Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

A Proof-of-concept Implementation of a Non-linear Video Player for HTTP-based Adaptive Streaming

by

Patrik Bergström

LIU-IDA/LITH-EX-G—13/002—SE

2013-02-18

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköpings universitet
Institutionen för datavetenskap

Final thesis

A Proof-of-Concept Implementation of a Non-linear Video Player for HTTP-based Adaptive Streaming

by

Patrik Bergström

LIU-IDA/LITH-EX-G—13/002—SE

2013-02-18

Supervisor: Niklas Carlsson
Examiner: Niklas Carlsson

Abstract

Video consumption on the Internet has been growing over the last decade and is expected to continue to increase. Video streaming is a widely used approach to viewing video on the Internet, which allows users to watch the video while it is being downloaded. Adaptive streaming is a video streaming technique that allows the player to change the downloading video's bit rate depending on the user's available bandwidth.

Another approach to a more personal viewing experience is non-linear videos. These videos can be played in a non-sequential order. For example, a viewer can be given the choice of what ending to watch in a movie, or the path through an exhibition.

This thesis will present the design and implementation of a novel structure for non-linear video. This structure is used by a video player for downloading and viewing an adaptive video intended for non-linear viewing, stored at some server. Media creators will also have an easier time to both visualize and create uniform video experiences.

This thesis presents modifications to Adobe's Open Source Media Framework and Strobe Media Playback which allow playing non-linear video. Presented in this thesis are the design and implementation details. Changes in the player include a user interface for non-linear media. The changes to the back-end include buffer management for parallel downloading and techniques to handle the new structure.

Finally, this thesis presents proof-of-concept validation tests that show the functionality of the design and implementation. The thesis is concluded with a discussion of future work in this area.

Keywords: non-linear video, streaming media, OSMF, video streaming over HTTP

Sammanfattning

Konsumtionen av video på Internet har ökat det senaste decenniet och trenden är förväntad att fortsätta stiga. Strömmande video är en ofta använd lösning för att se video på Internet. Tekniken tillåter användaren att se film medan den laddas ned. Adaptive streaming är en teknik som låter videospelaren ändra bit-raten på videon baserat på användarens tillgängliga bandbredd.

En annan lösning för att göra användarens upplevelse mer skräddarsydd är ickelinjära videor. Den här typen av videor kan spelas ur sekvens. Som exempel kan en tittare välja slutet i en film eller välja vägen genom en utställning.

Examensarbetet presenterar designen och implementeringen av en ny struktur för ickelinjär video. Denna struktur används av videospelaren för nedladdning och visning av adaptiv video avsedd för ickelinjära filmer, som sparas på en server. Mediaskapare kommer också att ha lättare att både visualisera och skapa filmer på ett enhetligt sätt.

Det här examensarbetet presenterar i detalj de modifikationer på Adobes Open Source Media Framework och Strobe Media Playback som tillåter spelning av ickelinjära videor. Ändringarna innefattar ett användargränssnitt för ickelinjär media. Back-end implementeringarna innefattar bufferthantering för parallell nedladdning och tekniker för att hantera den nya strukturen.

Slutligen presenterar det här examensarbetet proof-of-concept-tester för att styrka funktionaliteten av designen och implementeringen. Rapporten avslutas med en diskussion om framtida arbete inom fältet.

Nyckelord: ickelinjär video, strömmande media, OSMF, http-streaming

Acknowledgement

I would like to thank Niklas Carlsson for the opportunity to do this thesis, for his continual feedback during the project and for help with the more theoretical parts of the thesis.

I want to thank Vengatanathan Krishnamoorthi for taking time to help me start up the project and wrap it up towards the end.

Last, but not least, I want to thank my opponent Tim Samuelsson.

Linköping, February 2012

Patrik Bergström

Table of contents

List of figures

List of Abbreviations

DRM	Digital Rights Management
F4M	Flash Media Manifest
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
OSMF	Open Source Media Framework
RTMP	Real Time Messaging Protocol
SMP	Strobe Media Playback
SWC	Shockwave Flash Component
SWF	Small Web Format
XML	Extensible Markup Language

1 Introduction

The video usage on the Internet has been growing the last decade and watching the video while it is downloaded is now common place. For example, real-time entertainment in the US accounts for nearly 50% of the traffic during peak period, compared to 30% as of two years ago [1]. High cellular network traffic has also been observed with Netflix and YouTube, accounting for roughly 50% of the traffic [2]. In 2015, video is expected to make up for nearly two thirds of all Internet traffic [3].

Personalized user experiences are becoming increasingly important. One approach is adaptive streaming in which the quality of the video is changed based on current bandwidth conditions. In this work we also consider a more general type of personalization, in which the user is provided with multiple path choices through a media object. We call such media object non-linear. In contrast to traditional linear media (in which the user views the content along a single linear timeline), non-linear media lets the user view the content in several different paths (often structured in a graph-like timeline).

1.1 Motivation

Non-linear media offers the user to choose different paths through the movie; for example choosing a good or bad ending or navigate through a presentation. Google offers a service[1] where the user is able to virtually explore wonders of the world. Computer games are also examples of non-linear media since the player can decide what to view. There also exists gaming services[2] that renders the game on a server and show the user the resulting video frames. There currently are no open-source non-linear players.

1.2 Problem description

This thesis presents a new application for non-linear streaming application based on a pre-existing HTTP-based streaming service. It also contains preliminary results of using the proof-of-concept player.

1.3 Contributions

This thesis extends the Open Source Media Framework (OSMF) and Strobe Media Playback (SMP) to allow for creation and viewing of non-linear video. To increase the user experience the player downloads fragments in parallel, buffering multiple potential path choices in parallel, which allows deferred path decision and smooth transitions when the user decides to jump in the movie. To help media creators visualize non-linear media we also develop a structure for representing non-linear media. A customized button is used to provide users with path choices. No modifications are needed on the server side.

[1] http://www.google.com/culturalinstitute/worldwonders/
[2] http://www.onlive.com/

In summary, this thesis makes three significant contributions:

- Creating a new media file structure. The structure was created to be simple while still being efficient. Like previous research, [6] [7], the structure is based on trees.
- Modifying an existing player through design and implementation of buffer management and use of a metadata file to handle the media structure.
- Proof-of-concept validation tests based on different scenarios which show the results of the modifications to the player.

1.4 Structure

Chapter 2 provides an overview of existing streaming systems and the state of the art, including a summary of related works. Chapter 3 describes an overview of non-linear media, the system's structure design and the program's architecture. Chapter 4 describes the implementations. Chapter 5 presents the test methodologies that were used. Chapter 6 presents preliminary results and chapter 7 concludes with a discussion and summary of this thesis' contributions.

2 Background

2.1 State of the art

Today the majority of streaming is done using HTTP-based protocols. HTTP-based streaming protocols can be either progressive or adaptive. With progressive streaming the client will download the entire file in a given quality, regardless of the current bandwidth conditions. This technique is used by for example YouTube.

To increase the quality of service and to utilize the available network infrastructure better, as well as using the server's and client's resources better, adaptive streaming is being adopted by more and more service providers.

Also, by allowing the streaming client several different qualities of the video the client is able to choose the most suiting at the time.

There are two types of adaptive streaming: chunk-based and range-request-based. With range-request-based adaptive streaming, as used by Netflix, the client downloads ranges of bytes of the video. With chunk-based adaptive streaming the video is split up into a few seconds long fragments which are downloaded separately. This technique is used by Microsoft Smooth Stream and Adobe's OSMF.

In this thesis we consider chunk-based adaptive streaming. We select to use the OSMF player. In contrast to the Microsoft Smooth Streaming player which is proprietary, the OSMF player is open source and allows us to modify the player and implement our own protocols.

2.2 Adobe's Open Source Media Framework

OSMF is a back-end for streaming media, it downloads fragments and calculates bandwidth requirements but it can not play the media on its own. The project is compiled into a library which is used by a front-end to handle those parts while the front-end plays the video for the user.

OSMF is an open source solution that developers are free to add their own add-ons to. The application is very modular in its architecture which makes adding plug-ins easier but it can be hard to see the entirety of the architecture. Due to the open source license it is also easier to publish our own versions of OSMF. OSMF is a Flash based solution which means it is compiled into SWF files and executed in Flash Player.

The extensions are based on OSMF version 2.0.2494 which was released at the same time as the thesis project started. The new version added several new features and bug fixes and using an older version would be bad practice. OSMF was initially developed by Adobe and Akamai together before making the code available to the public in 2010.

OSMF stores its media in element classes, such as VideoElement and AudioElement which can be accessed by playback programs. To support logic OSMF abstracts the element classes with trait classes which describes how an element will load or play, for example. The modifications done to the project is implemented in a lower level of abstraction however, and does not use these elements.

For HTTP streaming to work it requires that the server the client is connecting to has a metadata file called a F4M file, containing XML data on how the file is stored on the server. This data can contain links to a set of video files, of the same movie, encoded at different qualities and a description of the bit rate of that file and DRM information. Once the metadata file is downloaded OSMF can generate the URL addresses needed to begin downloading. An examples request for a HTTP metadata file looks as follows:

```
http://www.example.com/media/http_dynamic_Streaming.f4m
```

The client can then parse the F4M for additional metadata, called bootstraps, which contain information about fragments such as their length and grouping. The video is divided into segments, which in turn are divided into fragments which represent a few seconds of video. Once it receives a fragment it will store it in an internal buffer and use index information to calculate at what time the fragment is supposed to be played. A request for a HTTP video fragment can look like the following:

```
http://www.example.com/hds-vod/sample1_1500kpbs.f4fSeg1-Frag4
```

To begin downloading and playing a movie the client will begin to request fragments of the video file in a sequential order. The rate of new requests is determined by the buffer, by default the client attempts to have between four and six seconds of buffered video. If the upper boundary is reached it will not request more fragments and will request at a faster rate if the buffer is below a lower threshold. The player maintains a single TCP connection in which it downloads both audio and video which both exist in the fragment.

The project has support for both RTMP streaming and HTTP streaming. While they both provide adaptive streaming through chunk-based algorithms they differ greatly in protocol.

This project is only focused on HTTP-based streaming. HTTP-based streaming is more commonly used for adaptive streaming and does not require additional ports to be opened [4].

2.3 Adobe's Strobe Media Playback

The strobe media playback presents the media downloaded and is integrated into a web page through a SWF file. Since the strobe playback is open source it is possible to add new widgets and alter behaviour of the player. Unlike Adobe's Flash Player, the Strobe Media Playback requires that the server the user is connecting to has the playback installed.

While it is possible to create a playback from scratch, using an existing one saved enormous amounts of time as well as making the finished prototype more realistic.

The playback is connected to the OSMF back-end on the computer which feeds it the video data. Because of this connection it is also possible for the playback to alter OSMF's behaviour such as setting states.

To use more advanced concepts for SMP such as HTTP streaming and using DRM with the videos it requires Flash version 10.1 or later.

The playback player can be configured through flash vars or by altering the HTML file that loads the SWF file. However, to do larger changes the source code must be altered.

2.4 Related work

Video delivery through rate-adaptive HTTP-based streaming has not been as extensively studied as other protocols and offers more room for research. Akhshabi et al. [4] presents a good overview of how the most popular stream clients work and show the importance of good rate-adaption. Rao et al. [5] and Erman et al. [2] both present measurements of the importance of video traffic on the Internet.

Other work [6][7] has focused on the server implementation to a larger extent while this thesis is mainly focused on the client implementation. This also differs from streaming of artificial worlds or computer games [8] since the clients do not need information about each other, such as displaying each other's avatars.

3 Design

While traditional media is played and stored in a sequential, (list-like) manner, non-linear media is easier described using a multi-path graph structure. For this purpose we will develop a tree structure for non-linear video. A tree will not have the same amount of possible operations as a multi-directional graph, but it is sufficient to test the structure in a short amount of time.

To achieve a structure in the media content, key points in the media must be identifiable. Since OSMF already splits the downloaded media, we pick fragment boundaries to be called decision points. At these points the player decides if it should continue to play in sequential order or if it should skip ahead, potentially several fragments, to a new play point.

To conform to OSMF's normal download structure, and to avoid creating unnecessarily many small files, the media file should still be stored as a single file, but divided internally into smaller clips that can be put together to form a larger cohesive video. Figure 1.a shows how a video file divided into branches is stored.

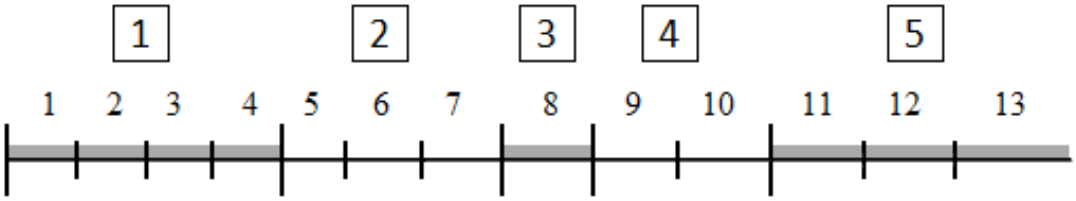

Fig. 1.a) Example of a movie file partitioned into clips and arranged in a list.

To limit the scope of the implementation it was decided to traverse the file from left to right in an ordered manner and not allow jumps backwards in time. This means a tree can be used to describe the structure of the media content where each node is a small clip. Each decision point will then result in two or more new branches from the node. The decision point is thus the last fragment in a series of one or more fragments which make up each node. The decision points require additional information outside of the tree to know which fragments they branch to.

To simplify the player, these tree structures will play in a depth-first left-most child way and jumping will mean taking a sibling to the right. A jump will cause the player to skip fragments in a sequential manner and begins to play a node not directly after, e.g. going from node one to five or fragment seven to nine. To calculate the jump length to a specific point the left siblings of the intended node and their entire subtrees' number of fragments are added together, and are then skipped over in the movie.

Figure 1.b shows the same video as Figure 1.a but structured as a tree.

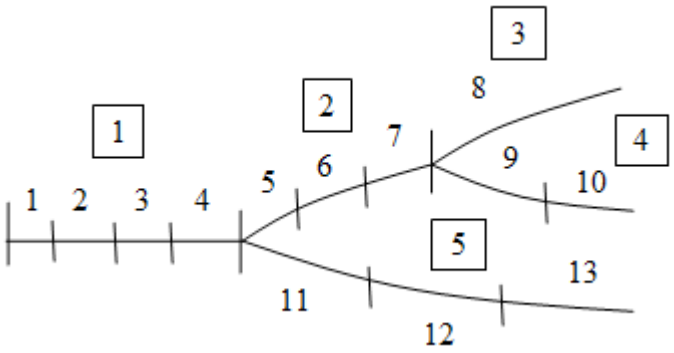

Fig. 1.b) Example of a movie file divided into clips and arranged into a non-linear tree structure.

Compared to other methods, such as breadth-first, using depth-first requires fewer jumps, which in turn means that less information is needed. The player will always continue to download and play sequentially ordered fragments. By using this as a default path, following the left-most child means continuing to play in sequence and these jumps can be implicit and no information needs to be saved. The numbering on a breadth-first implementation would be discontinuous, which would lead to the need to explicitly label every possible jump at each decision point.

To make the structure more general, such as being able to move in any direction in the tree, instead of it being unidirectional requires larger amounts of information and the above solution is likely not a good starting point.

3.1 Program design

Because of the size of OSMF and SMP and the time limit for the project the decision was made to focus on a lower level of implementation rather than a more complete implementation. This means we created a bare minimum of new classes and instead of properly sub-classing existing classes we directly changed behaviour in existing methods. Care was taken to not disturb normal behaviour of the program.

We set up a communication node, or mediator, between existing classes to pass information around as well as deciding suitable behaviour. This mediator class will also handle the fragments, such as downloading and storing them for later use.

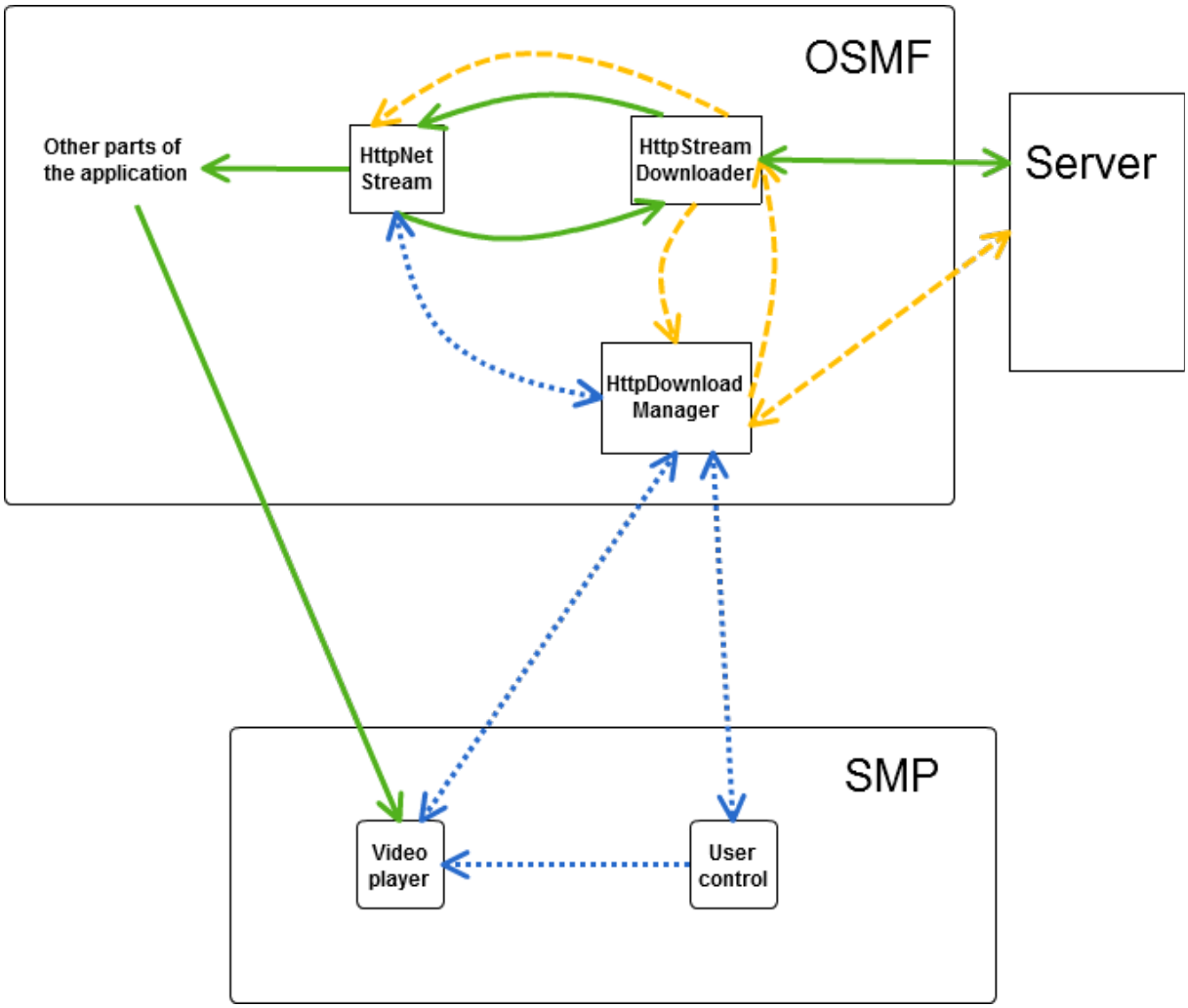

Fig. 2 Program design overview. Some classes have been left out for clarity.

Figure 2 describes the existing architecture along with the added functionality of the `HttpDownloadManager` class. The green solid lines describe the normal path for a fragment to travel. A request for a video begins at the `HttpNetStream` class which signals `HttpStreamDownloader` to begin downloading a fragment from the server. Once the download is completed the data is returned to `HttpNetStream` and back to the rest of OSMF which can pass the video to the playback player (SMP).

The `HttpDownloadManager` will begin to download future fragments in advance and will intercept `HttpStreamDownloader`'s attempt to reach the server and instead give the downloader the parallel fragments when applicable. From outside of this handover the download process looks like normal. A parallel fragments path through the application is shown in yellow.

The dotted blue line represents communication between the `HttpDownloadManager` and the other classes that is not necessary for downloading fragments. The user signals OSMF through the user controls in SMP to perform a jump or not. The `HttpDownloadManager` can tell the video player part of SMP to show messages about upcoming jumps. Both the

video player and `HttpNetStream` know the current playhead time and can inform `HttpDownloadManager`.

OSMF's framework source code is contained in over 350 files and has more than 70 000 lines of code including comments and line breaks. The SMP project is made up of over 20 000 lines of code including comments and line breaks split over 120 files. The full project folder has over 2 400 files in total.

4 Implementation

The implementation is strictly client based with the exception of a structure text file on the server and the requirement of a specific compatible movie file. This means normal servers does not have to change anything except making the extra file available for download. Furthermore, the implementation does not interfere with the player's normal behaviour which means an altered player will still be able to play linear movies without any change at all.

The programming language used was ActionScript 3.0 since both OSMF and SMP are written in it. Adobe Flash Builder 4.6 was used since it helps having a real IDE to work in. It simplified debugging and compiling and also structured the code easier. When compiling the project a SWF file is generated as well as additional files which help displaying it. This file is then loaded by an HTML file which also is created by the compilation process.

Most work on the existing architecture dealt with adding compatibility with the new download manager class. This often meant handling cases based on important fragment numbers for when the player need to access the download manager.

Much of the communication between the download manager and the SMP project was done through events since the program does not stop while the text file is being downloaded and many of the events are timing-based.

To instruct the client to download in a non-linear fashion an additional text file was added which describes the manner the fragments are supposed to be downloaded.

4.1 Download management

In the following we describe high-level solutions for the classes included in the download process.

To make the program function more robust and to make sure internal states and logic were correctly set the altered code often uses functions and similar paths as the normal program would.

To handle the parallel downloads the modified OSMF will open new connections to the server on the same port and store the downloaded fragments in the computer's RAM. Due to the small size of each fragment the increased size of the running program should not be a problem.

Seek functionality: To control the content of the buffer easily the seek method of `HTTPNetStream` was used. Seeking in the program clears the buffer and sets the internal states (such as fragment index) ready to continue playing at exactly the landing point.

Reading bytes: Once the seek function has run its course `HTTPNetStream` will begin trying to load bytes like normal through the `HttpStreamDownloader`, which in turn will return the prefetched data through the download manager, instead of downloading it on its own.

Managing fragments: The download manager manages much of the jumping logic on a branch level, where each branch is consisting of several fragments, usually beginning with a landing fragment and ending with a decision point. At the beginning of a branch, the download manager will find out which is the corresponding decision point and landing point and will immediately begin to download the landing point.

By using this policy the program will not overwhelm the connection by downloading all landing fragments at the start, and will only download the fragments which have potential of being played.

Timed events: Calculations on branches are done through periodic events which are timed to the end of the decision point for each branch. To get the time length for each branch a special time table containing the length of all fragments in the video is used. This table is available before or very early in the video and occasionally updated.

Downloading policy: Due to the looser time constraint on the landing point compared to the sequential fragments the landing point fragment can likely be downloaded in a higher bit rate than normal. The program will always download the highest rate available to show this. This policy is not optimal and does not take into consideration the adaptive streaming which is allowed by the program. For the purpose of a simple proof-of-concept with minimal playback implementation, it achieves the task. The policy of when to download fragments is also relatively naïve and an area for future work.

4.1.1 Alternative buffer handling

A previous method used to download the fragments was to at buffer time (when the fragment is downloaded rather than played) increment the number of the next fragment to be played through the fragment index handler. The next fragment to be downloaded would then be the landing point instead of following the sequential order. This way the buffer would begin downloading the landing point and the following branch immediately, without clearing the buffer.

This resulted in the buffer containing a gap between the first branch fragment numbers and the landing point number, but the data flow would be continuous. The player will continue playing as long as the buffer contains data regardless for order. By letting the downloader work this way it was able to download fragments following the landing point before playing the landing fragment, like normal. Ultimately this method was too unresponsive, as the actual jump is decided several fragments ahead of time.

The buffer is not directly accessible from the classes involved in the download process, instead the classes must invoke specific methods to handle the buffer, such as seek or append. Due to this, the buffer management is rather clumsy.

4.2 User interface

A button was added to the user interface to allow the user to choose whether to perform the jump or not. In the current version it looks like an arrow signifying which way to take and can only choose between two paths. When the button is pressed the arrow will change which direction it points towards, signifying which branch will be taken in the tree structure. In future versions it should be possible to change the behaviour to allow for multi-choice paths. The button was added to the SMP project which is mainly focused on presentation to the user.

To add new graphics to the project the graphic assets file must be updated. The assets are compiled into a SWC file which is loaded and parsed into the project. Changes must be done to a FLA file via a Flash editing program; Flash Professional was used for this. The new graphics must be added and then given names which correspond to what the project expects.

When a decision point is getting closer a message is shown to inform the user about what each path will show, if it is specified by the media creator. This is done by the `StrobeMediaPlayback` by listening to events from the `HttpDownloadManager`, which fires about two fragments before decision points and delivers the message to the player. The message widget itself is a sub-class of the widget classes that SMP uses for user interface.

4.3 Structure metadata file

The structure described in the previous chapter is encoded into a text file available on the server and downloaded by the download manager class once the application starts.

The file contains 4 different fields which describes the non-linear content. The first line describes which fragments of the movie are the decision points. The next line describes the length of each node, which is used to determine the length of jumps at a decision point. The following line describes the structure of the tree. The following is an example:

(1(2(3,4),5))

The line is parsed left to right. Each set of parentheses describes which nodes are accessible if the node to the left of the parentheses is followed. Each comma describes a new branch. Because of this structure it is possible to have several branches per node in the tree and not only a binary one. This example describes the same structure as Figure 1.

The last field is several lines long and contains the messages that will be shown to the user to inform him of his possible choices at the decision points. The size of the text file is kept small to keep the extra downloading and memory at a minimum.

A decision was made to use plaintext instead of a more complex format such as XML or JSON because it is simpler to implement and did not require additional libraries. The amount of data involved is relatively small and easily parsed.

4.4 Download manager

A new class was created to handle the parsing of the text file as well as downloading the extra fragments needed and handle the logic to decide whether to skip or not. The class has a static instance of itself, and is supposed to be used like a singleton but Actionscripts 3.0 lack private constructors. This allows other classes to be sure they are using the same instance of the class. Since the manager is responsible for communication between classes and prefetching fragments it is important that only one instance is used throughout the project.

The largest function of the class is the parse function. Upon creation the class will begin downloading the structure file and will send an event to itself to begin parsing. During the parsing the class will compute and store which fragments are the decision points, how far the jumps are in fragments and the messages that are supposed to be shown to the viewer. Once the parsing is done the manager will send an event to all listening classes. The class has several helper functions based on the fragments, such as finding time offset or index for fragments, and signal dependent classes. The class ended up being over 400 lines of code.

5 Test methodology

Debug messages were added to the program code to make it possible to see the internal variables and states of the player. This was also very helpful in the initial stage to find out how the program operated. By running scripts in Cygwin it was possible to extract information regarding the buffer and the download process from the log file generated by the Flash application.

Preliminary testing began with downloading fragments in parallel, to see how the system worked and identify important classes to the downloading process. This also showed if there was an actual possibility to store fragments in advance and download fragments out of order, or if some other method would have to be devised.

Later the testing iteratively became more like an actual use case. The tree in Figure 1.b was used as the non-linear tree structure in the text file. While the sample video was not created with non-linear media in mind it was still possible to see if the concepts work. Having a theoretical behaviour in mind helped to compare the actual behaviour of the program and see where the behaviour altered from desired behaviour.

Most of the initial testing was performed on sample video file for HDS supplied by Adobe's Flash Server installation. Because the fragments duration in the sample movie consisted of a six seconds long fragment followed by two three second long in a continuous sequence it was simple to calculate where landing points in fragments corresponded in time. The fragments' duration is decided by the creator and by using key frames in the scenes which means fragments can have virtually any duration.

Towards the end of the project the testing also included playing another video instead of the one provided by Adobe. This video did not have the same well structured fragment sequence as the sample video. The results of the tests were very similar in nature to the first video, for space reasons only the results from the first video are presented.

5.1 Test environment

Because all testing was done on a local set-up it was hard to test how the program would act under real circumstances such as a restricted amount of bandwidth or spikes in available bandwidth. Such tests would have been very useful to measure how well adaptive streaming policies worked.

We used Adobe's Flash Media Server 4.5 on the local computer during the testing to have an easily configurable server close to hand. The server supports many different types of streaming protocols including HTTP-based adaptive streaming. Setting up the server was very simple which made it faster to do the implementation part. The extra text file is not hosted by the server program but is instead stored on the local computer, and there was no attempt to include the text file to be part of the Flash server. The text file is downloaded through http and should not require additional ports opened.

The testing was primarily done in Internet Explorer 9 and Mozilla Firefox 17 and using a debug version of Adobe Flash version 11.

The project is tested by opening the generated HTML file containing the compiled application with a web browser. If debug flags are set, Flash will generate log files containing both predefined and the added debug messages.

6 Preliminary validation results

To validate the functionality of the implementation, this chapter presents the results of running the application in different scenarios. These scenario-based tests aim to show that the player is able to download in parallel, as well as the player's ability to seamlessly jump in the video.

In the following, we show results from three different run paths through the media object. These correspond to the three possible paths illustrated in Figure 1. Scenario a follows the default path, plays nodes 1-2-3 sequentially, and does not perform any jumps in the video object, but instead follows the left-most path in the figure. Scenario b switches to the right path at the decision point at fragment 9 and will perform a three second long jump, following the nodes 1-2-4. Scenario c takes the right path early at fragment 4 and will perform a much longer jump between fragment 4 and 11. Scenario c has the shortest path going only through node one and five.

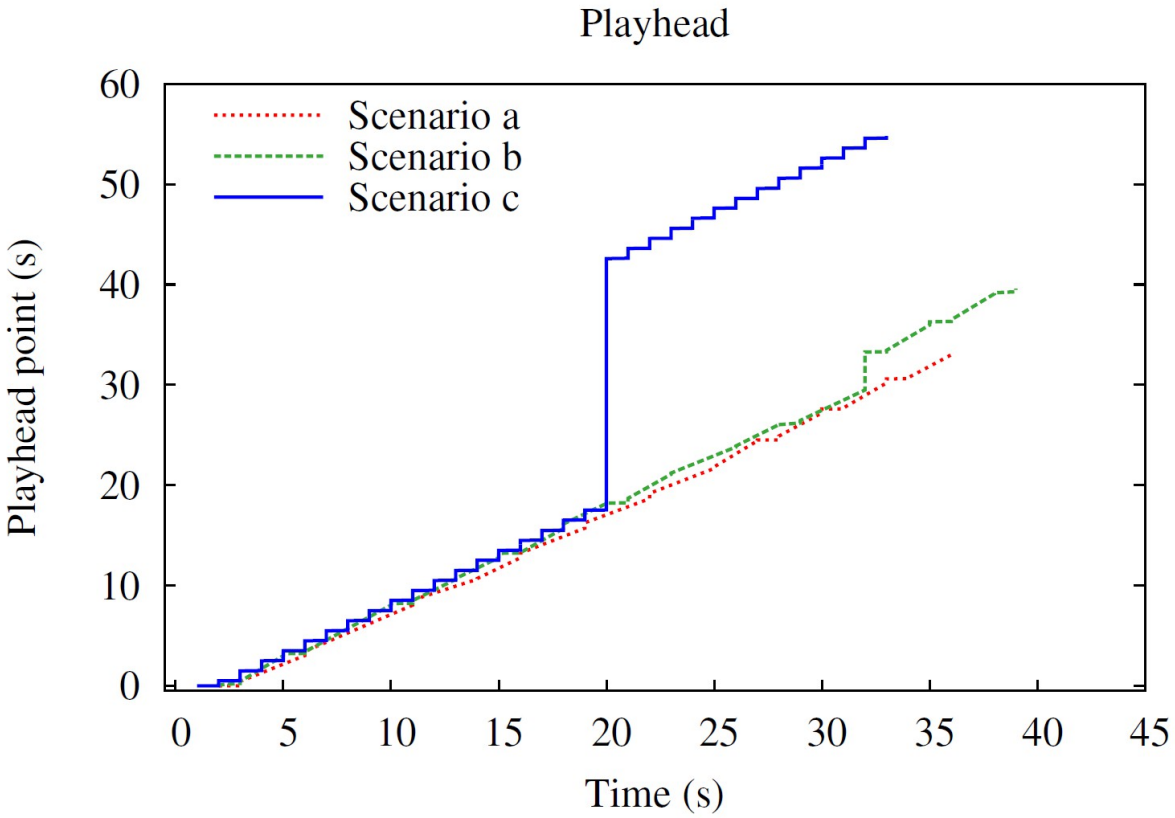

Fig. 3 The playhead time in the player compared to normal time.

The red line for scenario a increases linearly supporting that no jump has been taken. The small jump for scenario b corresponds to three seconds of skipped data, which happens when the player jumps from fragment 7 to 9. The much larger gap for scenario c can be explained by the large jump at fragment four when it skips all fragments associated with the entire left subtree and continues; resulting in a 24 second jump in the media. The data from scenario c was captured at a finer time granularity than the other scenarios to better show the jump in the media object, hence the more jagged appearance. These results confirm that the player is capable of performing jumps in a playing movie.

More careful examination of the results shows that these jumps can be done with minimal playback interruptions. At the time of the writing the player have a small buffering period where the fragment is moved from the internal buffer of the download manager to the playbacks' media buffer.

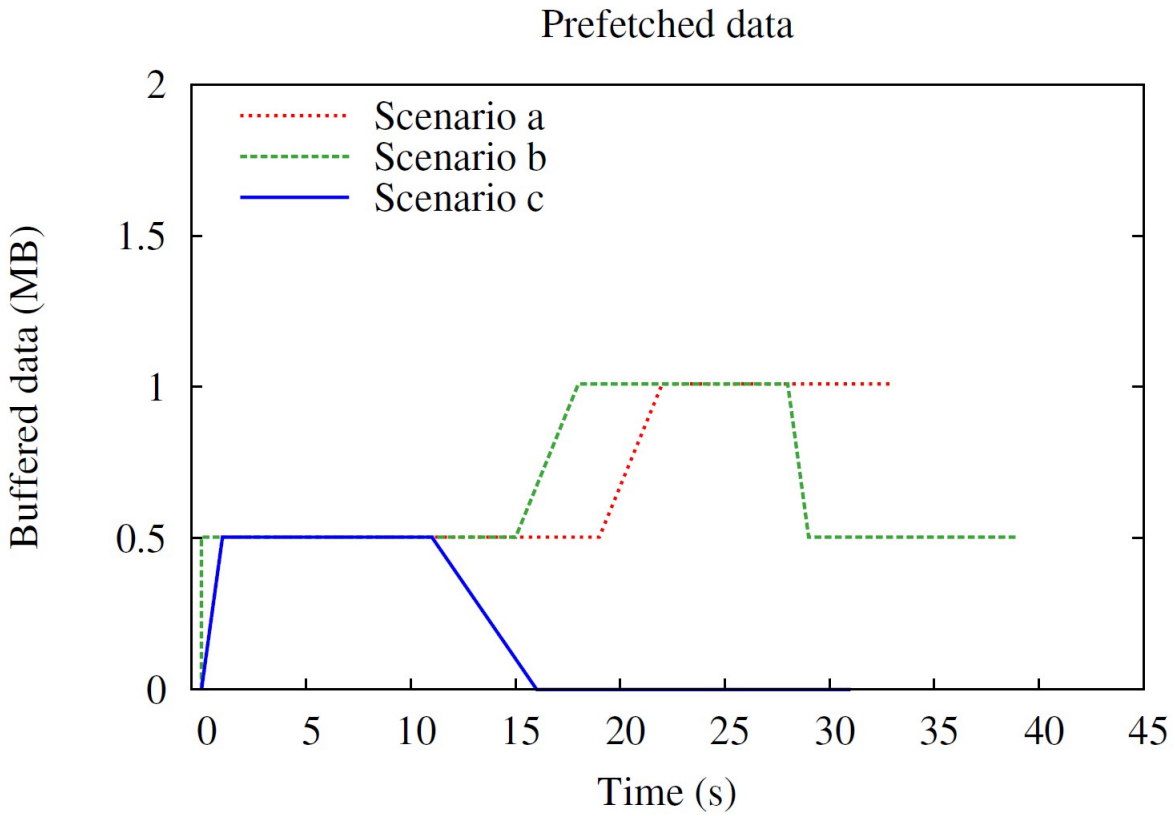

Fig. 4 The player's use of cached data.

We next take a closer look at the prefetching done by the download manager. Figure 4 shows the amount of cached video data in the download manager's buffer. In scenario a's case it does not require to use the cache and thus it does not decrease. Scenario c uses some of the cached fragments early, at 17 seconds, which explains the drop at that point. The same drop happens to scenario b which uses another fragment at a later point in the video. From the figure we can clearly see that the data downloaded by the download manager is being used when jumping in the media. Using this data, rather than downloading the data from the origin servers (at the time of the jump), is what ensures the minimal playback interruptions.

The results of having the landing point downloaded at the beginning of each node can be seen. While all three scenarios download the first landing point, only scenarios a and b download the second landing point. This is because these scenarios begin playing a new node with a landing point attached to it while the third scenario will end after playing its second node.

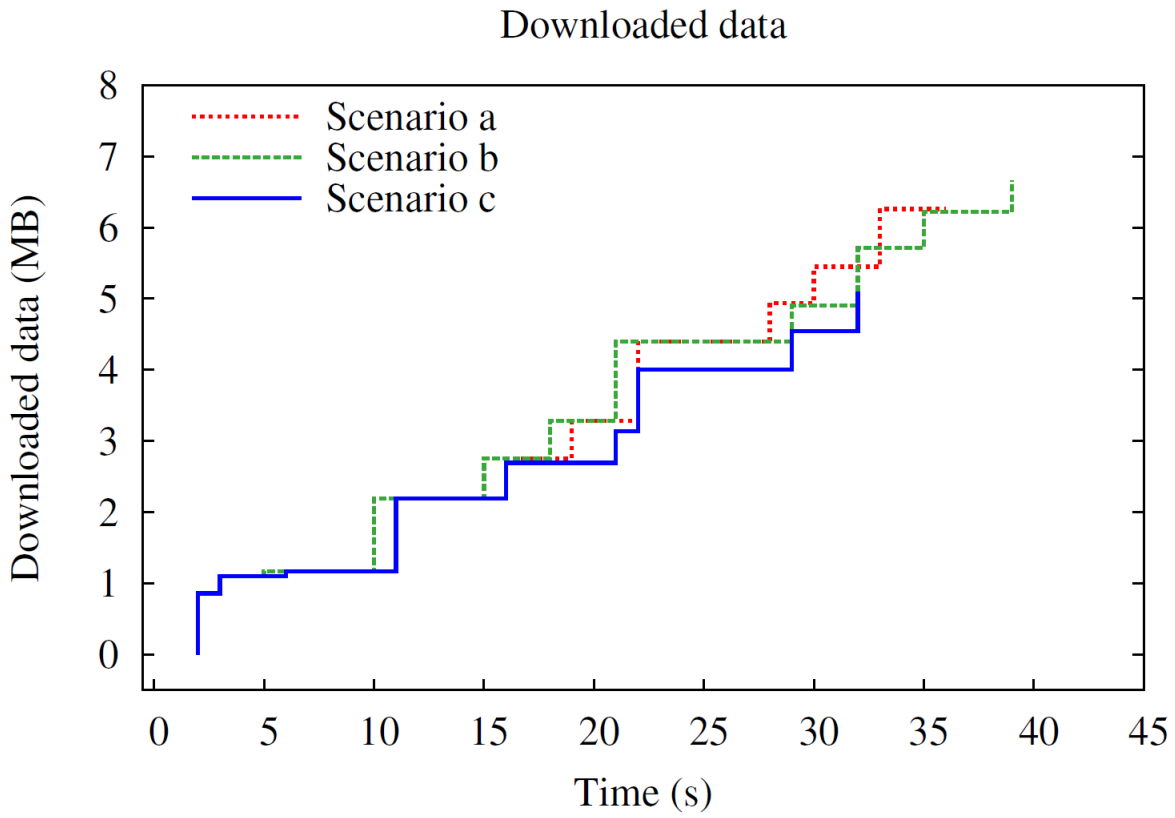

Fig. 5 Amount of data downloaded at a point of time.

We next illustrate that the overhead associated with preloading is small compared to the total amount of played data, and to show that the amount of data downloaded is proportional to the total playback time. Figure 5 shows the amount of downloaded data for the different playback scenarios.

The amount downloaded differs slightly between the scenarios which correspond to when the player reaches the larger six seconds long fragments. The tests show that the player downloads different amounts of data when it follows different paths. The lines end at the point where the player stops downloading.

Note, for example that the total amount of downloaded data (as defined by the maximum y-value), is 6.2, 6.7 and 5. Comparing this against the total played media, we note that scenario b (9 fragments) has the most data, and scenario c the least (7 fragments).

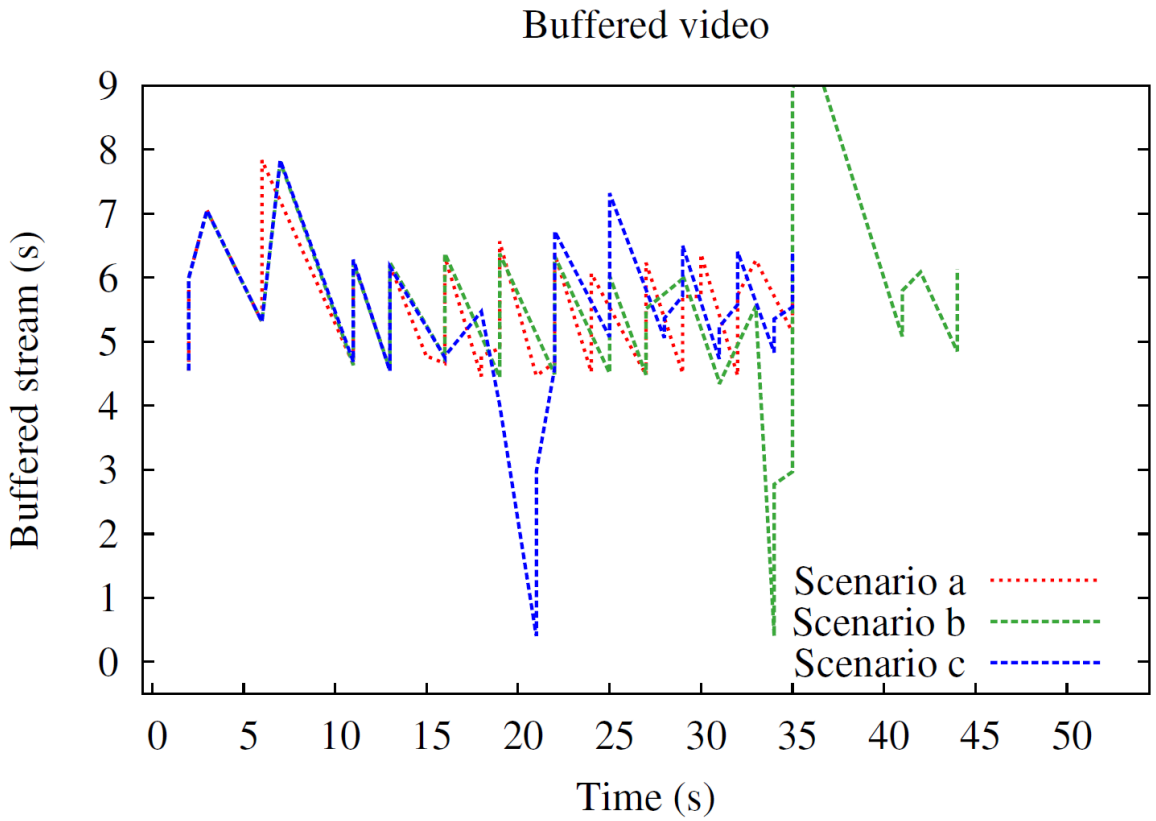

Fig. 6 Amount of time buffered at a point of time.

Finally, Figure 6 shows the media playback buffer time for each scenario. As with default OSMF behaviour, if the buffer dips below four seconds, the player would occasionally freeze up to re-buffer. Note that we use parallel downloading to mitigate such playback interruptions. From the graph it is possible to see that the player does not need long pauses during jumps, further validating the functionality of the player.

In summary, the results show that it is possible to conclude that the player behaves within the parameters to play a video normally. Also, that the player uses the internal cache to make sure the player can quickly continue to play the video after a jump.

7 Conclusions and discussions of future works

This thesis presents the design and implementation of a non-linear video structure in a prototype player based on Adobe's OSMF. The goal was to produce a proof-of-concept player. The first two chapters describe the current state of the art and presents OSMF. Because OSMF is open source it was a good candidate to implement the modifications needed for the new player.

Chapter 3 goes into detail about the structure design and mentions high level design of the OSMF and SMP applications. The decision to implement the structure as a depth-first tree made the structure very lightweight compared to other solutions while also being relatively simple to implement. Using the `HttpDownloadManager` as a communication hub made

communication between classes and projects easier and required less work than implementing a more modular approach.

Chapter 4 describes the modifications needed to implement the handling of a non-linear video file. The majority of modifications on existing classes were to allow communication between them and the `HttpDownloadManager`.

Finally chapter 5 and 6 describes the test methodology and presents the results of the proof-of-concept validation testing. The tests were done on a local set-up which made testing of how the player handled different bandwidth conditions outside of this thesis' time limits. The validation tests showed that the player could use pre-fetched fragments to handle jumping in the video with very small amounts of stuttering and apart from this, the player would not be affected at all.

This thesis makes three significant contributions:

- A lightweight structure for describing non-linear media. This structure is based on a tree structure where each node represents any number of fragments. The node then branches into one default path and one explicit jump path which skips forward in time in the media object.
- Modifications done to OSMF to support the new structure. OSMF and SMP were modified to support playing non-linear video. OSMF now also includes buffer management and parallel fragment downloading to increase the user experience when skipping forward in a video.
- A proof-of-concept player which can play non-linear video. The player was tested to prove that the design and modifications done were sound.

7.1 Future work

The following gives some examples of future work that was not solved in this thesis:

- Future work could include more extensive performance testing. For example test additional videos and a larger sample of different structures. Additionally, tests should be done under different bandwidth conditions such as spikes of different lengths.

- The architecture of OSMF is very solid and if more work is done on the non-linear part the new and modified classes should reflect the modularity of OSMF and be a self-contained part of the project.

- Even though the downloading policies perform the function they are created for, they are relatively basic. Further evaluation of different aspects, such as using the adaptive streaming functionality or ensuring the landing point is fully downloaded, is a potential field of research.

- The binary tree structure used in this thesis can be upgraded to a graph structure which can support multiple branches per node. This would allow the player to mix and match the clips to a much larger extent. By allowing this the video files' size

could be reduced as a clip could be reused for several different paths in the video, making it much more efficient.

- The user interface is at this point very basic and could become much more user-friendly. Additional work could include how to best present non-linear media to a user. The current user interface also does not allow for more than binary path choices.

- To allow finer granular control, future work could also use time instead of fragments to define branch points, or evaluate if this is a good solution.

Some of these extensions are currently being implemented and tested, as part of follow up work to this thesis.

8 References

1. Sandvine, "Sandvine Global Internet Phenomena Spotlight 2011 - North America", White Paper, 2011, http://tinyurl.com/SandvineNA2011.

2. Jeffrey Erman, Alexandre Gerber, K.K. Ramakrishnan, Subhabrata Sen, Oliver Spatscheck, "Over The Top Video: The Gorilla in Cellular Networks", in Proceedings: ACM Internet Measurement Conference (ICM), Berlin, Germany, Nov. 2011.

3. Cisco Systems, "Cisco Visual Networking Index: Forecast and Methodology, 2010-2015", White Paper, 2011.

4. Saamer Akhshabi, Ali C. Begen, Constantine Dovrolis, "An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP", in Proceedings: ACM Multimedia Systems (MMSys) San Jose, CA, Feb. 2011, pp. 157–168.

5. Ashwin Rao, Yeon-sup Lim, Chadi Barakat, Arnaud Legout, Don Towsley, Walid Dabbous, "Network Characteristics of Video Streaming Traffic", in Proceedings: ACM CoNEXT, Tokyo, Japan, Dec. 2011.

6. Niklas Carlsson, Anirban Mahanti, Zongpeng Li, Derek Eager, " Optimized Periodic Broadcast of Non-linear Media", IEEE Transactions on Multimedia, Vol. 10, No. 5 (Aug. 2008), pp. 871 – 884.

7. Yanping Zhao, Derek Eager, Mary Vernon, "Scalable on-demand streaming of nonlinear media", in Proceedings: IEEE INFOCOM '04, pp. 1522—1533, Hong Kong, China, Mar. 2004.

8. Matteo Varvello, Stefano Ferrari, Ernst Biersack, Christophe Diot, "Exploring Second Life", IEEE/ACM Transactions on Networking, Vol.19, No.1 (Feb. 2011), pp. 80—91.

Patrik Bergström

A Proof-of-concept Implementation of a Non Video Player for HTTP-based Adaptive Streaming

2012-02-18