Lab 9
Media Streaming over HTTP

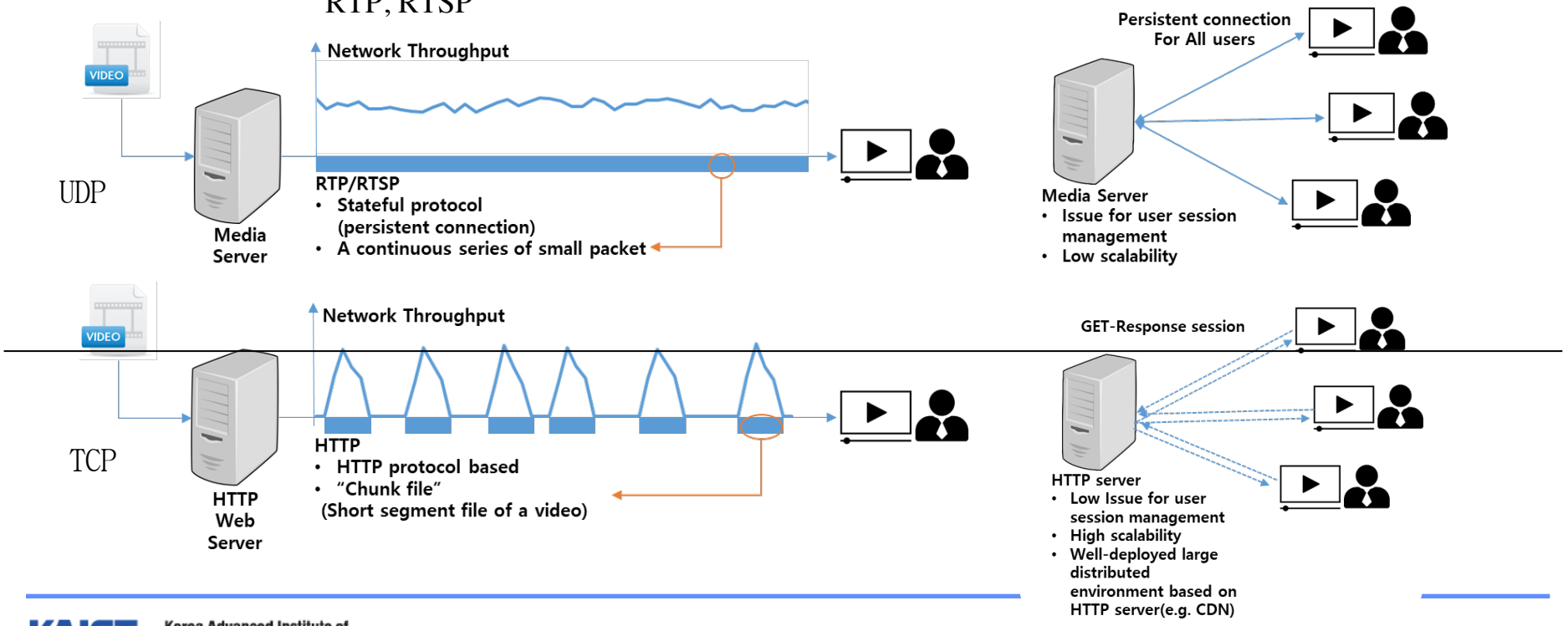
DASH Media Streaming Server

Contents

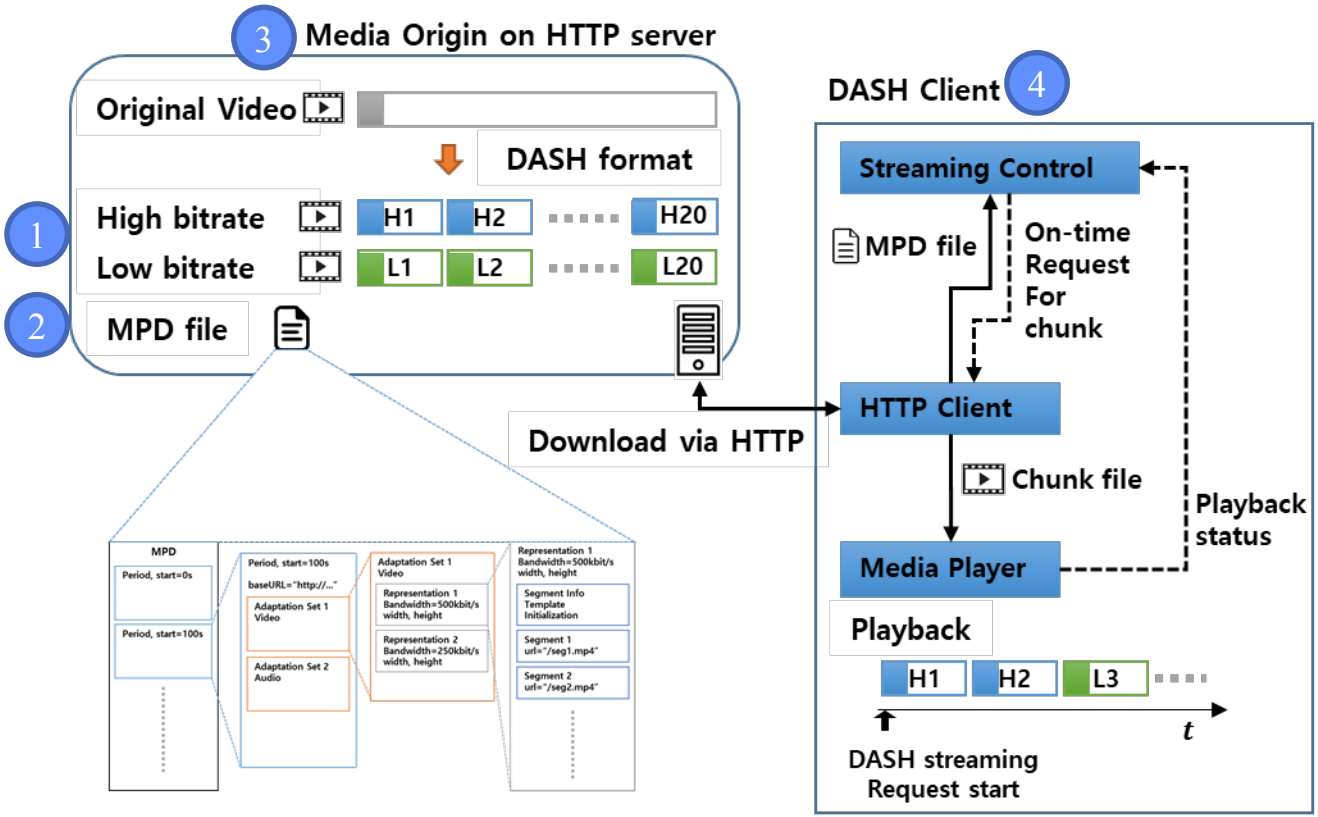
- NodeJS based HTTP server & DASH encoding
- Monitoring of User QoE

Stateless Media Streaming

- Dynamic Adaptive Streaming over HTTP
  - HTTP based stateless streaming protocol
    - Reasonable internet connectivity is nowadays available anywhere, anytime and almost on any device
    - HTTP does not cause any NAT/firewall issues as it is the case with other media transport protocols like RTP(Real-time Transport Protocol), RTSP(Real Time Streaming Protocol)
    - HTTP server does not have issues for management of users' streaming session unlike RTP, RTSP



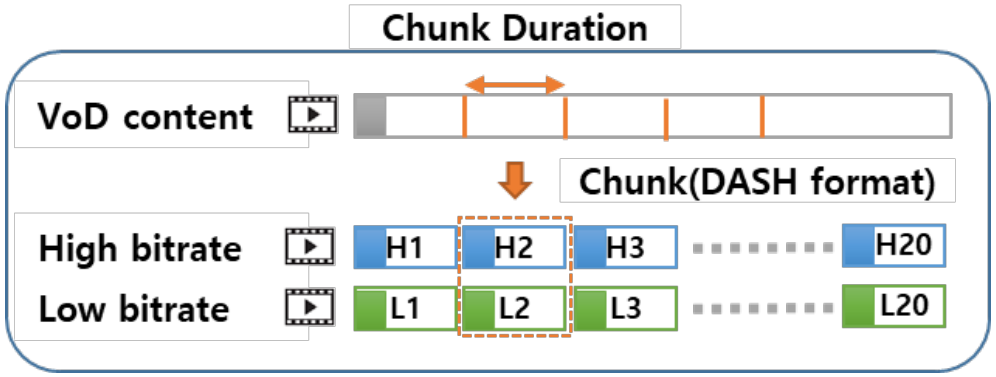
DASH streaming overview



1. **DASH formatted video files:** stored in the HTTP server and delivered via HTTP protocol to the client.
2. **Media Presentation Description:** Manifest file for DASH formatted video files.
3. **HTTP server:** Web-server for response to user streaming request via GET method.
4. **DASH client:** Download the MPD file, the video files and playback.

Chunk (Segmentation) for DASH

- For DASH streaming, a video file is divided into separate files called a **“chunk”** that can be played for a certain period of time.
- Each chunk has playback time units called chunk duration, and can be generated for each bitrate for the same playback interval.



- The user downloads the generated chunks sequentially through the DASH client during streaming procedure.
- Origin_input.mp4 (about 3min)
- segment_1.m4s ~ segment_37.m4s(5sec)

```
-rw-rw-r-- 1 ncl ncl 405838996 Aug  6 20:30 origin_input.mp4
-rw-rw-r-- 1 ncl ncl   1252225 Aug  7 11:28 segment_10.m4s
-rw-rw-r-- 1 ncl ncl   1366897 Aug  7 11:28 segment_11.m4s
-rw-rw-r-- 1 ncl ncl    488460 Aug  7 11:28 segment_12.m4s
-rw-rw-r-- 1 ncl ncl    577773 Aug  7 11:28 segment_13.m4s
-rw-rw-r-- 1 ncl ncl   1057339 Aug  7 11:28 segment_14.m4s
-rw-rw-r-- 1 ncl ncl   1764247 Aug  7 11:28 segment_15.m4s
-rw-rw-r-- 1 ncl ncl   1894627 Aug  7 11:28 segment_16.m4s
-rw-rw-r-- 1 ncl ncl   1307445 Aug  7 11:28 segment_17.m4s
-rw-rw-r-- 1 ncl ncl   2105255 Aug  7 11:28 segment_18.m4s
-rw-rw-r-- 1 ncl ncl   1741779 Aug  7 11:28 segment_19.m4s
-rw-rw-r-- 1 ncl ncl   1806405 Aug  7 11:28 segment_1.m4s
-rw-rw-r-- 1 ncl ncl    497305 Aug  7 11:28 segment_20.m4s
-rw-rw-r-- 1 ncl ncl    604917 Aug  7 11:28 segment_21.m4s
-rw-rw-r-- 1 ncl ncl    526928 Aug  7 11:28 segment_22.m4s
-rw-rw-r-- 1 ncl ncl    676758 Aug  7 11:28 segment_23.m4s
-rw-rw-r-- 1 ncl ncl    801124 Aug  7 11:28 segment_24.m4s
-rw-rw-r-- 1 ncl ncl    606644 Aug  7 11:28 segment_25.m4s
-rw-rw-r-- 1 ncl ncl    800980 Aug  7 11:28 segment_26.m4s
-rw-rw-r-- 1 ncl ncl   1543385 Aug  7 11:28 segment_27.m4s
-rw-rw-r-- 1 ncl ncl   2264417 Aug  7 11:28 segment_28.m4s
-rw-rw-r-- 1 ncl ncl   1310937 Aug  7 11:28 segment_29.m4s
-rw-rw-r-- 1 ncl ncl   1374609 Aug  7 11:28 segment_2.m4s
-rw-rw-r-- 1 ncl ncl   1354868 Aug  7 11:28 segment_30.m4s
-rw-rw-r-- 1 ncl ncl   1271213 Aug  7 11:28 segment_31.m4s
-rw-rw-r-- 1 ncl ncl    516087 Aug  7 11:28 segment_32.m4s
-rw-rw-r-- 1 ncl ncl   1006248 Aug  7 11:28 segment_33.m4s
-rw-rw-r-- 1 ncl ncl    947923 Aug  7 11:28 segment_34.m4s
-rw-rw-r-- 1 ncl ncl    889124 Aug  7 11:28 segment_35.m4s
-rw-rw-r-- 1 ncl ncl    978761 Aug  7 11:28 segment_36.m4s
-rw-rw-r-- 1 ncl ncl    101889 Aug  7 11:28 segment_37.m4s
-rw-rw-r-- 1 ncl ncl   1333865 Aug  7 11:28 segment_3.m4s
-rw-rw-r-- 1 ncl ncl   1488788 Aug  7 11:28 segment_4.m4s
-rw-rw-r-- 1 ncl ncl   1074935 Aug  7 11:28 segment_5.m4s
-rw-rw-r-- 1 ncl ncl    908818 Aug  7 11:28 segment_6.m4s
-rw-rw-r-- 1 ncl ncl   1024016 Aug  7 11:28 segment_7.m4s
-rw-rw-r-- 1 ncl ncl   1451144 Aug  7 11:28 segment_8.m4s
-rw-rw-r-- 1 ncl ncl   2474742 Aug  7 11:28 segment_9.m4s
-rw-rw-r-- 1 ncl ncl       766 Aug  7 11:28 segment_init.mp4
```

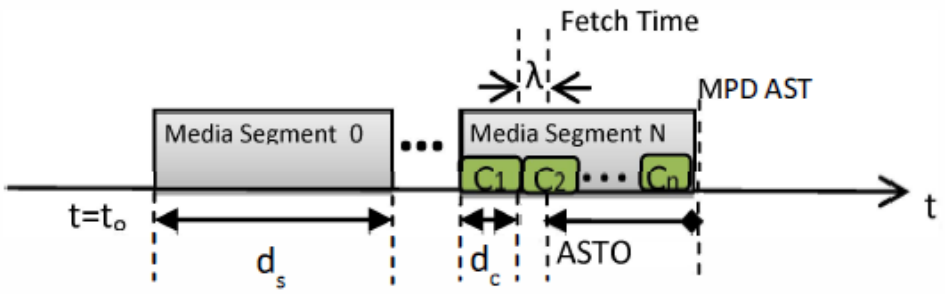
Live DASH Streaming Latency

- Basic DASH Latency
  -Client, MPD(Media Presentation Description), MPD@availabilityStartTime

- Low Latency DASH
  -Segments are further divided in smaller parts and these parts delivered using HTTP chunks
  -availabilityStartTimeOffset (ASTO)

Live DASH Streaming Latency

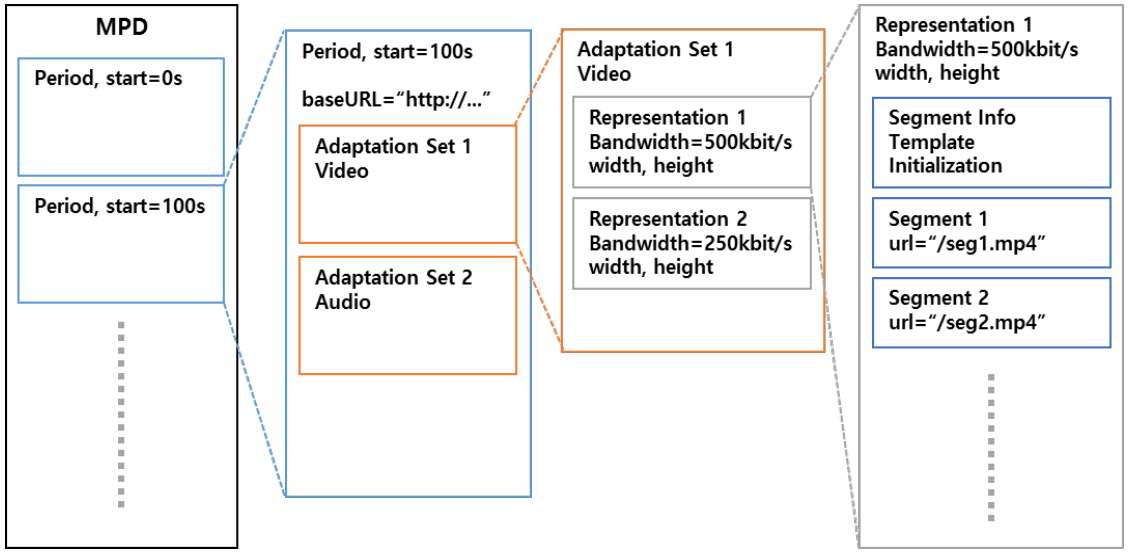
- Segment duration (ds)
- Fragment duration (dc)
- λ represents a margin introduced in the computation of ASTO to cope with UTC mismatch between the client and the server



$$\mathrm{ASTO} = \mathrm{AST} - (\mathrm{AST} - d_s + d_c + \lambda) = d_s - d_c - \lambda$$

Media Presentation Description

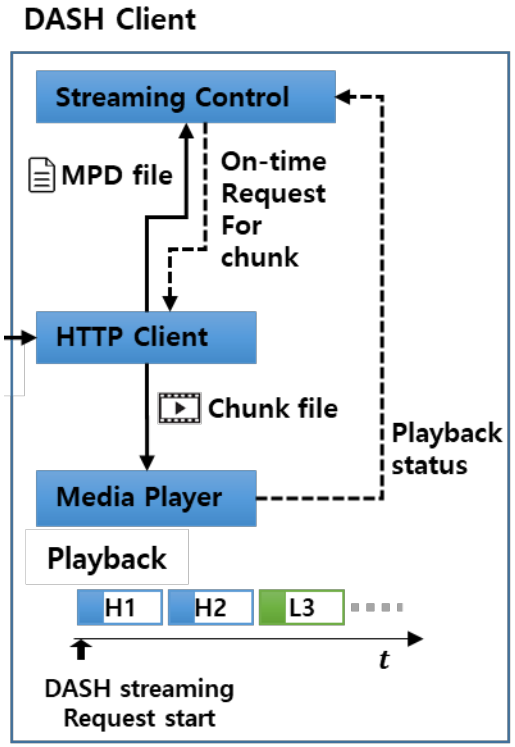
- Media Presentation Description (MPD) is a hierarchical data model that contains meta data for video divided into chunks.



- MPD contains the URL of each chunk so that the appropriate chunk can be downloaded via HTTP GET method at the appropriate time as the client plays the video.
- REF) Stockhammer, Thomas. "Dynamic adaptive streaming over HTTP--: standards and design principles." *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011. : 3.2 Media Presentation

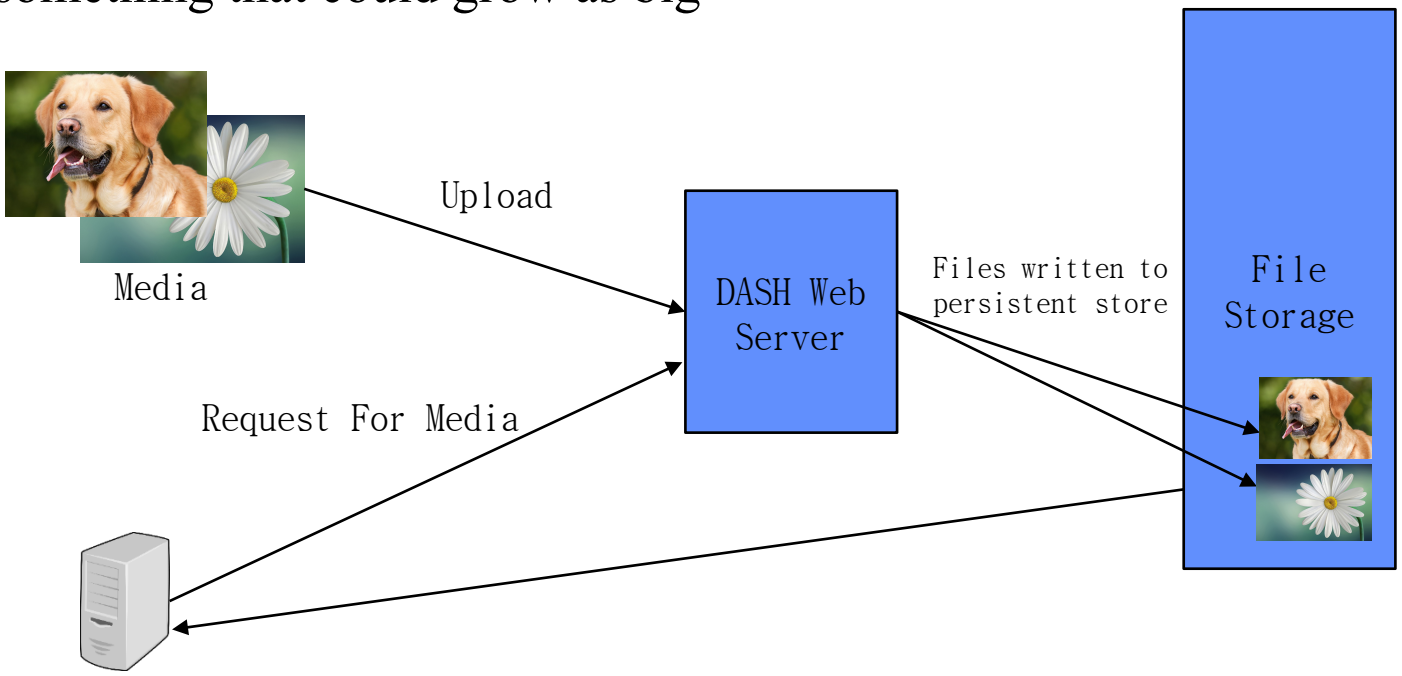
DASH client

- Application for DASH streaming
- Streaming control: Analyzes the MPD file to find the URL and time information for the chunk file that makes up the video which the user wants to watch. Based on MPD, the streaming control generates an HTTP request for the required chunk while playing the video.
- HTTP Client: It is responsible for downloading the MPD and chunk files from the HTTP server. It requests chunk files specified by "Streaming Control" through the HTTP GET method, and forwards the downloaded chunk file to the player.
- Media Player: A player that plays chunks in DASH format. It stores the downloaded chunk in the buffer(gray bar), plays it back and allows the user to watch the video, and sends the playback status, such as chunk time remaining in the buffer, to the streaming control.



Example: Media Streaming Application

- In this Media Streaming example, the system must be perceivably fast, its data stored reliably and all of these attributes highly scalable. Building a small version of this application would be trivial and easily hosted on a single server. Let's assume that we want to build something that could grow as big



DASH Implementation with Node.js server

NodeJS based HTTP server for DASH

- NodeJS HTTP server in Linux Environment
  - NodeJS(version 6.x) installation (http://nodejs.org/en/)

```
$ curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
$ sudo apt-get install -y nodejs
```

  - NPM installation (http://docs.npmjs.com/getting-started/what-is-npm)

```
$ sudo apt-get install npm
```

  - Project Directory & initialize project

NodeJSProject	- Project directory
└ app.js	- Entry point for HTTP server code
└ public	- Content directory

```
ncl@e3-2-cluster6:~/NodeJSProject$ ls -l
total 4
-rw-rw-r-- 1 ncl ncl    0 Aug 10 23:04 app.js
drwxrwxr-x 2 ncl ncl 4096 Aug 10 23:04 public
```

```
$ scp -r ncl@143.248.152.206:/home/ncl/NodeJSProject ~/
```

NodeJS based HTTP server for DASH

- NodeJS HTTP server in Linux Environment
  - Project Directory & initialize project(cont.)

```
$ sudo npm init
```

1

```
ncl@e3-2-cluster6:~/NodeJSProject$ sudo npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
```

2

```
name: (NodeJSProject) nodejsproject
version: (1.0.0)
description:
entry point: (app.js)
test command: nodejs app.js
git repository:
keywords:
author:
license: (ISC)
About to write to /home/ncl/NodeJSProject/package.json:

{
  "name": "nodejsproject",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "nodejs app.js"
  },
  "author": "",
  "license": "ISC"
}


Is this ok? (yes) yes
```

3

```
ncl@e3-2-cluster6:~/NodeJSProject$ ls
app.js  package.json  public
```

- package.json
  - Description for the project
  - Dependency Library for the project

NodeJS based HTTP server for DASH

- NodeJS HTTP server in Linux Environment
  - Project Directory & initialize project(cont.)

```
$ sudo npm install express --save
```

```
ncl@e3-2-cluster6:~/NodeJSProject$ ls
app.js  node_modules  package.json  public
```

```
$ cat package.json
```

```
ncl@e3-2-cluster6:~/NodeJSProject$ cat package.json
{
  "name": "nodejsproject",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "nodejs app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.15.4"
  }
}
```

NodeJS based HTTP server for DASH

- NodeJS HTTP server in Linux Environment
  - app.js

```
var express = require('express'),
    app = express(),
    path = require('path'),
    fs = require('fs');

app.use(express.static(path.join(__dirname, 'public')));

//HTTP server Listen with port 55555
var server = app.listen(55555, function(){
    console.log('This server is running on the port ' + this.address().port );
});

//GET "Hello" handle
app.get('/Hello', function(req, res){
    console.log('GET Hello from a client');
    res.send('Hello, world!');
    res.end();
});
```

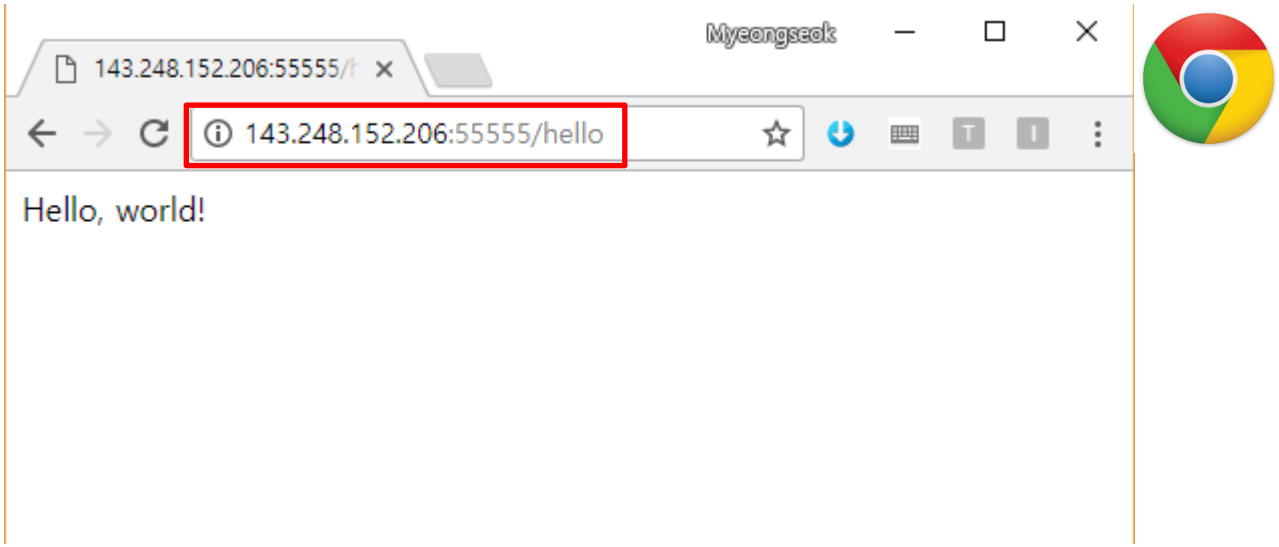
NodeJS based HTTP server for DASH

- NodeJS HTTP server in Linux Environment
  - app.js

```
$ nodejs app.js
```

```
ncl@e3-2-cluster6:~/suki/DASHstudy$ nodejs app.js
This server is running on the port 55555
```

  - Chrome Browser




```
ncl@e3-2-cluster6:~/suki/DASHstudy$ nodejs app.js
This server is running on the port 55555
GET Hello from a client
```

NodeJS based HTTP server for DASH

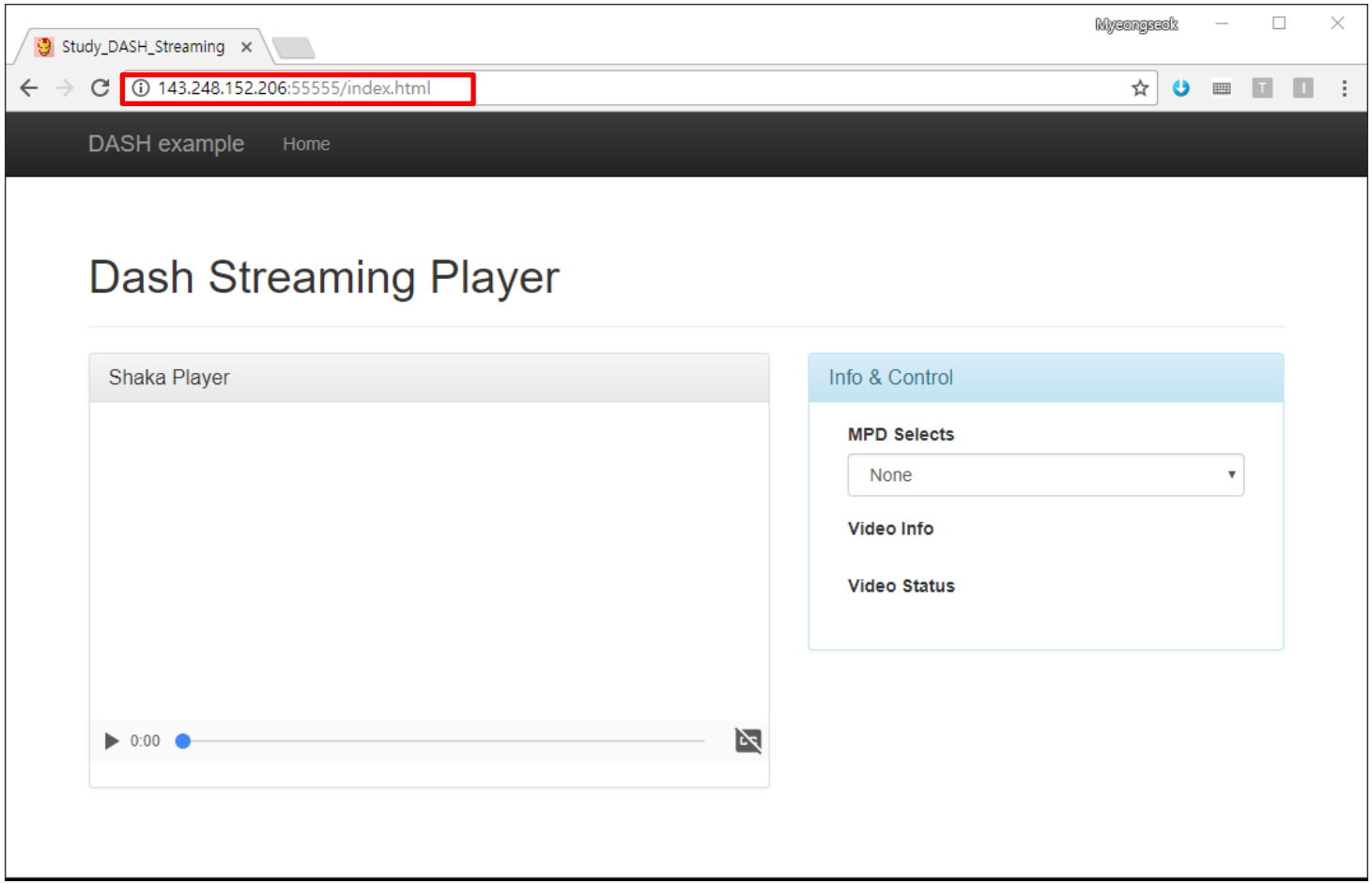
- NodeJS HTTP server in Linux Environment
  - Client Web page source code

NodeJSProject	- Project directory
└ app.js	- Entry point for HTTP server code
└ public	- Content directory
└ css	- css code directory
└ js	- JavaScript code directory
└ lib	- Library code directory
└ favicon	- favicon directory
└ vod	- VoD Content directory
└ live	- Live Content directory
└ index.html	- Index page: HTML code

```
ncl@e3-2-cluster6:~/NodeJSProject/public$ ls
css  favicon  index.html  js  lib  live  vod
```

NodeJS based HTTP server for DASH

- NodeJS HTTP server in Linux Environment
  - Chrome Browser

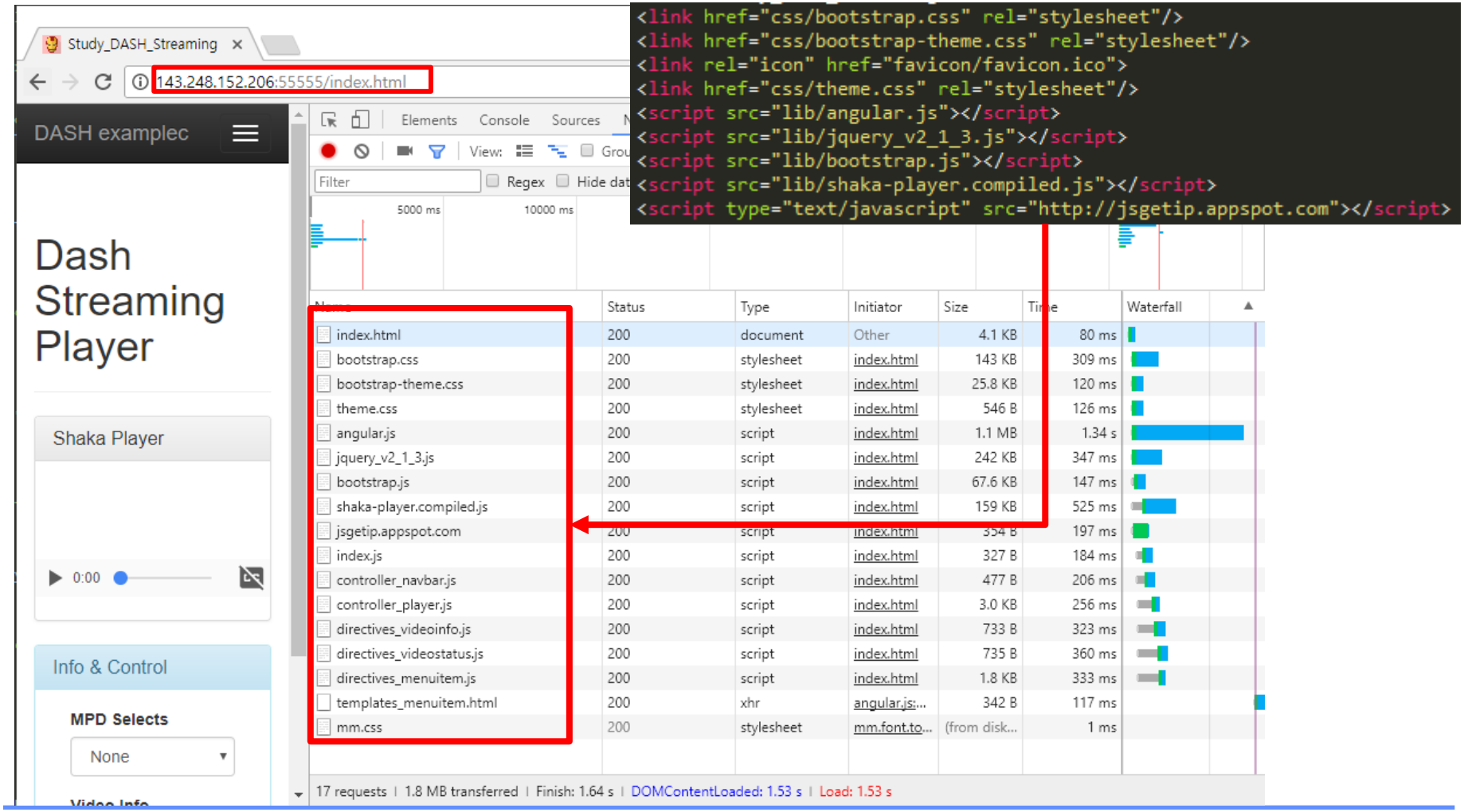


NodeJS based HTTP server for DASH

- NodeJS HTTP server in Linux Environment
  - Chrome Browser(Development Tool – Network tap)

In index.html

```
<link href="css/bootstrap.css" rel="stylesheet"/>
<link href="css/bootstrap-theme.css" rel="stylesheet"/>
<link rel="icon" href="favicon/favicon.ico">
<link href="css/theme.css" rel="stylesheet"/>
<script src="lib/angular.js"></script>
<script src="lib/jquery_v2_1_3.js"></script>
<script src="lib/bootstrap.js"></script>
<script src="lib/shaka-player.compiled.js"></script>
<script type="text/javascript" src="http://jsgetip.appspot.com"></script>
```



Media Encoding

- DASH encoding
  - Encoding video for DASH format 2)

```
$ cd ~/NodeJSProject/public/vod
$ ffmpeg -i origin_input.mp4 -vcodec libx264 -s 1280x720 -b:v 2000k -keyint_min 150 -an
full_output_1280x720_2000k.ts
$ MP4Box -add full_output_1280x720_2000k.ts full_output_1280x720_2000k.mp4
$ MP4Box -dash 5000 -segment-name ./segment_ full_output_1280x720_2000k.mp4
```

```
ncl@e3-2-cluster6:~/NodeJSProject/public/vod$ ls
full_output_1280x720_2000k_dash.mpd  segment_13.m4s  segment_1.m4s   segment_26.m4s  segment_32.m4s  segment_4.m4s
full_output_1280x720_2000k.mp4       segment_14.m4s  segment_20.m4s  segment_27.m4s  segment_33.m4s  segment_5.m4s
full_output_1280x720_2000k.ts        segment_15.m4s  segment_21.m4s  segment_28.m4s  segment_34.m4s  segment_6.m4s
origin_input.mp4                     segment_16.m4s  segment_22.m4s  segment_29.m4s  segment_35.m4s  segment_7.m4s
segment_10.m4s                       segment_17.m4s  segment_23.m4s  segment_2.m4s   segment_36.m4s  segment_8.m4s
segment_11.m4s                       segment_18.m4s  segment_24.m4s  segment_30.m4s  segment_37.m4s  segment_9.m4s
segment_12.m4s                       segment_19.m4s  segment_25.m4s  segment_31.m4s  segment_3.m4s   segment_init.mp4
```

  - -vcodec: output video codec
  - -s: video resolution
  - -b:v: output video bitrate
  - -dash: chunk duration

DASH Client

- Google Shaka Player
  - Shaka Player is a JavaScript library for adaptive video streaming. It plays DASH content without browser plugins using MediaSource Extensions and Encrypted Media Extensions.
  - REF) http://shaka-player-demo.appspot.com/docs/api/index.html
  - ❖ It is already stored in the provided code.

└ public	- Content directory
└ lib	- lib
└ shaka-player.compiled.js	- Shaka-player code

- ❖ In index.html @line 49: HTML video element

```
<video id="video" width="100%" controls autoplay muted></video>
```

DASH Client

- Google Shaka Player initialize
  - In js/controllers/controller_player.js

```
window.addEventListener('load', startup);
function startup(){
   initApp();
}
```

```
function initApp(){
   shaka.polyfill.installAll();
   if(shaka.Player.isBrowserSupported()){
      var video = document.getElementById('video');
      var player = new shaka.Player(video);
      window.player = player;
      player.addEventListener('error', onErrorEvent);
   }
   else{
      console.error('Browser not supported!');
   }
}
```

DASH Client

- Google Shaka Player Start
  - In index.html


```
<select class="form-control" id="mpdSelect" ng-change="onChangeMPD()" ng-model="manifestUrl">
    <option ng-repeat="option in availableMpds" value="{{option.id}}">{{option.name}}</option>
</select>
```

When select option changed

- In js/controllers/controllers_player.js @line 10

```
$scope.onChangeMPD = function(){
    console.log("onChangeMPD : " + $scope.manifestUrl);
    startView();
}
```

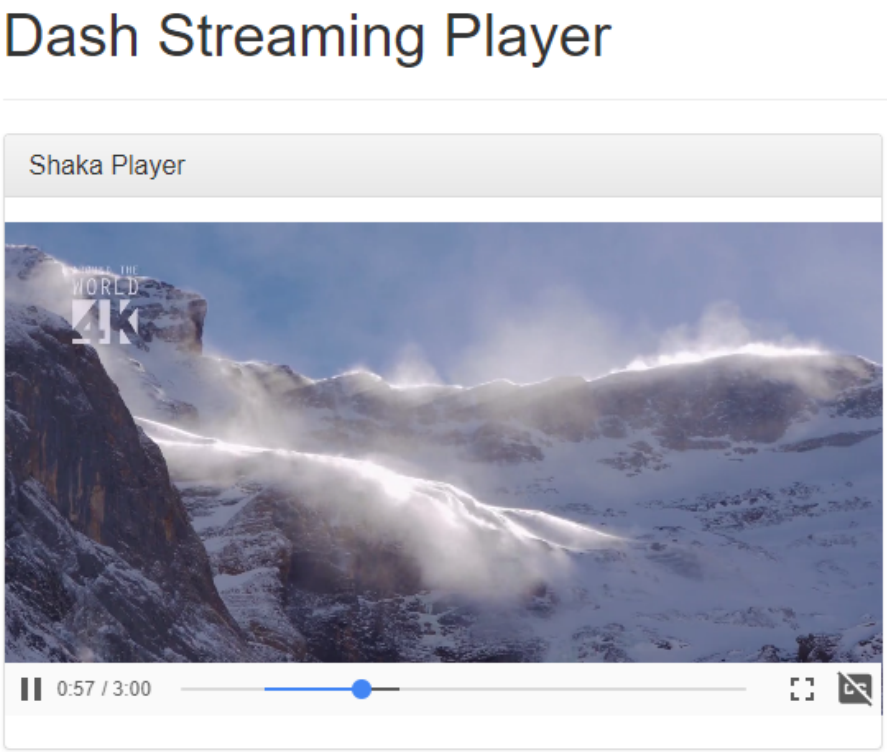
```
$scope.availableMpds = [
    {id: '/None', name: "None"},
    {id: '/vod/full_output_1280x720_2000k_dash.mpd', name: "Video"},
    {id: '/live/live_dash.mpd', name: "Live"},

];
```

- In js/controllers/controllers_player.js @line 87

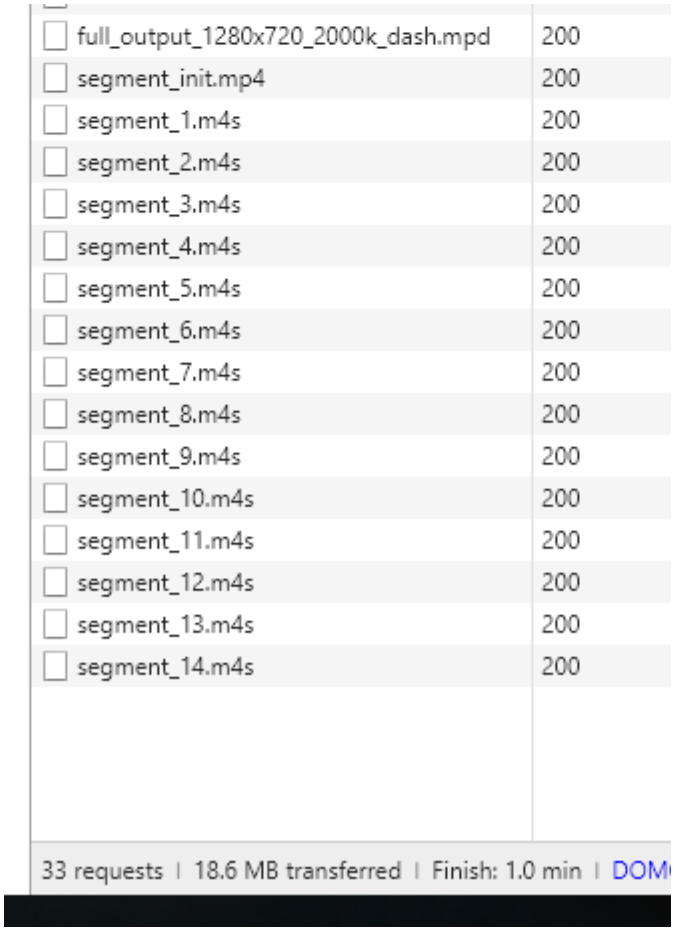
```
window.player.load("http://143.248.152.206:55555" + $scope.manifestUrl);
//GET "/video/full_output_1280x720_2000k_dash.mpd"
```

DASH Client

- Google Shaka Player Start
  - VoD streaming

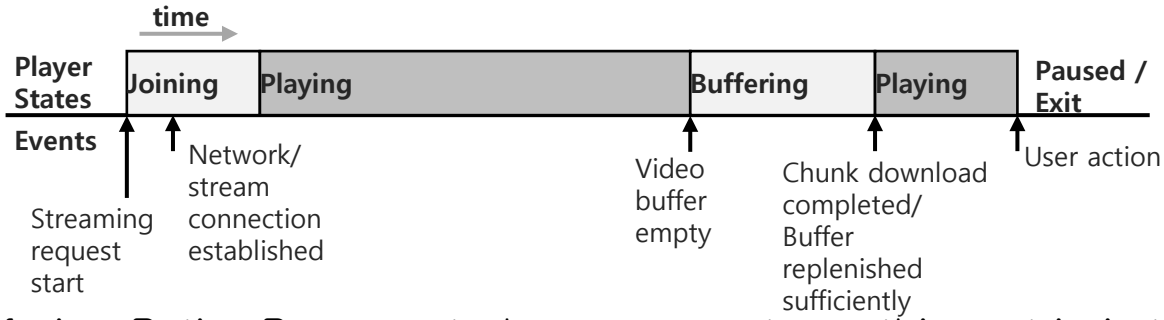


- Chrome Development Tool - Network



Lab)Streaming QoE

- QoE: Quality of Experience
  - Ref)Dobrian, Florin, et al. "Understanding the impact of video quality on user engagement." ACM SIGCOMM Computer Communication Review. Vol. 41. No. 4. ACM, 2011



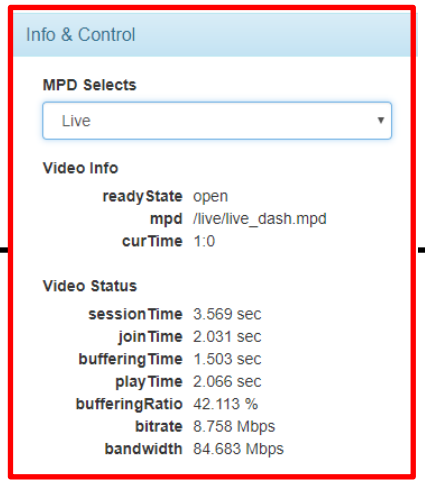
  - Buffering Ratio: Represented as a percentage, this metric is the fraction of the total session time (i.e., playing plus buffering time) spent in buffering. This is an aggregate metric that can capture periods of long video "freeze" observed by the user. As illustrated in above figure, the player goes into a buffering state when the video buffer becomes empty and moves out of buffering (back to playing state) when the buffer is replenished.

$$\text{video sesstion time} = \text{playing time} + \text{buffering time}$$

$$\text{buffering ratio} = \frac{buffering\ time}{video\ sesstion\ time} \times 100(\%)$$

Lab)Streaming QoE

- QoE: Quality of Experience
  - In js/controllers/controllers_player.js @line 20 ~ line 53




```
setInterval(updateStatus, 1000);
            ...
   var videoStats = window.player.getStats();

            ...


      $scope.videoStatus.sessionTime = (videoStats.bufferingTime + videoStats.playTime).toFixed(fixval) +
" sec";
      $scope.videoStatus.joinTime = (videoStats.loadLatency).toFixed(fixval) + " sec";
      $scope.videoStatus.bufferingTime = (videoStats.bufferingTime).toFixed(fixval) + " sec";
      $scope.videoStatus.playTime = (videoStats.playTime).toFixed(fixval) + " sec";
      $scope.videoStatus.bufferingRatio = ((videoStats.bufferingTime) / (videoStats.bufferingTime +
videoStats.playTime) * 100).toFixed(fixval) + " %";
      $scope.videoStatus.bitrate = (videoStats.streamBandwidth / 1024 / 1024).toFixed(fixval) + " Mbps";
      $scope.videoStatus.bandwidth = (videoStats.estimatedBandwidth / 1024 / 1024).toFixed(fixval) + "
Mbps";
```

Lab 9. Build up DASH Server and monitoring service quality

- Read text materials and test practices
- Use your notebook PC to examine context and produce the source code when you finish the successful practice.
- Objectives
  - Understand media streaming mechanism and it's service quality control
  - Build up DASH media server
  - Monitor service quality