2014

**An investigation into the Quality of Service and Quality of Experience of Microsoft Smooth Streaming**

**David Fisher**

Table of Contents

[Abstract: 2](#_Toc386714955)

[Chapter 1: Introduction: 3](#_Toc386714956)

[Chapter 2: Streaming Protocols 5](#_Toc386714957)

[2.1: Adaptive Bitrate Streaming 6](#_Toc386714958)

[2.2 MPEG-DASH 8](#_Toc386714959)

[2.3: RTP (Real Time Transport Protocol) 14](#_Toc386714960)

[2.4: RTCP: RTP Control Protocol 16](#_Toc386714961)

[2.5: RTSP (Real Time Streaming Protocol) 19](#_Toc386714962)

[Chapter 3: Streaming for Mobile Devices 22](#_Toc386714963)

[Chapter 4: Media and Cloud Platforms 24](#_Toc386714964)

[4.1: CDN Content Delivery Network 25](#_Toc386714965)

[Chapter 5: Commercial Streaming Protocols 26](#_Toc386714966)

[Apple 26](#_Toc386714967)

[Adobe 28](#_Toc386714968)

[Microsoft 30](#_Toc386714969)

[Chapter 6: Methodology 32](#_Toc386714970)

[Chapter 7: Design 33](#_Toc386714971)

[Chapter 8: Implementation 34](#_Toc386714972)

[Chapter 9: Analysis of results 37](#_Toc386714973)

[Chapter 10: Conclusion 49](#_Toc386714974)

[Appendix 51](#_Toc386714975)

[References 52](#_Toc386714976)

# Abstract:

Adaptive streaming has been adopted by the streaming industry, with a large number of applications implementing the technology. Adaptive streaming technology offers major quality of service and quality of experience advantages and benefits to end users unlike previous streaming technologies. Streaming in a low bandwidth environment is as seamless as streaming in a high bandwidth with adaptive streaming technologies.

Throughout this paper both emerging and previous streaming technologies that are phasing out are researched and documented giving the reader an insight into the streaming industry. The older streaming technologies include RTP, RTCP and RTSP. The newer emerging streaming technologies researched in this paper are MPEG-DASH, Microsoft Smooth Streaming, Apple HTTP Live Streaming and Adobes HTTP Dynamic Streaming.

In particular the main focus and implementation in this paper is the Microsoft Smooth Streaming protocol. An investigation into the quality of experience and quality of service is carried out by creating both a smooth streaming client and server. Various use case scenarios are carried out with the stream session data recorded and analysed to find out how the Microsoft smooth streaming protocol delivers both quality of service and quality of experience to the end user. Questions answered in this paper is how does the smooth streaming protocol behave in a streaming session, how does smooth streaming provide the end user with quality of service and what impact does a varied bandwidth network environment have on the quality of service of the stream.

# Chapter 1: Introduction:

Streaming is a continuous stream of data, allowing media content such as video and music to be viewed and listened to in real time. Media content is compressed and sent over the Internet in segments which are then played back by the media client. This provides a faster alternative way of viewing than waiting for the content to be fully downloaded. Whilst streaming media content, the computer temporarily stores data into a buffer which is a section of physical memory storage used to store data temporarily.

Media streaming service providers such as Netflix make use of content delivery networks to distribute content to end users. A content delivery network is a large set of servers or computers that provide web content to numerous end devices. These servers are located in different locations, so end users request data from the server closest to them. As the demand for multimedia content to be accessible at anytime and anywhere has dramatically increased, streaming technologies have been developed and enhanced to meet these requirements. The main goal of streaming is to provide the user with a seamless viewing experience, due to varying network conditions and environments this has become an issue, with media content stopping and stalling in low network speed areas. To prevent this from occurring, adaptive streaming technologies have been developed.

Today’s steaming technologies use Adaptive Streaming which is designed and based around HTTP. Adaptive Streaming makes use of the client to monitor the end users bandwidth, CPU capacity and other conditions, which in turn requests either a high or low bitrate quality from the server.

Other streaming protocols such as RTP (Real-Time Transport Protocol) and RTSP (Real-Time Streaming Protocol) are still used, but are being phased out in response of the adaptive streaming technology. The industry standard for adaptive streaming is MPEG-DASH, Dynamic Adaptive Streaming over HTTP. A group of over fifty companies and organisations collaborated together to create a standard for adaptive streaming. This new standard is a technology related to Apple’s, Microsoft’s and Adobes proprietary protocols, which are:

* HDS HTTP Dynamic Streaming (Adobe)
* HLS HTTP Live Streaming (Apple)
* MSS Microsoft Smooth Streaming

During the course of this research project, various media streaming protocols will be researched and examined in detail. These protocols will range from previous to today’s streaming technologies to previous used streaming technologies, giving an insight into the media streaming industry. From researching such technologies and protocols, the long term achievement is to carry out a series of tests which will evaluate and analyse the quality of service and experience of an adaptive streaming technology, in particular the Microsoft Smooth Streaming protocol. From the tests and research carried out, an understanding of the behaviour of the technology will be analysed and documented. Throughout the series of tests carried out, the same video file will be used; this video file will be encoded at different bitrates with varying quality which will enable the client player to seamlessly switch between each quality due to the network the conditions. From the user’s perspective, the overall product being developed will be a Smooth Streaming media player which enhances viewing experience, proving a smooth streaming experience to end users.

The research questions answered in the analysis chapter of this document is how the Microsoft smooth streaming protocol provides quality of service and quality of experience for the end user. This is answered by implementing a Microsoft smooth streaming client and server and simulating various use case scenarios whilst streaming a video.

From running the various streaming scenarios and analysing the data gathered from these streaming sessions, multiple quality of service and experience factors have been discovered, answering the research questions for the Microsoft preparatory protocol, smooth streaming.

# Chapter 2: Streaming Protocols

In this chapter, various streaming protocols will be researched and documented in detail. These protocols are still in use today, however adaptive streaming protocols are in wide spread use and are the future of media streaming. Older protocols such as RTP and RTSP are phasing out with support becoming very limited. Although these protocols are of adequate use, the constant demand and increasing amount of users utilising online media services has caused scalability issues. Adaptive streaming protocols overcome this issue and improve the user viewing experience. Recently a new standard has been developed an adaptive technology called MPEG-DASG. This technology will also be researched and examined in detail further in this chapter.

## 2.1: Adaptive Bitrate Streaming

Adaptive Bitrate Streaming is a technology used in today’s streaming services. Services such as YouTube and Netflix and other commercial applications utilise this streaming technology to give further QOS (quality of service) and QOE (quality of experience) to the end user. This technology provides an enhanced video and audio viewing experience compared to previous streaming protocols such as RTP and RTSP. Adaptive bitrate streaming uses HTTP (Hypertext Transfer Protocol) which is allowed through firewalls by default, unlike RTP. Adaptive streaming is also supported by almost all servers, since only a web server is needed.

Adaptive bitrate streaming uses the client player to monitor numerous factors which will in turn determine a decision to stream either higher or lower bitrate media content, providing the user with a smooth viewing experience, with minimal buffering and loading time. The main factors which are monitored are:

* Network Bandwidth:

Measured in bps (bits per second), network bandwidth refers to the capacity of the connection, the bigger the capacity the faster the speed.

* CPU utilisation:

CPU utilisation refers to the amount of CPU cycles being spent on a single process.

* Resolution:

Resolution refers to the amount of pixels displayed on a given monitor/screen. This is measured horizontal and vertical axes, for example full HD being 1920 by 1080.



Figure 1 shows the different bitrate video files (Laukens, 2011)

Other factors also have an impact on the quality of media to be streamed; one such factor could be whether the user is viewing a small thumbnail of the video (i.e. not full screen).

Depending on the factors outlined in above, the client chooses the appropriate bitrate quality to stream accordingly. At the start of a stream, usually the lowest available bit rate quality is selected to enable a faster start up time, during the playback of the first media segment; the client then monitors the various factors and selects a bitrate accordingly. The client does the majority of the work, where the server passes down the description of the media object. The format and syntax of the description file is in xml, however this may vary depending on the protocol used. The client retrieves individual segments by examining the description file and issuing HTTP GET requests to the server. These description files contain lists of URL links which point to individual segments of media content at various time intervals and bitrate quality. This results in reduced buffering time which is evident in YouTube’s service, which having implemented this technology saw a 20% reduction in buffering time (Roettgers, 2013) and a faster start time.

## 2.2 MPEG-DASH

MPEG-DASH is a technology developed under MPEG, becoming an international standard in November 2011. Companies involved in the standardisation of this protocol are Apple, Netflix, Samsung and many others. (DASH, dashif.org/about-dash/industry-forum) The standardisation of such a protocol provides universal deployment of streaming media services.

Dynamic Adaptive Streaming over HTTP or DASH for short is an industry standard protocol for streaming high quality multimedia content over the internet. The media content is broken down, segmented and encoded at varying bitrates. As content is played through the client, the client can select an appropriate bitrate to stream depending on the network conditions of the end user; this enables the end user to view the media content without any buffering, jitter or stalls.

Buffering is the time taken for a video to load enough data into the buffer to be able to play the video content. Jitter and stalls are pauses and stops during the video playback, these can occur due to low bandwidth, or when the rate of the video is greater than download speed.

Before the client can start to stream any content, the client must learn the characteristics of the media, such as the profile, duration, bandwidth and more. A profile is a set of restrictions that are typically features of the Media Presentation Description (MPD) which is described in detail further in this chapter. The duration refers to the length of a specific media segment and bandwidth is the amount of data that can be transferred from one point to another in a given time.

The client learns about this information by parsing the MPD (Media Presentation Description) which is an XML document that contains metadata which is required by the DASH client to build HTTP-URLs to gain access to segments to provide the streaming of content to the end user. Each segment is described by an MPD, which looks like the following:



Figure 2. The Media Presentation Description (Microsoft, DASH Content Protection using Microsoft PlayReady, 2013)

Profiles:

The above identifier specifies the profile used in this media content segment. Profiles are described in more detail later in the chapter. The profile identifier is mandatory and should be used in each MPD.

Type:

This attribute specifies whether the MPD may be updated or not. Being static defines that the MPD may not be updated where dynamic defines that the MPD can be updated. The attribute may be set to dynamic during a live event, for example a live news report. The default value is set to static.

MinBufferTime:

MinBufferTime refers to a common duration used in the representation data rate (see bandwidth attribute).

Duration:

The duration attribute specifies the duration of the period to define the start time of the next period.

More tags and attributes exist within an MPD but not all of them are present as different tags apply to different profiles such as the on-demand and live profile. These tags and attributes include:

* minBandwidth
* maxBandwidth
* minWidth
* maxWidth
* minHeight
* minFrameRate
* maxFrameRate

In MPEG-DASH, all of the work falls on the client player, where control lies exclusively with the client, having the responsibility of requesting certain bitrate media segments depending on network bandwidth. The MPEG-DASH protocol only describes the Management Presentation Description and segment formats. The encoding format of the various segments, the delivery of the MPD and the client behaviour are not in DASH’s scope.

Media Presentation Description model architecture:

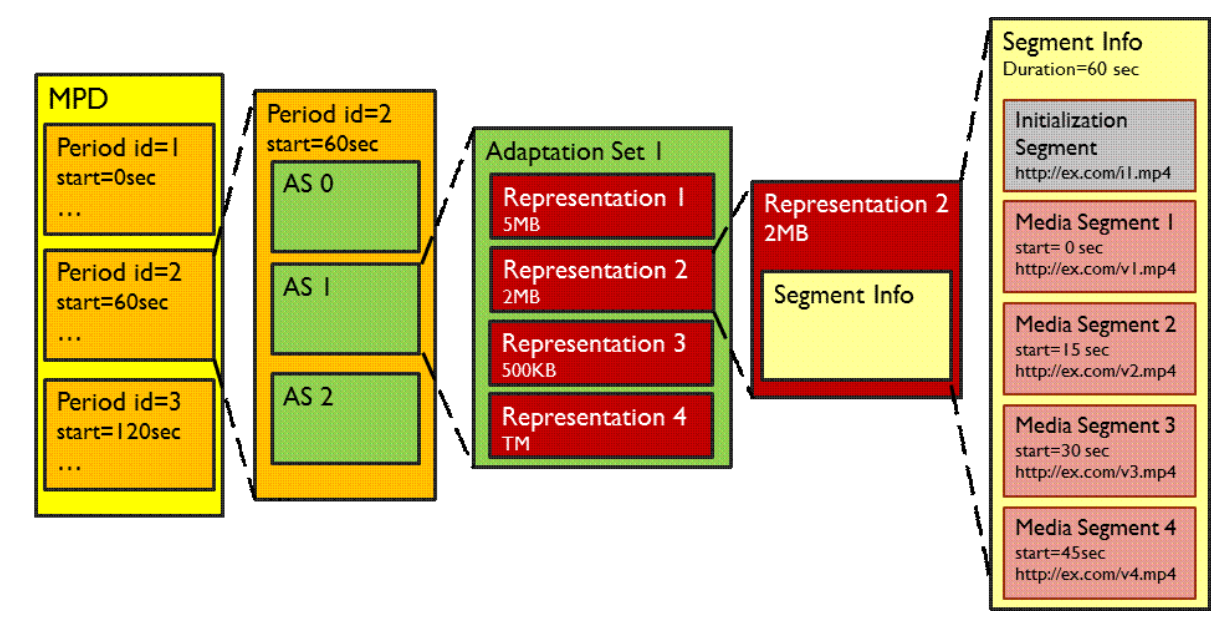


Figure 3 shows the MPD Hierarchical model. (Sodagar, 2011)

The MPD consists of multiple periods. Each period has both a starting and duration time consisting of one or multiple adaption sets. An adaption set provides information on media components and its varying encoded bitrates. Each adaption set normally includes multiple representations, where a representation is an encoded alternative of the same media content, varying from bitrate, resolution and other characteristics. Each representation is made of one or numerous segments. A segment is a sequenced chunk or segmented part of the media content. The first segment may be an initialisation segment which contains the required information for the start of client media decoder. Each segment has an addressable URI on a server for which the client can access and retrieve. (Sodagar, 2011)

As DASH uses HTTP as its underlying protocol, scalability and broadcast to millions of users is not an issue, unlike its predecessor RTP, where streaming requires the server to maintain an individual stream session for each client. As DASH is HTTP based most configured firewalls support the outgoing HTTP connections which make DASH easily accessible. Also edge servers delivering the media content are able to cache the content, making DASH scalable.

DASH Profiles:

There are many profiles outlined in DASH that impose a set of restrictions, usually restricting features of the MPD, segment formats or content delivered within segments. Profiles are defined to enable interoperability for various deployment cases by limiting the amount of features for each profile. Each profile has an identifier, a URN. As seen in the above example (in figure 1) the profile is indicated within the MPD. Profiles which are followed by the ‘profiles’ attribute are URNs which conform to the URN standard. Externally defined profiles can be either a URN or URL, these external profiles are called interoperability points.

DASH defines six profiles:

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Reference** | **Abstract** |
| urn:mpeg:dash:profile:full:2011 | ISO/IEC 23009-1 [[1](http://dashif.org/identifiers/references#23009-1)], section8.2 | identifier for MPEG-DASH Full profile. |
| urn:mpeg:dash:profile:isoff-on-demand:2011 | ISO/IEC 23009-1 [[1](http://dashif.org/identifiers/references#23009-1)], section 8.3 | identifier for MPEG-DASH ISO Base media file format On Demand profile. |
| urn:mpeg:dash:profile:isoff-live:2011 | ISO/IEC 23009-1 [[1](http://dashif.org/identifiers/references#23009-1)], section 8.4 | identifier for MPEG-DASH ISO Base media file format live profile. |
| urn:mpeg:dash:profile:isoff-main:2011 | ISO/IEC 23009-1 [[1](http://dashif.org/identifiers/references#23009-1)], section 8.5 | identifier for MPEG-DASH ISO Base media file format main profile. |
| urn:mpeg:dash:profile:mp2t-main:2011 | OISO/IEC 23009-1 [[1](http://dashif.org/identifiers/references#23009-1)], section 8.6 | identifier for MPEG-DASH MPEG-2 TS main profile. |
| urn:mpeg:dash:profile:mp2t-simple:2011 | ISO/IEC 23009-1 [[1](http://dashif.org/identifiers/references#23009-1)], section 8.7 | identifier for MPEG-DASH MPEG-2 TS simple profile. |

(DASH, dashif.org/identifiers/profiles/)

Three of the main widely used profiles today are described below:

Full Profile:

This profile contains all features and segment types. An example of the full profile is, streaming any media content; weather an audio or video file. All tags and attributes are included within the given MPD.

On Demand Profile:

This profile provides support for on demand content. The limitations forced by this profile are that each representation is provided as a single segment and that sub-segments are aligned across representations within an adaption set. This profile enables efficient use of HTTP servers. Tags and attributes must be included and set to a correct value. (MPEG, 2012)

Live Profile:

This profile is optimized for live encoding. Minimal latency will occur during the encoding of the live media, on top of the delivery of segments. Each segment of media requested is made available using a generated URL, sending an MPD update before each segment request is not necessary in this circumstance. The constraints issued by this profile are that segments are constrained so that accessing representations at segment boundaries is enabled, thus seamless switching within an adaption set cab be performed. The ‘type’ attribute in the MPD can also be set to static to distribute non-live content, for instance a media presentation has finished but kept available as an on-demand service. (MPEG, 2012)

Introducing the MPEG-DASH standard prevents the issue of providing support for all proprietary protocols (building a cross platform media player). Such as HLS (HTTP Live Streaming) Apple, Microsoft Smooth Streaming and Adobe HTTP Dynamic Streaming. Each of these proprietary protocols is very similar to one another, all using HTTP to receive content, although the way that this is done is very dissimilar. Writing code to handle each proprietary protocol is a tedious and expensive task.

One of the main disadvantages of DASH is that there is currently no supported DASH web client. Another issue, that Microsoft has stated that once the MPEG-DASH standard has been finalised, it will support the standard, however both Adobe and Apple have not said the same. Until Adobe and Apple provide support for DASH, DASH may be not be used as much as it was hoped for. One reason for the above is that DASH is codec agnostic meaning that DASH only supports the H.264 and WebM, with neither codec being supported in most HTML5 browsers. A codec is a piece of software used to encode and decode a data stream.

Licensing fees also arise as a major issue, depending on how you use H.264 you are required to pay a license fee to MPEGLA representing various patent holders. As of yet, it is unsure that the DASH standard is to be royalty free. Mozilla Firefox believes that the internet is a global public resource that must remain open and accessible. (Mozilla) Firefox has become a very popular web browser and without the implementation of DASH in its browser, DASH will lose traction.

As well as the features mentioned above, DASH has additional features such as:

* Ad-insertion
* Compact manifest
* Fragmented manifest
* Segments with variable durations
* Multiple base URLs
* Clock-drift control for live sessions
* Scalable video coding and multi-view video coding
* Flexible set of descriptors
* Sub setting adaption sets into groups
* Quality metrics for reporting the session experience (Sodagar, 2011)

Use Case:

The purpose of encoding the same piece of content at different bitrates is that the client player listens and watches network bandwidth. When there is a change in network bandwidth and speed, the client requests either a higher or lower segment. For example:

A video is encoded at three different bitrates, 5mbps, 2mbps and 500kbps. The accompanying audio is also encoded at various bitrates, AAC 128kbps and AAC 48kbps. A device starts to stream the content by requesting segments of the video content at 2mbps and the audio content at 128kbps. After monitoring the first segment of media content, the client realises that the available bandwidth available is lower than 2mbps, so at the next available point, the client requests the video segment at a low bitrate, 500kbps, but leaves the audio bitrate at 128kbps.

Summary:

From research the above protocol, it is clear that MPEG-DASH and dynamic adaptive streaming over HTTP is the most beneficial and satisfying technology to use when streaming media content over unmanaged networks (the internet) to both end users and media streaming providers.

## 2.3: RTP (Real Time Transport Protocol)

The Real Time Transport Protocol was developed by the IETF (Internet Engineering Task Force) and defines two protocols, RTP and RTCP (Real-Time Transport Control Protocol). RTP is an end-to-end transport protocol which is used to deliver multimedia data such as video and audio. RTCP provides a control steam which sends control information associated with a specific data flow. (Larry Peterson, 2007) RTP supports a wide variety of multimedia applications such as on-demand media and internet telephony, however as RTP runs on top of UDP, the delivery of packets cannot be guaranteed and the sequence of packets are not guaranteed to be kept in order. The benefit of RTP running on top of UDP is to make use of UDPs checksum and multiplexing services. However RTP is not bound to UDP and has the capability to run on other underlying protocols such as TCP. Running on top of TCP would generate a mass amount of bandwidth introducing latency and ‘lag’ due to packet retransmission and TCP error checks. TCP packet retransmission proves to be of no use regarding the transmission of video data, the time taken for the lost packet to be retransmitted and received by the end user will be too high making the frame arrive out of time and become useless.

RTP delivers packets in sequence enabling the receiver to reconstruct the packets in the correct sequence. Sequencing can also be used to define the location of a packet. (Javvin, 2005) RTP also provides support for data transfer to multiple destinations using multicast distribution; however this is only supported if the underlying protocol supports it, e.g. UDP. (Javvin, 2005)

RTP defines profiles and formats. The profile provides information that guarantees the understanding of the fields within an RTP header. The format determines how the data that follows the RTP header is handled or interpreted.

Header Format:

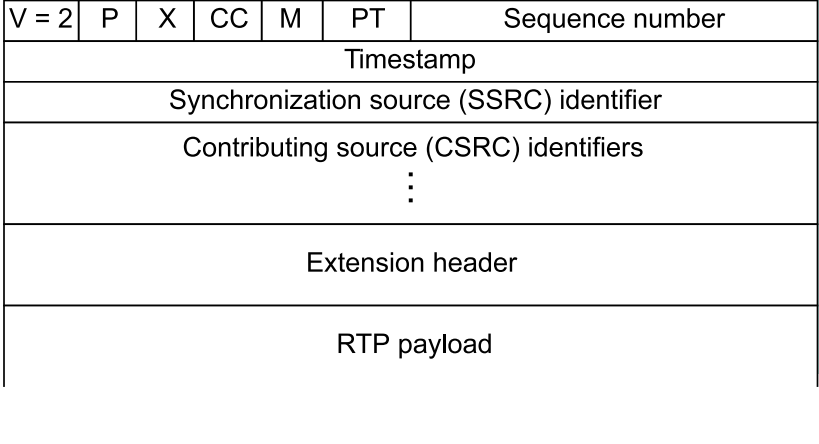


Figure 4 shows the header format of an RTP packet (Larry Peterson, 2007)

* V:

The first two bits of the RTP header are a version identifier. This identifies the RTP version.

* P:

Padding may be required to fill a block to a specific size, which is required by an encryption algorithm.

* The Extension bit (X):

This bit indicates the existence of an extension header. The fixed header will be set by an extension header if the extension bit is set.

* CC:

CC contains the amount of CSRC identifiers that follow the fixed header. (Arjan Durresi, 2005)

* M:

This indicates the marker. The marker is defined by a profile and is used to enable significant events, such as frame boundaries to be marked in the packet stream. (Arjan Durresi, 2005)

* PT:

Payload Type identifies the format of the RTP payload and determines its interpretation by the application. (Arjan Durresi, 2005)

* Sequence Number:

The sequence number is used to detect lost packets. This is done by incrementing by one each time an RTP data packet is sent. The initial value is set randomly to avoid attacks. (Arjan Durresi, 2005)

* Timestamp:

This value is a number that represents the time at which the first sample was generated. The time stamp is a result from a clock which allows synchronisation and jitter calculations. (Larry Peterson, 2007)

* SSRC:

The Synchronisation Source is chosen at random so that each SSRC has a different value throughout an RTP session. The SSRC field identifies synchronisation sources within the same RTP session, indicating the source of the data or if the data is combined. (Arjan Durresi, 2005)

* CSRC:

Contributing Source Identifiers identifies the contributing sources for the payload. CSRC is used when a number of RTP streams pass through a ‘mixer’. (Larry Peterson, 2007) A use case of a mixer is when, users in one area are connected to a low speed connection, whereas the majority of users are using a high speed connection for a conference. Instead of forcing every user into a low bandwidth and low quality, an RTP relay called a mixer is used near the low connection area. The mixer resynchronises incoming packets to reconstruct the constant spacing generated by the sender, mixes the reconstructed streams into one stream, translates the encoding to suite low bandwidth connections and forwards the low bandwidth suited packet stream across the low bandwidth link. (V. Jacobson, 2003)

## 2.4: RTCP: RTP Control Protocol

RTCP is a companion protocol of RTP which provides RTP session participants feedback control information concerning the quality of the transmission related to the RTP data being sent, as well as a method of synchronising different media streams that come from the same sender. RTCP can also provide the identity of a particular RTP participant. (Javvin, 2005) The underlying protocol must support multiplexing of data and control packets. This is usually implemented using UDP, both RTP and RTCP run on consecutive port numbers. Multiplexing is when multiple signals and streams are combined and sent over one signal transmission, this prevents interference occurring among the various signals. (FAS) In a given RTP session, RTP participants send RTCP packets to every member of the same RTP session using IP multicast. (Arjan Durresi, 2005) These RTCP packets contain statistics describing the quality of the RTP data stream. Including the amount of lost packets and the amount of packets sent. From reading and processing these statistics adaptive applications can determine to receive low or high quality data. (Arjan Durresi, 2005)

As mentioned above, RTCP can provide feedback information and identification of the participants in an RTP data stream. RTCP defines several packet types which are as follows:

* SR: Sender Reports:

These are generated by active senders. This enables active senders of an RTP session to report transmission and reception statistics. The SR report also contains information on inter-media synchronisation, cumulative packet counters and the number of bytes sent.

* RR: Receiver Reports:

Receiver Reports are generated by inactive senders, containing information on reception feedback quality regarding data delivery, the amount of packets lost, inter-arrival jitter and timestamps used to calculate the round trip delay between the sender and receiver.

* SDES: Source Description Items:

Source Description Items contain information which describes the sources. RTP data sources are identified by a thirty two bit randomly generated number. As these identification numbers are not user friendly, source description packets contain textual information which is called ‘canonical names’ which are unique to each participant. Canonical names may include a telephone number, username or email address.

* Bye:

This packet indicates the end of participation.

(Arjan Durresi, 2005) Both sender reports and receiver reports exchange information, enabling a TCP like flow mechanism using adaptive encodings. As more participants join, more RTCP messages are exchanged, when there is a large amount of participants, the fraction of control traffic is kept limited to adapt and suite scalability making up five per cent of total data traffic. (Tommi)

From reading and processing the reports outlined above, calculations and estimations can be made to better improve video/audio playback. Such estimations and calculations are, round trip delay and packet loss. From analysing receiver reports, a calculation can be used to determine the round trip delay between sender and receiver. As the receiver report contains the LSR (the time stamp of the last sender report received) and the DLSR (Delay since the last sender report received), the sender can calculate the round trip delay using the following formula. Where A is equal to the time instant when the receiver report was received by the sender.

D = A – LSR – DLSR

(Tommi)

As mentioned briefly above, weaknesses occur in the scalability of RTCP algorithms. These weaknesses involve:

* Congestion due to floods of RTCP packets.
* Delays between receipts of RTCP packets from a single user.
* Large size of group membership tables.

Timer consideration:

The current algorithm set to scale the transmission interval of RTCP reports is linearly proportional to the estimation of the group size. As the participant size increases, sender and receiver reports are sent less frequently, this is best suited to cope with a few hundred participants, but as the number of participants increase into the thousands congestion occurs and the sending of reports is not handled efficiently enough to cope with the increasing number participants. For example in large cable TV networks cable TV users’ change channels at the beginning and end of television shows at the same time, this is known as the step-join phenomenon. This causes congestion which worsens the situation as disappeared packets keep the group size inaccurate, exceeding the 5% target of total data traffic. (Tommi)

Restricting the amount of packets sent in step-join environments will help to improve congestion at peak times as the group size and the amount of participants rapidly increases.

As participation increases (as explained under timer consideration), the requirement to keep track of each member within an RTP session becomes an issue. An SSRC table with over a million active participants demands a high amount of storage capacity, this is inappropriate for devices with a limited amount of storage. To overcome this issue a sampling method of group participants is implemented, reducing the amount of storage space needed. (Tommi)

Summary:

From reading and researching both RTP and RTPs partner protocol RTCP, RTP has been a significant protocol in the use of multimedia applications. Since the launch of RTP back in 1996 (S. Casner, 1996) RTP has proved successful in delivering and handling media content to participants within an RTP session. Although the control protocol is powerful and extremely important in relation to the quality of an RTP session, issues and disadvantage appear. As media applications are becoming increasingly popular and used more frequently congestion and lag occur as RTCP reports are sent. The contents of these RTCP packets can make an impact on reducing the amount of congestion (by changing the quality of media), the issue of congestion still occurs as RTCP reports are sent to all users within the same RTP session. RTP also requires the server to manage a separate streaming session for each participant, making large scale applications resource intensive. MPEG-DASH and other HTTP based streaming technologies overcome this major issue by offloading the work onto the client player. Although algorithms and solutions have been implemented to help combat these RTP issues, a vast amount of bandwidth is still generated.

As the internet and its services are at a constant speed of change, CDN’s (Content Delivery Networks) are being used to deliver multimedia application, many of which do not support RTP as they are just HTTP web servers. Another important issue to be raised is that RTP and RTCP packets are often blocked by firewalls, preventing the service to be used.

## 2.5: RTSP (Real Time Streaming Protocol)

Real Time Streaming Protocol or RTSP for short is a multimedia streaming protocol developed and designed in April 1998 jointly by Real Networks, Netscape Communications and Columbia University. (H. Schulzrinne, 1998) The protocol establishes and controls single or several continuous streams of media data between servers and clients over the internet. (Arjan Durresi, 2005) The media data can have real time properties such as audio and video and be either in real-time or on demand.

RTSP does not deliver the media content but provides a framework to control the media data using TCP (Networks, docs.real.com/docs/proxykit/rtspd.pdf, 1998), acting as a network remote control for multimedia servers. (Javvin, 2005) UDP and multiplexing UDP are also supported however TCP is mainly used. As well as supporting and using the above transport layer protocols, RTSP is designed to work with lower level protocols such as the RTP application layer (MS\_RTSP, 2014) protocol which is used for the delivery of the media data. Feedback and quality statistics are also provided by the use of RTCP. (MS\_RTSP, 2014)

RTSP is very much similar to HTTP in which it has similar syntax; however the main difference here is that HTTP is a stateless protocol whereas RTSP has a state. Like RTP, RTSP has an identifier to keep track of participant sessions. Another difference between the two protocols is that both an RTSP server and client can issue requests. (Javvin, 2005) As RTSP is a client server protocol, there is no concept of an RTSP connection, the server maintains a session which is labelled by an identifier. During an RTSP session the client may open or close various reliable transport connections to the server to be able to issue RTSP requests. A request message sent from a client to a server includes the method to be applied to the resource, an identifier of the resource and the protocol version in use. (H. Schulzrinne, 1998) Requests can also be used to control and perform actions upon a particular session, affecting the state of a session. A session state is needed to be able to associate RTSP requests within a particular stream, as RTSP uses a separate protocol to that of the data transfer protocol. (H. Schulzrinne, 1998) Requests from clients and responses from servers are exchanged using RTSP methods. (MS\_RTSP, 2014) Such methods that affect the session state are, setup, teardown, play, record and pause.

* Setup:

This request stipulates the transport mechanism to be used for the media stream. The header of the setup request indicates the transport parameters acceptable to the client, the response will also contain the transport header which is selected by the server. The server also generates session identifiers in response to setup requests. Once the client receives a session identifier, the client then keeps the identifier until teardown has commenced. The client must provide the server with the identifier at each request. This identifies an RTSP session across transport connections, however there are other means of identifying a stream session without an identifier, one of which is the use of dynamically generated URL’s. (H. Schulzrinne, 1998)

* Play:

This method signifies the server to start transmitting the media data as specified in the SETUP method. Play requests can be queued and the server must queue play requests in order. The play request plays streamed data from the start of the range to the end of the range specified. (H. Schulzrinne, 1998)

Although a play request without a range header is acceptable, the media content will start playing from the beginning. A time parameter may also be included within a range header, which specifies a specific time in which playback should start; this time is represented in UTC (Current Local Time). (H. Schulzrinne, 1998)

* Pause:

The pause request temporarily stops the media stream delivery. As each RTSP header contains a URL, playback is paused for that particular stream. Server resources are kept, although some servers will close the session, freeing up the resources. This is done via a timeout parameter which is located in the setup message. (H. Schulzrinne, 1998)

* Teardown:

This request frees up resources on the RTSP server associated with a particular stream, removing the session and stopping the delivery of the media from the server for a given URI. (H. Schulzrinne, 1998) An example of a server sending a request to the client is when the server signifies the state of a session. Each request has a corresponding RTSP response that is sent in the opposite direction. (MS\_RTSP, 2014)

* Options:

An options request can be issued at any time, an example of such a request is, if the client is about to try a nonstandard request. Unlike the requests and methods above, the Options requests does not affect or influence the state of the server. (H. Schulzrinne, 1998)

* Announce:

This request serves two different purposes. When sent in the direction of the client to the server, the announce method describes the media object which is identified by the URL. When sent in the opposite direction, the announce updates the session description in real time. (H. Schulzrinne, 1998)

* Set parameter:

The SET\_PARAMETER method requests to set a value to a particular parameter for a media stream identified by the given URI. (H. Schulzrinne, 1998)

* Get parameter:

Similar to the SET\_PARAMETER, the GET\_PARAMETER retrieves the value of a parameter of a media presentation or stream identified by a given URI. (H. Schulzrinne, 1998)

* Redirect:

This request informs the client that the client needs to reconnect to a different server location. Before redirecting, the client must teardown the current connection and setup the new connection given by the URI which is indicated in the redirect header. A timestamp may also be existent in the request indicating to the client when to redirect. (H. Schulzrinne, 1998)

* Describe:

The Describe request is used to retrieve information about a specific media presentation identified by the request URL from the server. The server then sends a describe response to the client, containing all of the media initialisation information for the resource that it describes. (H. Schulzrinne, 1998)

RTSP describes the transmitted media content by using the SDP syntax (Session Description Protocol). Clients can receive this before a stream session has started, by sending RTSP requests such as the ones indicated above. (MS\_RTSP, 2014) Other formatting protocols include SAP (Session Announcement Protocol) and SIP (Session invitation Protocol). These protocols are used to assist the advertisement of multicast sessions. (Javvin, 2005)

# Chapter 3: Streaming for Mobile Devices

As mobile networks continue to evolve and grow over time, the demand of services becomes greater, such a service in great demand is the delivery of multimedia data such as video or audio. Wireless networks were originally designed for point to point communication, serving users individually, this is known as unicast. (JDSU) As unicast is inadequate for the purpose of media streaming, multicast broadcasting is used as an alternative, which has the capability of delivering data to many users simultaneously which is highly scalable (JDSU). The quality media content is also increasing at a fast pace, which is encouraging mobile operators to optimise media delivery over the cellular network. (JDSU) The cellular network is at a constant rate of change, with operators upgrading infrastructure and implementing new technologies to fulfil the requirements and demand of media streaming. Technologies have been put forward and implemented to solve the issue of a limited spectrum range, using the network capacity required to broadcast content to multiple users efficiently. Such technologies are described below.

* Multicast/Broadcast Mode Service (MBMS):

MBMS can be used in two different manners. The first being able to multicast media streams over a GSM network to multiple users, or the ability to broadcast media to users in a single geographical location. (JDSU)

* Digital Media Broadcasting (DMB):

DMB is the advancement of DAB (Digital Audio Broadcasting), DMB enables users to view TV content on their mobile devices. DMB also has the capability of carrying IP based data such as files and notifications as well as streaming media content. (JDSU)

* Media Flo:

Media Flo is an end to end system enabling the broadcast of video, audio and data casting which is the broadcast of data over a wide geographical area. Media Flo is designed to optimise coverage, capacity and power consumption. Media Flo can be implemented without the need to upgrade infrastructure, utilising existing broadcast infrastructure such as sites and antennas. (JDSU)

Mobile operators have been launching Mobile TV applications and services long before the development of 3G broadcast technologies. Instead mobile operators have been launching media services over existing 2.5G and 3G networks. (Frank Hartung, 2007) The demand for this type of service forced network operators to deploy such services over 3G unicast networks using packet switched streaming as the underlying technology. (Frank Hartung, 2007) PSS packet switched streaming provides an end to end streaming framework for network networks.

The advantage of 3G unicast is that network resources are only used when the end user is actively using the service and can optimise the transmission for each end user. However the main downfall here is the issue of scalability and how the network model will cope with a large amount of end users streaming content. Broadcasting solves this scalability issue, with the use of broadcast/multicast support to 3G networks. In 3GPP this support is called MBMS (Multimedia Broadcast and Multicast Service). (Frank Hartung, 2007) This technology brings small changes to the overall existing network reducing the cost in both upgrade of cellular network and terminals.

IMS (IP Multimedia Subsystem) is a next generation standard architecture which provides a service control platform permitting the creation of multimedia applications using wired and wireless capabilities. (JDSU) IMS merges both packet switched and packet services onto an IP platform. (JDSU) This helps to achieve the goal of IMS, providing all existent and future services that the internet provides. IMS has number of attributes, with the most important being:

* End to End Quality of Service
* Support for IP version 6

(JDSU)

Streaming media content on the move, such as on a train or in a vehicle is known as roaming, changing from network to network constantly on the move. In order to receive data the network must track the location of the end device. This is achieved by sending a series of messages over the cellular network. (JDSU)

# Chapter 4: Media and Cloud Platforms

Media and cloud platforms provide the means of delivering and storing media content such as audio and video to end users anytime anywhere. There are many platforms in existence, each with their own services and features. An example of such a platform and service is Microsoft’s media platform, Windows Azure. This cloud service has numerous platforms including media services, mobile services and storage solutions. As this research project focuses on the media streaming aspect, the Windows Azure Media Services will be explained in detail below.

The Microsoft media platform offers a vast array of features, with key functions such as media format conversion, content protection, live and on-demand adaptive streaming over HTTP, these services and functions can be easily integrated into the Windows Azure cloud platform under the Windows Media Services platform. (Microsoft, www.microsoft.com) Offering flexibility, scalability and reliability, handling rich media experiences for a global audience.

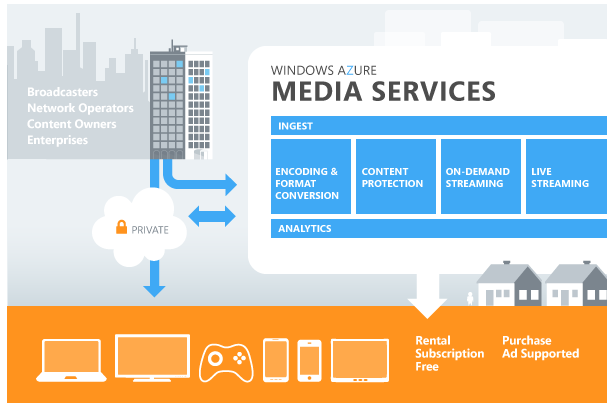
The Azure Media Services provides the ability to use the cloud to transcode videos. Azure also enables the user to transcode a video once and has the ability to deliver the video in multiple formats, with Microsoft’s smooth streaming supporting HLS (HTTP Live Streaming) and MPEG-DASH. As mentioned above, azure media services provides a service to encode and work with an array of standard codecs and formats including IIS Smooth Streaming, MP4 and conversion to Apple’s HTTP Live Streaming. (Azure W. , How to use Media Services (Java) - Azure feature guide, 2014) Azure media services also provide the ability to encrypt both live and on demand content, securing transport, storage and delivery. This is done by providing DRM (Digital Rights Management) technology. Most of all, azure media services enables the ability to stream content both on live and on demand scenarios. Supporting smooth streaming, http live streaming and other mp4 formats, delivery of such content is also provided by Windows Azure CDN or other third party CDN’s are supported. (Azure W. , How to use Media Services (Java) - Azure feature guide, 2014) 

Figure 5 Windows Azure Media Services Architecture (Azure W. , How to use Media Services (Java) - Azure feature guide, 2014)

Above is an image representing and displaying the current architecture of the azure media services platform. As well as supporting windows PC, azure media services also supports Apple Macs, Windows 8 applications, IOS devices, Android devices, Xbox’s, embedded or dedicated services smart TV’s and Blu-Ray players.

## 4.1: CDN Content Delivery Network

A CDN or Content Delivery Network are a set of servers located globally which distributes content to end users located near a server close by. A CDN helps improve the time it takes to load a piece of content, improves scalability and reduces the download time of the content by bringing the data close to the end users. These groups of servers bring down and store highly demanding content on them from the origin server which reduces download time and improves scalability by retrieving data from the server closest to them. (Ahmend Mansy, 2011)

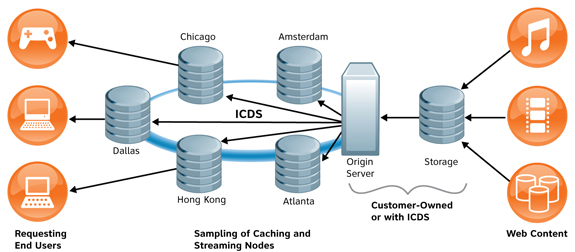


Figure 6 Content Delivery Network Architecture (Shawnee, 2009)

# Chapter 5: Commercial Streaming Protocols

## Apple

Apple uses HTTP Live Streaming (HLS), which is a proprietary adaptive bitrate streaming technology. HLS enables the delivery of audio and video over HTTP from a web server to IOS devices such as iPads, iPhones, Apple TV and Mac computers. HLS supports both live and on-demand media. (Apple, HTTP Live Streaming Overview: Introduction, 2014) One major downfall to Apple’s HLS is that only IOS and Apple devices are supported and devices running IOS 3.0 or later will have the client software built in to support this adaptive streaming technology. (Apple, HTTP Live Streaming Overview: Introduction, 2014)

Apple’s HLS consists of three components:

* The server component
* The distribution component
* The client software

The server component is responsible for taking input streams, digitally encoding them and encapsulating them ready for distribution. The input source can be either from a live or pre-recorded source, the media format is encoded into the mpeg-4 h.264 format and packaged into an MPEG-2 transport stream. The transport stream is then segmented and saved as a collection of files with a .ts extension. (Apple, developer.apple.com/library/ios/documentation.html) There are several components to a server, the media encoder, stream segmenter and the file segmenter. The media encoder takes real-time media and encodes and encapsulates it ready to be segmented at the segmentation component. The media is encoded at H.264 video and HE-AAC audio which is supported by the client devices. (Apple, developer.apple.com) The segmentation component then reads the transport stream and divides it into a set of small media files that are equal in duration, each segment being a separate file. The segmenter also creates an index file which contains references to the segmented set of files. These references contain the location and availability of the media files. Encryption can also take place at this process, encrypting each media segment and creating a key file. (Apple, developer.apple.com/library/ios/documentation.html)

The distribution component consists of web servers that are responsible for accepting client requests and delivering the encapsulated media prepared by the server component. With large scale demand, the use of CDN’s are utilised to deliver content to a vast amount of end users.

The client component starts off by retrieving the index file which is based on a URL identifying the stream. As mentioned above, the index file specifies the location of the media files along with decryption keys and other alternate streams that are available. The client then downloads each media segment in sequence. Once having a sufficient amount of data downloaded the client can then begin to reassemble the stream and playback the media. This is an on-going process until the end of the stream is reached. (Apple, HTTP Live Streaming Overview: HTTP Streaming Architecture, 2014)

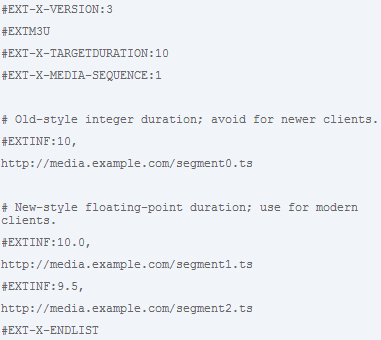


Figure 7 Apple's HLS Index File (Apple, HTTP Live Streaming Overview: HTTP Streaming Architecture, 2014)

Figure five shows a simplistic example of Apple’s index file. This file consists of tags which will be explained in the paragraph below.

The #EXTM3U tag is an extended M3U file distinguished from a basic M3U file. The #EXTNIF tag specifies the duration of the media segment, applying to the media segment that follows it which is followed by the given URL. The #EXT-X-TARGETDURATION specifies the maximum media segment duration. The #EXT-X-MEDIA-SEQUENCE tag specifies an individual media segment which is identified by a unique integer sequence number. (Apple, draft-pantos-http-live-streaming-12 - HTTP Live Streaming, 2013)

## Adobe

Adobe uses HTTP Dynamic Streaming (HDS), which is Adobes proprietary Adaptive Streaming technology, enabling high quality media streaming. HDS provides an open format solution which efficiently delivers media content to end users on the Flash platform.

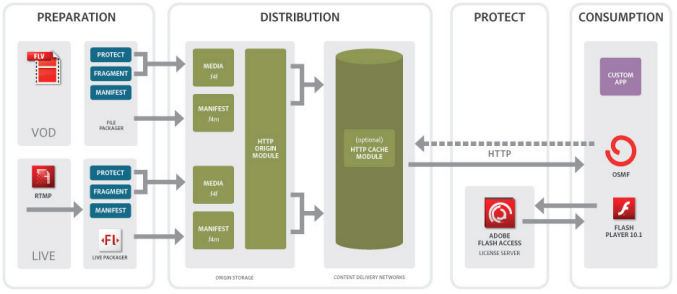


Figure 8 Illustrates the process involved to prepare, deliver, consumer and protect media content using HDS (Denis, 2011)

Adobe use tools called file packager and live packager to segment existing media content into a fragmented F4F format. Both tools generate and perform the same tasks, creating F4F fragment compliant files, to generate an XML based manifest file and to optionally encrypt and protect the media content. (Adobe, Adobe Flash Access Overview On Protocted Streaming, 2010) The file packager created F4F files taken from a previously encoded FLV or F4V file, also creating a manifest file (called F4M) in turn. This manifest file contains information describing the media segmented content, available bitrates and file locations as well as DRM (digital rights management) license server locations. (Adobe, Adobe Flash Access Overview On Protocted Streaming, 2010) The information provided allows the client to initiate and begin to playback the media. The manifest file is an XML document that provides the necessary information required to display and play the media content. The manifest file provides a description of the media content which contains Bitrate information, Bootstrap information, Moov atom and DRM authentication. The bootstrap information provides information required to begin play back. The Moov atom is an optional metadata for media representation. (Adobe, Adobe Flash Access Overview On Protocted Streaming, 2010) As HDS supports multiple bitrate delivery, each bitrate file segment is added to the manifest as a separate node. Files with multiple bitrates can share DRM and Bootstrap information, alternatively this information can be provided separately for each individual file. (Adobe, Adobe Flash Access Overview On Protocted Streaming, 2010) An example of a F4M manifest file is present below.

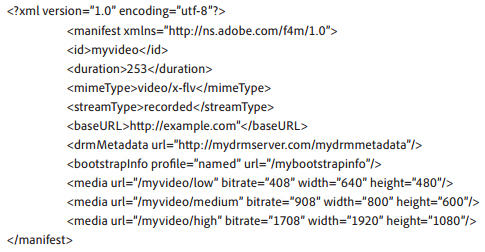


Figure 9 shows an F4M manifest file (Adobe, Adobe Flash Access Overview On Protocted Streaming, 2010)

In order to parse the manifest file, certain logic in the client player is required. Adobe states that the Open Source Media Framework (OSMF) has the necessary logic and can therefore be used to develop media players based on Adobe’s HDS. Below is a description of the operation of OSMF. (Adobe, HTTP Dynamic Streaming on the Adobe Flash Platform, 2010)

The first task falls upon the client, the client must a HTTP request to retrieve a given manifest file (F4M file). When the client receives the manifest file the client translates the desired time code to a segment number pair. (Adobe, HTTP Dynamic Streaming on the Adobe Flash Platform, 2010) Which then enables the client to construct the specific URL to request the specific fragment from the manifest file. This request is then sent to the origin module which then in turn delivers the requested file segment. (Adobe, HTTP Dynamic Streaming on the Adobe Flash Platform, 2010)

## Microsoft

Microsoft’s implementation of adaptive bitrate streaming is called Smooth Streaming. The smooth streaming protocol is used to deliver both on-demand and live media content from an IIS media server to a smooth steaming client. As Microsoft smooth streaming makes use of an MPEG-4 based data structure delivery over HTTP this allows seamless quality bitrate switching in almost real time, resulting in a constant playback experience for the end user. (Microsoft, [MS-SSTR]: Smooth Streaming Protocol, 2014)

As with all adaptive streaming technologies, Microsoft smooth streaming delivers media content in small fragments, these fragments are delivered every two seconds on average. A verification check is carried out on these fragments to see weather each requested fragment has arrived within the appropriate time and played back at the expected bitrate quality level. If these requirements are not met, then a lower bitrate quality is requested by the client, subsequently when conditions allow it a higher bitrate quality is requested. (Microsoft, Smooth Streaming : The Official Microsoft IIS Site, 2014)

Like all adaptive bitrate streaming protocols, the client first requests the manifest file belonging to the specific video start streaming. The Microsoft smooth streaming manifest file is an xml file which has an extension ISMC. The server then sends the client the requested manifest, which initiates and enables the client to request the first media fragment. Below is a communication sequence of the smooth streaming protocol.

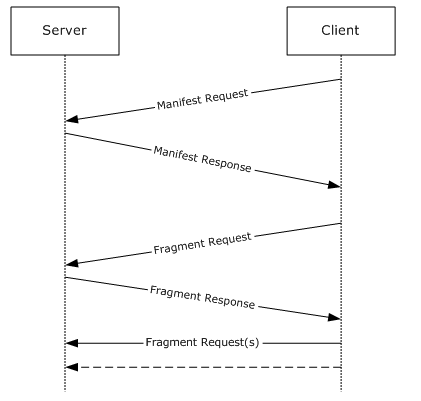


Figure 10 Microsoft Smooth Streaming Communication Sequence Diagram (Microsoft, [MS-SSTR]: Smooth Streaming Protocol, 2014)

The media content encoded at various bitrates levels are individual files relating to the bitrate quality level. Once the smooth streaming client requests a media segment the IIS server dynamically creates cacheable virtual fragments from the media files and delivers the best available bitrate quality to the client. The benefit of this is that the media content owner does not have to manage thousands of segmented media content, only whole files are managed.

Microsoft smooth streaming defines two parts, the wire format and the disk file format. The wire format defines the structure of the media chunks and the file format defines the structure of the contiguous file on disk, enabling better file management.

Smooth streaming media content consists of the various encoded media files, an ISM and ISM file. This is followed be either ISMV or ISMA files. The ISMV file contains both audio and video or only video whereas the ISMA files contains only audio, there is one ISMV file per encoded video bitrate. The ISM file is the server manifest file and the ISMC is the client manifest file. The server manifest file describes the relationships between the media tracks, bitrates and files on disk, the client manifest file describes the available streams to the client, the codecs used, the various encoded bitrates, the video resolutions and more. (Alex, 2009)

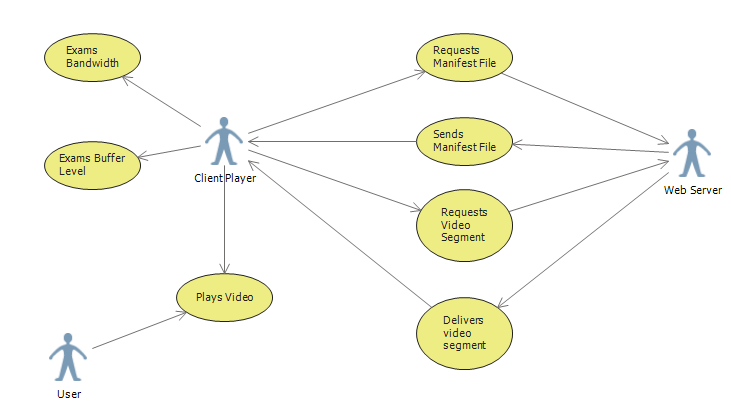
# Chapter 6: Methodology

Throughout the duration of this research paper, the main objective is to gain a better understanding of adaptive streaming technologies. For the implementation section of this research project an adaptive media player will be developed using Microsoft’s Smooth Streaming. The Media Player will be developed in Microsoft’s Visual Studio 2012 using C#. The implementation project will play a piece of video content using Microsoft’s proprietary Smooth Streaming Protocol. The video will be hosted and stored in the cloud using Windows Azure and the video content will be encoded at varying bitrates so that the video can be viewed smoothly, even if network /traffic bandwidth is high. Whilst the video is playing, the smooth streaming client will be monitoring any dropped frames and the speed of the internet connection. With this information the client player can switch bitrate quality accordingly. After and throughout the video is playing, the above information will be stored and displayed in a graph which will be analysed. The client player will use the same video through a series of network scenarios using a network bandwidth simulator. Again, the information generated will be stored and plotted into a graph which will be compared amongst all the other graphs generated.

# Chapter 7: Design

Use Case Scenario:

The user selects and plays a video via a Windows 8 store application media player. Unknowing to the user, the client player will monitor the current network environment and buffer levels. This will be performed by measuring the network bandwidth, buffer level and will change the quality of the video being streamed according to the results in real time.



# Chapter 8: Implementation

In this chapter a description of the experimental approach, the various tests performed and the data retrieved from these tests will be explained in detail. From researching various emerging adaptive streaming technologies, a media player consuming the Microsoft Smooth Streaming protocol has been developed which gives quality feedback data. This data will be analysed within the Analysis chapter of this document.

The developed Smooth Streaming media player streams smooth streaming content from an IIS (Internet Information Services) web server which is hosted locally. The video file streamed for experimental use is an open source animation called ‘Elephants Dream’ which was created by the Orange Open Movie Project studio in Amsterdam. (Dream, 2010) This video has been encoded for use with Microsoft Smooth Streaming using Microsoft’s Expression Encoder. The video file has been encoded at eight varying bitrates which are:

* 230000 bytes
* 331000 bytes
* 477000 bytes
* 688000 bytes
* 991000 bytes
* 1427000 bytes
* 2056000 bytes
* 2962000 bytes

The smooth streaming application has been developed in windows visual studio 2012 using both C# and XAML. XAML is short for Extensible Application Markup Language which is an XML based language developed by Microsoft (Microsoft, msdn.microsoft.com). The streaming client has been created as a standalone windows 8 store application.

When streaming a video from the Smooth Streaming media application each activity during the stream is recorded and outputted to a log textbox from which the data can be retrieved. Information such as a change in bitrate quality, chunk downloads and re-buffering of the video is outputted to the log textbox. Once retrieved, this data can be analysed to help give an understanding of the behaviour, QOE (Quality of Experience) and QOS (Quality of Service) of the smooth streaming protocol. Running this windows store application along with a packet sniffing application such as Wire Shark will provide enough data to analyse and learn about the adaptive smooth streaming protocol. Once the end of the video stream has been reached, the outputted data can be retrieved from the textbox and be placed into a text file. From there, a simple Java application has been written to retrieve specific information from the overall data gathered from the stream.

try {

Scanner txtscan = new Scanner(new File("File Path"));

while(txtscan.hasNextLine()){

String str = txtscan.nextLine();

if(str.contains("BitrateChanged")){

String time = str.substring(6, 22);

System.*out*.print(time + ",");

String bandwidth = str.substring(38);

System.*out*.print(bandwidth + "\n");

Figure 11 Java data extraction program (See appendix for full code)

To gain a better understanding of the Microsoft Smooth Streaming protocol a series of tests have been set to best simulate different user experiences whilst streaming a video. The numerous tests simulate high and varying bandwidth. Another experiment has been conducted to test a minimised sized down window and full screen viewing.

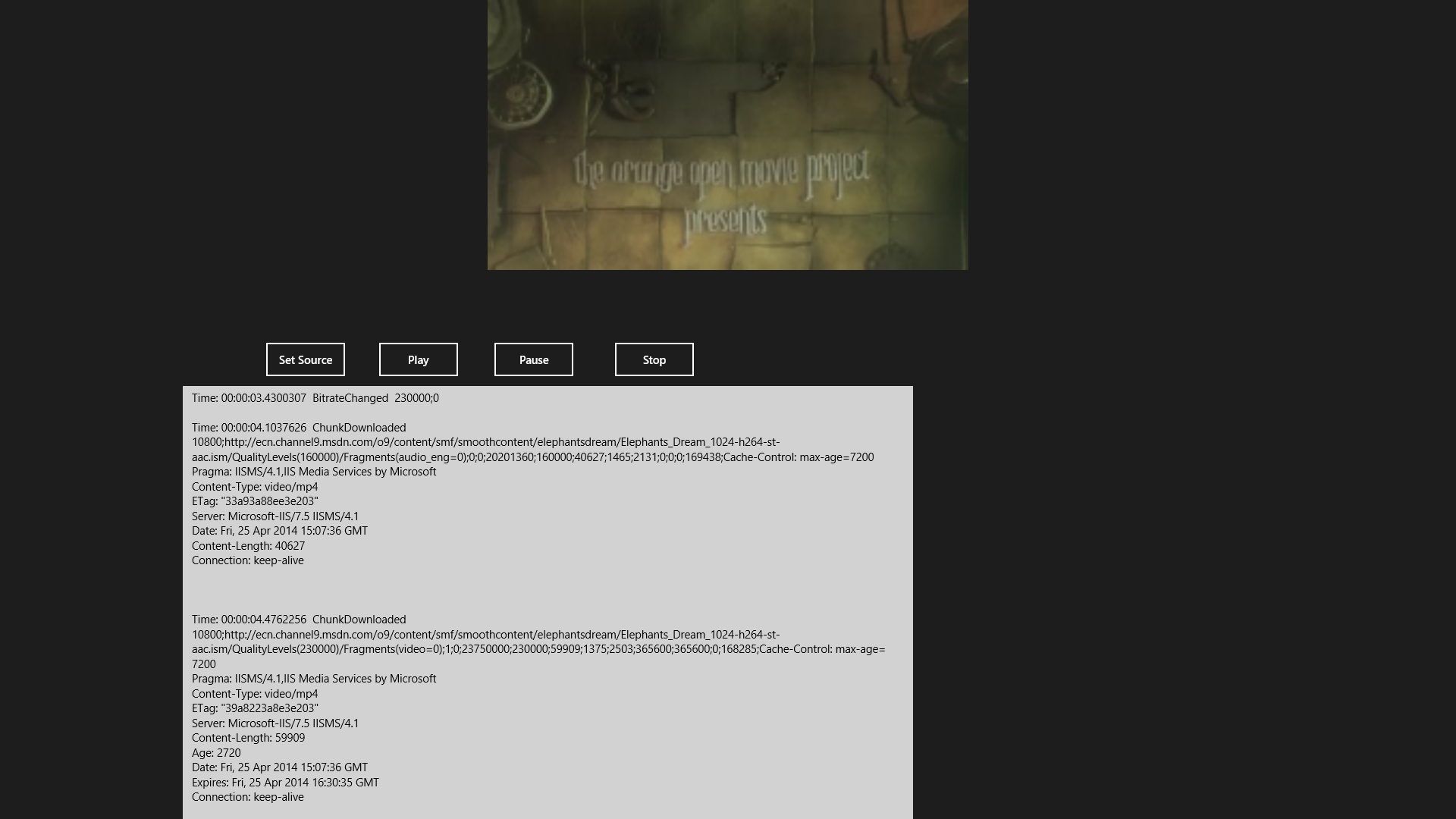


Figure 12 Smooth Streaming Media Player

The architecture of the media player is fairly simple, as seen in the architectural diagram below, the streaming application consists of a client and server. The server is an IIS server and the client is a windows store application.

Architectural diagram of smooth streaming project:



Figure 13 Architecture Diagram

This diagram illustrates the architecture of the smooth streaming project. The project consists of both a client and server (running IIS 7.0 with media services). Like any smooth streaming application, the client requests the manifest file and media chunks from the server. The server then accepts these requests and hands them over to the client to playback the media content.

As mentioned above, running this smooth streaming project requires both a client and server machine. In this test scenario the client machine must have the windows 8.0 operating system or above installed and the server machine must have IIS installed along with IIS media services 4.0. Once these requirements are met, the IP address of the server must be placed into the host file of the client machine along with a web address that the server IP address will be mapped to. This web address must then be placed within the client application. Inserting the IP address into the client machine along with a web address the IP address is mapped to is an efficient and inexpensive way of creating a domain name, as a computer first checks its host file before performing a Domain Name lookup elsewhere.

Whilst running the client program, Wire Shark will be running in the background at the same time. Wire Shark is an open source packet analyser which is used to capture and keep a trace of the various packets that are going both to and from the server during a current session.

# Chapter 9: Analysis of results

Throughout all of the experiments below the same video content was streamed, also the same hardware for the client and server was used. This gave an equal test on all of the experiments and recorded results. The data retrieved from wire shark and the stream event information were recorded and displayed in graphs below for representational purposes. These graphs include an Adaption Rate graph which is a measurement of bitrate quality change over a period of time (in seconds). Other graphs include a TCP time sequence graph and time sequence Stevens graph, these graphs are relevant as TCP (Transmission Control Protocol) is used to deliver the various video and audio segments of the video to the client.

In this section of the analysis chapter a video stream session from client to server will be analysed using the information obtained from the client (below in figure 14) and from the wire shark trace file relating to the stream.

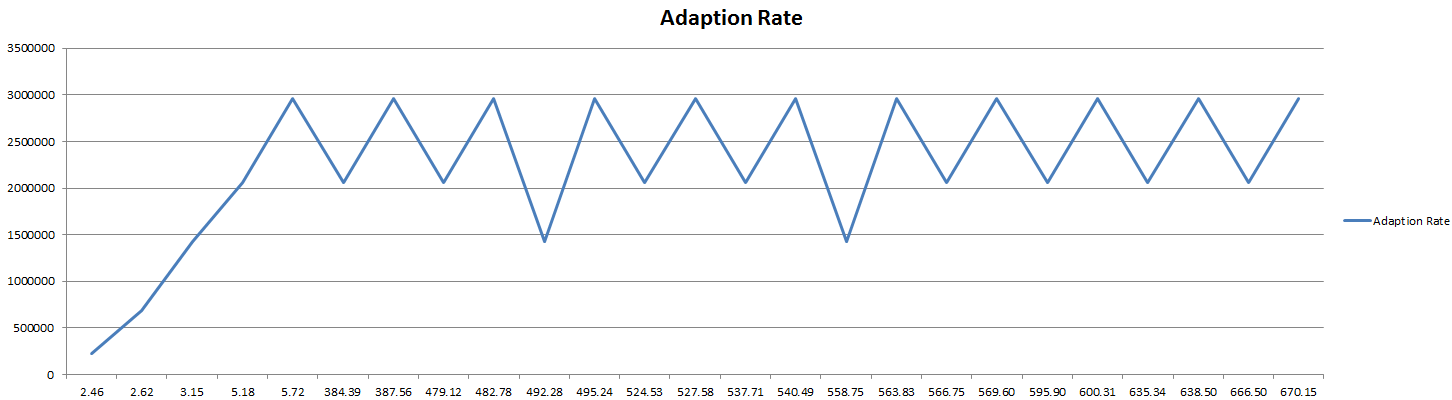


Figure 14 Rate Adaption Graph

The graph above shows the adaption rate for a particular stream session. In this streaming session bandwidth was not deliberately affected. The graph shown displays the time and bitrate the smooth streaming protocol changed to during the stream of the video ‘Elephants Dream’. On the X axis is the time (in seconds) and along the Y axis is the bitrate (in bytes).

This specific graph describes a smooth streaming video client streaming a video from a Microsoft IIS server. The video client played the video back on a small sized window. No bandwidth was changed during this test.

An important observation to note in this graph is that at the beginning of the video stream Microsoft’s Smooth Streaming protocol starts off by streaming the lowest available bitrate and then gradually climbs to the maximum available bitrate in four stages. The lowest available bitrate is streamed first to avoid and minimise buffering and loading time, which improves the viewing experience of the end user. Also the player avoids big jumps between the available bitrates. This is another quality factor, by avoiding huge jumps in quality change the end user will be less aware of sudden quality jumping, enabling a smooth streaming experience.

However, when the bitrate adaption reached the maximum available bitrate, the video quality dropped down one level which can be seen in the above graph. From there onwards, the video quality frequently changes bandwidth levels. This is not an expected result. The expected result here is to have a steady smooth stream, with a minimum of two bitrate changes (the lowest to highest available bitrate).

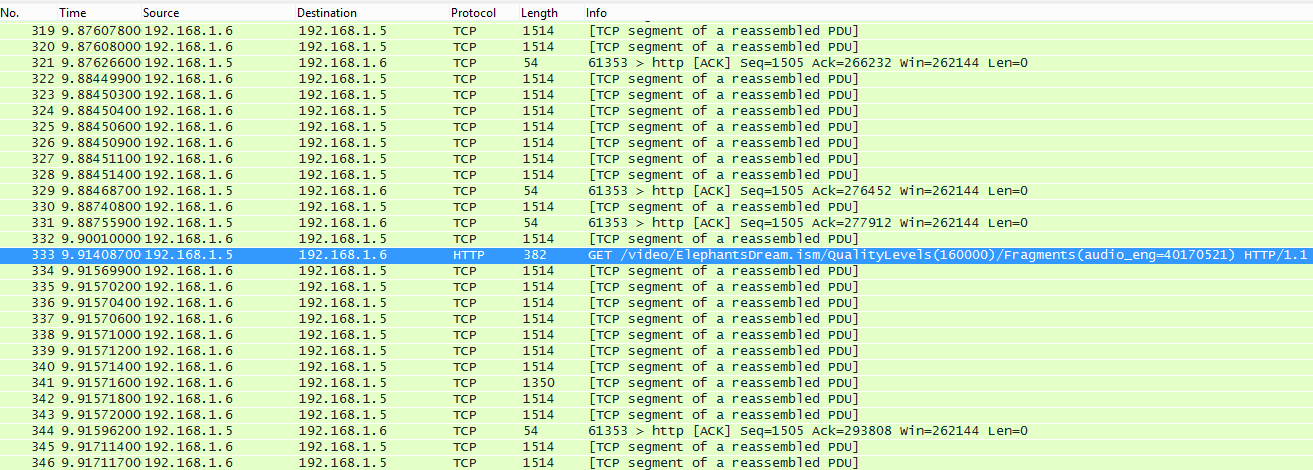
From further analysis, using a wire shark trace, the smooth streaming player only requests a new fragment once the previous fragment has been successfully downloaded, this can be seen below in figure 15. This shows that the Microsoft Smooth Streaming protocol does not use HTTP pipelining. HTTP pipelining is when a client sends multiple requests without waiting for each response from the server. The server must then send the various responses to the requests in the same order they were received (RFC). Implementing HTTP pipelining will introduce more processing on the server side, something that is avoided in HTTP adaptive streaming. 

Figure 15 Wire Shark Packet Trace

Above is a screen shot of a streaming session captured by Wire Shark. The highlighted packet shows a HTTP GET request to the IIS server, the other packets in the screenshot are TCP packets containing the requested video fragments. As evident in the above screen shot, a HTTP GET request is only issued once the previous requested fragment has been successfully downloaded, requesting a new video fragment every two seconds (on average) after the previous fragment was requested. At the start of a stream the player builds up and maintains a target playback buffer as quick as possible. This is evidence of another quality factor presented by the smooth streaming protocol. By maintaining a steady buffer rate, the viewing experience is enhanced. As the smooth streaming protocol requests video fragments every 2 seconds the player can aim to have a constant playback buffer, by requesting fragments every two seconds.

The length of the video segment sizes is also another important factor to decide when providing adaptive streaming content. Shorter segments result in more quality changes, have a low encoding efficiency and are more beneficial in an adjusting and varying bandwidth environment. Microsoft smooth streaming has a short time range between each video segment, two seconds to be exact. The decision of having two second segments makes the smooth streaming protocol more beneficial as it can adjust bitrate quality fairly quickly meaning that the smooth streaming protocol can handle most network environments. This can be seen in the screen capture of a wire shark trace below in figure 17. However, with small segment lengths comes a low encoding efficiency. As the smooth streaming protocol has two second video segments the client must process and play back each segment every two seconds, putting a strain and requiring more processing power on the client side. On the other hand longer video segments will not be able to handle varying network environments as well, especially on the unpredictability of mobile networks, as the client will not be able to switch the video quality fast enough. This raises the question; can a low powered media device such as a smart phone be able to efficiently stream smooth streaming content? And will it be able to stream the maximum available bandwidth level consistently?

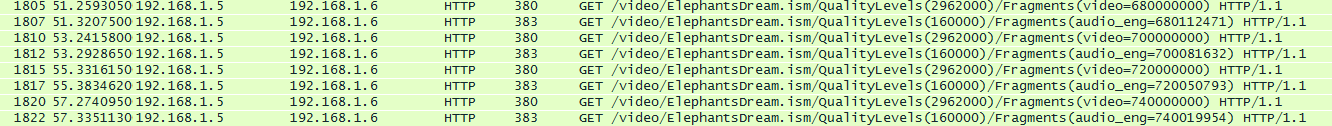


Figure 16 Wire Shark Packet Trace

Figure 16 shows the smooth streaming client requesting video and audio segments individually every two seconds. This is due to the video being encoded at varying bitrates and the audio encoded at one separate bitrate, although various languages could also be requested instead of different audio bitrate encodings.

Another observation from this wire shark capture is that during the initial time period of the stream, the player requests fragments more frequently than every two seconds, generating more traffic at the beginning of the stream. From analysing the captured stream data, the smooth streaming protocol requests more often than every two seconds at the start of the stream because the client player must build up and fill the buffer. This is clear as the player takes a few seconds to start the video playback, this can be seen in figure 14 in the adaption rate graph. This is yet another quality factor, improving the viewing experience of the end user.

After analysing a wire shark smooth streaming trace file, an in depth knowledge into Microsoft’s proprietary smooth streaming protocol has been gained, revealing both the quality of experience and quality of service factors. In this next section of the Analysis chapter, further experiments will be carried out and analysed, evaluating the smooth streaming protocol under multiple use case scenarios.

In this test, there has been no bandwidth limitation, a direct connection between the client and server has been achieved within a wireless LAN (local area network), as presented in figure 14. Also the video was viewed in a sized down window.

As can be expected, there were two bandwidth quality changes. The lowest available bitrate was selected; this is a normal characteristic with the smooth streaming protocol. Within a tenth of a second the highest available bandwidth was selected. This stayed consistent throughout the entire duration of the stream. This is an expected result, as there were no bandwidth limitations or traffic on the network. Below is a graph of the bitrate adaption and the time (in seconds) the protocol changed the quality.

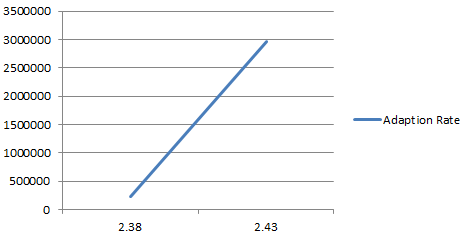


Figure 17 Adaption Rate Graph

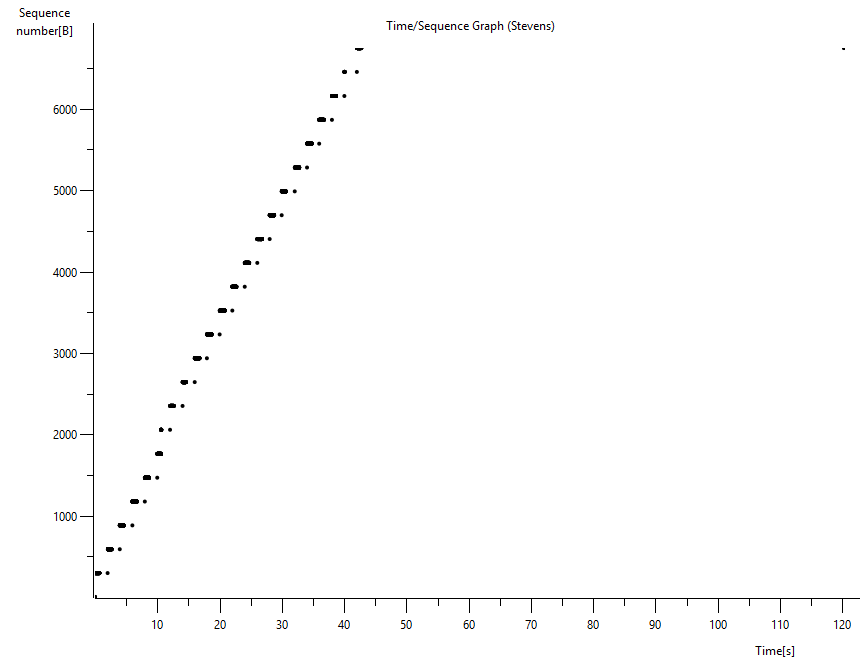


Figure 18 TCP time/sequence graph Stevens

Above is a TCP stream graph which has been formulated from wire shark. This particular graph is a time/sequence Stevens graph, which counts TCP sequence numbers over time. This graph gives a positive reading, which can be expected from the nature of this test. A diagonal line with a high gradient indicates fast data transfer. Although a diagonal line is not represented, instead a segmented line is shown. This is perfectly normal as the smooth streaming protocol downloads segments of video, not the whole file in one go.

Below is a screenshot of a TCP time sequence graph, generated from wire shark. In the graph below there are two lines, a grey line which is the uppermost line and a black line. The black line represents the advancement of byte transfer over time; the grey line represents the window size. The window size in this graph refers to the window size entry in a TCP header. The window size field controls the flow of data, its value is limited between 2 and 65,535 bytes, although this can be surpassed by using window scaling.

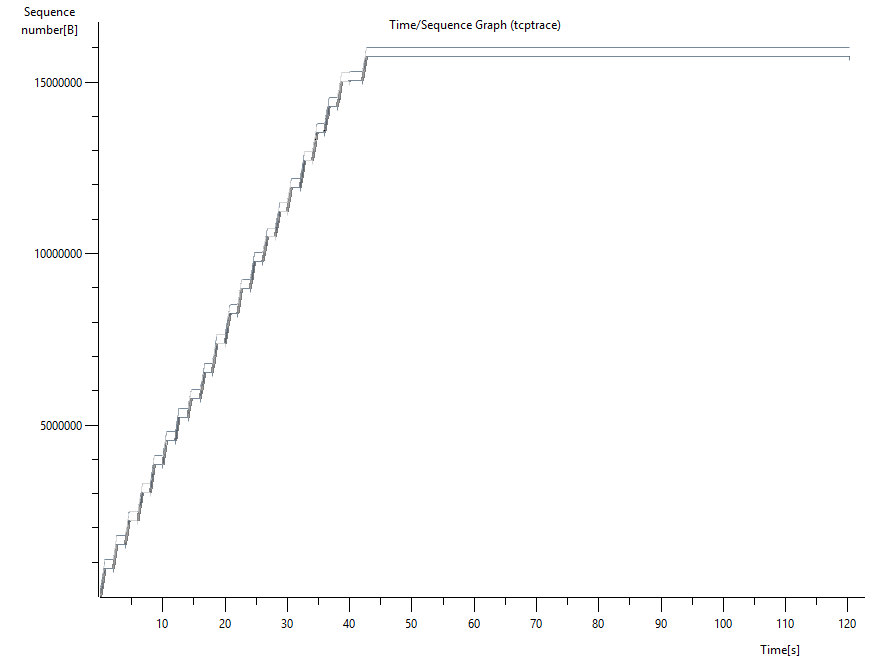


Figure 19 Wire Shark TCP Trace Graph

Figure 19 shows that the window size and window size are very close to one another, almost touching. This means that data transfer is going well throughout the duration of the stream. The data transfer practically matches the TCP window size, which is a good sign. Also the gradient of the graph is a good sign, representing a fast stream which is what was anticipated in this experiment.

Below is another test carried out which is almost identical to the experiment above, the only difference here is that the video was streamed in full screen (1920 \* 1080) with no bandwidth or networking limitation has been adjusted in this experiment.

Below is a screen shot of the client asking and retrieving the XML manifest file, this shows that the client contacts the server to request the ISMC manifest file before any segment requests are carried out, this is also evident in the smooth streaming communication sequence diagram in figure 10.

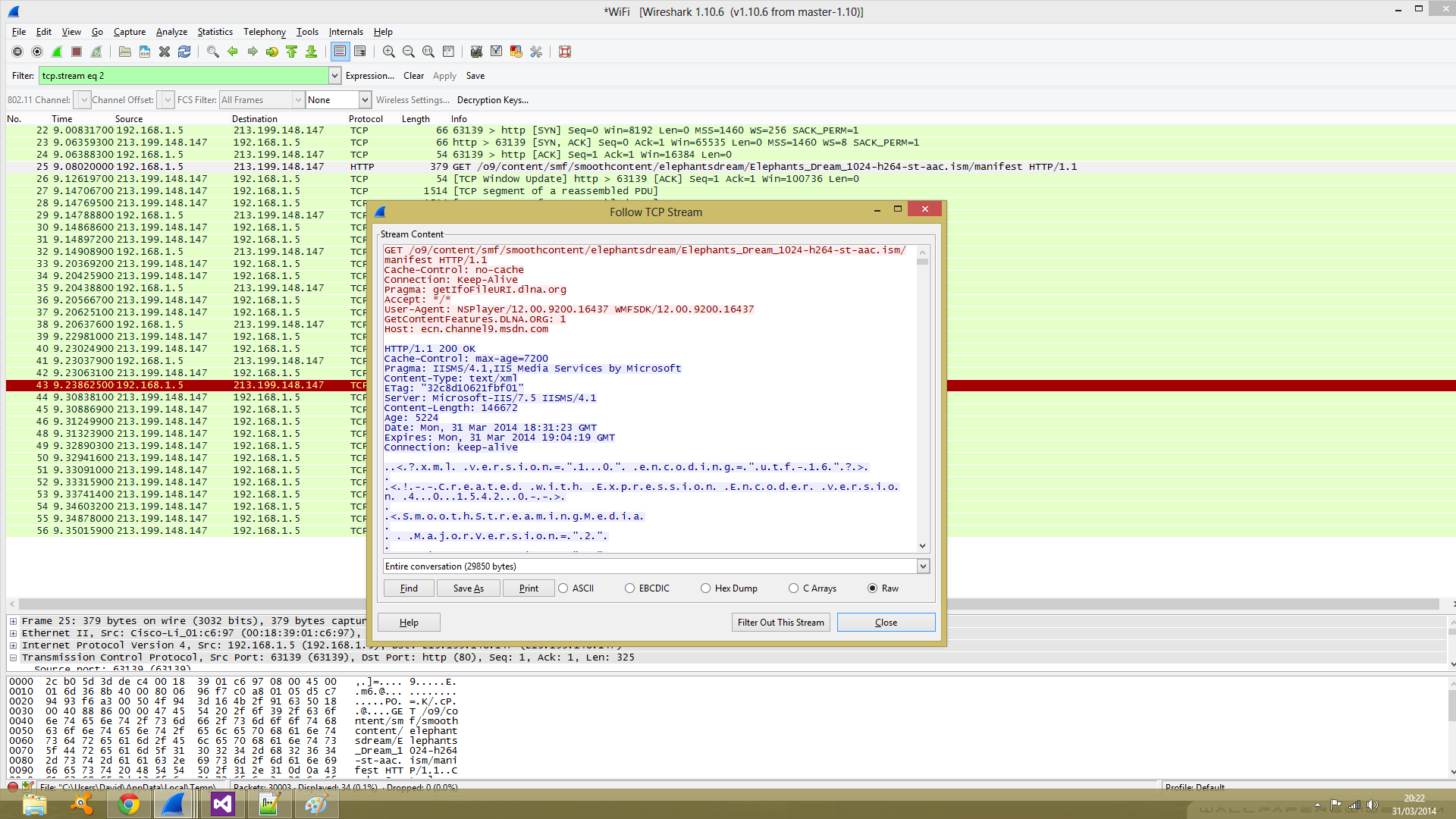


Figure 20 Smooth Streaming Manifest File

Below is another screen shot of the amount of time the media player had changed the bandwidth quality during the entire stream.

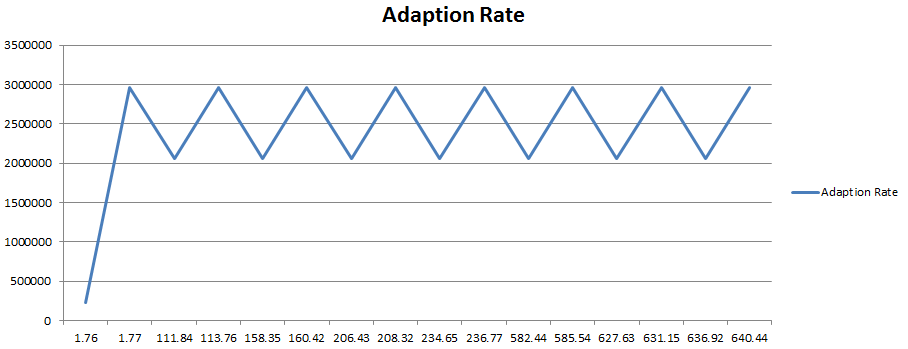


Figure 21 Rate Adaption Graph

This graph shows that the media player started off streaming the lowest available bitrate, from experiencing this in several experiments this can be characterised as normal behaviour of the smooth streaming protocol. In this experiment, it takes .01 of a second to jump straight to the maximum available bitrate. Similarly to the sized down window experiment, the time it took to transition to the maximum available bitrate was less than a second. However similarly to the other experiments carried out, the time it took to play the video after pressing the ‘play’ button. Another normal characteristic in this experiment is that the player requests more video segments at the beginning of the steam. An important note to make regarding the graph above is that the quality level continuously fluctuates between the highest available bandwidth (which is 2962000 bytes) and the level below it (which is 2056000 bytes). After delving into the wire shark trace relating to this stream session, the reason for the bandwidth fluctuation was due to TCP retransmissions. This can be seen in the screenshot below.

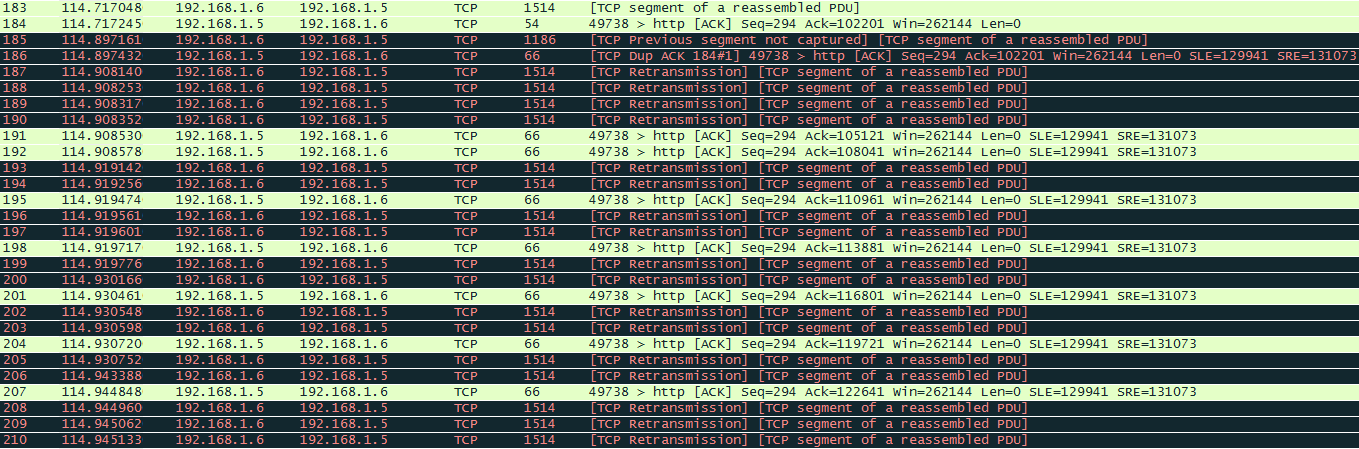


Figure 22 Wire Shark Packet Trace

The TCP retransmissions were requested throughout the stream session. There are several reasons for this happening, firstly the TCP segments were not captured which indicates dropped packets, there is also an ACKed segment that wasn't captured warning in the Wire Shark session trace, this indicates that either the packet was dropped or that Wire Shark did not successfully capture the packet fast enough to record it into the session trace. Finally there is an Out-Of-Order segment warning. As there was no bandwidth limitation in this streaming session, it is highly unlikely that packets were dropped. Also from analysing the smooth streaming client event data, there is no evidence of any video or audio segment being re-buffered. This now indicates that the requested media fragments did not meet the smooth streaming verification requirements. In the relating wire shark packet trace of the stream session, the packet arrival time makes a dramatic jump compared to the previous successfully arrived packets. This concludes that the reason behind the fluctuation in bitrate quality level is that the media fragments did not arrive within the appropriate time, which causes the TCP retransmissions and change in the bitrate quality level.

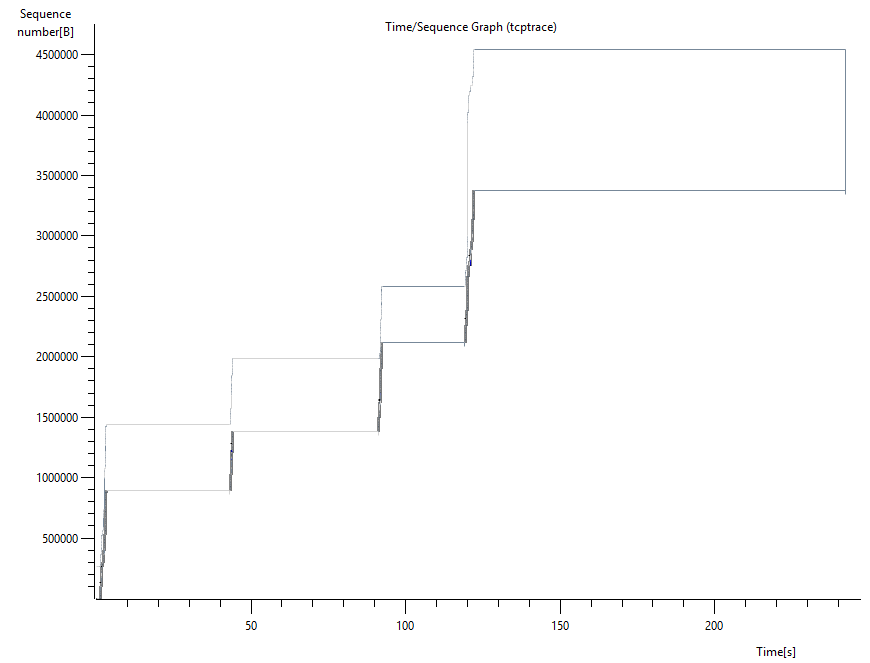


Figure 23 Wire Shark Time / Sequence TCP Trace Graph

Above is a TCP time sequence graph. In this graph the byte transfer and window size are not as close as they were expected to be, like in figure 19. As no bandwidth limitations were set in this experiment this result is unexpected. However, the reason behind these large gaps is due to the TCP retransmissions which can be seen in figure 22.

Another experiment carried out was to see the behaviour of the client player in a varied bandwidth environment. The bandwidth in this experiment was constantly throttled at various bitrates. This affected the quality of the stream, a graph can be seen below in figure 24 of the amount of quality changes during this experiment. The bandwidth was throttled using a piece of software called Net Balancer. Net Balancer displays every program using bandwidth and enables the bandwidth to be throttled for a chosen program, in this case the smooth streaming client was throttled.

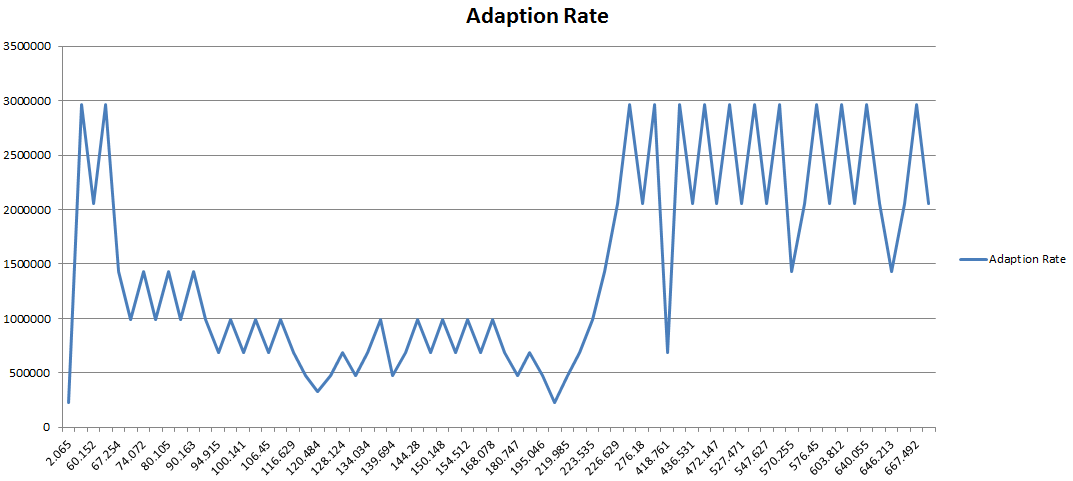


Figure 24 Rate Adaption Graph

The readings from this graph show that there are lot of quality switched throughout the entire duration of this streaming experiment. As usual, the client starts off streaming the lowest available bitrate and quickly jumps to the highest available bitrate. From there onwards the quality dramatically drops, eventually reaching the lowest available bitrate again. An observation to be made from this graph is that the smooth streaming protocol tries to avoid making big leaps in quality levels. This is a quality characteristic of the smooth streaming protocol. The reason behind the low bitrate quality streams is the low bandwidth which is forcing TCP retransmissions, packets being lost through the network.

Interestingly enough, the media player at low bandwidth does not request segments more often, like in the beginning of the stream. By building up the buffer as quick as possible in low bandwidth environments would be a benefit for the viewing experience of the user. However, this did not introduce any jitter or stalls to the video. The bitrate quality changes every two seconds (when needed), this is done in the HTTP GET request of the video segment. A screenshot can be seen below proving this.

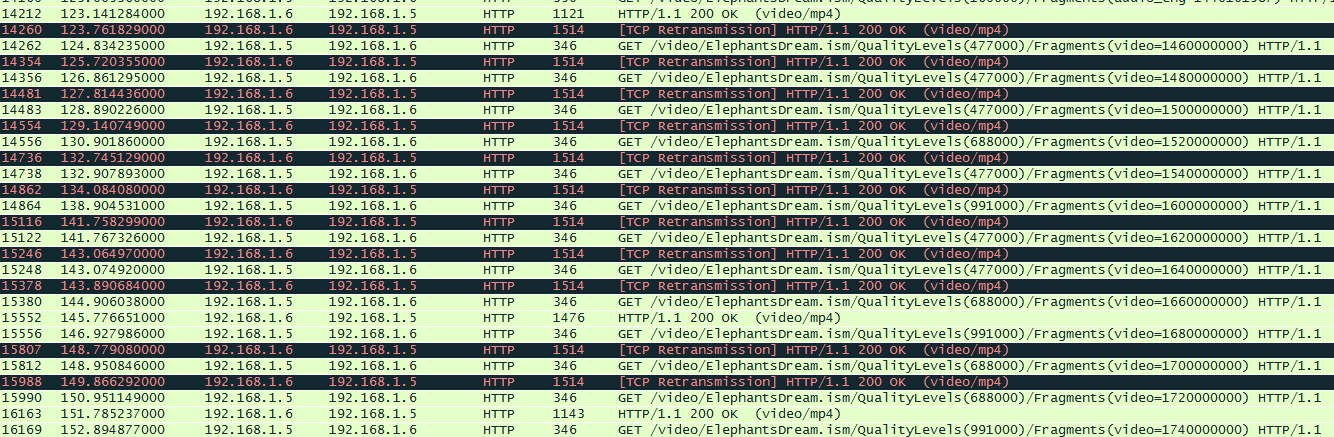


Figure 25 Wire Shark Trace File

Requesting quality change at the same time as requesting a new video chunk helps to reduce traffic to and from the server, which results in a better overall streaming experience for the end user.

Analysing the TCP time sequence Stevens graph, the graph starts off to arch. Then transmission stops until the 50th second. The transmission then continues and starts to arch again. Around the 200th second of the video, the network speed begins to speed up which can be seen in both the graph below in figure 26 and in figure 24 where the bitrate quality climbs up to the maximum available bandwidth once again. The graph below shows a gradual process through the video stream, although the network bandwidth dropped down to a low speed, the video continued to stream, and data was successfully downloaded at a lower bitrate to compensate for the connection speed.

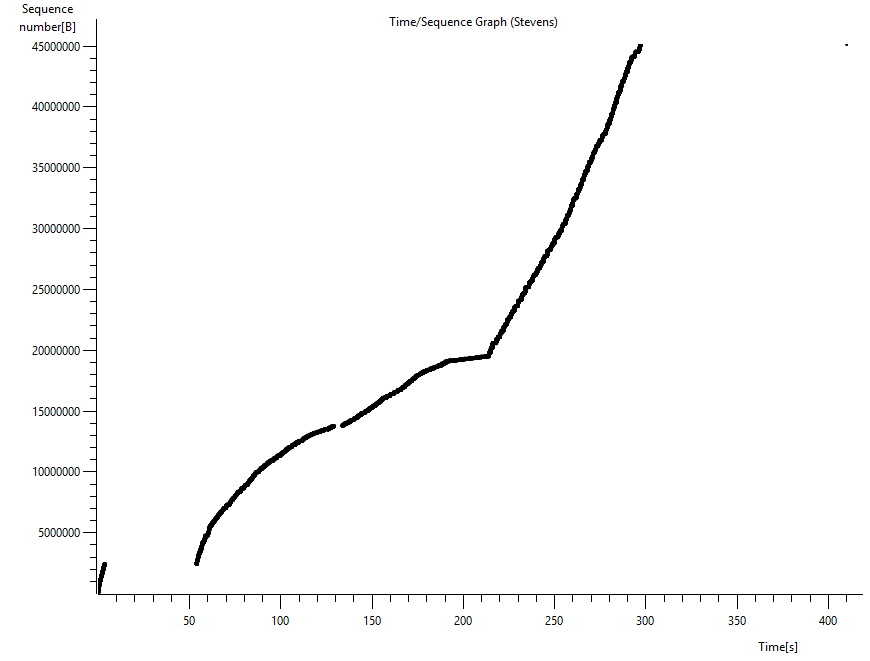


Figure 26 Wire Shark Time / Sequence Graph Stevens

Although this graph shows a solid black line, the graph in fact is very similar to the graph in the first experiment, figure 18. This graph looks like a solid line as there are a lot more video segment requests and TCP retransmissions due to the low bandwidth environment. This graph is evidence that the smooth streaming protocol does in fact handle a variety of network conditions. Less data is streamed, as the bitrate quality varies and drops down, and the overall streaming experience is smooth. There are also no zero window errors in the wire shark trace file relating to this stream session. A zero window error is when the client asks the server to stop sending data so that the client can have time to process it and playback the video. This proves that the Microsoft smooth streaming protocol delivers a smooth streaming experience, giving the quality of service and quality of experience to the end user.

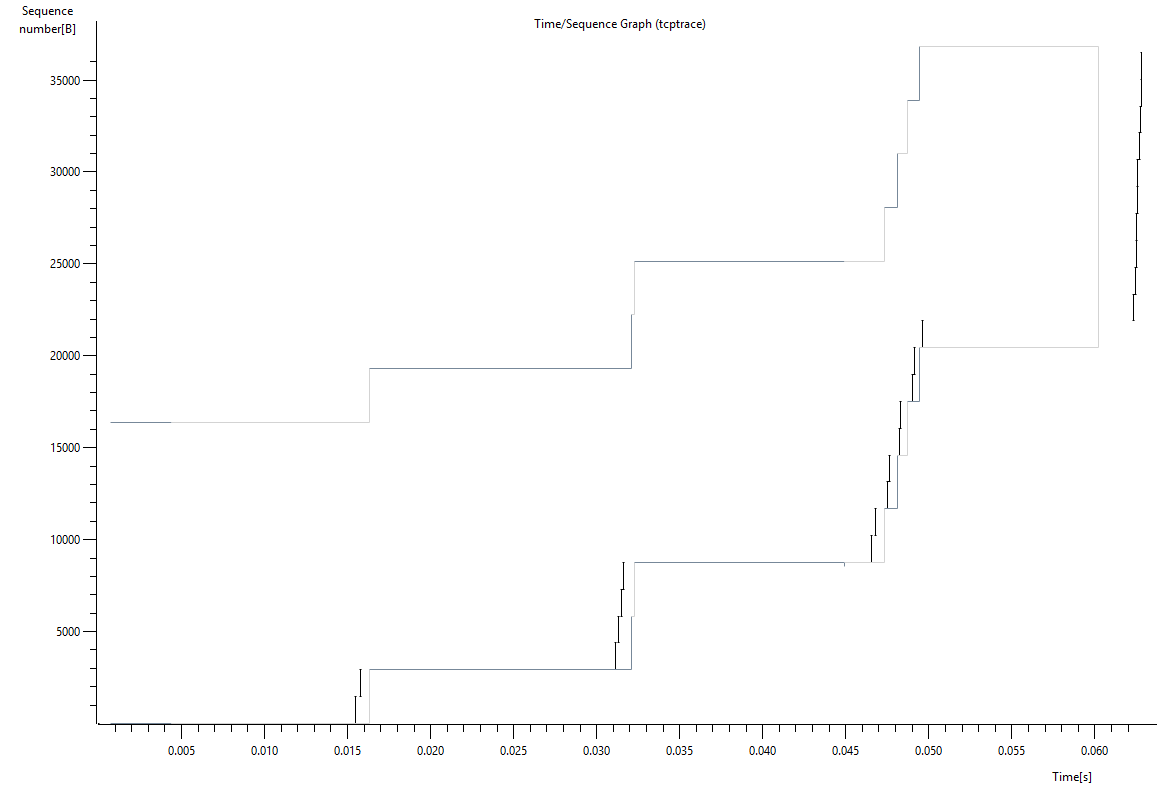


Figure 27 Wire Shark Time / Sequence TCP Trace Graph

Above is a TCP time sequence stream graph, generated from wire shark. As explained in the first occurrence of this type of graph, the black line represents the advancement of bytes over time and the grey line represents the window size. Here you can see that the TCP window size is much greater than the transfer of bytes over time. This is because of the bandwidth limitation applied during the experiment. As the bandwidth gradually speeds up, the two values get closer to each other.

Another important observation is that there was no TCP zero window. This is important as it indicates that the stream went smoothly, there was no evidence of requesting the server to stop transmitting data while the client process it. This is again proof that the smooth streaming protocol continues to smoothly stream video in a varying bandwidth environment.

# Chapter 10: Conclusion

This chapter will take in the various test results in the analysis chapter and summarise them, as well as inputting the opinions of the researcher. The quality of service and quality of experience of the Microsoft proprietary protocol smooth streaming has been analysed and displayed in the analysis chapter above. Creating different streaming scenarios in the tests above has highlighted both the quality of service and quality of experience of Microsoft smooth streaming. These quality factors include the smooth streaming fragment verification which verifies that a requested fragment has arrived at the appropriate time and played back at the requested bitrate quality. Another quality factor in the Microsoft smooth streaming protocol is at the start of the stream where the client player requests video fragments more frequently that every two seconds to build up and fill the buffer, which in turn provides the end user with a seamless smooth streaming experience. Also the length of the video fragments requested from the client to the server are two seconds long. This is an important quality factor of smooth streaming as the shorter the fragment the quicker the quality level can change according to the network environment. The Microsoft smooth streaming protocol makes managing smooth streaming content easy as each encoded bitrate quality is a separate file, media fragments are created virtually which prevents the difficulty of managing thousands of segmented video files.

From researching and developing a smooth streaming application, the development process is fairly easy to implement. Encoding a video for Microsoft smooth streaming is very simple to accomplish using Microsoft Expression encoder. Unfortunately to encode video content for smooth streaming, a paid licence is required. After encoding the chosen video for use with Microsoft smooth streaming the video must then be placed on a Microsoft IIS web server with media services. The downfall to this is that hosting the video on any other web server will prevent you from viewing smooth streaming content. The video can only be streamed off of an IIS web server.

Another downfall to the smooth streaming protocol is that when bandwidth variations and fluctuations take control of the streaming quality, the smooth streaming protocol still continues to request video segments every two seconds. In this case, if the bandwidth reached a certain threshold (below the lowest available bitrate), then the player could request video segments more frequently, filling up the buffer to provide the end user with a smooth streaming experience.

However from developing, implementing and testing the smooth streaming protocol, quality factors have been discovered that will overall improve the end users viewing experience. These factors include the two second fragments, requesting video fragments more frequently during the beginning of the stream to fill the buffer, and the unwillingness of jumping and switching to huge bitrate quality levels. Another good characteristic of the smooth streaming protocol is that the video and audio channels of the stream are requested separately. Audio can be encoded at one value where the video can be encoded at varied bitrate levels.

Developing a standalone Windows 8 store application is very limiting as there is limited support in terms of classes and methods. There is however an SDK (software development kit) for Microsoft smooth streaming. From using this, the smooth streaming client was able to interact with the smooth streaming client manifest file and stream the video content accordingly. Due to the limitation of the windows store application the current download speed in real time was unfortunately unavailable to record and gather information from. Another issue with the windows store application is that the Windows 8 operating system is required to run the application.

Future research:

From researching various adaptive streaming technologies, a comparison between the popular commercial adaptive streaming technologies and previous older technologies such as RTP could be carried out, running the same tests and metrics in the experiments in the analysis chapter above. Also testing the smooth streaming protocol on a variety of platforms and devices such as smartphones, tablets and on a web browser, using Microsoft Silverlight.

# Appendix

Figure 11 – Java data extraction program:

public class SsAdaption {

public static void main(String[] args) {

try {

Scanner txtscan = new Scanner(new File("Path"));

while(txtscan.hasNextLine()){

String str = txtscan.nextLine();

if(str.contains("BitrateChanged")){

String time = str.substring(6, 22);

System.*out*.print(time + ",");

String bandwidth = str.substring(38);

System.*out*.print(bandwidth + "\n");

}

}

txtscan.close();

}catch(Exception e){

e.printStackTrace();

}

}

}

# References

2616, R. (n.d.). *www.w3.org/Protocols*. Retrieved from http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html

Adobe. (2010). *Adobe Flash Access Overview On Protocted Streaming.* Retrieved from www.adobe.com: http://www.adobe.com/products/flashmediaserver/pdfs/flashaccess\_wp\_protectstreaming.pdf

Adobe. (2010). *HTTP Dynamic Streaming on the Adobe Flash Platform.* Retrieved from www.adobe.com/products: http://www.adobe.com/products/httpdynamicstreaming/pdfs/httpdynamicstreaming\_wp\_ue.pdf

Adobe. (2010). *www.adobe.com/products/flashmediaserver/pdfs/flashaccess\_wp\_protectstreaming.pdf.* Retrieved from www.adobe.com/products: http://www.adobe.com/products/flashmediaserver/pdfs/flashaccess\_wp\_protectstreaming.pdf

Ahmend Mansy, M. A. (2011). Analysis of adaptive streaming for hybrid cdn/p2p live video systems. Atlanta, Georgia, United States.

Alex, Z. (2009, March). *IIS Smooth Streaming Technical.* Retrieved from www.microsoft.com: http://www.microsoft.com/en-us/download/confirmation.aspx?id=17678

Apple. (2013, October 14). *draft-pantos-http-live-streaming-12 - HTTP Live Streaming*. Retrieved from tools.ietf.org: http://tools.ietf.org/html/draft-pantos-http-live-streaming-12#page-4

Apple. (2013, October 14). *tools.ietf.org/html/draft-pantos-http-live-streaming-12*. Retrieved from tools.ietf.org: http://tools.ietf.org/html/draft-pantos-http-live-streaming-12#page-4

Apple. (2014, February 11). *HTTP Live Streaming Overview: HTTP Streaming Architecture*. Retrieved from developer.apple.com: developer.apple.com/library/mac/documentation/networkinginternet/conceptual/streamingmediaguide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html#//apple\_ref/doc/uid/TP40008332-CH101-SW2

Apple. (2014, February 11). *HTTP Live Streaming Overview: Introduction*. Retrieved December 08, 2013, from developer.apple.com/: https://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/Introduction/Introduction.html

Apple. (n.d.). *developer.apple.com*. Retrieved December 08, 2013, from developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html: https://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html

Apple. (n.d.). *developer.apple.com/library/ios/documentation.html*. Retrieved December 08, 2013, from developer.apple.com: https://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html

Apple. (n.d.). *developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/Introduction/Introduction.html*. Retrieved December 08, 2013, from developer.apple.com/: https://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/Introduction/Introduction.html

Apple. (n.d.). *developer.apple.com/library/mac/documentation/networkinginternet/conceptual/streamingmediaguide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html#//apple\_ref/doc/uid/TP40008332-CH101-SW2*. Retrieved from developer.apple.com.

Arjan Durresi, R. J. (2005). *www.cse.wustl.edu/~jain/books/ftp/rtp.pdf.* Retrieved from www.cse.wustl.edu: www.cse.wustl.edu/~jain/books/ftp/rtp.pdf

Azure, M. W. (n.d.). */www.windowsazure.com/en-us/develop*. Retrieved December 08, 2013, from http://www.windowsazure.com: http://www.windowsazure.com/en-us/develop/media-services/how-to-guides/media-services-java/

Azure, M. W. (n.d.). *www.windowsazure.com/en-us/develop/media-services/how-to-guides/media-services-java/#what-are*. Retrieved December 08, 2013, from www.windowsazure.com: www.windowsazure.com/en-us/develop/media-services/how-to-guides/media-services-java/#what-are

Azure, W. (2013). *www.windowsazure.com/en-us/develop/media-services/*. Retrieved from www.windowsazure.com: www.windowsazure.com/en-us/develop/media-services/

Azure, W. (2014). *How to use Media Services (Java) - Azure feature guide*. Retrieved December 08, 2013, from http://www.windowsazure.com: http://www.windowsazure.com/en-us/develop/media-services/how-to-guides/media-services-java/

Azure, W. (2014). *How to use Media Services (Java) - Azure feature guide*. Retrieved December 08, 2013, from www.windowsazure.com: www.windowsazure.com/en-us/develop/media-services/how-to-guides/media-services-java/#what-are

DASH, I. F. (n.d.). *dashif.org/about-dash/industry-forum*. Retrieved November 08, 2013, from dashif.org: dashif.org/about-dash/industry-forum

DASH, I. F. (n.d.). *dashif.org/identifiers/profiles/*. Retrieved November 10, 2013, from dashif.org/: http://dashif.org/identifiers/profiles/

Denis, B. (2011, April 18). *HTTP Dynamic Streaming with Flash Access protection - Adobe Developer Connection | Adobe Developer Connection*. Retrieved from http://www.adobe.com/: http://www.adobe.com/devnet/adobe-media-server/articles/dynamic-streaming-protection.html

Dream, E. (2010, May 31). *www.elephantsdream.org*.

FAS. (n.d.). *www.fas.org/man/dod-101/navy/docs/es310/comnet/ComNets.htm*. Retrieved from www.fas.org: http://www.fas.org/man/dod-101/navy/docs/es310/comnet/ComNets.htm

Frank Hartung, U. H. (2007, March 1). *Delivery of Broadcast Services in 3G Networks.* Retrieved December 08, 2013, from http://www.ericsson.com: http://www.ericsson.com/res/thecompany/docs/journal\_conference\_papers/wireless\_access/ITB\_mar07\_huschke.pdf

Government, H. K. (2008, February). *www.infosec.gov.hk/english/technical/files/peer.pdf.* Retrieved from www.infosec.gov.hk: www.infosec.gov.hk/english/technical/files/peer.pdf

H. Schulzrinne, C. U. (1998, April). *www.ietf.org/rfc/rfc2326.txt.* Retrieved from www.ietf.org: www.ietf.org/rfc/rfc2326.txt

Javvin, T. (2005). *Network Protocols Handbook Fourth Edition.* Saratoga.

JDSU. (n.d.). Mobile Networks Tutorial - white paper.

Larry Peterson, B. D. (2007). *Computer Networks A Systems Approach Edition 4.* San Francisco: Morgan Kaufmann.

Laukens, N. (2011, February 4). *tech.ebu.ch/docs/techreview/trev\_2011-Q1\_adaptive-streaming\_laukens.pdf.* Retrieved from tech.ebu.ch/docs: tech.ebu.ch/docs/techreview/trev\_2011-Q1\_adaptive-streaming\_laukens.pdf

Microsoft. (2013, October Friday). Retrieved from download.microsoft.com: http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-RTSP].pdf

Microsoft. (2013, February 18). DASH Content Protection using Microsoft PlayReady.

Microsoft. (2013). *www.iis.net/*. Retrieved from www.iis.net/downloads/microsoft/smooth-streaming: http://www.iis.net/downloads/microsoft/smooth-streaming

Microsoft. (2014). *[MS-SSTR]: Smooth Streaming Protocol*. Retrieved from msdn.microsoft.com: http://msdn.microsoft.com/en-us/library/ff469426.aspx

Microsoft. (2014). *Smooth Streaming : The Official Microsoft IIS Site*. Retrieved from www.iis.net: http://www.iis.net/downloads/microsoft/smooth-streaming

Microsoft. (n.d.). *msdn.microsoft.com*. Retrieved from http://msdn.microsoft.com/en-us/library/cc295302.aspx

Microsoft. (n.d.). *www.microsoft.com*. Retrieved December 08, 2013, from http://www.microsoft.com/mediaplatform: http://www.microsoft.com/en-us/mediaplatform/overview.aspx

Mozilla. (n.d.). *www.mozilla.org/en-US/about/manifesto/*. Retrieved November 2013, 10, from www.mozilla.org: http://www.mozilla.org/en-US/about/manifesto/

MPEG. (2012). *Patent No. ISO/IEC 23009-1.*

MS\_RTSP, M. (2014, February 13). *www.download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-RTSP].pdf.* Retrieved from download.microsoft.com: http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-RTSP].pdf

Nelson, D. M. (2010, August). *www.microsoft.com/en-us/download/details.aspx?id=2547.* Retrieved from www.microsoft.com: http://www.microsoft.com/en-us/download/details.aspx?id=2547

Networks, R. (1998). *docs.real.com/docs/proxykit/rtspd.pdf.* Retrieved from docs.real.com: docs.real.com/docs/proxykit/rtspd.pdf

Networks, R. (n.d.). *www.realnetworks.com*. Retrieved October 13, 2013, from http://www.realnetworks.com/our-story/: http://www.realnetworks.com/our-story/

RFC, 2. (n.d.). *Hypertext Transfer Protocol - HTTP 1.1*. Retrieved from www.w3.org/Protocols: http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html

Roettgers, J. (2013, March 13). *www.gigaom.com/2013/03/13/youtube-adaptive-streaming-mobile-tv/*. Retrieved from www.gigaom.com: http://gigaom.com/2013/03/13/youtube-adaptive-streaming-mobile-tv/

S. Casner, A.-V. T. (1996, January). *www.ietf.org/rfc/rfc1889.txt.* Retrieved from www.ietf.org: www.ietf.org/rfc/rfc1889.txt

Shawnee. (2009). *www.shawnee.net*. Retrieved December 08, 2013, from http://www.shawnee.net/products-hosting.php: http://www.shawnee.net/products-hosting.php

Smith, J. (2013). *4444.* 444: 44.

Sodagar, I. (2011, November). *ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=6077864&tag=1.* Retrieved November 09, 2013, from ieeexplore.ieee.org: ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=6077864&tag=1

Tommi, K. (n.d.). *www.netlab.tkk.fi/opetus/s38130/k99/presentations/4.pdf.* Retrieved from www.netlab.tkk.fi: www.netlab.tkk.fi/opetus/s38130/k99/presentations/4.pdf

V. Jacobson, P. D. (2003, July). *www.ietf.org/rfc/rfc3550.txt*. Retrieved from www.ietf.org: www.ietf.org/rfc/rfc3550.txt