR\n");
                    break;
                default:
                    printf("\tMode: %i\n", roomMembers[i].roomMemberMode);
                    break;
            }
        }
    }

    // Save the room members in the room struct.
    room_->SetMembers(roomMembersVector);
    mutex_->unlock();

    // Set I'm ready to suggest others to start a match.
    RakNet::SetReadyStatus_Func setReadyFunction;
    setReadyFunction.userName = username_;
    setReadyFunction.isReady = true;

    roomsPlugin_->ExecuteFunc(&setReadyFunction);
}

/*
 * Callback when a member of the room chages its status.
 */
void Client::RoomMemberReadyStatusSet_Callback(
            const RakNet::SystemAddress &senderAddress,
            RakNet::RoomMemberReadyStatusSet_Notification *notification) {
    (void) senderAddress;
    notification->PrintResult();

    // Ask if everyone is ready.
    // NOTE: You can get ready and unready members from notification.
    RakNet::AreAllMembersReady_Func areAllMembersReady;
    areAllMembersReady.userName = username_;
    roomsPlugin_->ExecuteFunc(&areAllMembersReady);
}

```

```
/*
 * Callback for AreAllMembersReady_Func function. If everybody are ready set
 * the state of the room to ready.
 */
void Client::AreAllMembersReady_Callback(
                                const RakNet::SystemAddress &senderAddress,
                                RakNet::AreAllMembersReady_Func *callResult)…
    (void) senderAddress;
    callResult->PrintResult();

    if (callResult->allReady && room_) {
        std::lock_guard<std::recursive_mutex> lock(*mutex_);
        room_->SetState(ROOM_READY);
    }
    else if (!room_) {
        printf("Very weird, the room is nullptr.\n");
    }
}
```

RakNet client demo code (raknetclient.cpp)

```
/*
The MIT License (MIT)

Copyright (c) 2016 Indium Games

Permission is hereby granted, free of charge, to any person obtaining a copy…
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to …
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in a…
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

#include <string>
#include <iostream>
#include <stdint.h>
#include <unistd.h>

#include "client.h"

int main(int argc, char *argv[]) {
    unsigned short clientPort = CLIENT_PORT;
    unsigned short minimumPlayers = 2;
    char *serverAddress = SERVER_ADDRESS;
    int opt;

    while ((opt = getopt (argc, argv, "p:m:s:")) != -1) {
        switch (opt) {
            case 'p':
                clientPort = (unsigned short)atoi(optarg);
                break;
            case 'm':
                minimumPlayers = (unsigned short)atoi(optarg);
                break;
            case 's':
                serverAddress = optarg;
                break;
            case '?':
                if (optopt == 'p' || optopt == 'n' || optopt == 's') {
                    fprintf (stderr, "Option -%c requires an argument.\n", o…
                }
                else if (isprint (optopt)) {
                    fprintf (stderr, "Unknown option `-%c'.\n", optopt);
                }
                else {
                    fprintf (stderr, "Unknown option character `\\x%x'.\n", …
                }

                return 1;
            default:
                break;
        }
    }

    // At the beginning we allow as many concurrent connections as it is pos…
    // through out the game.
    // peersInMatch * (incoming + outgoing) + theServer * (incoming + outgoi…
    unsigned char maxPeers = minimumPlayers*2 + 1*2;
    Client client(serverAddress, SERVER_PORT, clientPort, GAME_TITLE, maxPee…

    // Not in use.
    //client.SetNatTypeServer(serverAddress, SERVER_PORT);

    // This should be used always, only if the server and clients are in the
    // same LAN this isn't needed.
    client.EnableNatPunchthroughClient(true);

    client.EnableFullyConnectedMesh();

    client.EnableRoomsPlugin();

    if (!client.Start()) {
        return 0;
    }

    // This should be run if you don't now if some one is behind a NAT route…
    // So almost everytime.
    client.PrepareConnection();

    while (client.GetState() == CLIENT_PREPARING_CONNECTION) {
        /* code */
    }

    if (!client.Connect()) {
        return 0;
    }

    while (client.GetState() == CLIENT_CONNECTING_SERVER) {
        /* code */
    }

    if (client.GetState() == CLIENT_CONNECTION_ERROR) {
        return 0;
    }

    // We will have exact count of players, as maximum is equal to
    // minimumPlayers. You can set maximum count parameter, if the room can …
    // variable count if players.
    client.QuickGame(120000, minimumPlayers);


    printf("\nStarting the main loop.\n");

    // Main loop
    while (1) {

        // Collect received packets.
        client.Update();

        if(client.IsRoomReady()) {
            client.ConnectToModeratorInRoom();
        }

        // This sleep keeps application responsive.
        std::this_thread::sleep_for(std::chrono::milliseconds(30));
    }

    return 0;
}
```

RakNet client demo code (commondefines.h)

```
/*
The MIT License (MIT)

Copyright (c) 2016 Indium Games

Permission is hereby granted, free of charge, to any person obtaining a copy…
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to …
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in a…
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

#include "MessageIdentifiers.h"

#ifndef __COMMONDEFINES_H
#define __COMMONDEFINES_H

#define CLIENT_PORT 61000
#define SERVER_PORT 60000
#define SERVER_ADDRESS "localhost"

#define CLIENTS_PER_MATCH 2
#define GAME_TITLE "Game Title"

#define MAX_CLIENTS 512
#define COORDINATOR_PASSWORD "Dummy Coordinator Password"

#define MESSAGE_ID_NAMES_SIZE 135

enum HactMessageIdTypes {
    ID_ANONYMOUS_USERNAME = ID_USER_PACKET_ENUM,
    ID_MYPROJECT_MSG_2
};


static const char *MESSAGE_ID_NAMES[MESSAGE_ID_NAMES_SIZE] = {
    "ID_CONNECTED_PING",
    "ID_UNCONNECTED_PING",
    "ID_UNCONNECTED_PING_OPEN_CONNECTIONS",
    "ID_CONNECTED_PONG",
    "ID_DETECT_LOST_CONNECTIONS",
    "ID_OPEN_CONNECTION_REQUEST_1",
    "ID_OPEN_CONNECTION_REPLY_1",
    "ID_OPEN_CONNECTION_REQUEST_2",
    "ID_OPEN_CONNECTION_REPLY_2",
    "ID_CONNECTION_REQUEST",
    "ID_REMOTE_SYSTEM_REQUIRES_PUBLIC_KEY",
    "ID_OUR_SYSTEM_REQUIRES_SECURITY",
    "ID_PUBLIC_KEY_MISMATCH",
    "ID_OUT_OF_BAND_INTERNAL",
    "ID_SND_RECEIPT_ACKED",
    "ID_SND_RECEIPT_LOSS",
    "ID_CONNECTION_REQUEST_ACCEPTED",
    "ID_CONNECTION_ATTEMPT_FAILED",
    "ID_ALREADY_CONNECTED",
    "ID_NEW_INCOMING_CONNECTION",
    "ID_NO_FREE_INCOMING_CONNECTIONS",
        "ID_DISCONNECTION_NOTIFICATION",
        "ID_CONNECTION_LOST",
        "ID_CONNECTION_BANNED",
        "ID_INVALID_PASSWORD",
        "ID_INCOMPATIBLE_PROTOCOL_VERSION",
        "ID_IP_RECENTLY_CONNECTED",
        "ID_TIMESTAMP",
        "ID_UNCONNECTED_PONG",
        "ID_ADVERTISE_SYSTEM",
        "ID_DOWNLOAD_PROGRESS",
        "ID_REMOTE_DISCONNECTION_NOTIFICATION",
        "ID_REMOTE_CONNECTION_LOST",
        "ID_REMOTE_NEW_INCOMING_CONNECTION",
        "ID_FILE_LIST_TRANSFER_HEADER",
        "ID_FILE_LIST_TRANSFER_FILE",
        "ID_FILE_LIST_REFERENCE_PUSH_ACK",
        "ID_DDT_DOWNLOAD_REQUEST",
        "ID_TRANSPORT_STRING",
        "ID_REPLICA_MANAGER_CONSTRUCTION",
        "ID_REPLICA_MANAGER_SCOPE_CHANGE",
        "ID_REPLICA_MANAGER_SERIALIZE",
        "ID_REPLICA_MANAGER_DOWNLOAD_STARTED",
        "ID_REPLICA_MANAGER_DOWNLOAD_COMPLETE",
        "ID_RAKVOICE_OPEN_CHANNEL_REQUEST",
        "ID_RAKVOICE_OPEN_CHANNEL_REPLY",
        "ID_RAKVOICE_CLOSE_CHANNEL",
        "ID_RAKVOICE_DATA",
        "ID_AUTOPATCHER_GET_CHANGELIST_SINCE_DATE",
        "ID_AUTOPATCHER_CREATION_LIST",
        "ID_AUTOPATCHER_DELETION_LIST",
        "ID_AUTOPATCHER_GET_PATCH",
        "ID_AUTOPATCHER_PATCH_LIST",
        "ID_AUTOPATCHER_REPOSITORY_FATAL_ERROR",
        "ID_AUTOPATCHER_CANNOT_DOWNLOAD_ORIGINAL_UNMODIFIED_FILES",
        "ID_AUTOPATCHER_FINISHED_INTERNAL",
        "ID_AUTOPATCHER_FINISHED",
        "ID_AUTOPATCHER_RESTART_APPLICATION",
        "ID_NAT_PUNCHTHROUGH_REQUEST",
        "ID_NAT_CONNECT_AT_TIME",
        "ID_NAT_GET_MOST_RECENT_PORT",
        "ID_NAT_CLIENT_READY",
        "ID_NAT_TARGET_NOT_CONNECTED",
        "ID_NAT_TARGET_UNRESPONSIVE",
        "ID_NAT_CONNECTION_TO_TARGET_LOST",
        "ID_NAT_ALREADY_IN_PROGRESS",
        "ID_NAT_PUNCHTHROUGH_FAILED",
        "ID_NAT_PUNCHTHROUGH_SUCCEEDED",
        "ID_READY_EVENT_SET",
        "ID_READY_EVENT_UNSET",
        "ID_READY_EVENT_ALL_SET",
        "ID_READY_EVENT_QUERY",
        "ID_LOBBY_GENERAL",
        "ID_RPC_REMOTE_ERROR",
        "ID_RPC_PLUGIN",
        "ID_FILE_LIST_REFERENCE_PUSH",
        "ID_READY_EVENT_FORCE_ALL_SET",
        "ID_ROOMS_EXECUTE_FUNC",
        "ID_ROOMS_LOGON_STATUS",
        "ID_ROOMS_HANDLE_CHANGE",
        "ID_LOBBY2_SEND_MESSAGE",
        "ID_LOBBY2_SERVER_ERROR",
        "ID_FCM2_NEW_HOST",
        "ID_FCM2_REQUEST_FCMGUID",
        "ID_FCM2_RESPOND_CONNECTION_COUNT",
        "ID_FCM2_INFORM_FCMGUID",
        "ID_FCM2_UPDATE_MIN_TOTAL_CONNECTION_COUNT",
        "ID_FCM2_VERIFIED_JOIN_START",
        "ID_FCM2_VERIFIED_JOIN_CAPABLE",
        "ID_FCM2_VERIFIED_JOIN_FAILED",
    "ID_FCM2_VERIFIED_JOIN_ACCEPTED",
    "ID_FCM2_VERIFIED_JOIN_REJECTED",
    "ID_UDP_PROXY_GENERAL",
    "ID_SQLite3_EXEC",
    "ID_SQLite3_UNKNOWN_DB",
    "ID_SQLLITE_LOGGER",
    "ID_NAT_TYPE_DETECTION_REQUEST",
    "ID_NAT_TYPE_DETECTION_RESULT",
    "ID_ROUTER_2_INTERNAL",
    "ID_ROUTER_2_FORWARDING_NO_PATH",
    "ID_ROUTER_2_FORWARDING_ESTABLISHED",
    "ID_ROUTER_2_REROUTED",
    "ID_TEAM_BALANCER_INTERNAL",
    "ID_TEAM_BALANCER_REQUESTED_TEAM_FULL",
    "ID_TEAM_BALANCER_REQUESTED_TEAM_LOCKED",
    "ID_TEAM_BALANCER_TEAM_REQUESTED_CANCELLED",
    "ID_TEAM_BALANCER_TEAM_ASSIGNED",
    "ID_LIGHTSPEED_INTEGRATION",
    "ID_XBOX_LOBBY",
    "ID_TWO_WAY_AUTHENTICATION_INCOMING_CHALLENGE_SUCCESS",
    "ID_TWO_WAY_AUTHENTICATION_OUTGOING_CHALLENGE_SUCCESS",
    "ID_TWO_WAY_AUTHENTICATION_INCOMING_CHALLENGE_FAILURE",
    "ID_TWO_WAY_AUTHENTICATION_OUTGOING_CHALLENGE_FAILURE",
    "ID_TWO_WAY_AUTHENTICATION_OUTGOING_CHALLENGE_TIMEOUT",
    "ID_TWO_WAY_AUTHENTICATION_NEGOTIATION",
    "ID_CLOUD_POST_REQUEST",
    "ID_CLOUD_RELEASE_REQUEST",
    "ID_CLOUD_GET_REQUEST",
    "ID_CLOUD_GET_RESPONSE",
    "ID_CLOUD_UNSUBSCRIBE_REQUEST",
    "ID_CLOUD_SERVER_TO_SERVER_COMMAND",
    "ID_CLOUD_SUBSCRIPTION_NOTIFICATION",
    "ID_LIB_VOICE",
    "ID_RELAY_PLUGIN",
    "ID_NAT_REQUEST_BOUND_ADDRESSES",
    "ID_NAT_RESPOND_BOUND_ADDRESSES",
    "ID_FCM2_UPDATE_USER_CONTEXT",
    "ID_RESERVED_3,",
    "ID_RESERVED_4",
    "ID_RESERVED_5",
    "ID_RESERVED_6",
    "ID_RESERVED_7",
    "ID_RESERVED_8",
    "ID_RESERVED_9",
    "ID_USER_PACKET_ENUM",
};

#endif
```

RakNet client demo code (upnphelper.h)

```
/*
BSD License
For RakNet software

Copyright (c) 2014, Oculus VR, Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without modificat…
are permitted provided that the following conditions are met:

*   Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

*   Redistributions in binary form must reproduce the above copyright notice…
    this list of conditions and the following disclaimer in the documentatio…
    and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" …
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIE…
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCL…
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIR…
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDIN…
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE…
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY…
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGE…
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

/*
The MIT License (MIT)

Copyright (c) 2016 Indium Games

Permission is hereby granted, free of charge, to any person obtaining a copy…
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to …
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in a…
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

#include "miniupnpc.h"
#include "upnpcommands.h"
#include "upnperrors.h"

#include <thread>

#include "RakNetTypes.h"
#include "GetTime.h"
#include "Itoa.h"

#ifndef __UPNPHELPER_H
#define __UPNPHELPER_H

// TODO: This is based on demo code from RakNet, modernisize it and use the
// newest UPnP.
static bool UPNPOpen(unsigned short portToOpen, unsigned int timeout) {
    bool success = false;

    // Behind a NAT. Try to open with UPNP to avoid doing NAT punchthrough
    struct UPNPDev *devlist = 0;
    RakNet::Time t1 = RakNet::GetTime();
    devlist = upnpDiscover(timeout, 0, 0, 0, 0, 0);
    RakNet::Time t2 = RakNet::GetTime();
    if (devlist) {
        printf("List of UPNP devices found on the network :\n");

        struct UPNPDev *device;
        for(device = devlist; device; device = device->pNext) {
            printf(" desc: %s\n st: %s\n\n", device->descURL, device->st);
        }

        char lanaddr[64]; /* my ip address on the LAN */
        struct UPNPUrls urls;
        struct IGDdatas data;
        if (UPNP_GetValidIGD(devlist, &urls, &data, lanaddr, sizeof(lanaddr))
                                                                        == 1…

            char iport[32];
            Itoa(portToOpen, iport,10);
            char eport[32];
            strcpy(eport, iport);

            // Version miniupnpc-1.6.20120410
            int r = UPNP_AddPortMapping(urls.controlURL, data.first.servicet…
                                        eport, iport, lanaddr, 0, "UDP", 0, …

            if(r != UPNPCOMMAND_SUCCESS) {
                printf("AddPortMapping(%s, %s, %s) failed with code %d (%s)\…
                                    eport, iport, lanaddr, r, strupnperror(r…
            }

            char intPort[6];
            char intClient[16];

            // Version miniupnpc-1.6.20120410
            char desc[128];
            char enabled[128];
            char leaseDuration[128];
            r = UPNP_GetSpecificPortMappingEntry(
                urls.controlURL,
                data.first.servicetype,
                eport, "UDP",
                intClient, intPort,
                desc, enabled, leaseDuration
            );

            if(r != UPNPCOMMAND_SUCCESS) {
                printf(
                    "GetSpecificPortMappingEntry() failed with code %d (%s)\…
                    r, strupnperror(r)
                );
            }
            else {
                printf("UPNP success.\n");
                success = true;
            }
        }
    }

    return success;
}

#endif
```