Experimental evaluation of a DASH implementation

Noor Afshan Fathima (A20385838), Arka Deb (A20386958)
Graduate Students: CS-544, Dept. of Computer Science (India Online)
Illinois Institute of Technology, Chicago

***Abstract*—Dynamic Adaptive (bitrate) Streaming over HTTP is heavily being adopted, as it offers significant advantages in terms of both user-perceived quality and resource utilization for content and network service providers. MPEG-DASH is picking pace along with other proprietary HTTP based adaptive streaming technologies and is expected to become the standard in the near future for interoperability. This paper presents a seamless integration of the MPEG standard on Dynamic Adaptive Streaming over HTTP (DASH) in the Web using the HTML5 video element. We further present DASH-JS, a JavaScript-based MPEG-DASH client. Furthermore, we present the integration of MP4 based media segments in DASH and a corresponding Media Presentation Description (MPD). Our preliminary evaluation demonstrates the bandwidth adaptation capabilities to show the effectiveness of the system.**

Keywords—Bitrate, Adaptive streaming, HTTP, dash.js, heterogenous networks, bandwidth MPEG-DASH over HTTP, HTML5, MP4, JavaScript, World Wide Web

I. Introduction

Over the last seven to eight years video-based applications, and video streaming in particular, have become utterly popular generating more than half of the aggregate Internet traffic. Cisco recently unveiled a report showing that by 2019, on-line video will be responsible for 80% of global Internet traffic. These online videos are used to showcase movies, series, concerts and what not. Mobile games landing pages usually don't come with a lot of content. All they need is one good landing page with a gameplay video that appeals to people, informs them, and triggers them to download/buy this game. A gameplay video is what users first prefer to land on when they first visit the website. Increasingly, users can watch this gameplay video over heterogeneous devices and networks. Many of these viewers/users also sit behind corporate firewalls that do not allow anything other than HTTP content to pass through. Game developers showcasing their gameplay video need to ensure an optimum and uninterrupted experience for the viewers on these various networks.

Perhaps, surprisingly though, video streaming today runs over IP without any specialized support from the network. This has become possible through the gradual development of highly efficient video compression methods, the penetration of broadband access technologies, and the development of *adaptive video players* that can compensate for the unpredictability of the underlying network through sophisticated rate-adaptation, playback buffering, and error recovery and concealment methods.

TCP currently is the most used underlying protocol. TCP's congestion control mechanisms and reliability requirement do not necessarily hurt the performance of video streaming, especially if the video player is able to adapt to large throughput variations. Further the use of TCP, and of HTTP over TCP in particular, greatly simplifies the traversal of firewalls and NATs. The first wave of HTTP-based video streaming applications used the simple progressive download method, in which a TCP connection simply transfers the entire video file as quickly as possible. One major issue of this approach is that all clients receive the same encoding of the video, despite the large variations in the underlying available bandwidth both across different clients and across time for the same client. This led to the development of a new wave of HTTP-based streaming applications that we refer to as Dynamic Adaptive Streaming over HTTP (DASH). Dynamic Adaptive Streaming over HTTP (DASH) is an adaptive streaming protocol. This means that it allows for a video stream to switch between bit rates on the basis of network performance, in order to keep a video playing.

Several recent players, such as Microsoft's Smooth Streaming, Adobe OSMF, as well as the players developed or used by Netflix, Move Networks and others, use this approach. In adaptive streaming, the server maintains multiple profiles of the same video, encoded in different bitrates and quality levels. Further, the video object is partitioned in *segments*, typically a few seconds long. A player can then request different segments at different encoding bitrates, depending on the underlying network conditions.

In DASH the rate-adaptation is done client-side. The adoption of rate adaptation gave way to modern Adaptive Bit-rate Streaming, with a bit-rate for every user's specific needs. This improves server-side scalability. Another benefit of this approach is that the player can control its playback buffer size by dynamically adjusting the rate at which new segments are requested.

In this paper we present an approach of implementing MPEG-DASH with JavaScript with the use of the HTML5 video element. HTML5 standard offers ways to integrate video and audio in Websites, leveraging built-in capabilities of the Web browsers. We finally publish our analysis report.

In second section we describe *System Model Outline* which explains about MPD file and MPEG-DASH Architecture. The subsequent section describes about tools and methodology for encoding of raw MP4 video into H.264/MPEG-4 AVC format using x264 codec and ffmpeg converter. Also to generate Media Presentation Description (MPD) file using GPAC's MP4Box. The integration of MPEG-DASH player in our landing page's source code, transfer of files on external server, setting up of local server, our DASH-JS client, and a preliminary evaluation thereof is presented in the subsequent sections.

The ending section concludes the paper and also includes future work.

II. SYSTEM MODEL OUTLINE

The basic idea is that the media content will be encoded in different versions, which differ in bitrates, resolutions, etc. and will then be chopped into segments that could be accessed individually by the client via HTTP GET requests.

We assume that we have a video stream consisting of n segments, each containing τ seconds of playback. We call τ the duration of a segment. (In order to simplify the presentation, we assume that all segments have equal duration. The presented mechanisms can, however, easily be extended to the general case.) Each segment is available in different representations. We denote by R the set of representations available for the given video stream. The representations in R might differ in various aspects. They might contain 2D or 3D video, different video quality levels, different spatial resolutions, etc. With MPEG DASH, the set of available representations is obtained by the client prior to the begin of the streaming session by downloading the XML-based Media Presentation Description (MPD) file. We assume that after the user requests to watch the video stream, (i) the client downloads the segments in chronological order, (ii) each of the segments is downloaded in exactly one representation, and that (iii) the downloads are non-preemptive (that is, the download of segment $i-1$ must be completed before the start of the download of segment i). After the data is downloaded, it is decoded and presented to the user. It is not required that a segment is downloaded completely before it can be decoded and played. Although the decoder requires a certain minimum amount of bytes to trigger decoding, it is usually less than the duration of a segment. During the time between arriving at the client and being played,the data is locally buffered.

A. Media Presentation Description

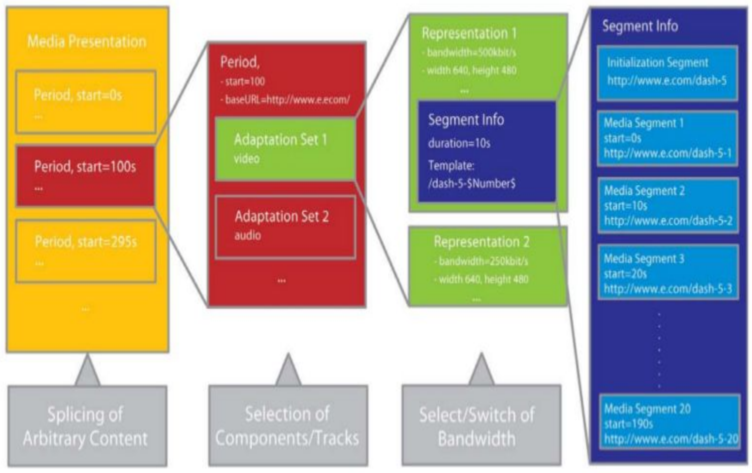


fig 1. Media Presentation Description

The MPEG-DASH Media Presentation Description (MPD) is an XML document containing information about media segments, their relationships and information necessary to choose between them, and other metadata that may be needed by clients.

Periods

Periods, contained in the top-level MPD element, describe a part of the content with a start time and duration. Multiple Periods can be used for scenes or chapters, or to separate ads from program content.

Adaptation Set

Adaptation Sets contain a media stream or set of media streams. In the simplest case, a Period could have one

Adaptation Set containing all audio and video for the content, but to reduce bandwidth, each stream can be split into a different Adaptation Set. A common case is to have one video Adaptation Set, and multiple audio Adaptation Sets (one for each supported language). Adaptation Sets can also contain subtitles or arbitrary metadata.

Adaptation Sets are usually chosen by the user, or by a user agent (web browser or TV) using the user's preferences (like their language or accessibility needs).

Representations

Representations allow an Adaptation Set to contain the same content encoded in different ways. In most cases, Representations will be provided in multiple screen sizes and bandwidths. This allows clients to request the highest quality content that they can play without waiting to buffer, without wasting bandwidth on unneeded pixels (a 720p TV doesn't need 1080p content). Representations can also be encoded with different codecs, allowing support for clients with different supported codecs (as occurs in browsers, with some supporting MPEG-4 AVC / h.264 and some supporting VP8), or to provide higher quality Representations to newer clients while still supporting legacy clients (providing both h.264 and h.265, for example). Multiple codecs can also be useful on battery-powered devices, where a device might chose an older codec because it has hardware support (lower battery usage), even if it has software support for a newer codec.

Representations are usually chosen automatically, but some players allow users to override the choices (especially the resolution). A user might choose to make their own representation choices if they don't want to waste bandwidth in a particular video (maybe they only care about the audio), or if they're willing to have the video stop and buffer in exchange for higher quality.

SubRepresentations

SubRepresentations contain information that only applies to one media stream in a Representation. For example, if a Representation contain both audio and video, it could have a SubRepresentation to give additional information which only applies to the audio. This additional information could be specific codecs, sampling rates, embedded subtitles. SubRepresentations also provide information necessary to extract one stream from a multiplexed container, or to extract a lower quality version of a stream (like only I-frames, which is useful in fast-forward mode).

Media Segments

Media segments are the actual media files that the DASH client plays, generally by playing them back-to-back as if they were the same file (although things can get much more complicated when switching between representations). The two containers described by MPEG are the ISO Base Media File Format (ISOBMFF), which is similar to the MPEG-4 container format, and MPEG-TS.

Media Segment locations can be described using BaseURL for a single-segment Representation, a list of segments (SegmentList) or a template (SegmentTemplate). Information that applies to all segments can be found in a SegmentBase. Segment start times and durations can be described with a SegmentTimeline (especially important for live streaming, so a client can quickly determine the latest segment). This information can also appear at higher levels in the MPD, in which case the information provides is the default unless overridden by information lower in the XML hierarchy. This is particularly useful with SegmentTemplate.

Segments can be in separate files (common for live streaming), or they can be byte ranges within a single file (common for static / "on-demand").

Index Segments

Index Segments come in two types: one Representation Index Segment for the entire Representation, or a Single Index Segment per Media Segment. A Representation Index Segment is always a separate file, but a Single Index Segment can be a byte range in the same file as the Media Segment.

Index Segments contain ISOBMFF 'sidx' boxes, with information about Media Segment durations (in both bytes and time), stream access point types, and optionally subsegment information in 'six' boxes (the same information, but within segments). In the case of a Representation Index Segment, the 'sidx' boxes come one after another, but they are preceded by an 'sidx' for the index segment itself.

B. DASH-JS Architecture

DASH-JS comprises the following components:

The Event Handlers act as an interface for the Media Source API and process events issued by the Media Source API. When the Media Source API issues the webkit source open event the Event Handlers will trigger the download of the MPD and afterwards the MPD will be handed over to the MPD Parser. The progress event is used to call the Buffer which is responsible for retrieving and buffering the segments of a media representation.

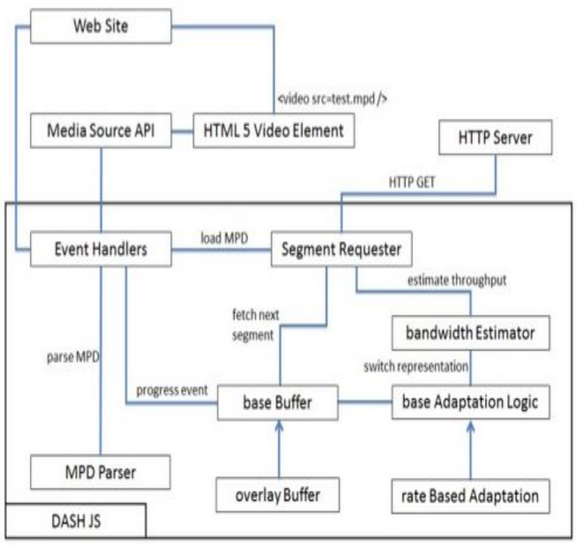


fig 2. depicts the architecture of our JavaScript implementation of MPEG–DASH.

The MPD Parser is used to parse the requested MPD. This will generate an object which holds all relevant information about the MPD such as the defined representations with their segments, the bitrate of each representation, the resolution of the video frames, and whether the segments are aligned throughout all representations. This information is used by the adaptation logic for determining which representations are available. Requesting the MPD is done with the Segment Requester, which uses the *xmlHttpRequest* to generate HTTP requests. Furthermore, the Segment Requester provides an asynchronous and synchronous method for requesting and receiving HTTP requests and responses. The Bandwidth Estimator is used by the Segment Requester to measure and estimate the current effective throughput. The class *Base Buffer* provides the base class for buffer implementations. It offers the possibility to register event handlers for specific events (e.g., criticalFillLevel).
Furthermore, two implementations of buffer types are provided. First, a buffer that operates on bytes, which can be used to store segments and query byte ranges of the stored segments and, second, a buffer that operates on the length of a segment.

Due to the fact that the Media Source API does not allow accessing the buffer of the HTML5 video element dash.js reference client has a so-called *Overlay Buffer*. This buffer inherits the Base Buffer and mimics the actual buffer of the video element. With each progress event issued by the Media Source API the method *bufferFillStateListener()* will be called. This method keeps track of the progress of the media being played back by subtracting the amount of time that has passed between the last call of this method and the current timestamp from the buffer. Additionally, it will trigger the download of further segments, which are then handed over to the video element and the buffer level is increased by the length of each segment. This buffer does not store byte chunks. It just keeps track of how many segments have been pushed into the video element in seconds and the current playback time of the video. The buffer and the estimated bandwidth can be used to decide which representation of the media stream should be selected by making the decision based upon the bandwidth and/or the fill state of the buffer. To allow the implementation of different adaptation logics the class Base Adaptation Logic is provided. Every adaptation logic must inherit this class in order to be used. The adaptation logic is called after each downloaded segment. The base class implements only a single method called *switchRepresentation()* , which should be overridden.
This method shall contain the adaptation decision logic. For our DASH-JS client we have used dash.js reference client's simple adaptation logic (*Rate Based Adaptation Logic*). Furthermore, our implementation does not need any third-party software or browser plug-ins. The DASH-JS client is purely written in JavaScript and makes use of the Media Source.

III. TOOLS

MP4 has penetrated into every aspect of our life. Teachers or trainers usually insert MP4 videos into PPTs to make further explanation. Companies always save their propaganda films as MP4 to fit for miscellaneous broadcast platforms flawlessly. Web surfers often download online videos as MP4 for fluent playback on media players, editing for further use, burning to DVD, etc. The majority of videos are compressed – which means they have been altered to take up less space on a computer.

A codec simply compresses and decompresses this data, interpreting the video file and determining how to play it on your screen. Then there are containers, which are essentially a bundle of media files.

A container usually consists of a video and an audio codec, but may also contain information such as subtitles. Containers allow you to choose one codec for your video and one for your audio – giving the user a little more control over how to record and rip your media. The appearance of MP4 brings us many benefits in terms of movie playback, video transport, online video streaming and more.

The MPEG-4 format is used to share files on the web. Video and audio tracks are compressed separately where the video file is compressed with MPEG-4 encoding and audio with AAC compression, which is the same audio compression type used in .AAC files.

The encoding of raw video into H.264/MPEG-4 AVC format using x264 codec and ffmpeg converter is further described in this section. Followed by describing method used to generate Media Presentation Description (MPD) file using GPAC's MP4Box which has options for interleaving, fragmenting and segmentation.

A. x264

x264 is an implementation of MPEG4 Part 10, also known as AVC or H264. x264 command (re-) encodes the video in H.264/AVC with the properties we will need. *x264 provides a command line interface as well as an API. We first experiment with command line interface and later also with API through ffmpeg interface. x264 forms the core of many web video services, such as Youtube, Facebook, Vimeo, and Hulu. It is widely used by television broadcasters and ISPs. It Provides best-in-class performance, compression, and features.*

All the command line parameters are explained after the code.

```
x264 --output intermediate_2400k.264 --fps 24 --preset slow
--bitrate 2400 --vbv-maxrate 4800 --vbv-bufsize 9600
--min-keyint 48 --keyint 48 --scenecut 0 --no-scenecut --pass 1
--video-filter "resize:width=1280,height=720"
inputvideo.mp4
```

--output intermediate_2400k.264 Specifies the output filename.

--fps 24 Specifies the framerate which shall be used, here 24 frames per second.

--preset slow Presets can be used to easily tell x264 if it should try to be fast to enhance compression/quality. Slow is a good default.

--bitrate 2400 The bitrate this representation should achieve in kbps

--vbv-maxrate 4800 Rule of thumb: set this value to the double of --bitrate.

--vbv-bufsize 9600 Rule of thumb: set this value to the double of --vbv-maxrate.

--keyint 96 Sets the maximum interval between keyframes. This setting is important as we will later split the video into segments and at the beginning of each segment should be a keyframe. Therefore, --keyint should match the desired segment length in seconds multiplied with the frame rate. Here: 4 seconds * 24 frames/seconds = 96 frames.

--min-keyint 96 Sets the minimum interval between keyframes. See --keyint for more information.We achieve a constant segment length by setting minimum and maximum keyframe interval to the same value and furthermore by disabling scenecut detection with the --no-scenecut parameter.

--no-scenecut Completely disables adaptive keyframe decision.

--pass 1 Only one pass encoding is used. Can be set to 2 to further improve quality, but takes a long time.

--video-filter "resize:width=1280,height=720" Is used to change the resolution. Can be omitted if the resolution should stay the same as in the source video.

inputvideo.mp4 The source video

B. ffmpeg

Like stated earlier, we also experiment with ffmpeg converter.

FFmpeg is the leading multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play pretty much anything that humans and machines have created. It supports the most obscure ancient formats up to the cutting edge. No matter if they were designed by some standards committee, the community or a corporation. It is also highly portable.
ffmpeg is not a codec – it's a set of codecs H.263, WMV….)including H.264. In simple terms it is used to encode video and pack it to different containers(MP4, MOV, AVI, WMV). It is also more than this. ffmpeg can be seen as a set of libraries allowing you to virtually read any media container using any codec (video or audio).
It is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter. ffmpeg reads from an arbitrary number of input "files" (which can be regular files, pipes, network streams, grabbing devices, etc.), specified by the -i option, and writes to an arbitrary number of output "files", which are specified by a plain output url. Anything found on the command line which cannot be interpreted as an option is considered to be an output url.

All the command line parameters are explained after the code.

ffmpeg -y -i inputfile -c:a libfdk_aac -ac 2 -ab 128k -c:v libx264 -x264opts 'keyint=24:min-keyint=24:no-scenecut' -b:v 1500k -maxrate 1500k -bufsize 1000k -vf "scale=-1:720" outputfile.mp4

-i	Input Source File
keyint	sets the maximum GOP (Group of Pictures)
min-keyint	Minimum GOP Size
no-scenecut	Completely disables the adaptive keyframe decisions
maxrate	Maximum bit-rate allowed
bufsize	Maximum buffer size
outputfile	(re-)encoded outputfile.mp4

C. MP4Box

The multimedia packager available in GPAC is called MP4Box. It can be used for performing many manipulations on multimedia files like AVI, MPG, TS, but mostly on ISO media files (e.g. MP4, 3GP).

In short, MP4Box can be used:

- *for interleaving, fragmenting and segmentation of videos files.*
- for manipulating ISO files like MP4, 3GP: adding, removing, multiplexing audio, video and presentation data (including subtitles) from different sources and in different formats,
- for encoding/decoding presentation languages like MPEG-4 XMT or W3C SVG into/from binary formats like MPEG-4 BIFS or LASeR,
- for performing encryption of streams
- for attaching metadata to individual streams or to the whole ISO file to produce MPEG-21 compliant or hybrid MPEG-4/MPEG-21 files
- *for preparation of HTTP Adaptive Streaming content,*
- and packaging and tagging the result for streaming, download and playback on different devices (e.g. phones, tablets) or for different software (e.g. iTunes).

All the command line parameters are explained after the code.

MP4Box -add intermediate.264 -fps 24 output_2400k.mp4

-add intermediate_2400k.264 The H.264/AVC raw video we want to put in a mp4.

-fps 24 Specifies the frame-rate.

[Note:H.264 doesn't provide meta information about the framerate so it's recommended to specify it. The number (in this example 24 frames per second) must match the framerate used in the x264 command.]

output_2400k.mp4 The output file name.

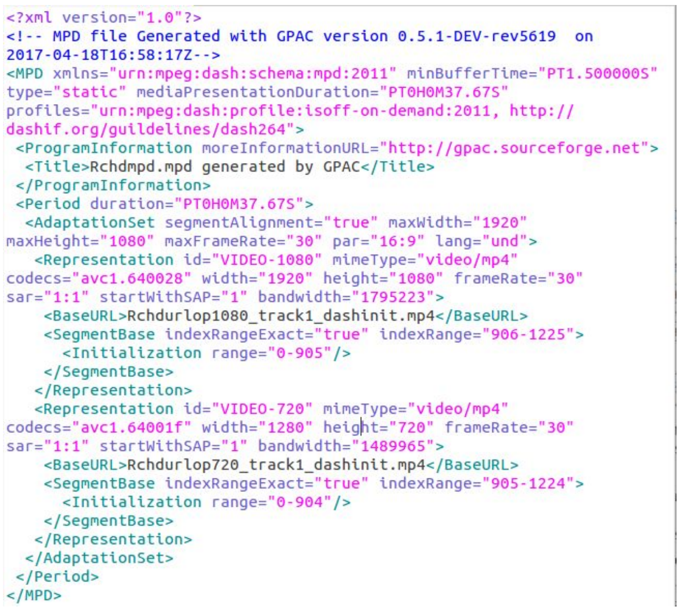
MP4Box -dash 4000 -frag 4000 -rap -segment-name segment_ output_2400k.mp4

-dash 4000 Segments the given file into 4000ms chunks.

-frag 4000 Creates sub-segments within segments and the duration therefore must be longer than the duration given to dash. By setting it to the same value, there will only one subsegment per segment.

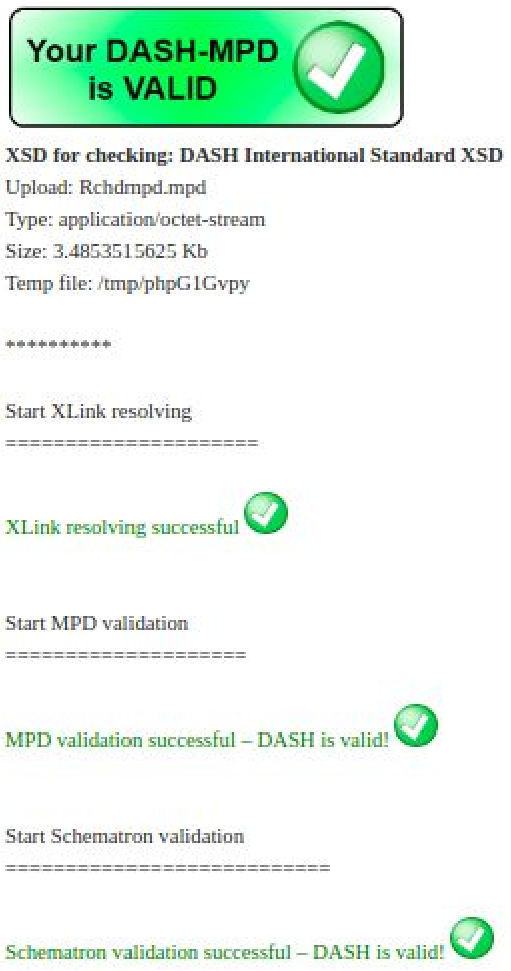
-rap Forces segments to random access points, i.e. keyframes. Segment duration may vary due to where keyframes are in the video–that's why we (re-)encoded the video before with the appropriate settings.

Figure 3 depicts a snippet of our obtained MPD file. We use DASH validator tool [9] [10] to validate the mpd file. Figure 3.1 shows one of the validated result.



```
<?xml version="1.0"?>
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev5619  on
2017-04-18T16:58:17Z-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500000S"
type="static" mediaPresentationDuration="PT0H0M37.67S"
profiles="urn:mpeg:dash:profile:isoff-on-demand:2011, http://
dashif.org/guildelines/dash264">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
  <Title>Rchdmpd.mpd generated by GPAC</Title>
 </ProgramInformation>
 <Period duration="PT0H0M37.67S">
  <AdaptationSet segmentAlignment="true" maxWidth="1920"
maxHeight="1080" maxFrameRate="30" par="16:9" lang="und">
   <Representation id="VIDEO-1080" mimeType="video/mp4"
codecs="avc1.640028" width="1920" height="1080" frameRate="30"
sar="1:1" startWithSAP="1" bandwidth="1795223">
    <BaseURL>Rchdurlop1080_track1_dashinit.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="906-1225">
      <Initialization range="0-905"/>
    </SegmentBase>
   </Representation>
   <Representation id="VIDEO-720" mimeType="video/mp4"
codecs="avc1.64001f" width="1280" height="720" frameRate="30"
sar="1:1" startWithSAP="1" bandwidth="1489965">
    <BaseURL>Rchdurlop720_track1_dashinit.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="905-1224">
      <Initialization range="0-904"/>
    </SegmentBase>
   </Representation>
  </AdaptationSet>
 </Period>
</MPD>
```

fig 3. depicts the snippet of our MPD using MP4 segments



Your DASH-MPD is VALID

XSD for checking: DASH International Standard XSD
Upload: Rchdmpd.mpd
Type: application/octet-stream
Size: 3.4853515625 Kb
Temp file: /tmp/phpG1Gvpy

****

Start XLink resolving
=====================

XLink resolving successful

Start MPD validation
===================

MPD validation successful – DASH is valid!

Start Schematron validation
===========================

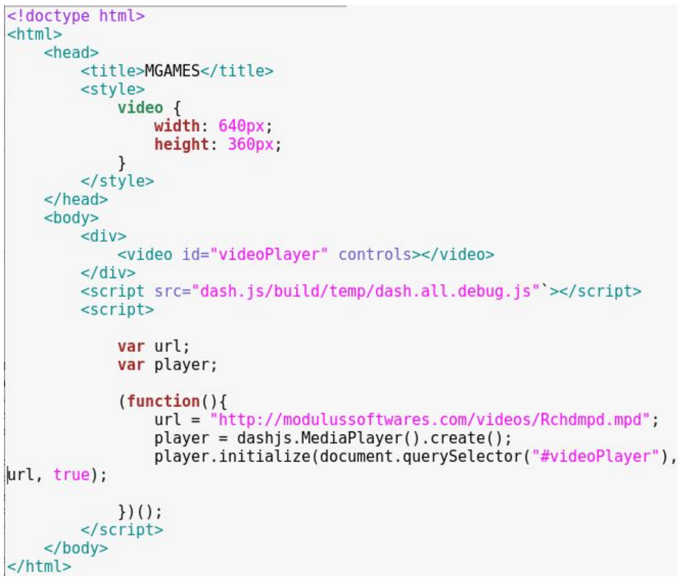
Schematron validation successful – DASH is valid!

fig 3.1 MPD validation

IV. MPEG-DASH FOR WEB

DASH in Javascript

We have a landing-page which has a game-play video. At present it is embedding using HTML5 <video> element. We now describe about integrating MPEG-DASH player into our website to play this video with DASH capabilities.
dash.js is an initiative of the DASH Industry Forum to establish a production quality framework for building video and audio players that play back MPEG-DASH content using client-side JavaScript libraries leveraging the Media Source Extensions API set as defined by the W3C. It provides time based and byte based buffers and has flexible adaptation logics.



```
<!doctype html>
<html>
    <head>
        <title>MGAMES</title>
        <style>
            video {
                width: 640px;
                height: 360px;
            }
        </style>
    </head>
    <body>
        <div>
            <video id="videoPlayer" controls></video>
        </div>
        <script src="dash.js/build/temp/dash.all.debug.js"`></script>
        <script>

            var url;
            var player;

            (function(){
                url = "http://modulussoftwares.com/videos/Rchdmpd.mpd";
                player = dashjs.MediaPlayer().create();
                player.initialize(document.querySelector("#videoPlayer"),
url, true);

            })();
        </script>
    </body>
</html>
```

fig 4. depicts the MPEG-DASH player implementation in HTML body

The above figure shows implementation of MPEG-DASH player using dash.js in our landing page. The implementation method uses javascript to initialize and provide video details to dash.js. The *url* points to our gameplay video in mpd format. The *player* creates a media-player and then initializes it.

V. ANALYSIS AND EVALUATION

For this experiment we implemented a simple reference player for the playback of MPEG-DASH via Javascript to output the current bit-rate, buffer length and dropped frames. We also used dash.js API reference client for obtaining real-time graphs. Our main aim is to test the bandwidth adaptation capabilities of the MPEG-DASH system. Additionally we also try to observe the Buffer Length behavior and deduce initial playout delay in each case.

1. For the evaluation of MPEG-DASH locally, we have used setup comprising a Debian host running an Apache Web Server, a client running the reference player in the Firefox browser and a router over a WiFi network.
2. For the evaluation of MPEG-DASH over the internet, we have used setup comprising an external linux server, where we have hosted the website resources. The client can run the reference player remotely in the Firefox browser over the internet. We test using 3G, 4G LTE and Broadband connections.

We have tested streaming the gameplay video in two scenarios. A single user trying to stream the video and two different users trying to stream the video sharing the same network at the same time.

Test 1:

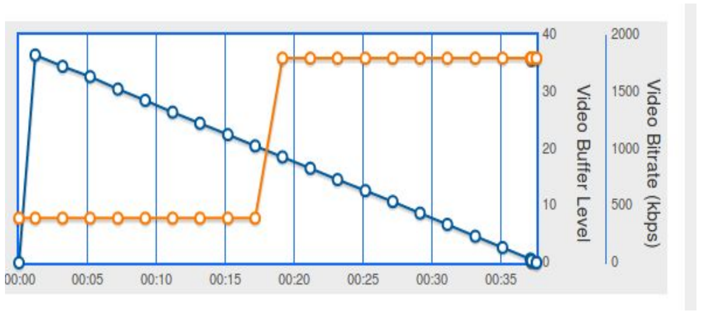


fig 5.1 Single Client, External host, 10 Mbps bandwidth

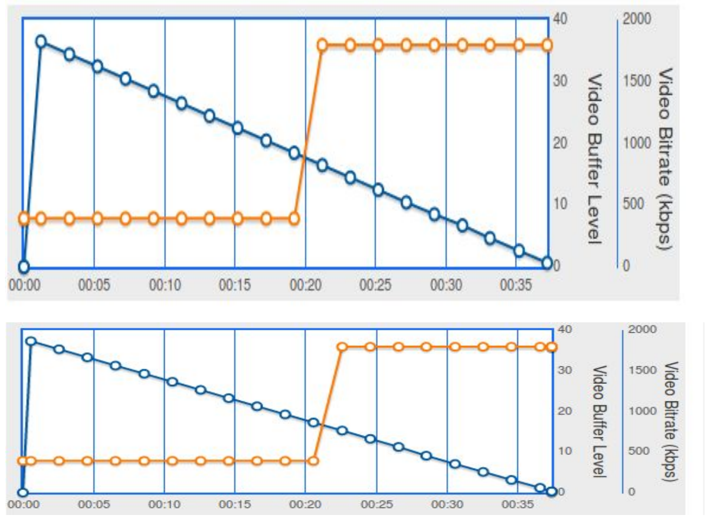


fig 5.2. Concurrent clients, External host, 10Mbps bandwidth

Test 2:

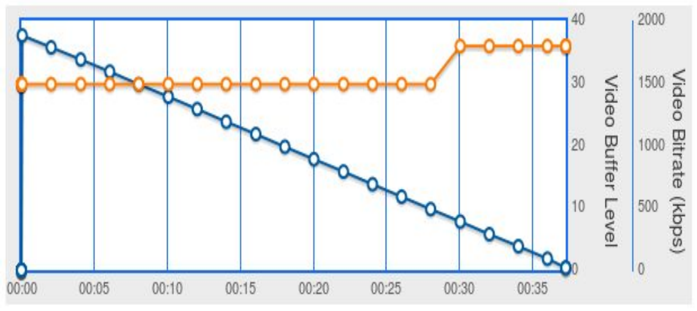


fig 6.1. Single Client, External host, 25 Mbps bandwidth

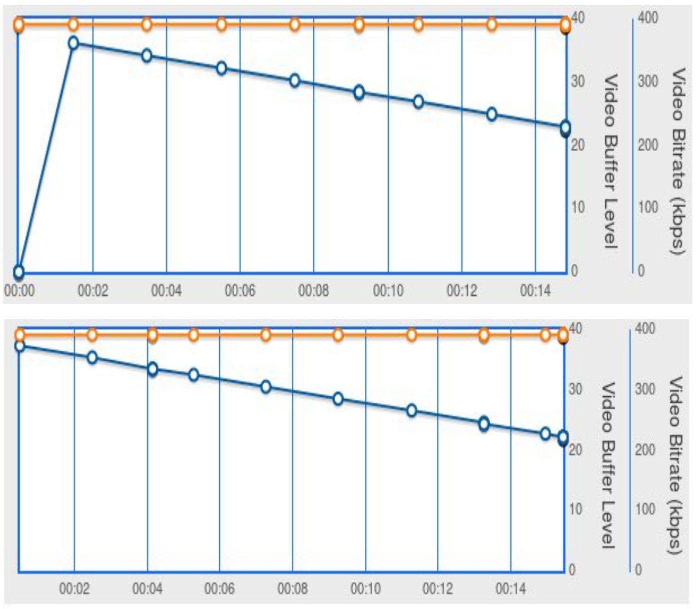


fig 6.2. Concurrent clients, External host, 25 Mbps bandwidth

Test 3:

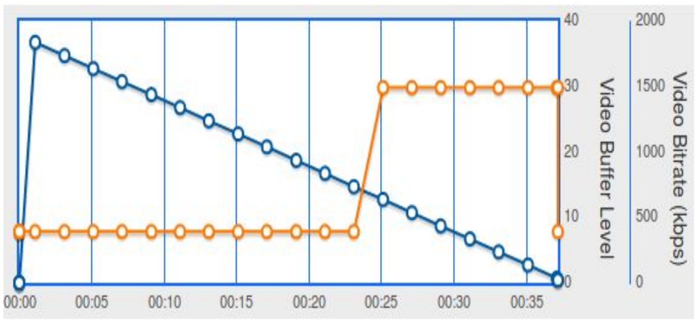
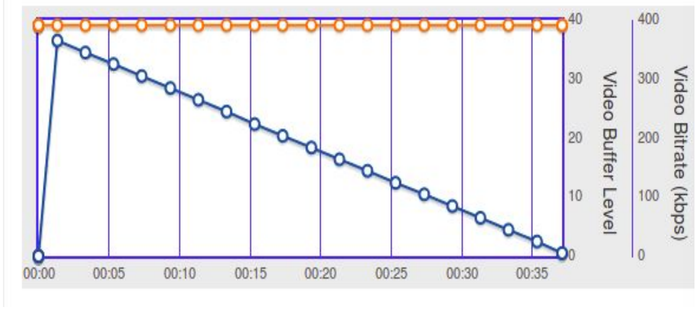


fig 7.1. Single Client, External host, 3G connection



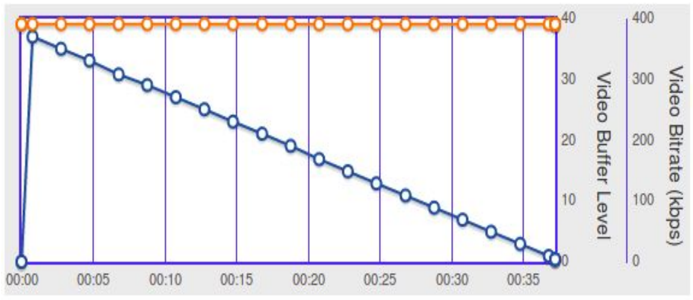


fig 7.2. Concurrent clients, External host, 3G connection

Test 4:

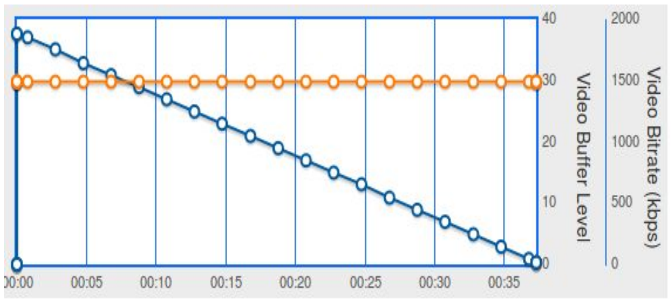


fig 8.1. Single client, External host, 4G LTE connection

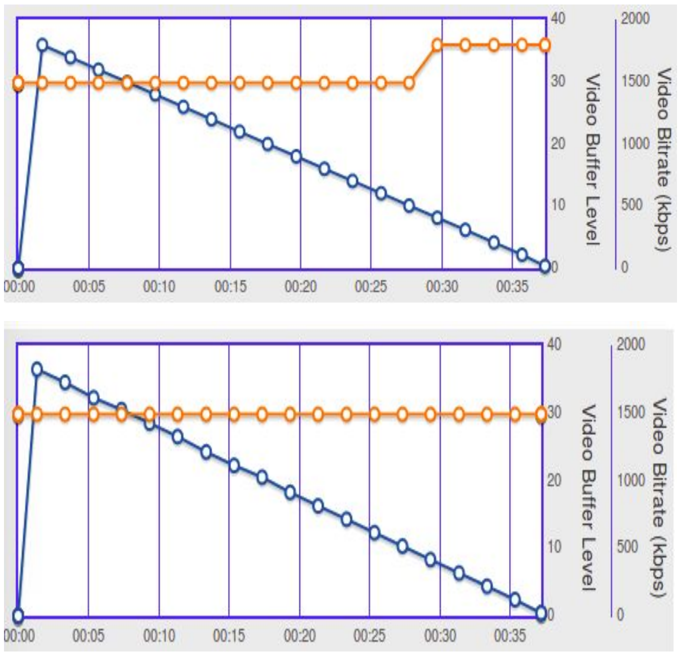


fig 8.2. Concurrent clients, External host, 4G LTE connection

Test 5:

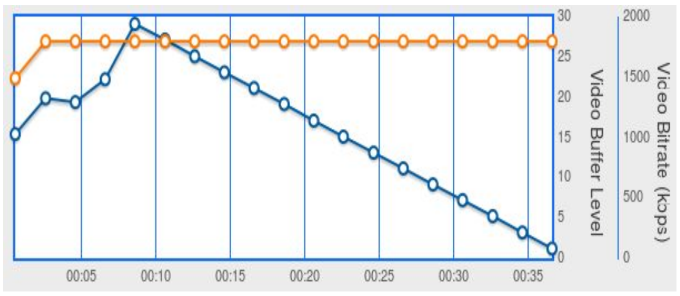
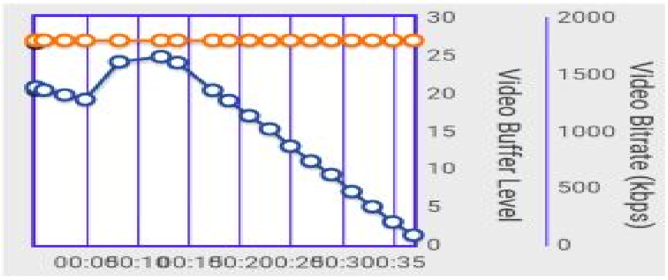


fig 9.1. Single client, Localhost WiFi Network



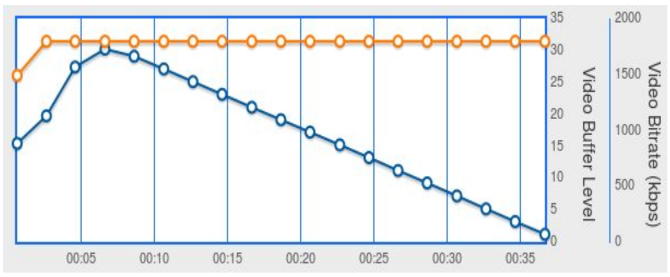


fig 9.2. Concurrent clients, Local host WiFi Network

We observed that the adaptation of the DASH player according to bandwidth performs remarkably well under every network that we tested. Further, it exhibits a stable and fair behavior if multiple clients share a common network path. We used our previously generated MPD which consisted of video segments. It has a duration of approximately 37 seconds. The table in fig 10 depicts the *max-rate* values that we have set for the corresponding video qualities.

Quality	*max bitrate*
360p	400 kbps
480p	650 kbps
720p	1500 kbps
1080p	1800 kbps

fig 10. Video qualities and corresponding max bitrate.

The Buffer Level is 37s (duration of video) from beginning for all the cases hosted on external server.

Fig 5.1 shows that the initial video bitrate is 391 kbps, therefore segments from 360p quality video is played out till 17s, after which the bit-rate steeply increases to 1795 kbps, now segments from 1080p quality video is played out. The initial playout delay is 2s.

Fig 5.2 shows similar results as Fig 5.1. The difference is seen only in terms of time at which the bit-rate steeply increases. For user1 it increases around 19s and the initial playout delay is 2s and User2 it increases around 21s, the initial playout delay is 1s.

Fig 6.1 shows that the initial video bitrate is 1410 kbps and increases after 27s to 1795 kbps. Therefore initially segments from 720p quality video are played followed by segments from 1080p quality video after 27s. The initial playout delay is 0.5s.

Fig 6.2 shows consistent video bitrate of 391 kbps without any change. Therefore segments of 360p video quality video are played out from beginning till the end for both users. For user1 the initial playout delay is 1.7s. For user2 The initial playout delay is 0s.

Fig 7.1. shows the initial bit-rate of 362 kbps, after 27s it shoots up to 1500 kbps and at the very last second it drops down to 391 kbps. Therefore initially segments from 360p quality video are played followed by segments from 720p quality video after 27s and at the very end segments from 360p quality video are played again. The initial playout delay is 1s.

Fig 7.2 shows that user1 gets consistent video bitrate of 382 kbps, The initial playout delay is 1.5s. and user2 gets 391 kbps respectively, the initial playout delay is 1s. Therefore for both users segments from 360p quality video are played out.

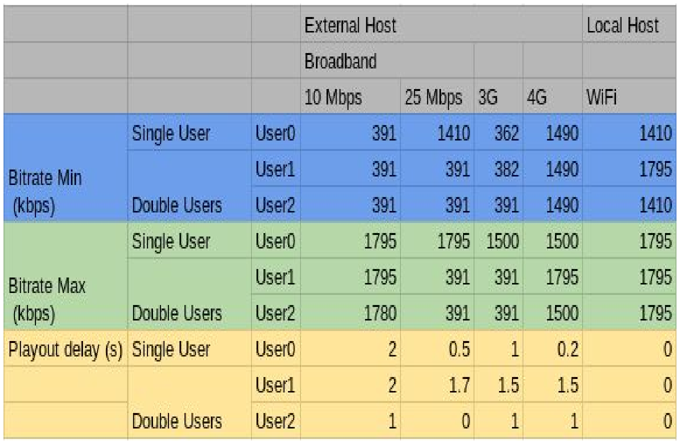
Fig 8.1 shows that the video bit-rate is consistently 1490 kbps. Therefore segments from 720p quality video are played out from beginning till end. The initial playout delay is 0.2s.

Fig 8.2 shows that user1 gets initial bit-rate of 1490 kbps and after 27 s it shoots up to 1795 kbps. Therefore initially segments from 720p quality video are played followed by segments from 1080p quality video after 27s. The initial playout delay is 1.5s. User 2 gets initial video bitrate of 1490 kbps and sees a very small increase by the end getting 1500 kbps. Therefore segments from 720p quality video are played out from beginning till end. The initial playout delay is 1s.

When hosted locally the Buffer Level is not same, it increases and decreases gradually.

Fig 9.1 shows that the initial video bit-rate is 1410 kbps, increases after 3s to 1795 kbps to remain constant till the end. Therefore initially segments from 720p quality video are played followed by segments from 1080p quality video after 3s. The initial Buffer Level is 14s and it increases gradually till 28s for a duration of 8s and then gradually decreases to 0s. There is no initial playout delay.

Fig 9.2 shows that user1 gets constant bit-rate of 1795 kbps.



			External Host				Local Host
			Broadband				
			10 Mbps	25 Mbps	3G	4G	WiFi
Bitrate Min (kbps)	Single User	User0	391	1410	362	1490	1410
		User1	391	391	382	1490	1795
	Double Users	User2	391	391	391	1490	1410
Bitrate Max (kbps)	Single User	User0	1795	1795	1500	1500	1795
		User1	1795	391	391	1795	1795
	Double Users	User2	1780	391	391	1500	1795
Playout delay (s)	Single User	User0	2	0.5	1	0.2	0
		User1	2	1.7	1.5	1.5	0
	Double Users	User2	1	0	1	1	0

fig 11. Summary of analysis results .

Therefore segments from 720p quality video are played out from beginning till end. The initial Buffer Level is 21s and it decreases gradually till 18s for a duration of 3s and then gradually increases to 24s and then gradually decreases to 0s. There is no initial playout delay.

User2 gets initial video bitrate of 1410 kbps, after 3s it increases to 1795 kbps to remain constant till the end. Therefore initially segments from 720p quality video are played followed by segments from 1080p quality video after 3s. The initial Buffer Level is 16s and then gradually increases to 30s and then gradually decreases to 0s. There is no initial playout delay.

VI. CONCLUSION

In this paper we presented an implementation of MPEG-DASH using JavaScript, which exploits the HTML5 video element and the Media Source API provided in browsers. Thus it is possible to provide DASH support for browsers without any further plugin necessary. Furthermore,
we have illustrated how raw video is converted into MP4 format and can be integrated into MPEG-DASH by giving a detailed description of the segment format as well as the corresponding MPD.

As JavaScript is available on a wide range of devices, it enables the integration of DASH-JS, e.g., within mobile devices such as smartphone and tablets.

We also presented an experimental implementation for dynamic adaptation of video quality to the available network bandwidth. We tested it over Broadband, 3G and 4G LTE connections by hosting over an external server and also over a WiFi Network by hosting locally. The implementation aims at analyzing video quality shifts, changing bit-rates and the delay between user's request and the start of the playback. The experiment was evaluated using a prototype of an HTTP streaming client compliant with the standard MPEG-DASH. The evaluation was performed in real-world scenarios and demonstrated stable and fair behavior under every used network conditions. We noticed that there was no play-out delay when we streamed the video over local WiFi network, but there was small (but negligible) playout delay when we streamed it over internet. Future-work can include analysing the results in by restricting the bandwidth resources.

REFERENCES

[1] Computer Networking A Top-down approach by Kurose and Ross.

[2] https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming

[3] http://www.cc.gatech.edu/~dovrolis/Papers/final-saamer-mmsys11.pdf

[4] https://github.com/slederer/DASHEncoder

[5] https://github.com/Dash-Industry-Forum/dash.js

[6] http://www.inase.org/library/2015/zakynthos/bypaper/COMMUN/COMMUN-07.pdf

[7] https://blog.streamroot.io/encode-multi-bitrate-videos-mpeg-dash-mse-based-media-players/

[8] https://bitmovin.com/mp4box-dash-content-generation-x264/

[9] http://dashif.org/conformance.html

[10] http://www-itec.uni-klu.ac.at/dash/?page_id=605