**ABSTRACT**

This thesis was created to support research and development of a virtual reality application allowing users to explore a virtual world. The main focus would be on creating a solution for streaming the screen of a PC to a mobile phone, for the purpose of taking advantage of the computer’s rendering capability. This would result in graphics with higher quality than if the rendering was done by the phone. The resulting application is a prototype, able to stream images from one PC to another, with a delay of about 200ms. The prototype uses TCP as its transmission protocol and also uses the Graphics Device Interface (GDI) to capture the host computer’s screen.

This thesis also provides some information and reflections on different concepts of tracking the movement of the user, e.g. body worn sensors used to capture relative positions and translating into a position and orientation of the user. The possibility of using magnetic fields, tracking with a camera, inertial sensors, and finally triangulation and trilateration together with Bluetooth, are all concepts investigated in this thesis. The purpose of investigating these methods is to help with the development of a motion tracking system where the user is not restricted by wires or cameras, although one camera solution was also reviewed.
# TABLE OF CONTENTS

LIST OF FIGURES .................................................................................................................. 1
EXPLANATION OF TERMS AND ABBREVIATIONS................................................................. 2
CHAPTER 1 - INTRODUCTION .................................................................................................. 3
CHAPTER 2 – PROBLEM DEFINITION ....................................................................................... 4
CHAPTER 3 – BACKGROUND .................................................................................................... 5
  3.1 SIMILAR PROJECTS ............................................................................................................ 5
    3.1.1 SAGE ............................................................................................................................ 5
    3.1.2 GAMES@LARGE ......................................................................................................... 5
    3.1.3 LOW DELAY STREAMING OF COMPUTER GRAPHICS .............................................. 5
  3.2 EXISTING SOFTWARE FOR STREAMING .......................................................................... 6
    3.2.1 LIMELIGHT ................................................................................................................. 6
    3.2.2 SPLASHTOP ................................................................................................................ 6
    3.2.3 VLC ............................................................................................................................. 6
    3.2.4 FFmpeg ....................................................................................................................... 7
    3.2.5 SUMMARY ................................................................................................................... 7
CHAPTER 4 – METHOD ............................................................................................................. 8
CHAPTER 5 - IMPLEMENTATION ............................................................................................... 9
  5.1 INVESTIGATION OF TECHNOLOGY ............................................................................... 10
    5.1.1 TRANSMISSION PROTOCOLS .................................................................................. 10
    5.1.2 EXTERNAL LIBRARIES USED ................................................................................. 11
    5.1.3 EXISTING VR-HARDWARE .................................................................................... 11
    5.1.4 RESEARCHING MOVEMENT CAPTURING ............................................................... 13
  5.2 CAPTURING THE SCREEN .............................................................................................. 16
    5.2.1 GDI ............................................................................................................................ 17
    5.2.2 DIRECTX .................................................................................................................. 17
    5.2.3 REFLECTIONS .......................................................................................................... 17
  5.3 TRANSFERRING DATA ..................................................................................................... 18
    5.3.1 TCP ............................................................................................................................ 18
    5.3.2 UDP ............................................................................................................................ 20
    5.3.3 HTTP .......................................................................................................................... 20
  5.4 EARLY TESTING AND REFLECTIONS .......................................................................... 21
  5.5 FINAL PROTOTYPE ........................................................................................................... 22
LIST OF FIGURES

Figure 1 - Diagram of bit rate in bits/frame with and without coding and caching ........................................ 5
Figure 2 - Image of the Oculus Rift DK2 (Oculus VR Inc.) .................................................................................. 11
Figure 3 - Image of the Razer Hydra controller (Razer Inc.) ............................................................................. 12
Figure 4 - Image of the Virtuix Omni (Virtuix) ............................................................................................... 12
Figure 5 - Picture of the MTi-30-AHRS-2A5G4-DK unit ................................................................................ 13
Figure 6 - Illustration of yaw, pitch and roll. (Strickland, 2007) ......................................................................... 14
Figure 7 - Code snippet of GDI screen capture. ............................................................................................... 17
Figure 8 - The start screen of the Android-client. ............................................................................................ 19
Figure 9 - Screenshot of the output window of the comparison-application ..................................................... 21
Figure 10 - A screenshot of the server window .............................................................................................. 22
Figure 11 - Photo of client and server in use ................................................................................................... 23
Figure 12 - Screenshot of Limelights starting screen ...................................................................................... 24
Figure 13 - Picture of the connection prompt of NVIDIA GeForce Experience being shown when connecting with Limelight ............................................................... 24
Figure 14 - Screenshot of the Android application of Splashtop ..................................................................... 26
Figure 15 - Picture of the output window of the ffmpeg executable ............................................................. 27
Figure 16 - Picture of the game Loadout being streamed showing the receiving frame rate in the top left corner ........................................................................................................... 30
Figure 17 - Picture of the game "Cthulhu saves the world" being received by the client, showing the frame rate in the top left corner ............................................................................ 31
Figure 18 - Picture showing a movie sample being received by the client, showing the frame rate in the top left corner .................................................................................................... 31
Figure 19 - A screenshot taken while streaming a movie clip. The smaller picture to the left is the host's screen and the bigger picture to the right is the screen of the client. The number showing in the top left corner of the right picture is the current frame rate. ............................................................ 32
Figure 20 - Screenshot of a broken frame received from the server ................................................................... 35
EXPLANATION OF TERMS AND ABBREVIATIONS

API – Application Programming Interface

CPU – Central Processing Unit.

FPS – Frames Per Second, meaning how many pictures are displayed every second.

GDI – Graphics Device Interface.

IDE – Integrated Development Environment

LAN – Local Area Network.

Mbps – Megabits per second. The measurement used while describing the speed of a networks transfer rate.

OSI – Open System Interconnection

RPG – Role playing game.

Steam – an entertainment platform specialized in computer games.

TCP – Transmission Control Protocol

Triangulation and Trilateration – two similar methods of calculating the position of an object using three nodes with known positions.

VR – Virtual Reality.
CHAPTER 1 - INTRODUCTION

This thesis describes the work put into the development of a prototype application able to stream a part of a PC’s screen to a mobile phone. The application would then be used in the development of a mobile head-worn virtual reality prototype, where the user will be able to control its in-game character just by moving around. The application, consisting of one server-part located on the host computer, and one client-part located on a mobile phone, would be used to stream the host computer’s screen to a mobile phone connected to a type of headgear which shows the received images to the user, regardless of the direction the user is looking at. A reflection about the suitability of TCP, UDP and HTTP for this project is offered, as well as a description of the implementation and testing of the prototype. It also informs about some already existing software available for this purpose.

While the main focus of this thesis was to develop a new streaming application and finding existing solutions and comparing them to one another, this report also reflects on new concepts of controlling the user’s in-game character. These methods are; tracking by magnetic fields, triangulation or trilateration together with Bluetooth, using inertial sensors and, finally, using a camera to track the movement.
CHAPTER 2 – PROBLEM DEFINITION

For a virtual reality to be as immersive as possible, the user should be able to see a complete and surrounding virtual world in 360 degrees, in all axes. The user should also be able to move freely, more or less, and these movements should be reflected by the in-game character in a natural way. For this, a way of showing the VR-environment for the user would have to be developed. When starting this project, a type of headgear using a phone to show a VR-environment, located on the phone, existed but in our case the VR itself is hosted by a computer to be able to render graphics of a higher quality. Hence, there was a need for a way to show the VR on the phone. Since the user should be able to move freely, a wired connection between the phone and the computer is undesirable.

We needed an application which can capture the screen, or even better just a part of the screen, of the host computer with a frame rate no lower than 15. The application should then send the images to a receiving client with as little latency as possible, preferably no more than 200ms. One way to stream video from peer to peer is to use a buffer in some way which smoothenes the playback and makes the receiving client being able to play the video without it having to stop to load the next part. This approach is called progressive downloading (Wavelength Media). A buffer is a container which contains the next part of the media being streamed (Siglin, 2010). For the buffer to be effective it has to be able to load the video faster than it plays or else the video will have to be paused to load the buffer again. For this project however, using a buffer is out of the question since the goal is for the client-side to always show the latest frame produced by the host computer.

Another solution to this problem would be to use some kind of existing hardware that mirrors the screen of the host computer onto another screen.

We also needed a way to capture the movement of the user, so we could move the in-game character accordingly. To make this application applicable to more than one type of room it would have to be flexible, while also making it more attractive for different audiences. To make it flexible, it would be desirable to use as little equipment as possible. Having a lot of cameras to capture the movement would therefore not be a good solution. Although there is a possibility to use the phones built in sensors, they’re not accurate enough for this project.
CHAPTER 3 – BACKGROUND

3.1 SIMILAR PROJECTS

3.1.1 SAGE
SAGE (Jeong, et al.), Scalable Adaptive Graphics Environment, is a solution for real-time streaming of high quality graphics over “ultra-high-speed networks”, designed to be used with high-resolution display walls such as the LambdaVision 100 Megapixel display wall. SAGE is a middleware which uses UDP to transfer the data, although some extensions were made to eliminate artifacts on the received graphics.

This project is highly interesting as SAGE shows that it’s possible to transfer high quality graphics without much latency and with a very limited loss of packages, which is desirable for most streaming applications as well as for this thesis.

3.1.2 GAMES@LARGE
Games@Large is a project which aims to develop a new platform used to provide a “richer variety of entertainment experience in familiar environments” (Nave, et al.). One focus of the project has been to create a means of streaming graphical output from a host to clients based on other devices. “This is achieved by capturing the graphical commands at the DirectX API on the server and rendering them locally” (Nave, et al.), meaning that the games will be run on a host computer/server which then streams the graphics to an end device, i.e. a mobile device. The main benefit by this approach is that the quality of the graphics will not be compromised, while using video streaming might lower it.

Like SAGE, GAMES@LARGE has shown that there are ways of streaming graphics without much latency and with a high frame rate. It also suggests that using DirectX would be more beneficiary than using some sort of video stream.

3.1.3 LOW DELAY STREAMING OF COMPUTER GRAPHICS
This project was about creating a new platform where games could be played on a device communicating with a server connected to the same LAN, where the game would actually be run (Fechteler). The creators of this project aimed for the platform to be used in home and hotel environments where the user would connect to a PC based server running the game and streaming the graphics to the
user’s device. The project recognizes the problem of delays in time critical gaming, e.g. first person shooters, and focuses on minimalizing it.

The solution they have created uses OpenGL to simulate the graphics at the server and then sends commands to the receiving client. By simulating the graphics at the server there is no problem with a round trip asking the client for the current projection matrix etc. This solution uses TCP to transfer the data from the server to the client.

Tests show that they did accomplish what they wanted to do although they feel that the command compression can be improved and the use of a different transmission protocol could also have an impact on the result.

This project is very interesting for this thesis as it accomplishes roughly the same thing as this thesis is about. The project has shown that it’s possible to use TCP and still achieve a delay low enough for the gaming experience to be acceptable although this project uses much more advanced techniques than this thesis.

### 3.2 EXISTING SOFTWARE FOR STREAMING

#### 3.2.1 LIMELIGHT
Limelight (Limelight) is an open source application developed by five students at Case Western Reserve University, Cleveland, Ohio, USA, during the MHacks hackathon 2013 (Mhacks, 2014). It was created by reverse engineering the NVIDIA Shield (NVIDIA Corporation). Limelight is designed to be able to stream Steam-games from a PC to a device using Android 4.1 or higher (Crider, 2014). Limelight requires your PC to have a NVIDIA graphics card. You’ll also need Steam (Valve Corporation) installed on your computer, as well as a router for a Local Area Network (LAN).

#### 3.2.2 SPLASHTOP
Splashtop Inc. is an award-winning company which has released a product with the name Splashtop (Splashtop Inc.). It’s a remote desktop solution able to mirror your computers desktop both over LAN, WAN and the Web. Besides from outputting video with very little latency (less than 200ms), it also streams sound from the computer to the phone. The option to zoom the pictures is there, as well as a built in keyboard and buttons that allows you to combine keys, such as Ctrl+Alt+Delete.

#### 3.2.3 VLC
VLC media player (VideoLAN Organization) is a free media player used for playing media and also sending and receiving media streams. It is developed by an organization called Video Lan which is a nonprofit organization (VideoLAN Organization) based in France, dedicated to creating open source tools for dealing with multimedia.
3.2.4 FFmpeg
FFmpeg (FFmpeg) is an open source project, created to be able to handle media in different ways. It’s able to encode, decode and transcode media as well as filtering and streaming it. FFmpeg provides multiple tools for this while also providing libraries for developers to use, just like Video Lan.

3.2.5 Summary
All of the projects described above are open source projects, except for Splashtop which is professionally produced software. VLC is primarily used to play and stream media between computers but looking at Splashtop it’s clear that there are ways to stream a computer screen to mobile device with good quality and without much latency. However, none of these solutions offers the ability to stream only a part of the screen, which is something desirable in this project. A more detailed evaluation can be found in
CHAPTER 4 – METHOD

This thesis was divided into three stages. The first stage was to come up with different ideas of means to capture the movement of the user as well as researching these ideas. Different possible solutions were thought of and information about what kind of hardware these different solutions required was gathered. The hardware was then investigated and its suitability was rated with focus on the cost, size and availability.

The second part of the thesis was to gather information about what means there are to stream a computer’s screen to a mobile device and what current solutions are already existing, both software solutions as well as hardware solutions.

The third part was to create and test a new application that would stream the screen of the host computer onto a mobile device.

Throughout this thesis weekly meetings were held with the supervisor where we looked at what had been done so far and what would be the focus for the coming week. If any problems occurred during the week they were discussed during these meetings, although if the problem was in need of a quick solution additional meetings were held. Besides these meetings, Skype was used for quick discussions and there was also frequent mail-conversations held.

While developing the prototype a method similar to the Prototype method (Neumann, 2004) was used, where a lot of prototypes were developed and evaluated.

To evaluate the final prototype, user tests were held. Seven test persons would test the application from three different angles and then fill out a short questionnaire where they would rate the experience from 1 to 5 with questions such as “How did you find the image quality during the stream?”.
4.1 APPROACH
Essentially, there were two questions to be examined during this thesis. The first one was “Is there a cost effective and relatively simple way of capturing the movements of a player and using that data as input in a real-time virtual reality?”, while the second question was “Is it possible to stream graphics from a computer to a smartphone, with a high frame rate and a slim to none latency, hence taking advantage of the computers better graphics rendering ability?”.

These questions were researched using a deductive approach and the results are based on quantitative testing although additional points of improvement are based on a personal, qualitative reasoning.
CHAPTER 5 - IMPLEMENTATION

5.1 INVESTIGATION OF TECHNOLOGY
Available techniques were investigated so that an approach on how to solve the problem could be chosen. Different transmission protocols were looked at and simple test applications were created in a way that would give some information of what suited this project the best. Numerous techniques of both capturing graphics and transmitting it were tested but not all are presented here.

Also, existing VR-hardware was examined to give some ideas of how to tackle the problem with movement tracking.

5.1.1 TRANSMISSION PROTOCOLS
Transmission protocols basically dictate how data will be sent between computers. Some are safe, in the way that it has error checking, like handshaking, which adds some overhead while some are more unsafe, meaning there is no way for the sender to know if data arrived or not.

TCP
TCP (Cisco Systems, 2005) stands for Transmission Control Protocol and is a protocol used for transferring data over a network. It is the fourth layer of the OSI-model. The main benefit of TCP compared to protocols like UDP is that it ensures that the data received is the same data as the one sent. When a packet is received it is checked for errors and if an error is found it request a re-send of the packet. It also ensures that a sequence of packets is aligned in its original order. However, this creates a lot of overhead which makes this protocol unfit for time-critical applications.

UDP
Like TCP, UDP (Ross & Kurose) is a protocol used for transferring data. UDP stands for User Datagram Protocol and it differs from TCP in the way that it only offers minimal transport service. Because of the lack of overhead it therefor delivers a quicker way to transfer data over a network. However, since UDP does no error checking and asks for no re-sending of data, the application receiving the data will have to handle all the possible errors itself. UDP is often used in system broadcasting data to one or multiple receiving clients, for example video conferencing-applications.

HTTP
HTTP (Kristol), Hypertext transfer protocol, is probably the most commonly known protocol. Since 1990, HTTP has been the protocol used for transferring data on the www (World-Wide Web). HTTP works as a “over the counter” protocol, a request-receive protocol. The client request something, the server handles the request and sends back the requested data to the client.
In this project it was used for testing purposes where an application, built in C#, would stream pictures over a port and an Android-application would connect to that URL, showing the pictures.

### 5.1.2 EXTERNAL LIBRARIES USED

**DIRECT X**

DirectX is developed by Microsoft and is widely used in programming for Windows applications, especially games. It was created to be a standard used by mainly game developers for the Windows platform. Simply put, DirectX is a collection of API’s which gives developers direct access to a computer’s hardware. GPU, graphics, input devices and sound are all components DirectX provides access to. (Sherrod & Jones, 2012)

### 5.1.3 EXISTING VR-HARDWARE

There already exists some hardware that makes games more immersive. Below is a small description of three different products with different functionality. These products are interesting for this thesis as they are all used for immersive gaming, much like the prototype of which this thesis supports the development of. If one would combine all these equipment, the resulting immersive experience would be similar to the one aimed for with this prototype, although for this project it’s more desirable to be able to move around instead of being locked to a point in the room.

**OCULUS RIFT**

Oculus Rift (Oculus VR Inc.) is developed by Oculus VR. It’s still in a development phase but development kits are available for purchase. Oculus rift is a type of headgear with a screen and a set of lenses, one for each eye, which resembles the feeling of looking through a pair of binoculars (authors note). It uses sensors to keep track of head movement and since the Development Kit 2, it also uses a camera for this purpose. The Oculus Rift is similar to the headgear which development this thesis supports.
**RAZER HYDRA**
Razer Hydra (Razer Inc.) is a pair of handheld controls, developed by Sixense. It uses a weak magnetic field to keep track of the position and orientation so that the user hands can be represented in-game. Aside from the motion tracking it’s also equipped with analog sticks, multiple buttons, triggers and bumpers. The technique used by the Razer Hydra is what inspired one of the ideas, for motion capturing, mentioned in chapter 3.3.

**VIRTUIX OMNI**
The Omni is a type of non-moving treadmill, developed by Virtuix, which combines a concave low friction surface and a pair of specially designed shoes to map the user’s movements (Virtuix, 2013). According to the manufacturer, the Omni is able to track movements such as walking, running, strafing, jumping and even sitting down, all this in 360 degrees. As of today, the Omni only works with a PC but will hopefully be available for other platforms as well. Furthermore, the Omni requires a VR-headset and a wireless controller. The Omni is interesting for this project as it makes the in-game character move, although restricting the user to one point in the room, resulting in an experience less natural than desired for the project this thesis supports.
5.1.4 RESEARCHING MOVEMENT CAPTURING

INERTIAL SENSORS
Accelerometers and gyroscopes are inertial sensors (Xsens). Using accelerometers alone would not be sufficient enough to obtain the orientation of an object, it would only give three degrees of freedom (3-DOF), but used together with gyroscopes it gives the possibility to get both the movement as well as the orientation, in other words, 6-DOF.

The sensors would be attached to the feet of the user, making it possible to track their movement and hence, the user’s movement. To track the user’s head’s movement, the phone’s built in accelerometer and gyroscope would be used.

I found a company called XSENS which specializes in this kind of hardware. According to their website (Xsens) their products has been used for motion capturing in the making of the movie Ted (2012) and the game Tomb Raider (2013). Before really exploring how to use this hardware, a price check was done. E.g. the price for a single MTi-30-AHRS-2A5G4-DK unit (Figure 5) is € 1,990,00 (as of 210514) which is far too expensive for us to buy.

At the start of this project the supervisor had already discussed the possibility of receiving some sensors from a company, based in Västerås, to use in the development of a motion capture-software. Unfortunately, this never materialized into something real and because of that, other options were considered.

MAGNETIC FIELDS
Using just three emitters it’s possible to detect six degrees of freedom, also known as 6-DOF, which means it’s possible to detect positions in a 3D space, as well as the orientation of the object.

I had previously worked on a project where we developed a game with support for the Razer Hydra which uses weak magnetic fields to capture the positions of the hands relative to a fixed center. This made me think that maybe I could use this technique to capture the movement of a full body. The idea was to use several signal emitters placed on different parts of the body to be able to calculate the position and orientation of the object. The orientation consists of the objects’ pitch, yaw and roll, as shown in Figure 6. This would make it possible to detect six degrees of freedom, also known as 6-DOF.
Two ideas were considered.

A) The idea here was to have the source of the field somewhere in the waist region and then track the movement of the feet by having one emitter per foot. To know the position and the orientation of the head three emitters would have been used, one placed in the fore-head and two placed in the back of the head. Because of the emitters we could track the relative positions of the emitters and then calculate the direction the user was looking in, as well as calculating the yaw, pitch and roll. Using the emitters stuck to the feet, we could track the feet’s position. However, this implementation comes with some flaws. For example; if the user step forward with one foot and then move the rest of the body, the information received would be the same as the information received when stepping forward with one foot and then just dragging the foot back again. The relative positions to the body would be more or less the same, making it hard to distinguish the two movements.

B) The second idea was to use a larger magnetic field along with three emitters on the head, just like in idea A, to track its movement and direction. The center point would be placed somewhere in the room instead of on the body. If needed, additional emitters would be put on the body too. This approach would eliminate the problem of two different motions being read as one.
I tried searching for articles about this topic but found it hard to find anything useful. Just recently however, the company behind the Razer Hydra, Sixense, has created a prototype full body motion capturing system called STEM (Sixense, 2014), designed to be used primarily in games.

**TRIANGULATION/TRILATERATION WITH BLUETOOTH**

Triangulation and trilateration is the technique the GPS (Global Positioning System) uses (Marshall & Harris, 2006). I had an idea that maybe it was possible to use triangulation or trilateration to calculate the position of the user by placing three Bluetooth receivers in the room and have one emitter placed on the user. By calculating the strength of the signal to the different receivers I would be able to calculate the position although, not the orientation of the player. I thought that if I were to add more emitters onto the user there would be a possibility to calculate the orientation as well. I searched the internet for some articles and I found that a group of people had made a test (Bekkelien, 2012) similar to what I wanted to do. The result of this test had unfortunately shown that using Bluetooth for triangulation was a bad idea as the Bluetooth-technology is too inaccurate and unreliable. Another report suggested the same thing (Feldmann, Kyamakya, Zapater, & Lue). I also found a report written by a group of students from Luleå University. They had tried to use Bluetooth for indoor-positioning but had come to the conclusion that Bluetooth may not be the best choice for time-critical systems (Josef Hallberg).

**POSITIONING USING A CAMERA**

The idea was to let a camera, e.g. a webcam or other camera connected to the computer, take a picture which would then be analyzed. The picture would be used together with a filter, e.g. the Kalman filter (Greg Welch), to calculate the position of the user. This approach would however not provide the orientation of the object.

Anticipated problems with this solution included:

- Bad images, making the processing inaccurate.
- Limited range, restricting the space to move in.
- Would possibly need calibration every time it was started.

Not very much research about this solution was done however as I didn’t view it as a good solution to our problem and also the focus had to be put on the streaming-part of the project.

**REFLECTIONS**

All of the concepts described above are already existing solutions developed to capture the movement of a user. As mentioned; Xsens has developed a suit which uses inertial sensors, Sixense
has developed a new prototype called STEM which uses magnetic fields and Kinect is a camera-concept. Bluetooth triangulation does capture the movement of a user although it’s not designed to be used for small areas as it is too inaccurate. For the VR-solution, that this thesis supports, to be as immersive as possible, the best fit would probably be using inertial sensors or developing a solution using magnetic fields as described above.

5.2 FRAME RATE EVALUATION
The idea for the streaming part was to create my own application, using wireless LAN to transmit the screen from the host to the receiving client. We wanted it to be able to stream just a part of the screen to the receiving client and we also wanted to test it and compare it to already existing applications. Every prototype application made during this thesis was built in Visual Studio 2013 and programmed in C#.

Early tests were being conducted continuously throughout the development of these different solutions for the purpose of being able to compare them. All frame rate-tests were conducted using the same two computers and the same power-scheme activated. The tests were conducted multiple times by capturing screenshots for one minute and then calculating the average frame rate.

Computers used:

- Computer A – Lenovo Laptop
  - Intel® Core™ i7-4700MQ CPU @ 2.40GHz
  - 8GB DDR3
  - 2GB Intel® HD Graphics 4600 @ 775MHz
  - Windows 8.1 x64

- Computer B - Custom built, stationary computer
  - Intel® Core™ i5-3570K CPU @ 3.40GHz
  - 8GB DDR3 @ 1600MHz RAM
  - 2GB AMD Radeon R9 200 Series @ 1050MHz
  - Windows 7 Ultimate x64
5.2.1 GDI
The first thing I did was to create a WinForm-application which only took a screenshot of the image shown on the display, using GDI. It’s pretty straight forward. With just three lines of code the capturing was done.

```csharp
    target = new Bitmap(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height);
    g = Graphics.FromImage(target);
    g.CopyFromScreen(0, 0, 0, 0, new Size(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height));
```

Figure 7 - Code snippet of GDI screen capture.

Although very simple, it’s also effective.

Results:

- Computer A: 29 FPS
- Computer B: 34-35

5.2.2 DIRECTX
Later, I discovered I could also use DirectX to capture the screen. On some forums on the internet, it was stated that DirectX would be faster than using GDI. I created an application using this, and expecting better results, I was a bit disappointed. Although a bit faster, the results weren’t mind blowing. While getting around 29 FPS from the GDI the DirectX-approach produced around 35 FPS, sometimes peaking at just above 40 with Computer A.

Results:

- Computer A:
  - 34-35 FPS, sometimes peaking at 40.
- Computer B:
  - 46-48 FPS

5.2.3 REFLECTIONS
As anticipated the DirectX was a bit faster. The difference however was only around 5 fps on Computer A while the difference on Computer B was 12-13 fps. One possible reason for this might be the hardware in the computers. Computer A has a rather fast CPU with a not so fast GPU, creating a relatively small gap in performance. The situation with Computer B is the reverse, making the performance gap bigger.
5.3 TRANSFERRING DATA
While testing this part, a 100Mbit wireless router was used. In this project, the server would be located on the computer running the game environment, continuously taking screenshots and sending them to the receiving client.

5.3.1 TCP
TCP is fairly simple to use and because of its error handling it’s very reliable. Although, in this scenario, the error handling is a bit redundant due to the fact that there is no need to retrieve old frames, we’re only interested in the latest frame.

SERVER
My first implementation of this server uses two threads, besides the main thread. One thread handles the sending of data and the other thread handles the capturing of the screen. When the server starts, it immediately creates a socket and starts listening for incoming connections, not capturing anything until one is established.

I did implement a TCP solution for both the GDI and the DirectX version. The GDI version creates an array of bytes out of the captured bitmap and then transfers that array over the socket-connection. During the implementation of the DirectX-solution I ran into a problem; there is no implicit way of transforming the captured surface directly to an array of bytes. Therefore, I created a bitmap out of the surface, saved it to a stream and finally created an array of bytes from that stream. This however puts a lot of, what seems like unnecessary, overhead on the application, making it slower.

This solution was put on hold for a bit while other options were explored but was later resumed. The code was reviewed and some code-optimization was done. One optimization done was to get rid of the memory leaks that previously haunted the application, slowing it down and creating crashes. Another optimization, which seemed to have a big impact on the latency-problem, was to remove the thread handling the transmission of data. Instead, the thread capturing the images now also handles the transmission. Anticipated problems with this approach would be a drop in frame rate.

Later, when the Android-client was developed, changes had to be made in the server code as well. The client side, being developed for Android using Java, needed to know how big the data packet was going to be or it would be stuck listening for more data forever. This problem was solved by adding a header to each packet containing information of the size. The header would always have the same size so the client could easily pick out the correct bits of the package.
WINDOWS CLIENT
The first implementation of the client was a WinForm-project and just like the server, the client used a total of three threads. The client connects to the given IP + port and when it was successfully connected it used one thread for receiving the byte-array, and one thread for translating those bytes to a picture and showing it in a simple PictureBox.

Just like in the case with the server, the client’s code was later reviewed and optimized. The optimization was similar to the one with the server; I removed the thread handling the received package and put that code in the same thread as the code receiving the data. This would maybe have an effect on the frame rate.

ANDROID CLIENT
Eventually, a client for Android was developed in Java using Eclipse IDE. Besides the main thread it uses an Async Task, which roughly works as a thread. This task handles the connection and streaming of data while also translating the incoming data to bitmaps and showing them on the smartphone’s screen. The main thread, or UI-thread, lets the user write the IP and press “Connect” and this screen will disappear and be replaced with the received images.

![Android client start screen](image)

**Figure 8 - The start screen of the Android-client.**

REFLECTIONS
With the first implementation of the GDI version, bitmap-images were transferred and the results were disappointing. A lot of dropped frames accompanied by huge memory leaks resulted in me putting this approach on hold and trying out new angles. When the code later was revised the
memory leaks was handled and both the server and client were more stable and the problem with dropped frames was also resolved.

5.3.2 UDP
Since UDP doesn’t contain any error handling, there is less overhead and the expected result was that this approach therefore should be faster. UDP is also connectionless and just sends over a port, not caring if it is received or not on the other end. A big advantage of UDP is that it supports multicasting. That means that if it’s desirable to send the same data to multiple clients, the data still only needs to be sent once.

SERVER
I started out by creating a simple server in WinForms, sending text-messages without encountering any real problems. When that was done, I continued by combining the GDI-capture with this UDP-server and tried to send bitmaps. Just like the TCP-server, this one used two additional threads, besides the main thread, one of them handling the screen capturing and the other handling the sending of the pictures.

CLIENT
When the implementation of the first UDP-server was done, the one sending text-messages, I created a client which received those messages and printed them to the screen. I then continued to build a client which would receive an array of bytes and translate it to a bitmap which then would be shown in a PictureBox.

The client could never receive anything due to the problems described below.

REFLECTIONS
It didn’t take long before I encountered a big problem, UDP is not meant for transferring large data-packets so, due to them being too big for a UDP-package to handle, the pictures could never be sent. However, one solution often used is to create RTP-packages (Rouse, 2007) and send those over UDP. I investigated it for a while, but later decided to try and come up with a different, easier solution.

5.3.3 HTTP
When I implemented this I started by looking for existing solutions. I found one open source demo project which streamed pictures from the computer.

SERVER
I didn’t do much tweaking of this server, I only changed the source of the stream so that there actually existed pictures for the server to stream.
CLIENT
Since this server was supposed to stream pictures I tried connecting with Google Chrome but without luck. I then wanted to try and connect with a Samsung Galaxy S3 so I used an app called “tinyCam Monitor Free” (Tiny Solutions LLC) and was successful. The app also calculates and outputs the frame rate.

REFLECTIONS
While successful in receiving images on the phone, the frame rate was low and very unstable. Due to the unstable frame rate, the amount of time each picture was shown before being replaced by the next was really uneven. Also, there was no way of knowing how much latency existed.

5.4 EARLY TESTING AND REFLECTIONS
To be able to create an application, able to stream with a high frame rate and a low delay, I created several applications to test the different approaches (see chapter 5.2 and 5.3) and did some testing. Eventually, I created an application which uses both techniques to capture the screen and compares the results.

![Figure 9 - Screenshot of the output window of the comparison-application](image)

The results showed that even though DirectX is typically faster than GDI, there is a bottleneck in the conversion from a surface to a bitmap, just as anticipated.

The table below describes the results of the comparisons I made. During the transmission tests the server was located on Computer A, with the client located on Computer B.

<table>
<thead>
<tr>
<th>Method of capturing</th>
<th>Transmission protocol</th>
<th>Frame rate</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capturing surface with DirectX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capturing surface with DirectX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capturing surface with DirectX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capturing surface with DirectX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Method</td>
<td>Width/Height</td>
<td>Delay</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>GDI</td>
<td>No transmission</td>
<td>29-35</td>
<td>N/A</td>
</tr>
<tr>
<td>DirectX</td>
<td>No transmission</td>
<td>34-48</td>
<td>N/A</td>
</tr>
<tr>
<td>GDI</td>
<td>TCP</td>
<td>Est. 10-15</td>
<td>200ms</td>
</tr>
<tr>
<td>DirectX</td>
<td>TCP</td>
<td>Est. 10-15</td>
<td>290ms</td>
</tr>
<tr>
<td>FFmpeg</td>
<td>UDP</td>
<td>10</td>
<td>Over 2000ms</td>
</tr>
<tr>
<td>Existing images</td>
<td>HTTP</td>
<td>1-4</td>
<td>N/A</td>
</tr>
</tbody>
</table>

To summarize, DirectX is faster in means of capturing the graphics of the host computer than GDI and if a way of speeding up the transmission, such as using UDP or finding a more effective way of converting the graphics to a byte array, this would probably be a better solution than using TCP with GDI. Also, utilizing DirectX to render the images received by the client could maybe eliminate some of the delay.

5.5 FINAL PROTOTYPE
The final prototype consists of two applications; one server and one client.

My final application gives the user a choice to either use GDI or DirectX to capture the screen while using a TCP-socket connection to transfer the bitmaps. GDI is recommended as of now. The server application also gives the user full control of what part of the screen to stream to the receiving client. The user can choose to write the coordinates of the top left corner of the rectangle to be captured and together with the values given to the width and height the rectangle is created. There is also an option to set the rectangle using the mouse cursor.

5.5.1 SERVER
The server is supposed to be run on the host computer, streaming the computer’s screen to the receiving client. Although the DirectX-implementation (described in chapter 5.2) captured the screen faster than the GDI-implementation (described in chapter 5.2), there was a delay-problem while translating the surfaces to arrays of bytes (read chapter 5.4). Because of this and the fact that I was unable to get UDP working the way I wanted, the GDI-approach is recommended.

5.5.2 WINDOWS CLIENT
The applications were tested and the result was better than anticipated.
The picture above shows a test being made with the server located on the left computer and streaming its screen to the right computer. As can be seen in the red squares, the latency is now roughly 200ms. The tools used for this test was just a simple online stop watch running in a web browser (Chrome). The two laptops were put close to each other and then a picture was taken of the two screens.

The latency didn’t change very much even if the stream was active for several minutes, which is good.

**5.5.3 ANDROID CLIENT**

There is no real difference between the final Android client prototype and the one first developed. It was concluded that the client already produced the wanted results and no changes needed to be done except for some error handling.
CHAPTER 6 – EVALUATION

6.1 LIMELIGHT

Setting up Limelight was a challenge. Firstly, there is a need for the host computer to have a NVIDIA graphics card and secondly, it has to be a relatively new NVIDIA graphics card. Furthermore, to use Limelight with your Android-device you also need to have NVIDIA GeForce Experience installed on the host computer, as well as Steam. If all these requirements are met, there should be a good chance of getting Limelight to work.

Both my computers use different AMD Radeon graphics cards so I had to try and borrow a computer with a NVIDIA card. I was able to borrow a computer with a NVIDIA GeForce GT 240 from a friend but when I tried to use Limelight with the computer I ran into some problems. When using Limelight the user is supposed to connect the Android-device to the host computer using LAN, Bluetooth etc. and a connection-prompt will appear on the host computer’s screen.

![Figure 12 - Screenshot of Limelights starting screen](image1)

![Figure 13 - Picture of the connection prompt of NVIDIA GeForce Experience being shown when connecting with Limelight](image2)

When I tried this on the borrowed computer that prompt would never appear and Limelight (running on my Android-phone) would eventually just tell me that the connection could not be established. There was also a problem with NVIDIA GeForce Experience not approving of the graphics card of the computer, although it was a NVIDIA card. My conclusion is that Limelight could not connect to the computer because of the graphics card being too old.
Luckily for me another friend was able to provide another computer. When NVIDIA GeForce Experience had been installed, an attempt of connecting Limelight to the computer was made. While thinking it would work, it turned out unsuccessful. More literature searching was done and requirements previously not known of were found. Beyond the previously stated requirements there was also a need for the NVIDIA GeForce 337.50 beta driver or newer ones. The driver was downloaded and installed and finally Limelight was able to connect.

When the connection was finally made Steam’s Big picture mode was automatically started and the game Antichamber was selected to play. One advantage with Limelight is that it is possible to control Steam’s Big picture mode from the phone by using it as a kind of remote control, although it is a bit buggy and awkward. The streaming itself was very good with a high frame rate and a low latency between the computer and the phone as well as really good picture quality. There were some problems with corrupt packages though which made the stream stop momentarily but it picked up on its own soon enough. One major disadvantage though is that if the user tries to switch to another program while the game is streaming, the stream stops working and the connection has to be set up again.

Summary

PC Requirements (Gutman, 2014):
NVIDIA GeForce GTX 600/700/800 series or GTX 600M/700M/800M series GPU
NVIDIA GeForce Experience (GFE) 2.0.1 or later
NVIDIA GeForce 337.50 beta driver or later
Steam enrolled in beta updates

Positives:
- High quality streaming.
- Different options for the streaming.
- Freeware.
- Open source.

Negatives:
- Only works with NVIDIA GeForce GPUs and has very specific requirements about cards and drivers.
- Can only stream from Steams Big picture mode.
- Dropped packages.
- Crashes when switching between programs.
6.2 SPLASHTOP

Splashtop is a remote desktop application not really designed to stream games with a high frame rate. It comes in different versions, both a free one and some commercial ones. The free version offers streaming of HD video (and also audio) which is what this project is about and therefore I choose to go with this version.

The difference I noticed about the two versions was that if you want to access your computer without being connected to the same LAN you need the commercial version. This is not the case for this project so I used the free version.

I downloaded and installed the application to my phone and the corresponding program to the computer and created an account at Splashtop. Then I only had to log in to my Splashtop account on my computer and a list of my connected computers would appear on the Android-application (picture #). When choosing the computer I wanted to connect to, it took about 2 seconds and then I saw the computers screen on my phone. I was able to control the mouse by just touching the screen and I was also able to zoom in on the desktop if I needed. There is also a built in keyboard, this is however not relevant to the purpose of this project. I started the game Formula 1 2013 on my computer, the same game I used for testing my own application. I noticed that there isn’t much latency but the frame rate however was not good enough for this application to be a solution to our problem.

Summary:

PC requirements (Splashtop Inc., 2014):

Windows or OSX

Positives:

- High quality streaming of both video and audio.
- Low latency.
- Easy and quick setup.
- Easy to switch between host devices.
- Freeware.

Negatives:

- Not a lot of options, e.g. there is no option to choose what part of the screen to stream.
- Frame rate inadequate to this project.

Figure 14 - Screenshot of the Android application of Splashtop
6.3 FFMPEG
There are many ways to use FFMPEG to stream. I used the FFMPEG executable, which uses a command line interface, to stream the desktop of one computer to another. The FFMPEG executable can’t capture the screen faster than a frame rate of 10 FPS so for this a filter is needed. I used a filter called UScreenCapture which is sent to FFMPEG as a parameter in the command line.

A lot of options were available for customizing the stream, although the documentation could have been better. A lot of the different commands I entered, like trying to push the frame rate higher, didn’t seem to have any effect on the resulting stream. I ended up trying to stream the desktop from one computer to another with the receiving end using VLC to connect via UDP.

![Figure 15 - Picture of the output window of the ffmpeg executable](image)

The images outputted by the receiving computer were almost always distorted and there was a latency all too high, over 2 seconds, for us to work with.

Summary:

Positives:
- Freeware

Negatives:
- High latency
- Distorted pictures
- Hard to customize
6.4 OWN IMPLEMENTAION
While doing tests with my software I used GDI for screen capturing and TCP as the protocol for data transmission. I calculated the frame rate and, depending on what was being performed on the host computer, the frame rate differs between 15 to 30+ FPS. Besides this, the latency can sometimes wander from being not noticeable up to around 200ms. The receiving client was the Android implementation.

Summary:

Positives:
- Quick set up
- Gives the option to select what part of the screen to stream and what technique to use
- Would be free if released to the public
- Can stream in ~25 FPS
- Can stream with unnoticeable latency
- Possibility to modify source code if wanted or needed

Negatives:
- Doesn’t work with full screen applications
- Issues with varying latency
- Occasional drops of frame rate

6.5 SUMMARY
If only considering the quality of the stream, Limelight would be the undisputable number one. Its streaming capabilities are unmatched by the other software it was compared to. However, Limelight has very specific requirements which might be hard to match, making it less flexible than the other software it was compared to. Furthermore, Limelight doesn’t provide the option to stream anything other than Steams Big picture mode as of now although, it’s open source so changes in the code could be done if wanted.

Splashtop is good but doesn’t quite reach the same frame rate as Limelight. It doesn’t provide the option to select a portion of the screen to be streamed either, just like Limelight.

FFMPEG provided the worst results during the comparisons and it’s not recommended to use it in the way I did.

My own prototype has the benefits of an open source code and it has no specific requirements other than that Windows is used. If the problems listed above can be resolved, together with a small increase of the DTR, this application would be a very good fit for a VR-application.
Different tests were conducted with the purpose of exposing weaknesses of the prototype application. The router used while conducting these tests was a Belkin F5D8235-4 with a wired transfer rate of 1000 Mbps and a (theoretical) wireless transfer rate of 300 Mbps.

7.1 USER TESTS WITH WINDOWS CLIENT
User tests were conducted to determine the usability of the created prototype application. Seven people with different previous experience in gaming, ranging from total beginners to experienced gamers, were selected and two different gaming environments were tried with all of the test persons. Aside from the gaming, the test persons also watched a short high definition (1080p) movie sample and got to rate the experience and usefulness of the prototype in media streaming.

The games used for testing was “Loadout” and “Cthulhu saves the world”. The tests were conducted such as the test person was playing the games with controls connected to the host computer but looking at the client’s screen.

“Loadout” is a third-person-shooter where quick visual feedback is important for the gaming experience to be enjoyable. Without quick visual feedback the game would be hard to play and the user would quickly become bored.

“Cthulhu saves the world” is a role playing game (RPG). As the in-game combat is turn based, the game doesn’t demand the same quick visual response as “Loadout” which makes this game more suitable for the prototype.

7.1.2 RESULTS

LOADOUT
Four of the test persons found this game acceptable to play although they all would prefer to play it with less latency as they found the controls to respond just a bit late. They also felt that in-game enemies got a slight advantage due to the latency. The other three persons first found this game too hard to

Figure 16 - Picture of the game Loadout being streamed showing the receiving frame rate in the top left corner
play with this latency but when asked to play the game without the stream they tried and still found the game too hard.

**Cthulhu Saves The World**

All seven of the persons found this game to be playable with the application. The “delay of the controls”, which is really a latency of the frames, was not an issue since this game isn’t time critical in that sense.

**Movie Sample**

While watching the movie sample, five of the test persons found the stream to be watchable and said they would not have a problem watching a whole movie like this, if only considering the visual aspect. The remaining two people found the frame rate to be a bit low and one of them said that she didn’t know if she could get used to it or if she would feel bothered if trying to watch a whole movie. The other one said he would not be able to enjoy a movie like this due to the frame rate and also due to the sound being a bit delayed compared to the video.
One mutual opinion throughout the tests was that the quality of the images was superb although sometimes there were a bit too many dropped frames, making the stream unsmooth and the experience unpleasant.

7.2 USER TESTS WITH THE ANDROID CLIENT
During this test, the same router was used and as a client a Samsung Galaxy S3 was used. The phone was connected to a TV via an adapter. Unfortunately, connecting the phone to a TV seemed to add a small bit of delay to the output but it was so small that it was often overlooked. The test was conducted in the same way as the movie clip-test of the Windows client but with the animated movie Tintin.

7.2.1 RESULTS
Not one of the persons had any problem watching the movie through the application due to the frame rate being quite high (~30) and an unnoticeable delay between sound (from the host computer) and the images. If the delay would have been greater the test persons would have noticed the sound and picture being out of sync.

Figure 19 - A screenshot taken while streaming a movie clip. The smaller picture to the left is the host’s screen and the bigger picture to the right is the screen of the client. The number showing in the top left corner of the right picture is the current frame rate.
7.3 ADDITIONAL TESTS
Aside from the user tests, an additional test was made where the movie sample would be streamed over a wired connection instead of a wireless one and the frame rate would be compared to the corresponding frame rate of the wireless connection. The purpose of this test was to find out if the wireless connection induced a bottleneck compared to a wired connection.

The results show that there was no bottleneck induced by the wireless connection. This suggests that the created prototype’s code must be revised and that it might be the use of TCP-sockets which slows down the transfer rate, as suspected.
CHAPTER 8 – CONCLUSIONS

The final prototype is a two component solution which uses TCP to send and receive the images rendered by the host computer. Although the projects called “Low delay streaming of computer graphics” and “Games@Large” showed that the technique where the graphics are simulated (but never rendered) at the host computer and commands are sent to the receiver is an effective one, the final prototype uses a much simpler technique but still manages to transfer images at an acceptable rate. The main benefit of the created prototype compared to the other software mentioned is that the server application offers the user to choose what part of the screen to be streamed, something the other software doesn’t provide.

The comparisons made between the existing software and the prototype that was created showed that Limelight provides the best results when streaming but when taking the very specific requirements in to consideration and the inability to choose what part of the screen to stream, my verdict would be that the best choice would be to keep developing the prototype created during this thesis.

The purpose of researching the different concepts of movement tracking was to provide some insight in what is possible to do and what’s not, while also providing some benefits and deficits with the different approaches. This has been achieved.
8.1 ISSUES
The biggest issue of all was to figure out a way to get the frame rate as high as possible and the latency as low as possible. Although I’m quite satisfied with the current latency, I need to get the frame rate up. Also, there was a problem with some frames arriving with just a part of the original picture, resulting in a “broken” frame, as shown in the picture below. This issue has however been resolved.

![Screenshot of a broken frame received from the server.](image)

Furthermore, I had big issues trying to get my UDP-server working and never resolved this. Also, the server won’t work while running full screen applications on the host computer.

The main issue is still the latency and frame rate. For the application to be perfect it would preferably have to reach a frame rate of around 60 fps with a latency close to 0ms. This is most likely not possible using the same method as the final prototype of this thesis does. It would need to get the graphics before it has been painted to the screen and would also require the use of a faster transfer protocol than TCP currently is.

8.2 OWN THOUGHTS
Though I am somewhat close to completing an adequate prototype of a streaming application, knowing what I know now, I would have investigated how to send the graphics captured over a network in a more effective way. If more people were to join in on the project, maybe the technique used in the “Games@Large”-project could be utilized and a more effective prototype could be developed. I would also have wanted to work a lot more on the movement tracking-part of the thesis, maybe even trying to implement one of the concepts.
In retrospect, I can say that the task to complete both the streaming part as well as the movement tracking would have been too great a task for me to complete during this thesis, but I’m confident I can complete it in the future, if given the opportunity.

8.3 FUTURE WORK
I will probably keep working on the streaming application until I am happy with the result. The next step would be to use UDP and possibly use a fast method of compressing the picture so that the whole picture could be sent in one single UDP-packet. I would also try a different method where the server would split the captured image into different smaller pictures and send each via UDP where the client would then assemble the pieces to one whole image again. This would require me to write a new header for the packets as well, containing information about which image the packet is a part of etc.

Furthermore, the problem with the prototype not sending new images while in a full screen application would be addressed. I would also try and convert the images to a video format and transmit those just to see if this can speed up the transfer rate.

If possible, I would like to keep working on the movement tracking as well, maybe trying out my idea using magnetic fields, or by using sensors.
REFERENCES


