OULUN SEUDUN
AMMATTIKORKEAKOULU

Jukka Vatjus-Anttila

MULTIPLE CONNECTIONS IN REALXTEND ARCHITECTURE

MULTIPLE CONNECTIONS IN REALXTEND ARCHITECTURE

Jukka Vatjus-Anttila
Bachelor's thesis
Spring 2012
Information Technology and
Telecommunications
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology and Telecommunications, Software Development

Author: Jukka Vatjus-Anttila
Title of thesis: Multiple Connections in RealXtend Architecture
Supervisor(s): Veijo Väisänen (OUAS), Jarkko Vatjus-Anttila (CIE)
Term and year when the thesis was submitted: Spring 2012
Pages: 37 + 6 appendices

RealXtend is an open source virtual space platform implementing both client and server functionality. In the default implementation of realXtend, the client could only log in to one virtual space server at any given time. In this research an ability to make multiple simultaneous connections to virtual spaces was experimented. Focus of the research was on how to control multiple virtual spaces within the same client window from a technical point of view.

This bachelor thesis presents methods of implementation in various software-layers of realXtend, and also makes proposals for use cases for future usage of multi-connection feature. Virtual game entities, called portals are examined and multiple virtual spaces are used via tabbed browsing user interface. Tabbed browsing is a common usage behaviour of modern web browsers, and it is expected that same implementation will emerge with virtual space viewers as well.

Measurements are made with a single viewer with multiple connections as well as multiple viewers with a single connection each. Memory consumption, throughput and network round trip times are measured and compared.

Future development proposal would be to consider possible passive-mode for connections which are currently not viewed in main camera. Active synchronization of hidden virtual spaces consume lots of precious resources, which is troublesome especially for mobile devices.

Keywords: Qt, realXtend, virtual space, concurrency, synchronization

Table of Contents

TERMINOLOGY AND ABBREVIATIONS

API: Application Programming Interface

CIE: Center for Internet Excellence

Chiru: Name of the project this thesis was comissioned by

C++: Programming language created by Bjarne Stroustrup

Entity: Single object in a virtual space which aggregates components

Component: Component for entity which defines the entity attributes

Game object: See Entity

GCC: GNU Compiler Collection

GDB: GNU Project Debugger

Git: Free and open-source version control system

Github: Remote repository service for Git version control system

Qt: Cross-platform application framework for GUI-programming

RealXtend: Free and open-source virtual space platform

Redmine: Project-management tool

Scrum: Agile development method

Tundra1: First evolutionary version from realXtend Naali

Tundra2: Second evolutionary version from realXtend Naali

UI: User Interface

UUID: Universally Unique Identifier

UX: User Experience

1 INTRODUCTION

In the future of the Internet and social media, 3D is going to be as common as 2D is now, because of the ever growing capabilities of the mobile device's processing power. Today people have access to Facebook, LinkedIn and other social networking sites that are purely 2D experiences, but a shift towards 3D with the Second Life and similar virtual spaces are under way. The 3D Internet is definitely the next big element among a mass of consumers, but it begs the question, how to handle all the 3D asset data for every virtual space visited with 3D avatar, and how to keep all this data synchronized. (21., 25.)

In this thesis the possibilities are researched for handling multiple simultaneous client-server connections in the client implementation. In addition it is studied, how to multiplex the data flow coherently from the 3D user interface (UI) to the network-layer, which is connecting the client and the server. Moreover, proposals for possible future use cases specific to multiconnection are made, in hope for opening the field of discussion for efficient power/resource saving methods in the area.

The measurements are made from two points of view, including memory consumption and network throughput in different situations. Possible bottlenecks in the logic are pointed out for the future research on the topic.

The research is done in an agile and iterative process called Scrum. Scrum was originally invented for extreme and rapid programming, but it has been modified to be more suitable for a slow paced research project as Chiru project. The prototype of the implementation is created on the realXtend architecture, and the target is to include it in to the latest production version of the project. This way feedback can be received from the realXtend open source community and the future development can be discussed among them.

2 DEVELOPMENT METHODS AND MATERIALS

This chapter introduces the research methods and approaches for the reader to better understand the progression how this research was executed. The agile development method is explained and some initial future use cases are introduced. The tools used are more specifically introduced in the later chapters. In addition, the background literature is reviewed.

2.1 Methods

The agile development methods are very popular in the modern software development, and this project is no exception. (1., 3.) The project followed the Scrum style progression including weekly formal or non-formal scrum meetings. Everyone had tasks signed up in the Redmine project management tool (Figure 1.), which was hosted on a dedicated server (20). Each sub-task was documented into a Redmine issue, which was used by the project administration to track the progress. Each Redmine issue was updated on a daily basis to reflect the current issue progression for the management to supervise.

The Scrum method usually has brief daily meetings where tasks are given and usually also completed on the same day. This aspect does not fit on the research project because the progression is slower. That is why scrum meetings are done on a weekly basis and one sprint is six months long.

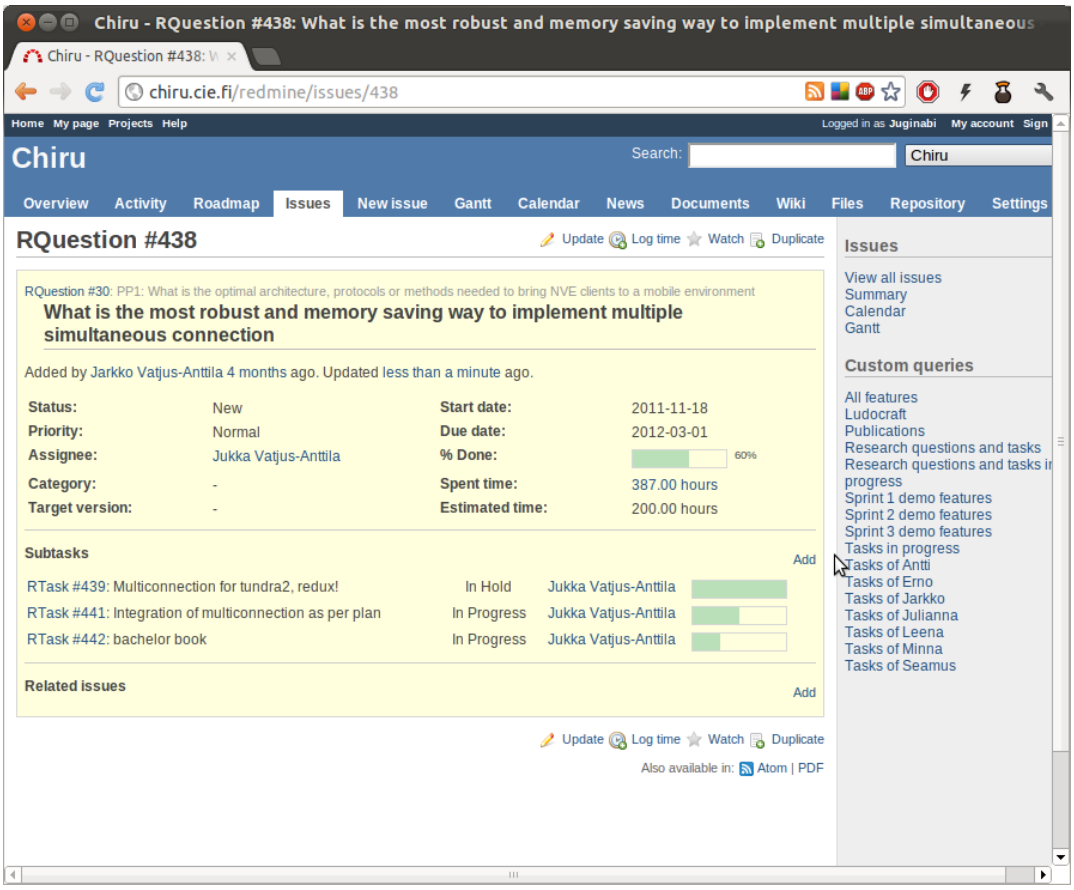

FIGURE 1. *Project management with Redmine*

All project development on realXtend code base was done in a source code repository hosted in the Github service. Git is a version control tool which can be used to track the changes in the project data. Github is a service, which enables free remote hosting of the data and uses Git as a version control tool. When the data is pushed to the distributed remote hosting service, it nullifies the risk of the local physical damage destroying data from the project. Many different features were at the scope of this project and everyone always had the latest changes from each other work, and thus the integration testing was an ongoing work from start to finish. This was an educating method in a sense that one had to keep in mind the scope of all of the code changes and what other features they affect. Several times there was a scenario where one small change broke something, which was completely irrelevant on the topic at hand.

2.2 Approach

The multiconnection implementation was first done in realXtend tundra1 branch, but a move to tundra2 brought a massive code base overhaul with it, and thus

this implementation was not valid any more. The tundra1 work was not thrown to waste completely, because a few flaws were discovered in the architecture messaging chain. This caused multiple connections to crash the program or odd behaviour between the simultaneous connections.

One of the problems of implementing the multiple connections resided in the whole messaging chain between the server and the client. When the server sent a data package to the client, there was no way of knowing who sent it if the client had more than one connection established to different servers. Universally Unique Identifier (UUID) implementation to realXtend client-server messaging chain was a top priority in this thesis. This topic alone brought up multiple parts of architecture, which needed modification for the multiconnection to work properly.

A general approach with the multiconnection implementation was to map all the private class members in the code, which had to be duplicated and stored. The network-layer and synchronization-layer contained majority of these members. These members are specified in more detail in the later chapters. The initial tundra2 connection procedure graphs were drawn to help in the specifying of all the problem areas.

2.3 Proposed use-cases

A general vision of the multiconnection does not specify which UI is the best for browsing the 3D internet. In the Center for Internet Excellence (CIE), one of the key focus areas is the UI and the user experience (UX) research. UI could be something radically different that has not been even imagined yet, or it could be something already invented with a little polish. Anything is possible, but considering the UI was not the actual focus of this thesis, the UI design is kept at the minimum although some use case proposals are introduced for handling the multiple connections. The UI modifications are also made to allow the multiconnection testing to be more convenient.

2.3.1 Tabbed browsing

The tabbed browsing has been increasingly popular in text editors since 1988, and this technique was also later included in web browsers to handle multiple

concurrent work flows. The research even shows that people today use tabs on one browser more likely than open multiple browser windows for different web pages. (22.)

A natural progression would be to include the tabbed browsing functionality with the multiple connections on the realXtend as shown in Figure 2. The login-tab would handle all connection requests from the user, and additional tabs would open on every successful connection attempt. If one wishes to change the main camera view from one virtual space to another, s/he would only click on one of the already opened tabs.

This would introduce the user to a new browsing paradigm using the standard UI one already knows how to operate. A low learning curve with this standard usage pattern would be beneficial for this UI approach, but considering the growth of the touch screen based mobile device market share, it is viable to presume large group of users on 3D internet are going to be mobile device users. (7., 17., 24.) In this case, the tabbed browsing might be a difficult method to be used with fingers only.

FIGURE2. *RealXtend viewer with tabs*

2.3.2 Portals

A portal is intended to be a completely new kind of a standard for making new connections to an arbitrary server, and is intended to be easy and simple to use.

The only restriction in portals is their target server is predefined. Defining a a portal with single sentence would be: ”*A component on a realXtend scene entity which allows the user to create a connection to another predefined virtual space by clicking the portal-entity using a mouse or gesture.*” Another way of implementing this feature would have been using a JavaScript component on the target entity but this option was not tested on this thesis.

The portal is intended to be an alternative method for generating new connections to different servers considering various kinds of mobile devices. A scenario could be imagined where a user wants to connect from one service to different services without defining the URL or other attributes. With the portal component on a 3D object, one can just move his/her avatar close to this object and make a touch gesture over the portal object and it initiates a new connection attempt. Various scenarios can be theorized, but for this thesis simply connection establishing to another server is tried from one of the virtual space objects.

2.4 Tools

This chapter introduces the tools used with this project. Linux operating system and Qt-Framework are described in detail as well as the programming language used in the project. The Git version control tool is also introduced for those not familiar with it. In addition, various other tools are briefly mentioned.

2.4.1 Linux operating system

The development environment was based on the Ubuntu Linux 11.10 operating system. (26.) The Linux OS generally provides very robust, powerful and comprehensive command-line tools compared to the Microsoft Windows counterpart. This allows a faster development, primarily through a powerful scripting in a Linux shell. Git, the GNU Compiler Collection (GCC) (5.) and the GNU Project Debugger (GDB) (6.) are tightly integrated with the command-line and provide vast amounts of information and debugging data. This combined with powerful data mining tools such as cat, grep, etc. combined with Linux command-line piping feature makes Linux simply the most versatile development environment. Support for various scripting languages like python,

perl and bash out-of-the-box is also considered de-facto development features what Linux OS has over Microsoft Windows counterparts.

All these features and powerful tools come with a price. The learning curve is quite steep to efficiently use all the provided tools. There is also a chance that one can cause damage or ultimately wreck the whole environment when using extensive *linux-spells* with a great deal of piping combined with commands like rm, xargs or similar. When one gets the hold on these tools and understands how to use them, it is going to make working on Linux faster and easier.

2.4.2 Qt-Framework

RealXtend virtual space software core is written completely in the Qt/C++ programming language. Qt is a cross platform application and UI framework. Qt consists of the cross-platform enabled libraries, development tools and Integrated Development Environment (IDE), which is called Qt-Creator, which also works on multiple platforms. Qt-Creator is shown in Figure 3. Choosing Qt as the base has allowed an easy portability of the realXtend software on multiple platforms i.e. Windows, Mac and various embedded operating systems such as Meego and Maemo.

Qt itself is based on the C++ programming language with the Qt specific additions called meta-object system, which includes a signal/slot mechanism allowing the inter-object communication. Especially signals and slots have proven useful on realXtend tundra2 software platform, because it has removed the need of using any custom event system to transmit data between modules and objects. Tundra1 had a mixture of custom-created events and signalling with the Qt meta-object system, but the tundra2 architectural changes removed the need of events.

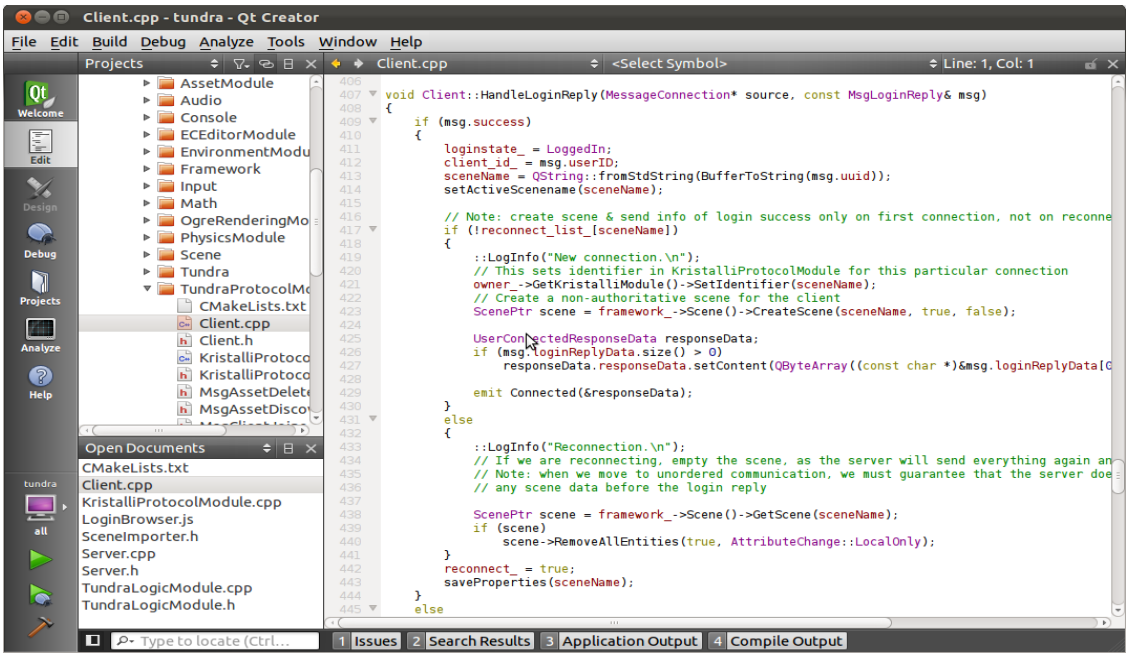

FIGURE3. *Qt-creator development environment*

2.4.3 RealXtend

RealXtend is a free and open source virtual space platform originally started and funded in 2007 by Juha Hulkko, a Finnish investor from Oulu (14). Today group of funders and supporters have grown extensively and includes, but is not limited to city of Oulu, ADMINO Technologies, LudoCraft and CIE. In April 2011, the realXtend association was founded and it currently contains the following members: CIE, Adminotech Ltd, Evocativi Ltd, Ludocraft Ltd, Playsign Ltd, Letwory Interactive and ENNE Ltd (13). There is also a realXtend foundation which owns the realXtend name (12).

RealXtend software is build from ground up using cross-platform compatible Qt programming language. This allows deploying, testing and development to be done on multiple platforms, including various Linux distributions, Microsoft Windows and MacOS.

Observed from the licensing's point of view, realXtend has undergone two generations. The first generation was the Second Life viewer based prototype, which was licensed under GPL, but the later introduced Naali viewer was build from a scratch and is based on the Apache 2 type of BSD license as shown in Appendix 4. The Naali viewer is the base for the later introduced evolutionary steps called Tundra v1.0 and Tundra v2.0.

RealXtend is based on the entity-component architecture. This means instead of deep class inheritance hierarchies for game objects, or entities, data is aggregated via components. This means the entities are only global identifiers for the game objects and they only hold lists of the aggregated components they currently have. The components contain attributes, which are updated in the component systems keeping track of specific type of components created. (2.)

2.4.4 OGRE

The Object-oriented Graphics Rendering Engine (OGRE) is an open-source library written in C++, which abstracts the use of low level libraries such as Microsoft DirectX and Khronos Group OpenGL (9). The license allows the usage of OGRE in commercial and open-source applications free of charge. The focus of OGRE-project was to create a graphics engine, which is easy to use for developers. Developers do not have to worry about platform specific capabilities when designing a graphics engine supporting multiple platforms. OGRE contains a modular architecture, where one can add new supporting features through the plugins. Main camera view in realXtend is handled by OGRE.

2.4.5 KNet

RealXtend uses the low level network transport library called kNet, which operates on top of TCP or UDP. The focus has been on the bit-efficient realtime streaming for the custom application specific messages. KNet is open-source and written in C++ and uses the same Apache based license as realXtend. (2., 10.)

2.4.6 Wireshark

When one needs a tool for network protocol analysing, Wireshark is the tool of the trade. It allows the real-time package capturing and an immediate browser interface to skim through the results. Filters can be added to reduce the shown packages on the Wireshark browser. Wireshark is an open-source and cross-

platform tool as any tool used in this thesis. Wireshark contains a Plug-in to analyze the kNet package traffic over the network. (28.)

2.4.7 Valgrind

Valgrind is a memory profiler, which has built-in tools for various tasks. It can be used to debug memory allocations, which essentially means Valgrind can find the possible memory leaks from a program in question. (27.) The essential tool in this thesis was the Valgrind Massif-tool (16). It is a profiler, which keeps track of almost all heap memory allocations done by the program under profiling. This can give very accurate profiling data concerning the memory usage and which methods or functions in the program are responsible for these memory allocations. The Valgrind core functionality is to setup a virtual machine environment for the application and to use the Just-In-Time compilation methods to translate the machine language code to the intermediate language used by the profiler tool in question. The relevant data is extracted and the intermediate language is recompiled back to the machine language for CPU to run. This transformation takes considerable amounts of CPU time and therefore the applications usually runs 1/100th of the actual speed without the profiler.

2.4.8 Git

Git, originally created and developed by Linus Torvalds, is an open source, free and popular tool for the project version control. Generally, it is used to handle the versioning of the project code base, but it is not limited to it. Git can also be used to handle versioning of any assets generated inside the project, for example, graphical assets, sounds or music and documentation. The assets excluding a programming code are usually quite large in size and they probably do not change too often, so it is not viable to store each version in the Git repository. Therefore, some other means of backup, version control and storage are usually used for the assets. As mentioned previously in this thesis, the Git allows version control to be distributed and accessible from any location containing an internet connection. Git can be locally used to track changes between versions, but it is highly recommended to set up a remote repository

where all the changes are pushed, to minimize the effect of the local physical failure of the components.

2.4.9 Github repository

The Chiru project uses the Github-service as a cloud storage for the project version control. Github is a service, which provides free public repositories for anyone to use and allows the unlimited amount of participants to access each repository. It is safe, because all project data is stored in a cloud service in case of a physical accident, which destroys all local data from the office.

Github works as a remote storage for the Git-tool. One can create a repository, which is also a new project, in Github and set it as a remote storage location or clone existing repository, which creates remote links automatically, as was done in this thesis. There are various graphical UIs available for the Git, but with the Linux powerful command-line integration of the Git, it was decided to use the command-line interface.

Working with Git is simple and straightforward. The Git clone makes a copy of the remote repository to the local machine. All file modifications are tracked automatically by Git. When one has finished with the changes, the Git commit is initiated, which marks the changes to one package. When one is ready to upload all the changes to the remote repository, the Git push is initiated which copies the changes to the cloud for everyone to get. If the push fails, it means someone did a change between the clone and the commit. In this case Git offers a merge tool. The changes always have to be merged before pushing, unless pushed into separate branch.

2.5 Testing

Testing is done from the technical point of view, not from the UI point of view. The UX evaluation is not a part of this thesis, and therefore the actual user feedback of the portals and tabbed browsing is not acquired. These features are solely used to handle the creation and disconnection of multiple concurrent virtual space connections.

The first question is how a single viewer with multiple connections compare to multiple viewers with a single connection each from the cumulative memory consumption's point of view. The memory consumption usage is measured using the Linux provided top-tool (23). Top-tool is a command-line tool, which displays virtual (VIRT), resident (RES) and shared (SHA) heap memory allocations made by a particular program. The RES memory allocation is the meaningful metric for this thesis, because it shows the heap memory usage reserved for the process in question.

The second question is about the network throughput, which would present the initial data for the multiconnection ready realXtend scalability for the next-generation of virtual space requirements. A similar scaling is expected as with the multiple viewers because every virtual space connection is run concurrently in the same way as they would be in the case of multiple viewers. The network measurements are done using the Wireshark network analyser tool. Wireshark has built in module that can be directly used to measure the package traffic produced by the kNet transport library. The throughput measurements are done using a single realXtend viewer with up to five concurrent connections and compared to up to five separate viewer instances producing traffic. The network tests are done in the wireless environment, which can be problematic considering the stability in the client-server based architecture (18).

3 IMPLEMENTATION

This chapter introduces all the abstract software-layers, which needed altering for multiconnection implementation. The implementation is separated into three core layers. The lowest layer is the network-layer, which handles all the incoming and outgoing data packages between the client and the server. In the middle there is a synchronization-layer, which handles the scene related data packages to/from the network-layer. Finally a graphics-layer is introduced, which contains the graphical representation of every virtual space.

3.1 Current implementation of realXtend

The default implementation in realXtend architecture did not support any number of concurrent virtual space connections in one viewer application. In a situation where a user wanted to change from one virtual space to another, the current connection was disconnected and all the assets were unloaded and the data members reset, which contained information about this particular connection. This procedure was followed by a fresh connection procedure, which loaded assets for the new virtual space and initialized every layer from the information chain to support this newly created connection. If one wanted multiple simultaneously created connections, one had to use multiple separate realXtend viewers.

Various software layers in the realXtend architecture were identified, which had to be changed in order to make the multiconnection possible within one viewer application. There are going to be changes in the core architecture logic, which is separated under its own subtopic in Qt-layers chapter, and there are going to be changes even to the higher level scripting layer, which is separated for another subtopic. This is done to differentiate core changes from UI related changes. Figure 4 shows the simplified and general architectural diagram of the initial state of realXtend architecture. Various modules not part of this implementation are on purpose omitted from the figure, for example an asset API, which is responsible for all asset related network traffic for the active connection.

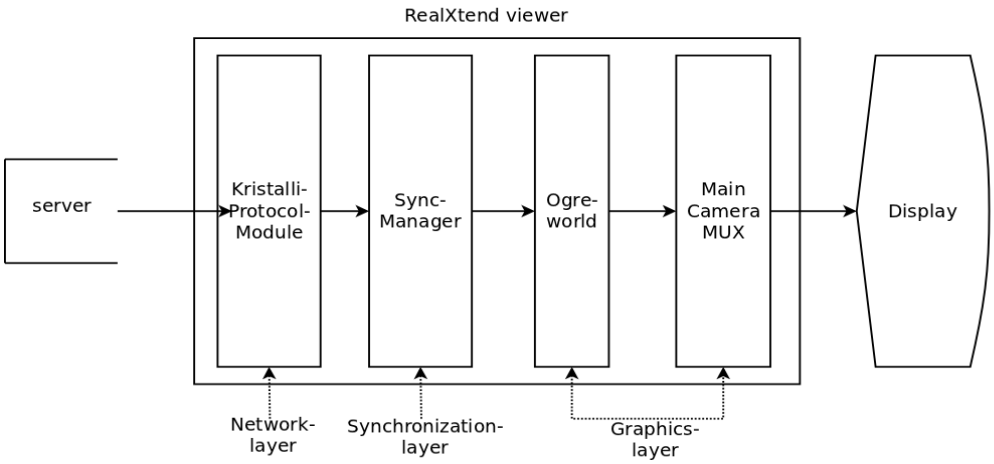

FIGURE4. *General realXtend architectural diagram*

3.1.1 Network-layer

The KristalliProtocolModule (KPM) handles all the network related work in the realXtend architecture. KPM acts as an interface for the underlying network protocol called Kristalli Protocol (kNet). KPM basically defines an easy-to-use interface for creating new message connections, disconnecting the existing message connections and updating the currently established message connections. KPM reads the data inbound from the message connection and sends it to the upper layers to handle it properly. KPM also handles all the messages received from the synchronization-layer and transmits them to the server via the message connection.

When the client initiated a new connection procedure, KPM received the connection parameters containing the host IP address, port and protocol. These parameters are then saved to the local data variables in case of a network failure and reconnection. This was the first problem discovered. All those data members had to be duplicated for every connection established. The message connection data member itself was also a singleton member inside the KPM. If a new connection procedure was initiated, the previous one was disconnected and overwritten.

Aside from the previously mentioned data members, several API methods had to be changed as well. The disconnections and the message connection queries from upper layers needed a key to know which message connection is the target for the intended action. Naturally, the update logic had to be

addressed properly to update the concurrent message connections simultaneously.

3.1.2 Synchronization-layer

The synchronization-layer encapsulates into the TundraProtocolModule containing all the code for the client, server, synchronization managers, scene managers and many other classes handling the data flow between the upper and lower software-layers. Considering the data package route coming from the server, the key classes in this layer are Server, Synchronization Manager (syncManager), Scene Manager (sceneManager) and the Client. There were multiple problem areas discovered in this software-layer.

A client keeps the state of the established connections and it kept the state of that single connection if any in the initial state. If multiple connections are made, the client-class needs a method to successfully analyse all connection states and process it accordingly. The same problem area was noticed here as in the KPM. Data members for each connection had to be duplicated and the API methods for the logout, update and several other logical methods had to be altered to handle the operations towards a specific connection.

The syncManager is the core class that is connected to each virtual space created to handle the data flow in and out of the virtual space. With the multiple virtual space connections a new syncManager instance had to be created. The instance creation should be tied to a global connection event, for example Qt-signal.

As learned from the Tundra1 implementation, when the data packages from the server do not have any identification, then there is no way of telling if the correct syncManager is handling the message. The UUID was considered to be the solution for this. If the server sends the UUID with every package, it could be guaranteed that the correct syncManager synchronizes the data to the correct scene. This approach would bring changes to the client, the server and the syncManager code in a multitude of places.

3.1.3 Graphics-layer

The OgreRenderingModule handles all the graphics related tasks in the Tundra architecture. This module handles the sceneManager creations, which are the containers for all the graphical data in a specific scene. In the Tundra1 architecture a robust implementation of the multiconnection support was made in the OgreRenderingModule. The sceneManager creation, switching and deletion actions were tied to the signals emitted from the connection, disconnection and scene switching from TundraProtocolModule. Only problem with this was the lack of identifiers in the data packages, which sometimes caused wrong assets to appear in a wrong scene.

Since realXtend moved to the Tundra2 architecture, a very matured version was implemented by other realXtend community members. The only main virtual space switch to another on demand is a minor problem, but can be solved on the scripting-layer. This made it possible to concentrate the efforts on the rest of the layers and especially on the UUID implementation, which was a considerable advantage.

3.1.4 User interface

RealXtend allows the java- and pythonscripts to be loaded on the application start-up. These scripts are used to handle various light tasks, for example, building the UI or loading a script to handle the behaviour of some object or avatar. An easy way was needed to handle the multiconnection implementation. A logical way to proceed was to extend the browser UI provided by realXtend. There are also users who do not use the graphical UI with realXtend, and therefore some kind of a command-line implementation was going to be created for switching the main virtual space view.

RealXtend can be modified externally by writing the UI using javascript or python. The existing UI had a browser style interface with multiple tabs, but the tabs were restricted to the WWW-tabs only. There was only one 3D-tab which showed the main view of the connected virtual space. This had to be changed so that a user can still create multiple WWW-tabs and use the login tab to generate more 3D-tabs. Also, some kind of logic had to be created, which

automatically changed the tab focus on the event of connection if that address/port/protocol combination had already been established. In addition, considerations should be made with the simultaneous use of the login-tab and the command-line interface, because if the connection is initiated from the command-line, the tab behaviour should reflect the current connection states in the same way as the connection was made from the login tab.

It was anticipated that new signals should be added to the realXtend Qt code, which signal certain events to the javascripts, for example a tab focus switch of an active connection. The UI design was going to be the last part to be modified, hence it would be best to consider these problems early in the development process.

3.2 Multiconnection changes

This section introduces the implementation procedure, which gradually expands to the fully fledged multiconnection implementation. Starting from the network level, which is considered as the lowest level, and then gradually steps are taken upwards and implementation procedure is explained on the way.

3.2.1 Network-layer

The first step was to change all singleton type member variables to the proper container types, which can handle the multiple instances of that type. The address, port, protocol and various other variables were changed to the QMap type containers, which stored the data in the key-value pairs, the key being the UUID transferred from the server and the value being the data. QMap stores the data sorted by the keys, which allows fast access to the data elements (4). QMap also provides an iterator class for the safe data processing (4). Appendix 1 shows the network-layer data containers for the multiconnection.

When a user initiates a new connection, the network-layer needs to save the data into the QMaps even though the actual login has not been made yet. This means there is no UUID available yet. This problem was solved by using the key ”NEW” for the newly created connection, and if the login was successful, the ”NEW”-key was updated with an appropriate UUID. An additional method was created to do the UUID copy over the NEW-tag.

The update method was the core of the whole network-layer. The purpose of it was to poll if the message connection was established and then process the connection. On the connection state change, various tasks were executed, for example if the state changed from connected to not connected, the message connection was terminated and deleted. The update method relied solely on the singleton type data members, and thus, after all the members had been changed to the QMap type storages, the update method had to be completely rewritten to handle the QMap storages accordingly.

The disconnection method also went through the same logic change as the connection and update method. Initially, the disconnection method disconnected the singleton message connection, but now the disconnection was done with the UUID as a parameter for allowing the disconnection of the specific message connections.

3.2.2 Synchronization-layer

The synchronization-layer changes began from adding the UUID generation in the server-class. Qt's own Quuid-class was used on the generation. The quuid class reference states that on any platform other than Microsoft Windows, the UUID is unique if /dev/urandom exists in the system. Otherwise, the pseudo-random generator is used, which does not provide genuinely unique identifiers. On the Microsoft Windows platform the UUID generated is unique regardless. (19.)

The UUID was incorporated with the MessageLoginReply network package which is sent to the client on a successful login. This way the client can extract the UUID on the login and distribute it for all required participants. The UUID propagation on the login can be seen on Figure 6.

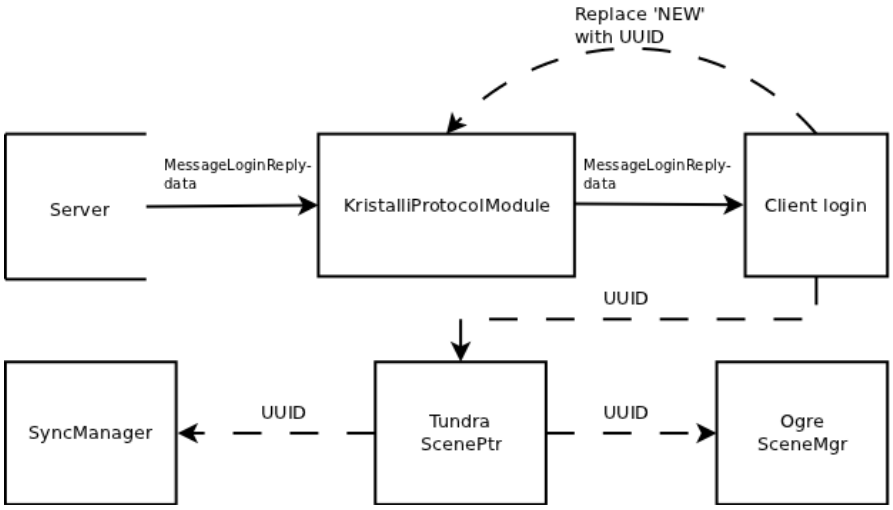

FIGURE 6. *UUID propagation in realXtend login process*

The client creates a new tundra scene for the virtual space on each login. The UUID is used as a scene name, and when the scene creation is ready, the Scene API emits a scene creation Qt-signal with the UUID as a parameter. This signal launches the syncManager creation which then attaches to this newly created scene and marks this specific UUID as the only source it accepts network packages from.

The syncManager dynamic creation brought a few changes to the TundraLogicModule-class update method. Initially, update method in the TundraLogicModule merely called the update method of the singleton static syncManager, which was created on the application start-up, but now, with the dynamic syncManager allocation, a group update was needed. Appendix 2 shows the syncManager dynamic runtime creation code.

3.2.3 Graphics-layer

As mentioned earlier, the Tundra2 architecture brought up a new implementation of the OGRE sceneManager handling. Every successful login now creates a separate ogre world container, which holds all the graphical data belonging to the scene. The view to this scene is accessed through the OGRE cameras. The scene can contain a free look camera or an avatar camera if the user has controllable avatar in this particular scene.

The main camera view change was handled within the scripting-layer because every user interaction is tied to the UI anyway. A further description of how this main camera switch was done is in the Section 3.3.

3.3 User Interface with javascript

The tabbed browser UI for realXtend was done with the javascript using QtWebKit as a browser engine. The browser UI was built in a traditional way where the user can add WWW-tabs at will and change between them by clicking the required tab. What UI did not allow was the creation of multiple 3D-tabs, which viewed the virtual spaces. There existed only one login tab which initiated a connection to the required server. When the connection was ready, the login widget was hidden and the main view from the virtual space was shown on this login tab.

There were Qt signal hooks in the javascript UI code, which tracked the connection state, and it was assumed only one connection can be established at any given moment. With the Qt code modified to support the multiple connections, this logic did not work any more. The connection state for every established connection had to be tracked separately. The UUID of the established connection had to be stored in a list when the login was successful and the tab order had to be tracked in order to retrieve the correct main camera view when the user clicked the required tab.

Disconnecting a virtual space connection was programmed to initiate from the tab close request. The same logic also had to be added to the menu bar disconnection item and for the *disconnect*- console command. This was done by tracking the active main camera view in the client-class and querying this variable from the javascript.

3.4 Portal concept

A portal is a simple tool to help the user to initialize the connection within the virtual space the user currently occupies with. It was designed with mobile devices in mind to allow the user to tap on virtual space object, which contained a portal component, without needing to use awkward text input with the login screen to establish a new connection to another virtual space. This method also

has the drawback of being predefined to a certain address, port and protocol combination.

This thesis did not focus on the implementation of the UX, and therefore it was not under evaluation whether this concept was usable or not. An interactive favourites toolbar, for example, is proposed where the user can approach different 3D objects and initiate connections from them.

3.4.1 Implementation

The portal component was written in Qt C++. The implementation was an entity component, which means it can be attached to any entity in the existing scene. This brings the property of a portal to the entity, which can be predefined in the scene XML-file or change the attributes at runtime by using realXtend build-in editor tools, but this is highly problematic if one is a mobile device user. These editor tools are nothing but developer tools and not meant for end user interaction, and therefore ease-of-use might be slightly over rated in this context.

Implementation follows the rules of the realXtend entity-component guidelines (16). The portal component updates itself with the parent entity location information provided by the EC_Placeable component, and if the user initiates the mouse left button click, it checks where the free look camera or avatar resides, and initiates a connection attempt if the distance is smaller than a predefined threshold distance. The actual connection procedure is handed to the client-class if all the requirements are met in portal component. The portal component itself works as a watcher in the scene entity which tracks if the requirements are met and fetches the needed class pointers and delegates the work for them. This allows usage of existing and proven connection methods and no additional logic has to be written for a portal to allow a new connection attempt to be created. Figure 7 shows the high-level portal activity-diagram.

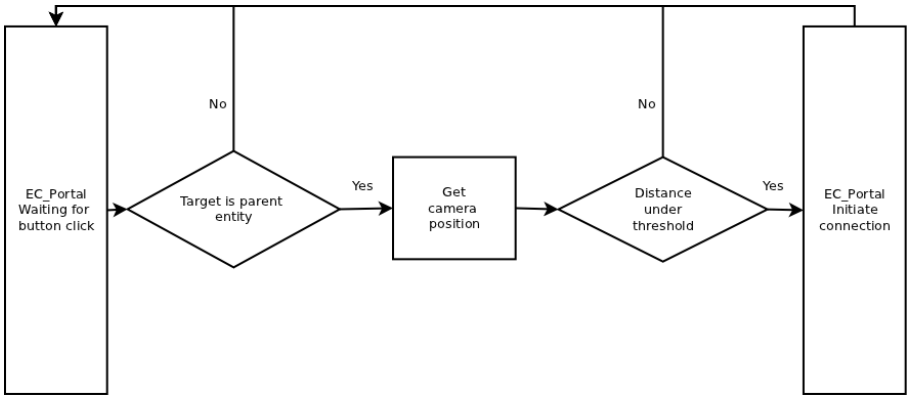

FIGURE 7. *Portal operation procedure*

4 RESULTS

The virtual spaces used in the testing are presented in Table 1. The connections to the virtual spaces are made progressively starting from the top of the list (Avatar) and capturing the memory consumption data before proceeding.

#	Virtual space name	Server location	# of entities	# of components
1.	Avatar	Local server	7	19
2.	DayNight	Local server	4	13
3.	Avatar	Remote server	7	19
4.	Physics	Local server	94	377
5.	Clubhouse	Remote server	162	486

TABLE 1. *Virtual spaces used in the testing procedure*

The memory consumption results are shown in Figure 8. The blue line presents the memory consumption of a single viewer with the growing number of concurrent connections. The yellow line presents the memory consumption of multiple viewers each holding a single connection.

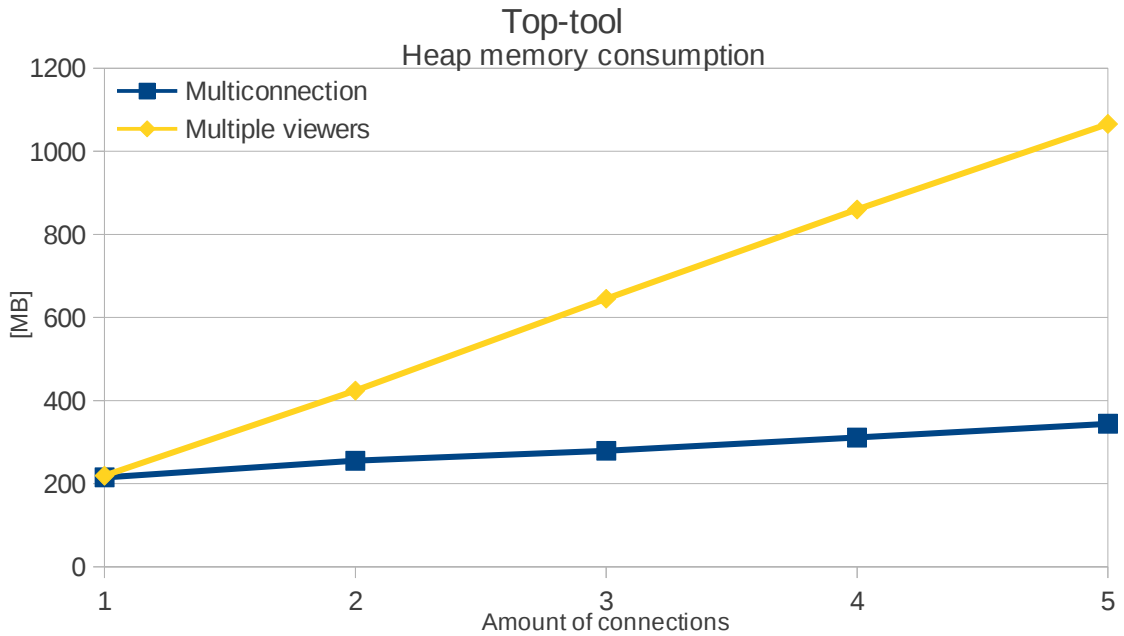

FIGURE 8. *Heap memory usage shown in megabytes*

For comparison, Figure 9 shows the memory consumption of a single viewer with the similar setup as in Figure 8 but this time measured by Valgrind. Figure 10 shows the same measurements done with multiple viewers for comparison. The legend is shown on the first graph in Figure 10 and the same colour codes apply for the rest of the graphs. The X-axis shows the time from the application start, and therefore the memory consumption seems to be gradually rising when new connections are made and assets are loaded to the scene. The Y-axis shows the heap, which is memory allocated by application, and the extra_heap, which is an additional memory reserved on top of the heap by the operating system kernel, for the application. Approximate time positions for the new virtual space connections are labeled by numbers from 1 to 5 in Figure 9. Valgrind stops the heap memory profiling on the application exit, and therefore the graph shows no memory deallocations at the end.

The reason for the lower heap memory usage figures compared to the top-tool is because Valgrind does not measure the size of application code nor the uninitialized data segments in the memory. The low-level system memory allocation calls also are the omitted from the Valgrind heap memory calculations. Therefore the comparison between Valgrind and top-tool measured graphs should only be done from a relative point of view. Direct comparison is not valid in this case.

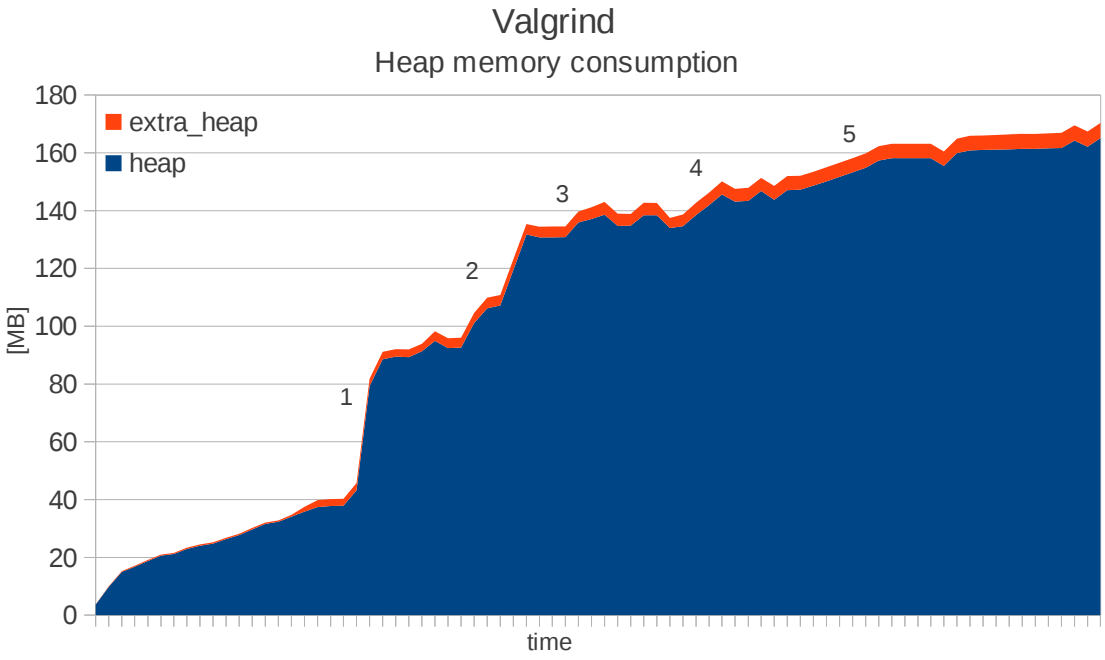

Figure 9. *Heap memory consumption, single viewer with multiple connections*

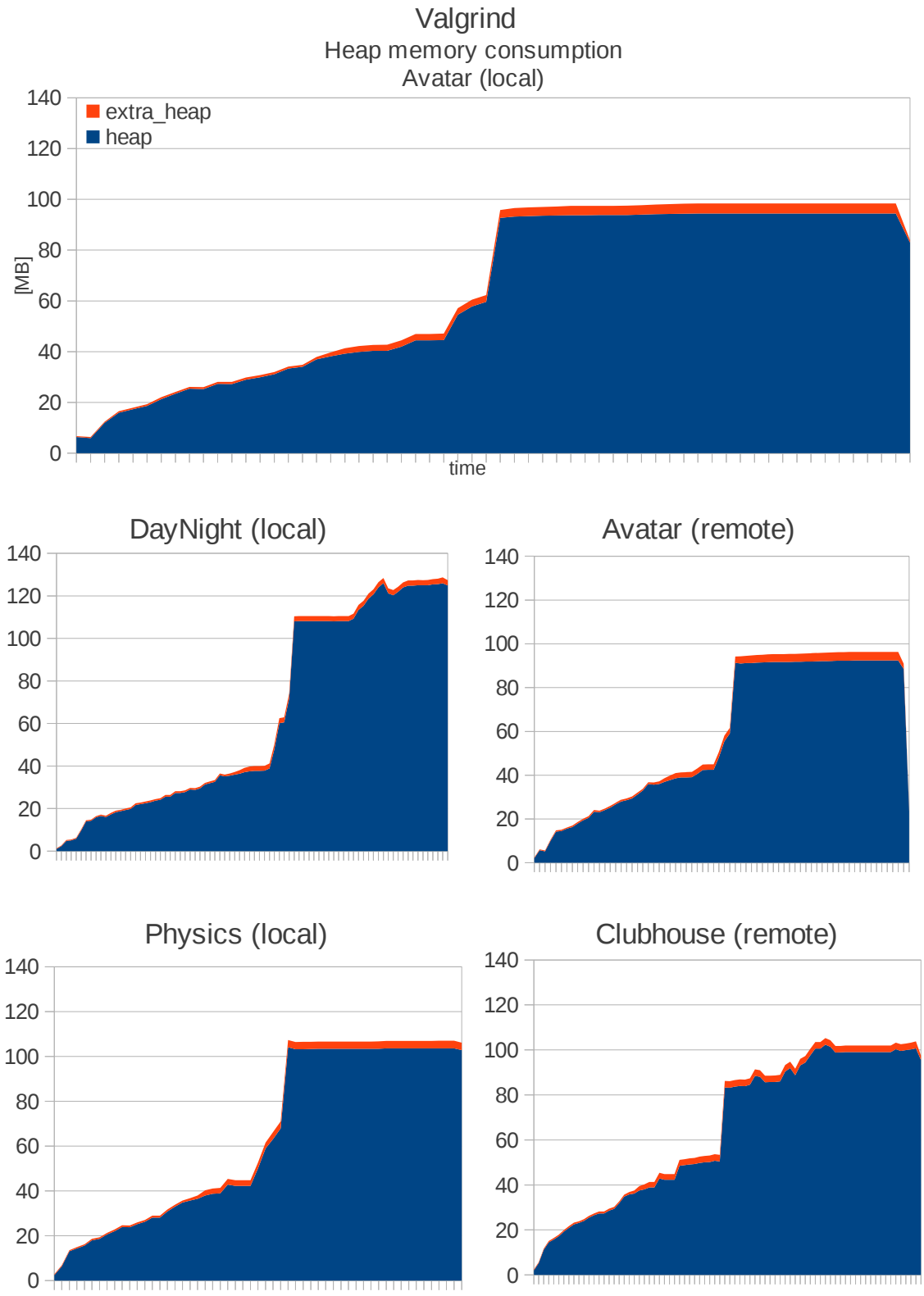

FIGURE 10. *Heap memory consumption with multiple viewers*

The network throughput tests were omitted because there was no reason to believe the figures would have been any different from the pre- to the post-multiconnection code. This is because the signalling between the client-server is still the same as before. The connections are simply handled concurrently, and the same data is transmitted for each virtual space regardless of how many connections are established.

The above was previously verified on the Tundra1 multiconnection implementation. Even though the architecture has been reworked for Tundra2, the data transmission and core functionality between the client and the server message connection remained the same. The package traffic showed a linear growth when concurrent connections were added to the realXtend Tundra1 viewer shown in Figure 10.

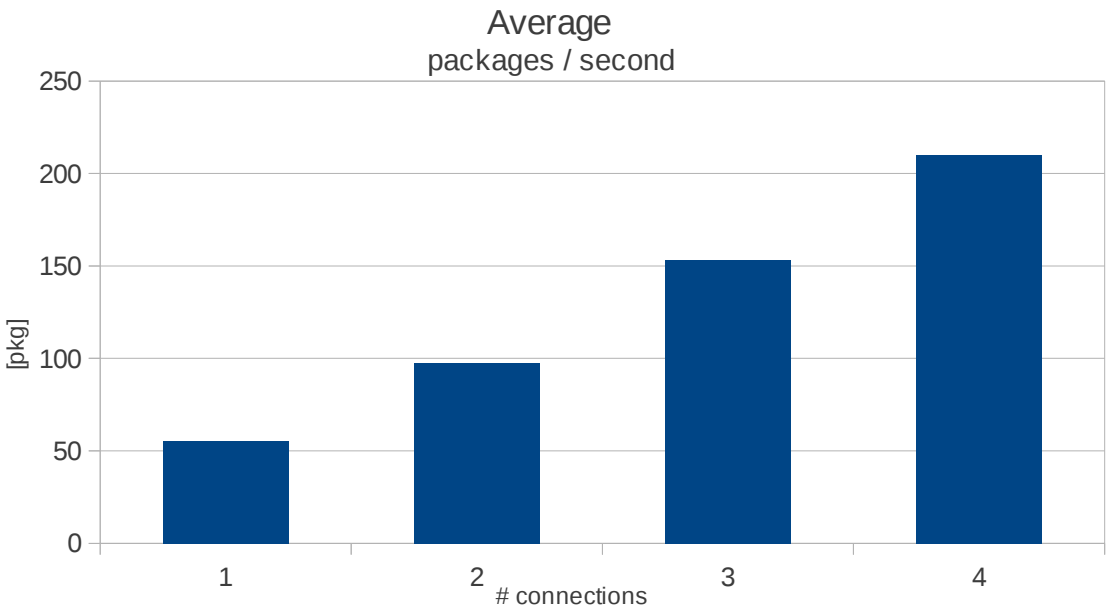

FIGURE 10. *Average network traffic on Tundra1 architecture*

5 CONCLUSIONS

The memory consumption results show the multiconnection implementation to be more memory efficient compared to the multiple viewers. This is because every realXtend viewer instance reserves ca. 170MB on the start-up for the basic services. Every scene reserves approximately 20MB – 50 MB of the memory for the scene specific assets. The multiconnection implementation only adds one time reservation for the basic service memory requirements, which reduces memory footprint greatly when new virtual spaces are opened. These results proved the multiconnection to greatly improve memory efficiency when running multiple concurrent virtual space connections.

The network throughput measurements were omitted because there was no expected change compared to the previous measurements on the Tundra1 architecture. The scaling was linear, as expected. The future research could be done to improve the efficiency with various distribution and partitioning techniques for the networked virtual environments, as was proposed in the paper from Bouras et al (8). This would mean large structural changes for the realXtend architecture. Peeking in to the more distant future, another possible research subject could be a hypergrid proposed by Lopes, C. (11). These methods are core-changing but should be discussed.

Ultimately, the multiconnection implementation required 1804 lines of code insertions and 699 lines of code deletions over the multiple files in realXtend. This was an expected metric considering the scope of this thesis. For now the implementation is found in the Github Chiru-fork of the realXtend naali.

REFERENCES

1. 3rd Annual Survey: 2008 “The State of Agile Development”. Available at: http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf. Date of data acquisition 5 April 2012.

2. Alatalo T. An Entity-Component Model for Extensible Virtual Worlds. Internet Computing, IEEE 2011 sept.-oct.;15(5):30.

3. AGILE METHODOLOGIES Survey Results. Available at: http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf. Date of data acquisition 5 April 2012.

4. Blanchette, J. & Summerfield, M. 2008. C++ GUI Programming with Qt4 (2nd Edition). the United States of America: Prentice Hall.

5. GCC. Available at: http://gcc.gnu.org/. Date of data acquisition 5 April 2012.

6. GDB. Available at: http://www.gnu.org/software/gdb/. Date of data acquisition 5 April 2012.

7. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016. Available at: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html. Date of data acquisition 6 April 2012.

8. Distribution and partitioning techniques for NVEs: the case of EVE. Challenges of Large Applications in Distributed Environments, 2006 IEEE; 2006.

9. Junker, G. Pro OGRE 3D Programming (Expert's Voice in Open Source). the United States of America: Apress; 2006.

10. KNet. Available at: https://github.com/juj/kNet. Date of data acquisition 7 April 2012.

11. Lopes C. Hypergrid: Architecture and Protocol for Virtual World Interoperability. Internet Computing, IEEE 2011;15(5):22-29.

12. Presentation of realXtend.org by Toni Alatalo at NVWN Monthly Meeting. Available at: http://nordicworlds.net/2011/04/16/presentation-of-realxtend-org-by-toni-alatalo-at-nvwn-monthly-meeting/. Date of data acquisition 4 April 2012

13. RealXtend Association. Available at: http://realxtend.wordpress.com/realxtend-association/. Date of data acquisition 4 April 2012.

14. REALXTEND: Avoimen lähdekoodin virtuaalimaailmajärjestelmä realXtend luo pohjan 3D Internetille. Available at: http://www.kauppalehti.fi/5/i/yritykset/lehdisto/hellink/tiedote.jsp?selected=kaikki&oid=20081101/12270739024580. Date of data acquisition 4. April 2012.

15. RealXtend Tundra SDK. Available at: http://realxtend.org/doxygen/. Date of data acquisition 8 April 2012

16. Massif: a heap profiler. Available at: http://valgrind.org/docs/manual/ms-manual.html. Date of data acquisition 19 April 2012.

17. Mobile Internet Research Report Reveals Massive Growth. Available at: http://www.mobilebeyond.net/mobile-internet-research-report/. Date of data acquisition 6 April 2012.

18. Performability evaluation of mobile client-server systems. Proceedings of the 2008 ACM symposium on Applied computing New York, NY, USA: ACM; 2008.

19. Quuid class reference. Available at: http://doc.qt.nokia.com/4.7-snapshot/quuid.html#createUuid. Date of data acquisition 20 April 2012.

20. Redmine. Available at: http://www.redmine.org/ Date of data acquisition 5 April 2012.

21. Social networking and context management for the future 3D Internet. Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International; 2009.

22. A study of tabbed browsing among mozilla firefox users. Proceedings of the 28th international conference on Human factors in computing systems New York, NY, USA: ACM; 2010.

23. Top-tool man-page. Available at: http://www.linuxmanpages.com/man1/top.1.php. Date of data acquisition 19 April 2012.

24. The Mobile Internet Report Setup. Available at: http://www.morganstanley.com/institutional/techresearch/pdfs/2SETUP_12142009_RI.pdf. Date of data acquisition 6 April 2012.

25. Towards 3D Internet: Why, What, and How? Cyberworlds, 2007. CW '07. International Conference on; 2007.

26. Ubuntu. Available at: http://www.ubuntu.com/. Date of data acquisition 5 April 2012.

27. Valgrind. Available at: http://valgrind.org/. Date of data acquisition 19 April 2012.

28. Wireshark. Available at: http://www.wireshark.org/. Date of data acquisition 7 April 2012

APPENDICES

Appendix 1. SyncManager update.

Appendix 2. Dynamic syncManager creation.

Appendix 3. Portal clicked code.

Appendix 4. RealXtend license.

```
void TundraLogicModule::Update(f64 frametime)
{
    *snip*

    // Update client & server
    if (client_)
        client_->Update(frametime);
    if (server_)
        server_->Update(frametime);
    // Run scene sync
    if (!syncManagers_.empty())
        foreach (SyncManager *sm, syncManagers_)
        {
            //::LogInfo("Updating!");
            sm->Update(frametime);
        }
    // Run scene interpolation
    Scene *scene = GetFramework()->Scene()->MainCameraScene();
    if (scene)
        scene->UpdateAttributeInterpolations(frametime);
}
```

```
void TundraLogicModule::registerSyncManager(const QString name)
{
    // Do not create syncmanager for dummy TundraServer scene.
    if (name == "TundraServer")
        return;

    // If scene is real deal, create syncManager.
    SyncManager *sm = new SyncManager(this);
    // Had to move some logic from Init to here because
    // --netrate cmdLine option needs syncManager.
    if (netrateBool)
        sm->SetUpdatePeriod(1.f / (float)netrateValue);
    ScenePtr newScene = framework_->Scene()->GetScene(name);
    sm->RegisterToScene(newScene);
    syncManagers_.insert(name, sm);
}
```

```
void EC_Portal::parentClicked(Entity *ent, Qt::MouseButton button)
{
    *snip*

    // Get placeable for portal position in 3D-space.
    EC_Placeable* placeable = parent_->GetComponent<EC_Placeable>().get();
    if (placeable)
    {
        position_ = placeable->transform.Get().pos;
    }

    TundraLogic::TundraLogicModule* tundra = framework->GetModule<TundraLogic::TundraLogicModule>();
    TundraLogic::Client *client = tundra->GetClient().get();

    EntityPtr avatar = scene_->GetEntityByName("Avatar" + QString::number(client->GetConnectionID()));
    if (avatar)
    {
        EC_Placeable* placeable = avatar->GetComponent<EC_Placeable>().get();
        if (placeable)
        {
            float3 avatarPos = placeable->transform.Get().pos;
            float distance = avatarPos.Distance(position_);
            if (distance < 3)
            {
                LogInfo("Connection initiated from portal!\n");
                client->Login(address.Get(), port.Get().toInt(),"PortalMan","", protocol.Get());
            }
        }
    }
    else
    {
        // No avatar present in scene. Get freelook camera pointer.
        EntityPtr camera = scene_->GetEntityByName("FreeLookCamera");
        if (camera)
        {
            EC_Placeable *placeable = camera->GetComponent<EC_Placeable>().get();
            if (placeable)
            {
                float3 cameraPos = placeable->transform.Get().pos;
                float distance = cameraPos.Distance(position_);
                if (distance < 9)
                {
                    LogInfo("Connection initiated from portal!\n");
                    client->Login(address.Get(), port.Get().toInt(),"PortalMan","", protocol.Get());
                }

            }
        }
    }
}
```

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS