

COMPARISON OF MODERN VIDEO STREAMING PROTOCOLS OVER HTTP:

MPEG-DASH AND HLS : REPORT

PROJECT MEMBERS

- Siddhartha Mishra (ES15BTECH11018)
- Raaghav R. (ES15BTECH11021)

OBJECTIVE

The goal of the project is to implement a **Media server** compatible with both MPEG Dynamic Adaptive Streaming over HTTP (DASH) and HTTP Live streaming(HLS).

PROCEDURE:

The structure for both HLS and DASH is :

For both the methods the common structure for the Media server/client is :

Server :

- Segment the media or streaming buffer to make a playlist using ffmpeg[3] (Handle audio and video stream appropriately).
- Add appropriate signature and track information and append the parsable format as mentioned in RFC [1] for HLS and RFC [2] for DASH.
- Encode and encrypt the segments (END-to-END) and send securely over HTTP.

Client :

- Decrypt the segments upon receiving them (Handle audio and video stream appropriately).
- Parse the receiving segments and use header information to recreate the proper sequence and adapt to the bitrate according to throughput and request different segments to server.
- Decode the sequence and recreate the playlist.
- Read from the buffer and stream it using openCV [4].

Other:

1. Also implement protocol specific logging, adaptive features etc.
2. Evaluate performance metrics to decide which method is better with respect to which parameters.

PARTS WE WORKED ON

Siddhartha Mishra

- DASH : Implementation of the segmentation, encryption and specific features and working prototype as mentioned in RFC.
- Client side : Parsing segment and decoding them to recreate playlists on client side.

Raaghav R.

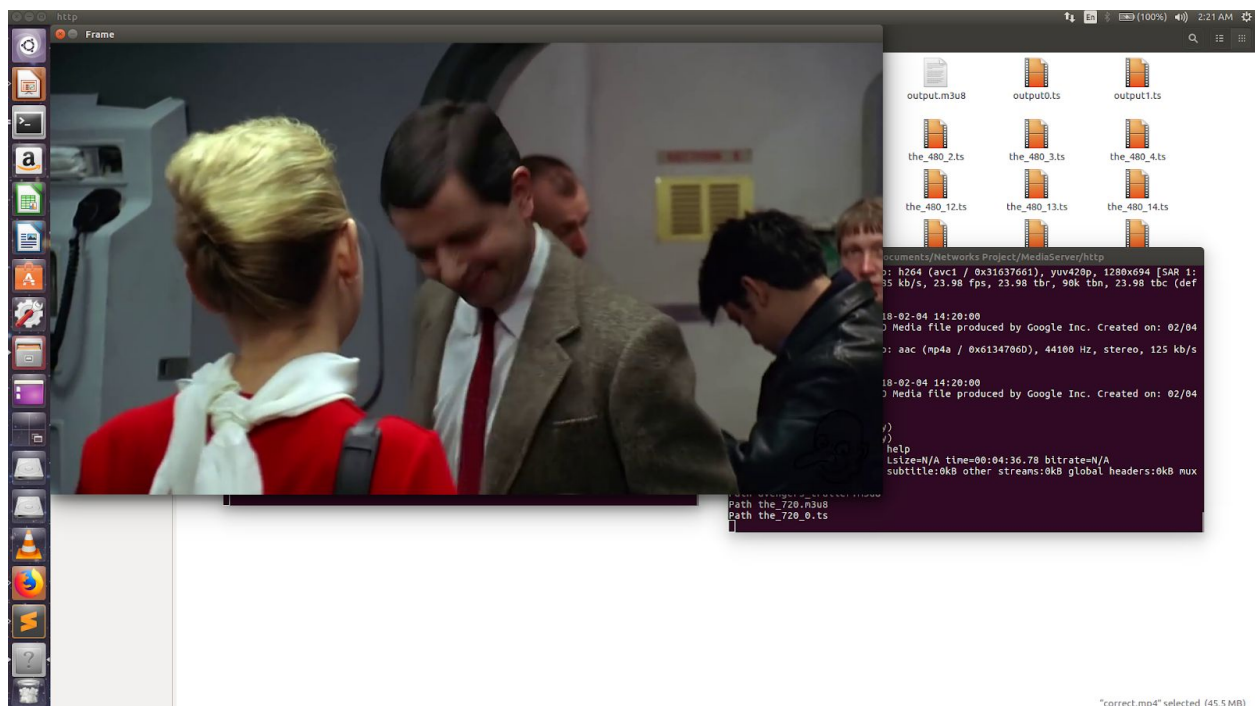
- HLS : Implementation of the segmentation, encryption and specific features and working prototype as mentioned in RFC.
- Client side : Render streaming buffer as received in order on display using OpenCV.

RESULTS:

1.HLS and MPEG server: Generates Playlists/MPDs according and triggers appropriate actions on those segments upon request by client for HLS/MPEG DASH respectively.

2.HLS Video Streaming Client :

Below is the working of our own custom client, that parses through the master playlist, that adapts to the current bitrate, demands the ts files needed from the quality that it can afford and plays it using OpenCV video tools.




Testing correctness of server on existing external HLS client :

This was a famous open source hls client and it our server is able to respond to it's request and adapt to bitrates properly.

Player size: Large (720p)

Current video-resolution: 1280 x 694

Permalink: http://127.0.0.1:8080/index.html?src=http%3A%2F%2Flocalhost%3A6724%2Favengers_trailer.m3u8&enableStreaming=true&autoRecoverError=true&enableWorker=true&dumpMP4=false&levelCapping=1&defaultAudioCodec=undefined&widevineLicenseURL=



manifest successfully loaded, 2 levels found

Playback controls Quality-level controls Audio-track controls Metrics-display Stats-display

Real-time metrics

[toggle sliding fixed window](#)

Window Alt: 2s 5s 10s 30s 60s 120s

Window Zoom: Window Zoom Out Window Zoom In

[Window Slide](#) [Window Slide Vol](#)

[metrics link](#) [metrics permalink](#) [copy metrics to clipboard](#)

play posbuffer
last pos:39016 ms
last buffer:233256 ms
max buffer:272773 ms
nb samples:294

last bitrate:145.01Mbps
min bitrate:55.55Mbps
max bitrate:347.09Mbps
min/max level:1/1
nb level switch:0
average level:1.00

play posbuffer zoomed
[10180,30186]
focus time:19186 ms
focus position:18993 ms
focus buffer:231941 ms

video event
frag changed:2 @ 1
frag changed:3 @ 1

time [duration] 21550 | 32284 |

load event
main fragment 25 @ 1
main fragment 26 @ 1

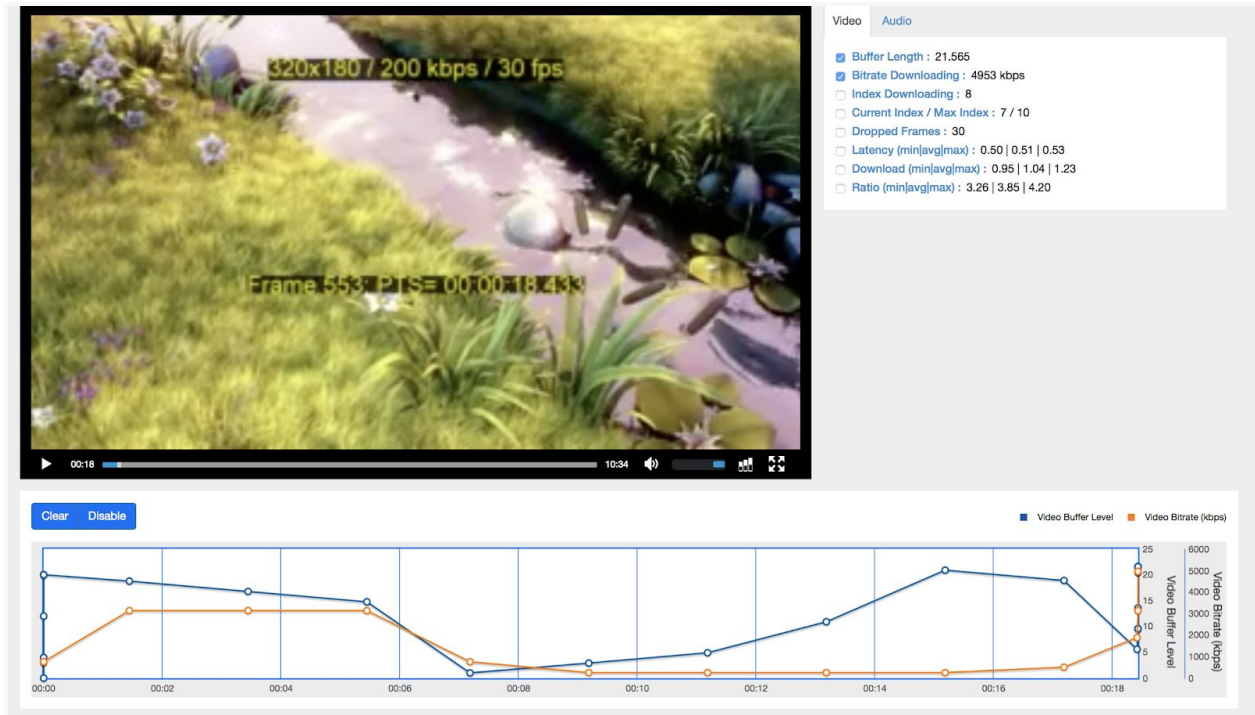
start-end [parsing loading parsing appending] size bitrate
27838 | 27914 [26 8 38 4] 1.4MB 145.1Mbps
38480 | 38562 |

Stats

```
{
  "autoLevelAvg": 1,
  "autoLevelCappingLast": -1,
  "autoLevelCappingMax": -1,
  "autoLevelCappingMin": -1,
  "autoLevelLast": 1,
  "autoLevelMax": 1,
  "autoLevelMin": 1,
  "autoLevelSwitch": 0,
  "fragAvgKbps": null,
  "fragAvgLatency": 6,
  "fragAvgProcess": 47,
  "fragBuffered": 28,
  "fragBufferedBytes": 50030560,
  "fragChangedAuto": 3,
  "fragLastKbps": null,
  "fragLastProcess": 22,
  "fragMaxKbps": null,
  "fragMaxLatency": 24,
  "fragMaxProcess": 116,
  "fragMinKbps": null,
  "fragMinLatency": 2,
  "fragMinProcess": 22,
  "fragLastLatency": 4,
  "fraggarsingKbps": 519798,
  "fraggarsingMs": 770,
  "levelNb": 2,
  "levelParsed": 1,
  "levelParsingUs": 2000,
  "levelStart": 1,
  "tagList": {
    "0": {
      "0": "INF",
      "1": "10.677322"
    }
  }
}
```

Duration:276.82356
Buffered:[0.041711,276.818211]
Seekable:[0,276.82356]
Played:[0,21.170294][45.571,52.963857]
audio Buffered:[0.0417,276.82356]
video Buffered:[0.041711,276.818211]
Dropped Frames:0
Corrupted Frames:0

Testing correctness of server using MPEG DASH external client
STATS(Dash.js) :



LIST OF TOOLS, SOFTWARES AND OTHER REFERENCES:

- [1] <https://tools.ietf.org/html/rfc8216>(RFC for HLS)
- [2] <https://tools.ietf.org/html/rfc6983> (RFC for MPEG-DASH)
- [3] <https://ffmpeg.org> Used to create ffmpeg segments of a continuous stream using a common encoder so that it can be decoded properly on the client side.
- [4] <https://opencv.org> To render video on client side stream from received buffers over network from media server.