Geo-based media player

An interactive interface for geo-based video streaming

Andreas Nordberg
andno793@student.liu.se

Jonathan Sjölund
jonsj507@student.liu.se

ABSTRACT

Being able to interact with a video stream can be both fun and educational however, to achieve the best user experience, the interaction must be seamless. Creating a media player that can allow users to view a video stream from different geographical positions is a challenge that is tackled in this paper. This paper describes the material and methods used to accomplish this kind of media player. And it explains how to achieve a seemingly good and, to higher extent, enjoyable video streaming experience. A proof-of-concept is shown of a mediaplayer that enables a user to see an overview over a geographical streaming location. By seeing each streams relative location to the point-of-interest the users are able to click on that stream and switch to it in a seamless way. It is shown that it is possible to switch between most nearby videos without interrupting or impairing the video streaming experience. (This is not final or necessary correct but hopefully something we can accomplish)

Keywords

HTTP Adaptive Streaming (HAS), OSMF video player, Interactive video streaming, Geographical based streaming (GBS), Seamless playback

1. INTRODUCTION

Streaming has evolved and become more popular over last couple of years. Thousands upon thousands of different streams are being watched every day[1], thus the demand for better and more ways to stream are longed for. If we could stream videos in different ways we can create a more interesting streaming environment, this can provide both better entertainment but also a better way to potentially improve observation in science and other areas more reliably. If a stream can provide the possibility for watching a video from different angles it can give people the option to observe and also enjoy something from different perspectives.

[1]Twitch statistics https://stats.twitchapps.com/, Fetched: 2016-04-01

This project focuses on accomplishing that, by creating a geo-based video player that uses HTTP-adaptive streaming (HAS) that can allow people to view a video from different angles and change between them seamlessly without any buffering delay or stuttering. By looking at an existing video streaming player and improve it to accomplish this task we show that it is something worth implementing in already existing media players.

In this project we design and develop a geo-based command-and-control video streaming player using geo-tags. In practice this means that a service in which you can choose between a set of recording streams of the same event, for example, but slightly different locations and angles. This would be a useful feature to have in any larger event where you would want to show the same scene from different locations and angles. The interface will be useful for both event-organizers that hire staff to make several different recordings of the same scene for simultaneous viewing, but could also be used by the public who volunteer to record the event live.

1.1 Boundaries

The application we provide is only going to be a proof-of-concept which means that we will only focus on the functionality of the video player. Factors like a pretty interface and usability on broader spectrum will be neglected. We will focus on making the application work for only one user to verify the functionality we want to accomplish. The number of video streams that we will be able to switch between will,for the purpose of testing, be limited to a few, but then expanded upon to support any reasonable number of streams. This is because we firstly want to make sure that it is possible to switch between video streams and not that it can be done over large numbers. The reason for this is that pre-buffering many videos can be difficult to accomplish with a large number of video streams. As long as we provide a way to do it the solution can be expanded upon.

2. BACKGROUND AND RELATED WORK

To be able to grasp the concept of how HTTP-adaptive streaming (HAS) and geo-based streaming (GBS) works we first present background on HAS and GBS in order to further strengthen our methodology and the interpretation of our result. Since we are using HAS and GBS when programming our interface there is a need to study existing works and articles. By using that knowledge it becomes possible to implement a new upgraded media player that is adapted to streaming from different geographical positions.

There are many related works to our work. Many of these works are focusing on branching videos in media players, which describes ways to allow for users to seamlessly switch between videos without quality degradation or interruptions [2, 5, 7]. Zhao et al. [7] propose protocols that enables the possibility of scaleable on-demand content with minimal server load and developing a way that limits the lower bound bandwidth requirement using multicast [7]. Other works talks about policies for providing a good way of prefetching several videos in different ways, providing means of allowing prefetching and instantaneous playback without playback quality degradation. The work studies the off-periods observed in HAS-players to utilize it as effectively as possible [6].

There are also works that have looked at optimiziation of video quality by observing and controlling the playback buffer by in turn looking at the network capacity, providing an algorithm for optimizing the video quality without any unnecessary buffering [8].

2.1 HTTP-based Adaptive Streaming

Mobile users streaming media sometimes suffers from playback interruption when faced with a bad wireless connection. HTTP-adaptive streaming seeks to resolve this by dynamically changing the bitrate and therefore quality of the stream to make do with the connection that is available to the user. To ensure smooth transitions between these quality swaps HAS also tries to predict the swaps in advance using various methods depending on the HAS framework. There are many algorithms for these predictions, but a brief example would be to use previous logged connectivity history and future connectivity using geo-based methods to make predictions [1]. Most HAS players uses weighted average of past download time/rates in order to estimate download rate of available bandwidth [2]. With these HAS predictions, a stream quality fitting the user's network quality can be buffered [1].

With HAS-adaptive streaming it is needed for us to prefetch data from several close-by streams (if not all, depending on number of them) and build up a small enough buffer that makes switching between different streams seamless. By looking at how HAS is used when implementing an interactive branched video we can say that parallel TCP connections is a must in-order to achieve this with a cost of wasting bandwidth and lower playback quality. This depends mainly on the number of videos that needs to be prefetched. Most HAS video players has a cap on the buffer size in order to avoid wasting bandwidth. Krishnamoorthi et al. [2] use a customized HAS player that solves the problem of trade-off between quality and number of chunks downloaded. The playback chunks are stored in the playback buffer while the prefetched chunks are stored in a browser cache thus allowing those chunks to be retrieved quickly. This ensures that no playback interruption occurs for the user. The way they download the chunks are done in a round-robin way to ensure that a buffer workahead is built up enough for seamless playback in parallel TCP downloading.

Downloading chunks in a round-robin way is how chunks will be downloaded in our media player. This way will be used together with the idea of prefetching in the downtime of a HAS-player. Most HAS-players has some kind of buffer treshold T_{max} where downloading is interrupted when reached and will resume only when the minimum buffer T_{min} is reached. This kind of behaviour can be called an *on-off behaviour* which can lead to poor performance under conditions with competing traffic. It is common in several HAS-players like Netflix and Microsoft Smooth Streaming for example. Krishnamoorthi et al. [6] provide policies and ideas that reduce the start-up time of videos by an order of magnitude and ensures the highest possible playback quality to be viewed. A way of improved channel utilization that allows for instantaneous playback of prefetched videos. They provide an HAS solution which we will take advantage of together with prefecthing nearby streams in a round-robin way. The solution allows for prefetching and buffer management in such a way that videos can be downloaded parallel and switched to instantaneous without interrupting the user experience. By using a novel system to utilize the unused bandwidth during off periods this allows for videos to simultaneously be prefetched while maintaining a fair bandwidth share. It also increases the playback quality in which a video is downloaded [6]. This idea will be discussed further in section 2.2 when we describe our idea of downloading streams.

There can occur several problems in HAS players [2]. Huang et al. [4] show that when a competing TCP flow starts a so called "downward spiral effect" occurs and the downgrade in throughput and playback rate becomes severe. This is caused by a timeout in the TCP congestion window, high packet loss in competing flows and when a client has a lower throughput the playback rate is lower due to smaller buffer segments which makes a video flow more susceptible to perceiving lower throughput and thus creating a spiral. A possible solution is to have larger segment size and by having an algorithm which is less conservative, meaning that requesting a video at lower rate than is perceived. This is something that we can keep in mind since quality can decrease drastically when having several videos buffering in parallel, though we will not have to buffer a full video at the same time but only chunks of a video while the main stream is being watched.

2.2 Non-linear Streaming and Multipath

Krishnamoorthi et al. [5] use this technique of storing prefetched chunks in a playback buffer simliar to [6]. Prefetching for different branches to allow seamless switching between videos, using the notion of multi-path nonlinear videos to stitch together videos using a novel buffer management and prefetching policy. This prefetching decreases the time it takes to switch between branches considerably and is something we will take advantage of since the code we use from [2] are based on a similar policy [5]. If we look at what Zhao et al. [7] wrote they describe how choosing a correct branching point sufficiently ahead of time with an accuracy of 75 % greatly reduces bandwidth requirements. Requesting nonlinear video content where chunks are downloaded parallel without causing jitter. This is something which is really important and efficient for users that would like the ability to switch between different videos on-demand. Selecting what type of chunks should be downloaded is hard to accomplish, atleast on a broader context when considering watching TV stream during TV broadcasting [7]. Most of these works are mostly focused on branching videos which is simliar but

not entierly simliar to what we will be doing [2, 5, 7]. We will contribute more to the possibility of prefetching several videos parallel and then be able to switch to any of them on-demand. However the ideas used when handling branching videos is something that will be used in our mediaplayer.

Figure 1 and 2 illustrate an example of a stream consisting of chunks being played, how these chunks are prefetched and stored and a swap between two streams.

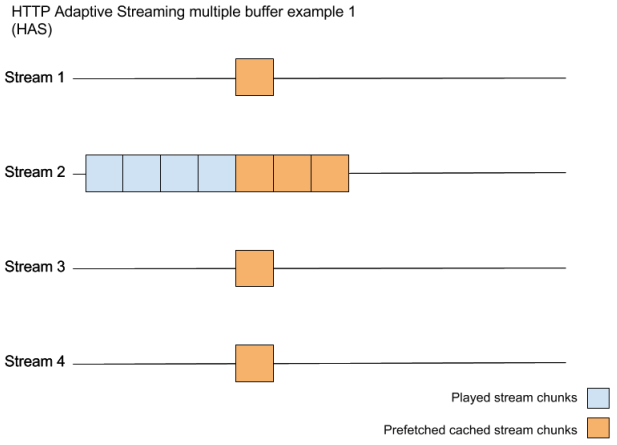

Figure 1: HAS Parallell Stream Buffer 1

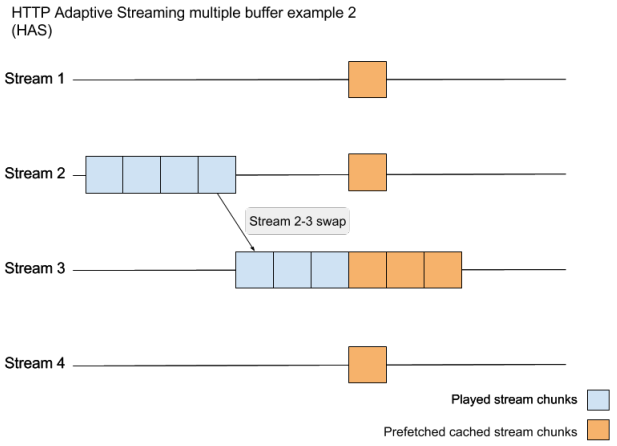

Figure 2: HAS Parallell Stream Buffer 2

2.3 Strobe Media Playback

To display the stream in our application we will be using a media player called Strobe Media Playback (SMP), created with the Open Source Media Framework (OSMF) by Adobe Systems. The OSMF itself is build upon Adobe Flash Player, while becoming more outdated by day and discontinued by some, is widely used for media and other graphic applications and suffices to use for the proof-of-concept of our application. In practice this means that the media player is created using the tools that OSMF provides, compiled into a runnable flash file bytecode and run by Adobe Flash Player. OSMF supports a number of important features that will be used within our interface. Most importantly it enables the use of HAS with its HTTP streaming support and progressive downloading. It also enables the player to seamlessly switch between several media elements by using a composition of "nested serial elements", which will be prominently used within our application. [3]

3. SYSTEM DESIGN

To advance in this project we will mainly be programming, designing and developing the application. The programming language of choice will be Java Action Script and the IDE Flash Builder, which is very similar to the IDE Eclipse. The functionality of the interface to be developed is multiple. We want the interface to accept incoming video streams tagged with a location and cardinal direction from expected sources. The video streams will have to be tagged with these geographical datas somehow, which is not a common included feature with most video recording softwares. Developing a separate recording application to create these kind of video streams, for the sake of this project, might be outside of our goal of limitations for this project. If it is, we will prove the functionality of our interface with fabricated video geo-tags. These streams will then be made to work with the custom OSMF player. Under-the-hood features will include HAS to ensure a smooth playback of the streams, both for buffering a single stream but also for prefetching and buffering a fraction of the other streams to ensure uninterrupted playback during stream swaps. To help us focus on the main problem of developing this interface, we are being provided with some existing code by our supervisors. This includes a working SMP player created with a modified version of OSMF with code from an existing HAS-interface using prefetching [2].

3.1 Interface design

The main part of this project is to expand upon the existing user interface (UI) of the default SMP player, as seen in Figure 5, and create a new section of it where we can implement the new desired functionality of this project.

In practice we decided to go with adding an additional button to the control bar of the UI. When pressed, the graphical interface similar to the one in Figure 4 is shown in the media player. Within this graphical interface, the user can hover over the arrows representing the available video streams located at different geographical locations and angles. While hovering over an arrow a tooltip is shown with some info about the video in question, such as GPS Coordinates and the angle, to give the user a comprehensive overview of the available streams. Finally, when an arrow is clicked the selected video is played with a seamless transition thanks to HAS.

The layout will also display a view of every stream and an optional "Point of interest" with its own geographical position. This point of interest is the center of all the different video streams and can be anything from a concert to some other large event. The added geographical view will also display the north, west, east and south cardinal directions to know the angle of the every stream relative to them. The angle θ in Figure 4 can be calculated by taking the magnitude heading from a recording client. This will give us the

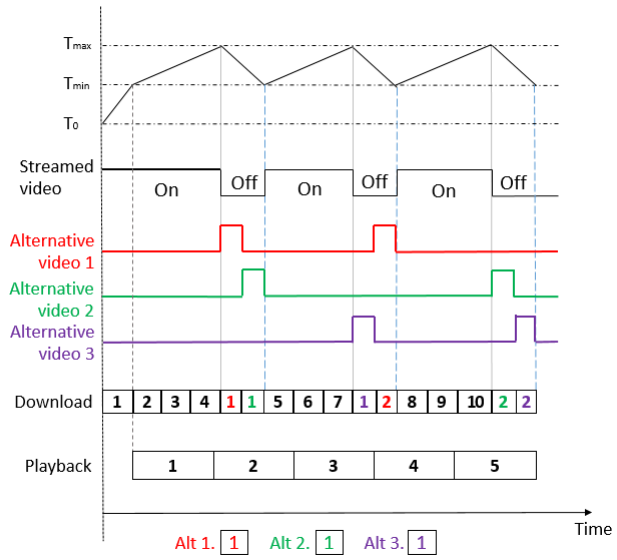

Figure 3: Prefetching overview

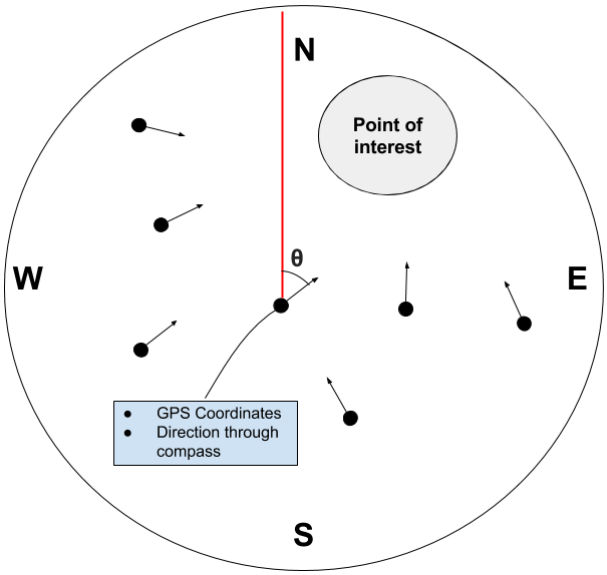

Figure 4: Concept interface of GPS and Direction selection map

direction relative to the north cardinal direction.

3.2 Multipath

As mentioned briefly in section 2.1 chunks will be downloaded in a round-robin way and chunks will be downloaded only during the downtime of the HAS-player. Krishnamoorthi et al. [6] mention a policy called *best-effort* that we will use, in which chunks are only downloaded after the buffer size has reached T_{max} and will start to prefetch chunks from several other videos. These chunks are only going to download as long as the time it takes to download them doesn't go below T_{min} of the currently streamed video. The policy adapts to available bandwidth and varying network conditions. It is also one of the better policies discussed since it downloads chunks of as many videos as possible which is a needed and important functionality in scenarios with many different streams [6]. In Figure 3 an idea of this can be seen. Other nearby streaming videos will only be downloaded ones T_{max} is reached. A nearby video will be prefetched only in few chunks and the videos are downloaded in a round-robin way. Alternative video 1 followed by 2 and so on. Ones the T_{min} is reached the main video is resumed downloading. The idea that would be best but will not be implemented is what video should be prefetched first or if that could be chosen. Prefetching distant videos may be better because they are probably more likely to be switched to. An interesting idea but not considered here for our proof-of-concept.

The SMP player is by default set to play a stream of video located at a server supporting HTTP-streaming. For this project we'll be using the Adobe Media Server 5 for enabling the chunked video streaming needed for our HAS functionality. Since we will be using a similar OSMF player that were used in Krishnamoorthi et al. [5], the quality of our prefetched chunks will be adaptive to the available bandwidth. [5].

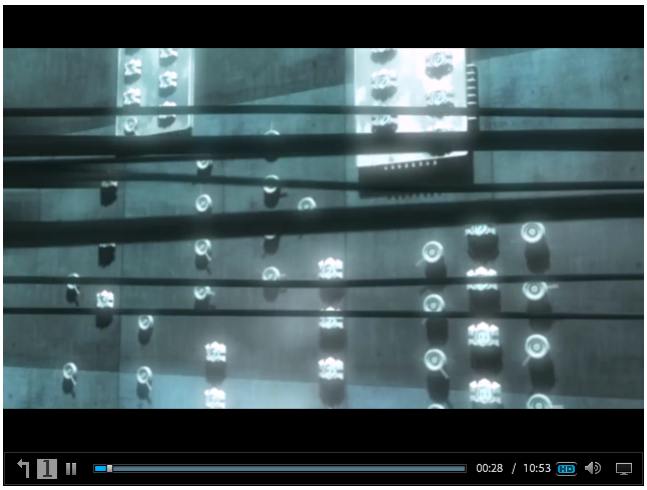

Figure 5: Strobe Media Player

3.3 Relative placement of Geographical Points

The interface accepts an arbitrary number of video streams coupled with a cardinal direction and GPS-coordinates, including latitude and longitude values. The graphical points representing these video streams with coordinates should then be placed and scaled relatively to each other on the interface's geographical map. To accomplish this automatic placement and scaling we developed an algorithm to calculate where the points should be drawn to keep their relative positions between each other, so the graphical points accurately represents the real life locations of the recordings.

3.3.1 Geographical Position algorithm

The way the algorithm works is that every streamer and point of interest is an object in a list. Every object starts

with having it's position in the middle of the so called geomap, a view similar to the interface in 4. What the algorithm does is that it goes through each and every object in the list and checks it's coordinates relative to the other ones. How this works is that the objects checks the distance between each other object by calculating the difference between the real-life coordinates of each one and gets a relative distance between each other. This is effectively done by using Pythagoras theorem like this:

$$\begin{aligned} x &= (longitude2 - longitude1) * 40000 \\ &\quad * cos((latitude1 + latitude2 * \pi/360)/360) \\ y &= (latitude1 - latitude2) * 40000/360 \\ z &= \sqrt{x^2 + y^2} \end{aligned}$$

In the equation above x and y are the difference between two objects longitude and latitude values translated to x- and y-coordinates to fit our geomap as it represents a view on a flat plane. If we had used latitude and longitude as x and y values instead then the positions wouldn't translate as well to our view because they translate to the earth's spherical plane and as result the relative distance between each object would be weird. The view wouldn't give a good accurate picture of reality.

With x,y and z can the relative move distance be calculated. The distance in which an object will move relative to another one.

$$\begin{aligned} valueX &= moveDistance * |\frac{x}{z}| \\ valueY &= moveDistance * |\frac{y}{z}| \end{aligned}$$

ValueX and ValueY are the distance in which an object will move in x-axis and y-axis respectively and moveDistance is a constant value which represents how much an object should move, this is set as the radius of the geomap to help with scalability. The reason that we take the absolute vale of x/y divided by z is because of how the checking of where an object should move is done. When we have our scaling value the algorithm will check how to move the object relative to real-life. A check to see if the object in real-life is more to the west, north, south or east in-order to know the direction for moving the object in x-axis and y-axis on our flat plane.

For scalability of our algorithm the value of moveDistance will be changed accordingly to the number of objects to place. This is done by dividing the moveDistance to number of users. The reason for this is because every object will move relative to every object in the list thus the distance one object has to move will be moveDistance times the number of objects. Another reason for doing this scalability is to ensure that an object won't be placed outside the geomap, however if that would happen then there is another algorithm which will check if an object is outside the map and adjust it accordingly.

3.3.2 Limitations and Accuracy

The geographical position algorithm places every object relatively good compared to reality but it is not perfect and there is room for improvements. The accuracy of the algorithm is generally good for the most cases but sometimes the relativity can seem a bit of and values aren't placed in the right place so to say. However the object is not placed in a completely wrong place and is more that the objects x- and/or y-coordinate is wrong relative to another object. Now this may be the case because we translate from latitude and longitude coordinates but if do the same positions but if we try with fewer objects then we can see how the position should be for the view in a flat plane.

When testing some cases it shows that when values the number of objects exceeds 10 then a few objects are placed in relatively different way compared to when the objects are fewer than 10. We are not entirely sure why but since our algorithm uses very simple equations and ideas then accuracy may decrease for many values because of our way of scaling the distance to move. The scaling distance may get so small that when rounding up to an integer the value will be of bay a small margin. This is the case when the few objects are placed relatively wrong, even though it seems wrong the margin of error is not big. Every cases tested shows that the margin in which the placement seem off is by a very small position in x-axis and y-axis. This will be shown in *Section 4*.

For now the algorithm will utilize a countermeasure, which also can cause the relativity to seem off but it seems relatively better. This countermeasure is that the way x-coordinates and y-coordinates is calculated in another way.

$$\begin{aligned} x &= longitude1 - longitude2 \\ y &= latitude1 - latitude2 \end{aligned}$$

The equation above is simpler and doesn't translate longitude and latitude to x- and y-coordinates so there is room for improvement but for many objects that isn't as big of an issue. When the number of objects are many then there isn't as much of and issue since every object will move so little compared to one another that how much they move in x- and y-axis isn't as important.

(Tänkte skriva i resultatdelen om hur placeringar ser ut och vart saker placeras samt nämna hur många object som algorithmen klarar av att placera ut på relativt bra sätt mm.)

3.4 Technical details

Här skriver vi vad vi gjort i koden för att få allt att fungera. Att vi använt Advertisement plugin för att få saker att funka, och att vi dragit om lite kablar så att advertisementplugin ska fungera med flera videos och att det ska kunna gå att pausa och byta tid i filmerna (vilket inte gick innan ty advertisementplugin är gjort för opausbara reklamer).

To be able to accomplish switching between videos and getting a functional UI there are a lot of technical details to be explained in-order to get a full understanding of how the

code works. Since we used the code from Krishnamoorthi et al. [2] there is first a lot to understand before even starting doing anything. The problems we had and complications we encountered will be explained in the discussion while the focus here will be on **our** code and implementations.

Our progressing can be divided into different sections which will be explained in a general detail:

1. Making a button to open the view.
2. Making a view appear with point of interest and cardinal directions.
3. Making clickable geo-map objects appear.
4. Connecting each geo-map object to a video and be able to play it through a class called AdvertisementPluginInfo.
5. Making the geo-map videos interactable.
6. Attempting to make the geo-map switching seamless.
7. (Adjustments and improvements of the code.)

The details of the code and implementation will not be explained line by line but will give more a general idea of what was done.

The first step was to make an interactive button which we open our view with. For this we had to create three different assets for how the button should look like, which we created in photoshop. The button is illustrated as three arrows with a dot at each end facing a general direction. This shows that a view is opened with objects similar to those. These buttons where then added to a SWC (ShockWave Component) file which stores the assets. The assets was then handed an assets id and name so they could be retrieved. A class for the button was created that was added to the control bar. The button extended *ButtonWidget* where we could add the assets to a face which allowed us to switch between the different assets when changing face.

The second step was to make a view appear that is represented as circle to better fit with how geo-map objects will be placed. For this a widget and sprite class was created. The geo-map widget class handles size of the clickable layout, creation of the geo-map view and handling of fullscreen. The geo-map view is placed in the middle of the stage for the player and when fullscreen is initiated everything will be scaled such that relativity is kept. In the geo-map sprite class the position algorithm, creation of every object and cardinal direction is handled.

In the third step we created a new class called *GeoMapObject* which will hold all our functions of the streaming video is shown on the view. This class will have functions to add and get the position of the geo-map object, the latitude and longitude, direction, setting the URL to be connected with the object etc. The geo-map object which is created in geo-map sprite is added to a list. This list will handle all the geo-map objects on the view and is used for when clicking on an object. Together with a function in the geo-map object class it helps to show which object is clicked on and make sure that no more than one object is highlighted at the same time.

When getting to the fourth step the technicalities became a bit more complicated and here is when the servers are used and getting the videos to play was about to happen. The server will be explained more later and also the difficulties and problems that occurred. For this step each video in the geo-map objects needed to be played and therefore a class called *AdvertisementPluginInfo* which is a class used for playing videos in the beginning, middle or end of a video. In this project a modified function of playing the video in the middle was used and made it so that instead of waiting for the video to change the switch and loading of the video would happen directly when the object is clicked on. To get this to work the class needed to first stop the main video and signal that another video is playing. For this the main media player from the Strobe media playback needed to be fetched and sent in to the *AdvertisementPluginInfo* class. This was solved by creating the geo-map button in the SMP class and then sending in an instance which was forwarded to the geo-map objects. This way the media container and media player that the SMP played mainly could be stopped and removed. When this was done the *AdvertisementPluginInfo* class could change between the different videos as they were an advertisement, this meant that only playing the video was possible but not interacting with it.

Step five, which was about getting the interaction for the video to work, is the most difficult task of them all.

3.5 Server and video application

Här skriver vi hur vi praktiskt tillämpat adobe flash-servern och hur den hänger ihop med Apache-servern. Även hur spelaren och mediaservern bara accepterar .f4v-filer och att vi konverterat våra egna filmer till det formatet.

As said earlier the server used is the Adobe Media Server 5 (AMS 5), which is primary used for downloading videos from cache as similar to the works described in *Section 2*. The AMS 5 is a server used for HTTP streaming which is needed to use HAS. The AMS 5 uses something called an Apache server which enables a video to be called with HTTP. To stream videos with the AMS 5 there can be a need to allow the the flash player to stream an HTTP through the local media player[2] otherwise there can occur an security error.

4. VALIDATED RESULT

Testfall Mätningar Lite bilder där vi kör programmet med egna filmer

5. DISCUSSION

Här prata om vilka svårigheter vi haft, vilka ändringar i planen vi gjorde från början till slutet, vad vi kompromissat med. Även att vi hoppade prefetching/överlämnade det jobbet till Vengeth?

6. CONCLUSION

[2] *https : //www.macromedia.com/support/documentation/en/flash*

7. REFERENCES

[1] Jia Hao, Roger Zimmermann, Haiyang Ma, *GTube: Geo-Predictive Video Streaming over HTTP in Mobile Environments.*
Published: 2014. Fetched: 2016-03-09
URL: http://goo.gl/6DiQiW.

[2] Vengatanathan Krishnamoorthi, Niklas Carlsson, Derek Eager, Anirban Mahanti, and Nahid Shahmehri, *Quality-adaptive Prefetching for Interactive Branched Video using HTTP-based Adaptive Streaming.*
Proc. ACM International Conference on Multimedia (ACM Multimedia), Orlando, FL, Nov. 2014, pp. 317–326.

[3] Greg Hamer, *Open Source Media Framework: Introduction and overview.*
Published: 2010-03-15. Fetched: 2016-03-09
URL: http://www.wi-fiplanet.com/tutorials/article.php/3433451/Implementing-Wi-Fi-Multicast-Solutions.htm.

[4] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, *Confused, timid, and unstable: Picking a video streaming rate is hard..*
In Proc. ACM IMC (November 2012).

[5] Vengatanathan Krishnamoorthi, Patrik Bergström, Niklas Carlsson, Derek Eager, Anirban Mahanti, and Nahid Shahmehri, *Empowering the Creative User: Personalized HTTP-based Adaptive Streaming of Multi-path Nonlinear Video.*
Proc. ACM Proc. ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking (FhMN), Hong Kong, Aug. 2013, pp. 53–58.

[6] Vengatanathan Krishnamoorthi, Niklas Carlsson, Derek Eager, Anirban Mahanti, and Nahid Shahmehri, *Bandwidth-aware Prefetching for Proactive Multi-video Preloading and Improved HAS Performance.*
Proc. ACM International Conference on Multimedia (ACM Multimedia), Brisbane, Australia, Oct. 2015.

[7] Zhao, Y., D. L. Eager, and M. K. Vernon, *Scalable On-Demand Streaming of Nonlinear Media*
IEEE/ACM Trans. on Networking, Vol. 15, No. 5 (Oct. 2007), pp. 1149-1162.

[8] T. Huang, R. Johari, and N. McKeown. *Downton abbey without the hiccups: Buffer-based rate adaptation for HTTP video streaming.*
In Proc. ACM SIGCOMM FhMN Workshop, Aug. 2013.