Institutionen för datavetenskap
Department of Computer science

Examensarbete

Geo-based media player

An interactive interface for geo-based video streaming

by

Andreas Nordberg and Jonathan Sjölund

LIU-IDA/LITH-EX-A--16/001--SE

2016-05-13

Linköpings universitet
Institutionen för datavetenskap

Examensarbete

Geo-based media player

An interactive interface for geo-based video streaming

by

Andreas Nordberg and Jonathan Sjölund

LIU-IDA/LITH-EX-A--16/001--SE

2016-05-13

Handledare: Niklas Carlsson and Vengatanathan Krishnamoorthi
Examinator: Nahid Shahmehri

Abstract

Being able to interact with video streams can be both fun, educational and provide help during disaster situations however, to achieve the best user experience, the interaction must be seamless. Creating an interface for a media player which allow for users to view multiple video stream of the same event from different geographical positions and angels 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. An algorithm is developed for placing each video stream object on the interface's geographical based map automatically. These objects are placed to ensure the relative positions of the objects compared to the real world. The end result of this project is a proof-of-concept media player which enables a user to see an overview over a geographical streaming area. By seeing each streams relative location to the point-of-interest the users are able to click on that stream and switch to it, however not in a seamless way. The result of this project shows the command-and-control center as initially envisioned but without seamless, uninterrupted, switching between the video streams. Instead the work done and the end result code provided will allow for prefetching to be easily patched in into any future work.

Acknowledgments

We would like to thank our supervisor, Niklas Carlsson, for this project assignment, continuous support and assistance during the project. Niklas' colleague Vengatanathan Krishnamoorthi has also been an immense assistance to help us understand how we would go about with the huge chunk of provided code and has helped us pass many code-related obstacles throughout the project. We also want to acknowledge Adobe Systems Incorporated for the development enviroment used during this project to create our interface. This enviroment includes the tools used such as the IDE Flash Builder[1], the Open Source Media Framework[2] and the Strobe Media Playback[3] built upon this framework.

---

[1] Adobe Flash Builder http://www.adobe.com/products/flash-builder.html
[2] OSMF https://sourceforge.net/projects/osmf.adobe/files/
[3] Strobe Media Playback https://sourceforge.net/projects/smp.adobe/files/

Contents

List of Figures

1   **Introduction**

Streaming has evolved and become more popular over the 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 and view streams 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. This project focuses on accomplishing this, 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 improving it to accomplish this task, we show that it's a feature 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 is 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 should then be able to automatically accept multiple coordinates for these streams and draw them in this graphical interface. 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. One major thing this interface also could be used for is during a disaster event or something of the sort, by helping the police, medical or the emergency service by allowing them to view a disaster scenario from multiple angles to help them in their communication. In such a scenario, being able to swap between different video streams would give them a better understanding of the scenario and what needs to be done in their work.

---

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

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 designing a pretty interface and a more extensive focus on user friendliness 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 initially 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 our main focus is to make sure that it is possible to switch between video streams, not that it's possible to do so with a large number of streams. The reason for this is that pre-buffering many videos can be difficult to accomplish with a large number of video streams and it can come with a trade-off of bandwidth usage [**watchingprefetching**] and less efficient bandwidth when in downloading in parallel [**scalableOnDemand**]. As long as we provide a way to do it the solution can be expanded upon afterwards.

2   Background and Related Work

To be able to grasp the concept of how HTTP-adaptive streaming (HAS) and geo-based streaming (GBS) works a background is presented on HAS and GBS in order to further strengthen our methodology and the interpretation of our result. Since the use of HAS and GBS is essential when programming the functionalities of the interface there is a need to study existing and related works. This knowledge is important to be able to implement an upgraded media player that is adapted to streaming and seamlessly switching between videos from different geographical positions.

There are many related works to our project. 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 [**qualbranch**, **hasmultipath**, **scalableOnDemand**]. Zhao et al. [**scalableOnDemand**] 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 [**scalableOnDemand**]. 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 [**bandawarePrefetch**]. 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 [**bufferbased**].

Some works have dwelved more into how to prefetch in an effective way. Khemmarat et al. [**watchingprefetching**] have looked at different ways to prefetch videos in-order to allow for the best user experience. While watching a Youtube video, the user experience is significantly increased when the time the video is paused and buffering is minimal. They provide an approach that tries to predict which video a user will click and and prefetch it. By looking at three different schemes they found that by combining caching and prefetching the hit-ratio, for which a clicked video is prebuffered, would increase up to 80 %. Their proposed schemes improves video playback in a way that it avoids playback delay. The trade-off of a higher bandwidth usage is minimized when combinding prefetching and caching of videos and by not having large amount of videos to prefetch. If this were to be combined in some way with what we want to accomplish with prefetching then it will allow for a much better video experience.

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 [**gtube**]. Most HAS players uses weighted average of past download time/rates in order to estimate download rate of available bandwidth [**qualbranch**]. With these HAS predictions, a stream quality fitting the user's network quality can be buffered [**gtube**].

By implementing HAS into our interface we need to prefetch data from several close-by video streams at the recording area (if not all, depending on number of them) and build up a small enough buffer that makes switching between these different streams seamless. By looking at how HAS is used when implementing an interactive branched video we can say that parallel TCP connections are a must in-order to achieve this with the 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. [**qualbranch**] 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 we plan chunks will be downloaded in our media player. This method 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 [**bandawarePrefetch**]. Krishnamoorthi et al. [**bandawarePrefetch**] 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 prefetching 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 instantaneously 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 [**bandawarePrefetch**]. 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 [**qualbranch**]. Huang et al. [**streamrate**] 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. Then 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 sizes and by having an algorithm which is less conservative, meaning that a video is requested at lower rate than it's perceived. This is

something to 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. [**hasmultipath**] use the 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 Krishnamoorthi et al.[**qualbranch**] are based on a similar policy [**hasmultipath**]. If we look at what Zhao et al. [**scalableOnDemand**] wrote they describe how choosing a correct branching point sufficiently ahead of time with an accuracy of 75 % greatly reduces bandwidth requirements by requesting nonlinear video content where chunks are downloaded in parallel without causing jitter. This is something which is really efficient and important 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 [**scalableOnDemand**]. Most of these works are mostly focused on branching videos which is simliar but not entirely simliar to what we will be doing [**qualbranch**, **hasmultipath**, **scalableOnDemand**]. 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 media player.

Figure 2.1 and 2.2 illustrate an example of a stream consisting of chunks being played, how these chunks are prefetched and stored and a swap between two streams.
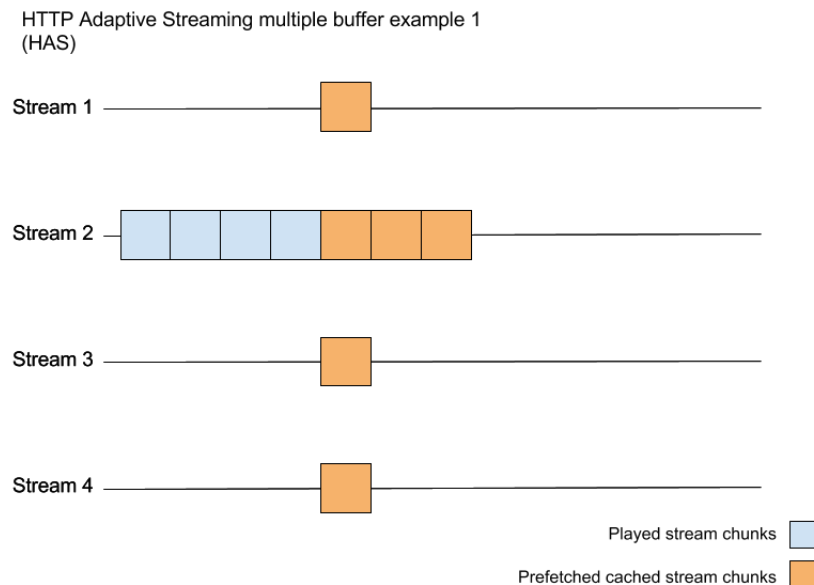

Figure 2.1: HAS Parallell Stream Buffer 1

There have been works that have looked at a way of optimizing periodic broadcast delivery for non-linear media, by creating functions and an algorithms that provides a way to effectively control over quality of service for clients with varying playback paths. They
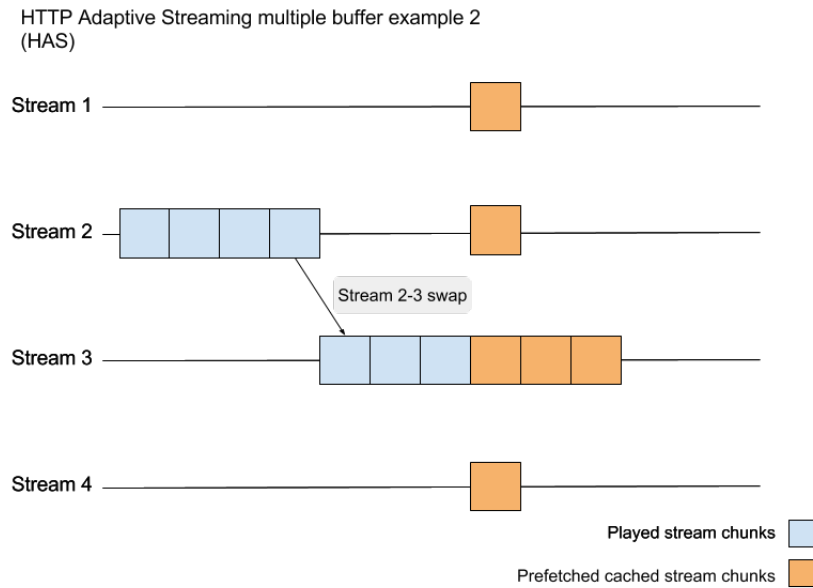
Figure 2.2: HAS Parallell Stream Buffer 2

look at cases were clients makes a path selection at their arrival instance over branching tree paths and graphs and show that the start-up delay increases exponentialy with the number of branching paths and that linear increase in bandwidth decrease the start-up delay exponentialy.[**optimizedbroadcast**]

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.[**osmf**]

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 Adobe ActionScript 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 geo-tagged video streams, for the sake of this project, is outside of our goal of limitations for our interface. Because of this, 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 [qualbranch].

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 3.3, 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 3.2 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 in the area. While hovering over an arrow a tooltip is shown with some information 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.

Along with these arrow objects representing the video streams in the graphical interface, the layout will also an optional "Point of interest" with its own geographical position. This
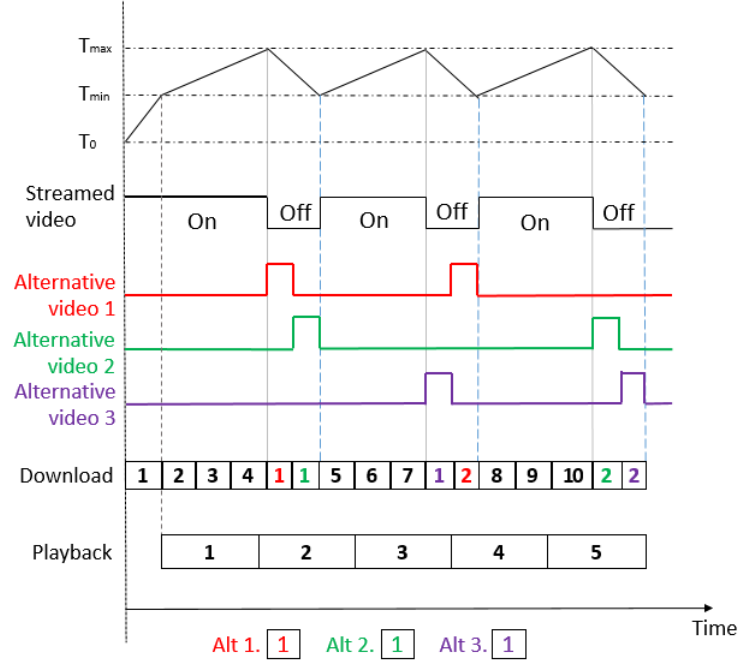
Figure 3.1: Prefetching overview

point of interest is usually the center of attention of all the different video streams and can be anything from a concert to some other large event. The implemented 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 3.2 can be calculated by taking the magnitude heading from a recording client. This will give us the direction relative to the north cardinal direction.

3.2 Prefetching principle

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. [**bandawarePrefetch**] 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 [**bandawarePrefetch**]. In Figure 3.1 an idea of this can be seen. Other nearby streaming videos will only be downloaded once T_{max} is reached. A nearby video will be prefetched only in a few chunks and the videos are downloaded in a round-robin way. Alternative video 1 followed by 2 and so on. Once the T_{min} is reached the main video resumes its downloading. The idea that would be best, but will not be implemented, is what video should be prefetched first, or if it should be chosen. Prefetching distant videos may be a better choice because they are probably more likely to be switched to. An interesting idea but not considered for our proof-of-concept interface.
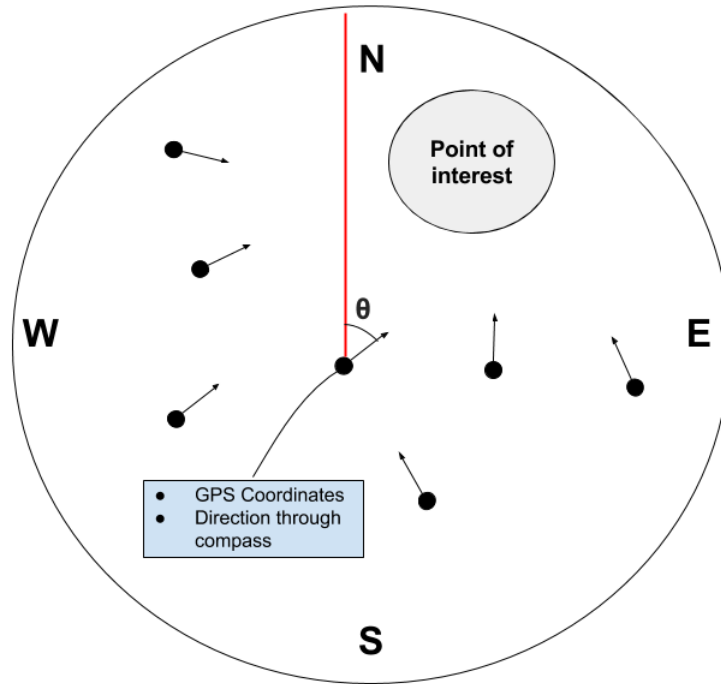
Figure 3.2: Concept interface of GPS and Direction selection map

3.3 Server Integration

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. [**hasmultipath**], the quality of our prefetched chunks will be adaptive to the available bandwidth. [**hasmultipath**].

3.4 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, as shown in Figure 3.2. To accomplish this automatic placement and scaling an algorithm was developed to calculate where the objects should be drawn to keep their relative positions between each other, so that the graphical points accurately represents the real life locations of the recordings.

3.5 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 geographical map. What the algorithm basically does is that it goes through each and every object in the list and checks its coordinates relative to the other ones. This is done be checking the objects' actual real-life distance between each other by calculating the difference between the coordinates of each object to get a relative distance between them. This is effectively done by using Pythagoras theorem like this:
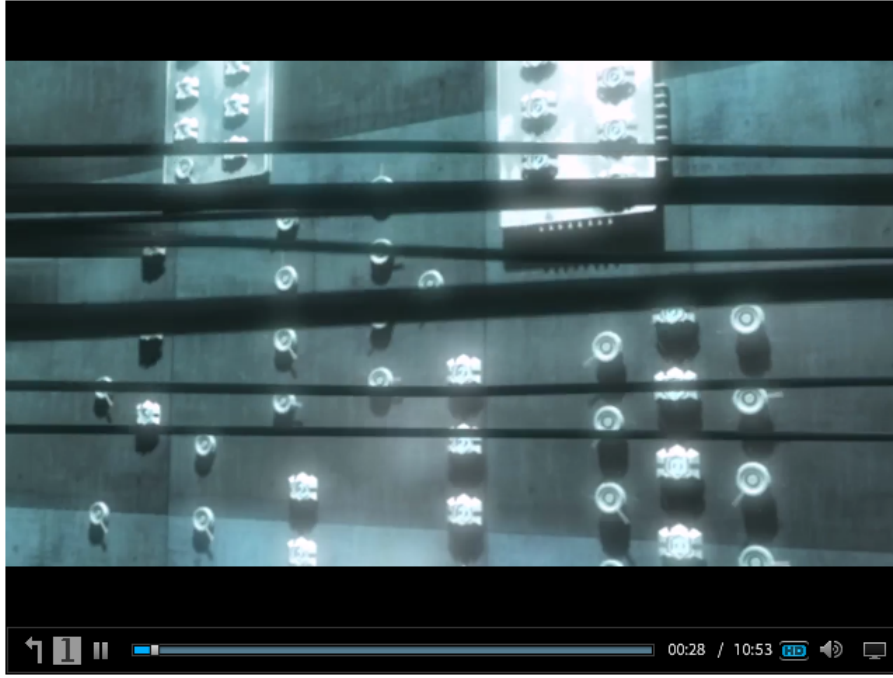
Figure 3.3: Strobe Media Player

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

The equations above show the equireqtangular approximation formula. It's a simple method of calculating the distance between two geographical points on the surface of a spherical area [**equi**]. In the formula we approximate the earth's circumference as 40000 km. X and y are the difference between two objects' longitude and latitude values translated to x- and y-coordinates to fit our geographical map as it represents a view on a flat plane. If we had used latitude and longitude as plain x and y values instead the positions wouldn't have provided a good enough accuracy to the flat x-/y-plane because of the spherical nature of latitude and longitude coordinates. An alternative method of calculating the distance between two objects would have been the Haversine formula, which excels at accuracy along high distances [**haversine**]. For smaller distances used however, as in our project, equireqtangular projection suffices.

With x, y and z the relative "move distance" can be calculated. This move distance is a fixed distance in which an object will move relative to another one in the geographical map, depending on the number of objects to be drawn.

$$valueX = moveDistance * |\frac{x}{z}|$$
$$valueY = moveDistance * |\frac{y}{z}|$$

ValueX and valueY are the distances in which an object will move in x-axis and y-axis respectively and moveDistance is the 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 we take the absolute value of x/y divided by z is to check how much an object should move in the x- and y- axis. When we have our scaling value the algorithm will check how to move the object relative to real-life. This check is 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 by the 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 fail-safe algorithm which will check if an object is outside the map and adjust it accordingly.

3.6 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 in most cases but sometimes the relative placement can seem a bit off. However in this case the object is not placed in a completely wrong place, but the object's x- and/or y-coordinates is wrong relative to another object. This mostly happens if we have too many objects, one or two objects relativity will be off but not with a large margin. The reason can be because we convert the latitude and longitude to x- and y-coordinates, however this cannot be the whole problem since relativity is generally good for a few objects (less than ten objects).

While testing, in some cases when the number of objects exceeds 10, 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 the 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 by a small margin. This may be the case when a few objects are placed relatively wrong, even though it seems wrong the margin of error is not big. Every case 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 *Chapter 4*.

For now the solution to relative placement of many objects, when the number exceeds ten, will be handled by another equation.

$$x = longitude1 - longitude2$$
$$y = latitude1 - latitude2$$

The equation above is simpler and doesn't translate spherical longitude and latitude coordinates to flat x- and y-coordinates which leaves room for improvement, but for many objects in a small area this isn't as big of an issue. When the number of objects are many then there isn't as much of an 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. This does not fix relativity entirely for many objects but when tested it looked better compared to the previous equation.

3.7 Technical details

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.[**qualbranch**] there was first a lot to understand before we could start doing anything. The problems we had and complications we encountered will be explained in the discussion section of this paper while the focus in this section will be on **our** code and implementations.

Our progression 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 a 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. Adjustments and improvements of the code and the implementation of a position algorithm.

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

The first step was to make an interactive button which we open our graphical interface with. For this we had to create three different colored assets for how the button should look like, which we designed on our own 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 were then added to a SWC (ShockWave Component) file which stores the assets. The assets were then given an assets id and name so they could be retrieved using these as references. 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", a kind of state, 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 the layout of the clickable layout, the creation of the geo-map view and the handling of fullscreen. The geo-map view is placed in the middle of the stage for the player and when fullscreen is initiated the graphical interface will be moved and scaled in such a way 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 hold all our functions of the streaming video to be shown in the mediaplayer. This class have functions to add and get the position of the geo-map object, the latitude and longitude of the real life recording position, 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.

Continuing to the fourth step, the technicalities became a bit more complicated and this is the part when the servers came into play and getting the videos to show up on the mediaplayer. More details about the server will be explained in 3.8, and also the main problems and difficulties that occurred for us. For this step each video in the geo-map objects needed to be played and we were provided with a class called *AdvertisementPluginInfo* for this, which is a class created for the purpose of playing advertisement videos in the beginning, middle or end of a video. In this stage we modified the functionality of this class to instead of playing the video halfway through the video, we changed the time from "halfway through the

video" to at the start of the video instead, so that instead of waiting for the video to change the video switch 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 mediaplayer 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 a reference 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 multiple advertisement, which meant that only playing the videos was possible but not being able to interact with them.

Step five, which was about getting the interaction for the video to work, was the most difficult task of them all. Since the videos were played as an advertisement some things needed to be changed, because these advertisement videos was set to not be interactable through the user interface. The main thing here is mediaplayer still recognizes the non-advertisement video as the main media from the Strobe Media Playback while the geo-map interface's videos was only some advertisements on top of it. What we did to fix this was to rewire all of the graphical user interface in a way that you would be able to control the advertisements with it. In other words instead of playing, pausing and interacting with the user interface for the main video, a check is done for the controls. What this check does is that it checks if an "advertisement" is being played and if it is, then the controls will be changed to affect the advertisement instead.

In the last step adjustments and improvements was done to the code and also the implementation of the position algorithm. Here the code was adjusted and improved to make sure that the implementations which were done wouldn't crash anything else. Here the *PointOfInterest* class was implemented to better fit the relative position algorithm. Since the algorithm uses a list of all geo-map objects there was need for *PointOfInterest* to be an object that uses similar functions to the ones in the geo-map object class.

3.8   Server and video application

As previously mentioned in the report 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 *Chapter 2*. The AMS 5 is a server used for HTTP streaming which is needed in order to use HAS. The AMS 5 uses something called an Apache server, specifically Apache 2.4, 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 a HTTP-video through the local media player[1] otherwise security errors may occur. The reason for this security error being that a call is made in the code to a plug-in which allows for sending and requesting a URL to be played.

Except for using the AMS 5 to play a video through HTTP the video also needs to be in format of F4V or FLV which are two different video file formats used for delivering videos over internet using Adobe Flash Player. Every video that has been filmed has been converted to FLV with FFmpeg[2] which is a free open source software project that produces libraries and programs for handling multimedia data. It contains a program which allows us for transcoding media files.

---

[1]Global Security Settings panel: https://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html
[2]FFmpeg: https://ffmpeg.org/

4 Validated Results

To demonstrate our geo-based mediaplayer, we went out and did some recordings to test every functionality we've been working on in the mediaplayer during this project. We went to "Blåa havet" in front of Kårallen located at Linköping Universitet where some students were promoting an upcoming event with some activities. We found that this was a suitable point of interest to record from different angles for our testing case. As we only had two cameras available at the time we made three sets of recordings consisting of two recordings each, with each set displaying the same scene from two different locations and angles at the same time. The desired outcome of this test was to be able to swap between the recordings to view the same object at one point in time from different angles.

4.1 Position Algorithm

To demonstrate the accuracy of our relative placement of geographical points in our interface, we noted the GPS-coordinates and angles at the used recording locations. We then input the coordinates into Google Maps as seen in Figure 4.1, which we will use here as a reference to prove the accuracy of our placement algorithm. We then input the same latitude, longitude and angle values into our interface and the result is shown in Figure 4.2.

The arrow-points in the geographic map is almost an exact match to those in the Google Maps screenshot, at least in terms in relativity. There is a slight difference between the two and the reason for this is that the points in our geographical map is scaled to make use of the entire map, in such a way that the distance between them are increased while their relative locations between each other remain intact. This would prove the accuracy of our relative placement of the geographical points, at least for a number of objects less than 10.

4.2 Geo-based Streaming

As we've mentioned before, our implementations is as shown in Figure 3.2 where we have a button that opens the geographical map, a circle that represents a "map" and arrows pointing in a direction that represents streamers/videos. When a video is selected the arrow is highlighted and that video is then played. In our test case, we set up two cameras at a time and did recordings of 90 seconds each. In these videos we captured plenty of people doing
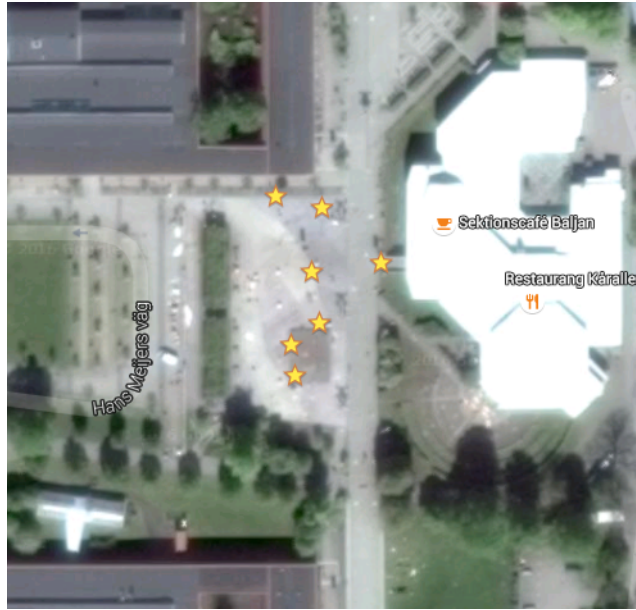
14

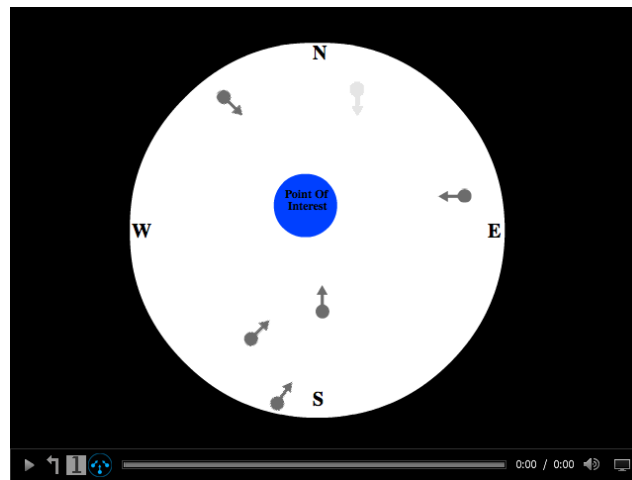Figure 4.1: Google Maps view of the Streaming locations

Figure 4.2: Geo-map view of the Streaming locations

plenty of stuff. There were people jumping the trampoline, using hoverboards, walking and biking around. When we input these 3 sets of two recordings each into our mediaplayer, we could swap between the two recordings of each set and watch these same events unfold from different positions and angles. In Figure 4.3 and 4.4 two different recordings are selected and they show the same event where, for example, the guy inside the red circle in the pictures are hoverboarding in front of the red shirt guy the same time of the videos. This would prove the desired functionality of where the user can display the same event from different geographical positions and angles.

Figure 4.3: Test view 1

Figure 4.4: Test view 2

5   Discussion

5.1   Understanding the provided code

When we started working on this assignment to make an interactive Command-and-Control center with geo-tagged streaming we first had to install and adjust to the tools given to us to develop the interface, being OSMF and SMP. These tools consisted of an extensive amount of existing code which we had to delve into and understand for us to implement our features. This was a process which took some time since we also weren't very familiar with the language enviroment, Adobe ActionScript 3.0. ActionScript is an object-oriented programming language developed by Adobe Systems and influenced by JavaScript, while its syntax still being relatively similar to Java which we had previous experience with. Eventually we got a better understanding on how to operate in this new enviroment and reverse engineer the provided code. However there were still many sections of the code which we didn't understand or knew that we would need in our work, and wrapping our heads around this took more time than we initially expected.

5.2   Issues with HAS and prefetching

At the start of this project we focused much of our time to understand the principles of HAS, geographical based streaming, prefetching and how to implement them into our own interface. While we did have a good grasp on how these principles works and had a good idea of how we would go around to implement them, we couldn't quite get it to work. Since we used code from a previous work we made the assumption that as long as our implementation of our interface's features was similar to that previous work, the HAS would function. Flash builder, SMP and the HAS-functionality in the provided code required the video files to be split into the formats F4M, F4X and F4F when doing the prefetching. We were also provided with some video test files from our supervisor which he had successfully used when he worked on the HAS-functionality in his code. This however didn't work for us since some codebits didn't run properly. There are two things that may be the cause of this. The first thing is that we didn't do what was neccessary to get it to work because our lack of understanding of how the HAS-functionality actually operates in the code and how we would need to rewrite the existing code to function with swapping between several videos. It didn't work out of the box because HAS in the provided code was hard coded to only support one video

17

and our attempts at supporting multiple video streams ended in failure even with the help of the HAS-functionality code's author himself. The second cause of this might be because the changes we did to the provided code in our implementation ruined the functionality of HAS. If we were to look at those two cases the first one seems to be the more plausible one, since we assumed that the code we got would just work as long as we had the assets and did a similar implementation to the one our supervisor had done. The second one seem less likely since the changes we made to the code was so that it wouldn't disrupt the HAS or media player in anyway, however it could also be a possibility.

Because we couldn't get the HAS-functionality to work properly we therefore couldn't get the prefetching of different video streams to work. Our focus and time throughout most of the project was very much put on the prefetching, but since we couldn't get it to work we switched our focus to a better implemented and functional command-and control interface. This included improving the interface to work properly whether the player was in standard or fullscreen mode, each geographical map object displaying GPS-coordinates and direction of the video stream while hovering over it and the relative position placement algorithm for drawing the objects. The position algorithm took some time to implement but we had initially a general idea of how it should work. When we developed it we worked on two similar but separate solutions each to see which one worked best, but since it took more time than expected only one solution was finished in time which proved feasible and then used. The main challenge with developing this algorithm was to provide relativity, scalability and accuracy up to our standards which caused the algorithm to take some time to create.

5.3 Improvements the position algorithm

When developing the position algorithm we looked at several ways to translate the spherical longitude and latitude to accurate grid x- and y-coordinates. In the end the choice was made between the two formulas haversine and equirectangular approximation. The formula we decided to use in the end was equirectangular projection because that's the first one we tried to implement with the algorithm and it worked well. Since the accuracy of equirectangular approximation apparently is slightly worse than that of the haversine formula, we could have compared the use of both formulas to see if there were any significant difference in the implementation between the two. Nonetheless the final algorithm is up to the standard that we envisioned.

5.4 The test case

For our test case, there is one thing we in hindsight would have changed if we would have redone it. In our case we set up only two cameras at a time to get multiple views of what was happening at the scene from different locations, simultaneously. To further and better prove the functionality of our user interface in a test case, we should have brought some more volunteers and cameras along with us to get even more point of views of the same scene at one point in time. While doing two recordings at once was enough to prove the functionality of this feature, more recordings would have been a better addition.

5.5 Adobe Flash

Furthermore, as mentioned previously in this report, Adobe Flash is becoming more deprecated by the day even by Adobe themselves. Because of this, if the project was redone the interface would be better suited to be implemented in the mediaplayer built from a more modern alternative such as Flash's main competitor, or rather replacement, HTML5.

5.6 Issues with the server

One big obstacle which unnecessarily cost a lot of time was setting up the server we used. Initially we used something called a WAMP server at the start of the project which enabled us to stream videos using HTTP through an Apache HTTP Server. However, since idea of prefetching was still present at that point of the project there was a need to switch to Adobe Media Server 5 since it would allow us to stream chunked bits of video used for the prefetching. While setting up the servers we ran across numerous problems with different kinds of security errors which wouldn't allow us to stream the videos using HTTP. While trying to solve these issues we found that since the Apache server ran on a Windows 10 client there was a process that blocked the server that was needed to be stopped. Only then was the server able to run and allow videos to be streamed with HTTP.

5.7 Project structure improvements

If the project was redone we would have made a more definite time plan of what was needed to be done. Our time plan, even though straigthforward, wasn't very detailed. We knew what we wanted to accomplish and when but we didn't really know how we would go about to accomplish it. This ended up unnecessarily consuming a lot of time since we didn't know where to look in the giant web of provided code to solve any eventual issues or where exactly to implement the changes and solutions.

6   Conclusion

This project provides a command-and-control center UI which allows for video streams to be changed on-demand with the use of an interactive geographical position map, with video stream locations tagged with GPS-coordinates including latitude and longitude values and cardinal directions, implemented in Strobe Media Playback. The interactive map provides details of where streamers are positioned relative to each other. This is handled through an algorithm that places every object on the map relative to how the objects, representing the recordings, in the real world are located. The accuracy of the algorithm is shown to be placing each object relatively good to one another with a good accurracy, at least when the number of objects doesn't exceed ten. By creating an object with a latitude, longitude and direction it will show this information of the object while hovering over it. Besides the locations of the streams there is also a point of interest drawn in the map which is also using this algo-rithm. Each stream is clickable and displays its representative video in the mediaplayer when clicked. The object representing the current recording playing will then be highlighted when the geographical map is reopened to show that this video is currently loaded and played. All the videos are interactable and when switching videos the point of time in the recording will be transferred to the next video in order to allow for the user to see the same situation and swap between different locations and angles. Even though seamless, uninterrupted switch-ing through prefetching was not achieved because of several difficulties the code provided is made in a way that prefetching should be easy to implement with our developed interface. The features of this interface should allow for a good way for people to stream and interact with videos during a concert or disaster event in way that will make experience and work easier to accomplish. For future work, when prefetching is added, the switching between different videos will be seamless and further improve the quality of service of the geo-based media player.

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/