

Media Processing

ffmpeg

ffmpeg

- We sometimes want to process video files
- ffmpeg is *the* answer for video manipulation
- Need to install ffmpeg
 - Include the installation in your Dockerfile if using Docker
- Invoke ffmpeg
 - Command line examples today
 - Make system calls from your code
 - -or- use a client library that makes the sys calls for you
(You still need to install ffmpeg)

ffmpeg

```
ffmpeg -i inputVideo.avi -f mp4 outputVideo.mp4
```

- Example of basic ffmpeg usage
 - Converts inputVideo.avi into an mp4
- The -i flag indicates the input filename
- The -f flag indicates the output format
- The last argument is always the output filename
 - No flag for the output filename

ffmpeg

```
ffmpeg -i inputVideo.avi -s 640x360 -f mp4 outputVideo.mp4
```

- We can add more arguments for more control
 - Output filename is still the last argument
- The -s flag is sets the resolution of the output file
 - We convert the file to 640x360

The Problem

- You host a video on your web app
- You want high quality so you host a large 1080p mp4
 - The entire file is 100's of MB
- Every user visiting your page has to download the entire video before playback can begin
 - Very slow to load
 - Entire file must download even for users who will only watch for a few seconds

Chunking

- Avoid the requirement of downloading the entire video before it plays
- Provided a way to request short segments of the video
- Download one "chunk" of the video at a time
 - Typically ~2-10 seconds of playback
- Advantages:
 - Only a few seconds need to be downloaded before playback starts
 - If the user skips around in the video, only request the chunk they skipped to
 - If the user leaves the page without finishing the video, the entire file doesn't have to be downloaded

Chunking - Range

- Even mp4 videos can be chunked
- The browser may end a request for a large file that includes a Range header
 - The Range header specifies a range of bytes being requested
 - eg. A request for `"/video.mp4"` with a header `"Range: 10000-20000"` is request the 9999th through 19999th bytes of video.mp4
- The response code should be 206 Partial content
- The response should contain a Content-Range header
 - Contains the range of bytes being sent and the to total size of the file
 - eg. `Content-Range: 10000-20000/500000`
 - eg. `Content-Range: 0-499999/500000` if the entire file is sent

Chunking - Range

- Using the Range header can be effective in some cases, however:
 - It adds extra complexity to the server
 - It relies on the browser to request useful ranges
 - Some browsers might not implement Range and ask for the entire file
- We would like a more robust solution

Chunking - Multiple Files

- Instead of relying on the browser asking for a specific range of bytes..
- We host a single video in multiple files
- Each file contains a few seconds of playback
- The browser requests each segment as needed
 - User only watches a few seconds -> Only need to request the first few segments
 - User skips to the middle of the video -> Request the middle segment

HLS vs MPEG-DASH

- Two major protocols support the idea of breaking a video into smaller segments/chunks: HLS and MPEG-DASH
- **HTTP Live Streaming (HLS)**
 - Developed by Apple
 - Only supports the H.264 encoding for video
 - Wide-spread adaptation
 - Spec freely available in RFC8216
- **Dynamic Adaptive Streaming over HTTP (MPEG-DASH)**
 - Developed by Moving Picture Experts Group (MPEG)
 - Supports any video encodings
 - No support on Apple devices
 - Spec published as ISO/IEC 23009-1:2022 - Available for \$245 (!)

HLS

- Divide the video into multiple .ts files
 - MPEG Transport Stream files
- One .m3u8 index file containing information about each .ts file and how they combine into a single video
- Your server hosts all files
- Set the video source as the index file
- Browser reads the index file to know when to request each ts file

 space.m3u8
 space0.ts
 space1.ts
 space2.ts
 space3.ts
 space4.ts
 space5.ts
 space6.ts
 space7.ts
 space8.ts
 space9.ts
 space10.ts
 space11.ts
 space12.ts
 space13.ts

```
1 #EXTM3U
2 #EXT-X-VERSION:3
3 #EXT-X-TARGETDURATION:8
4 #EXT-X-MEDIA-SEQUENCE:0
5 #EXTINF:6.773433,
6 space0.ts
7 #EXTINF:8.341667,
8 space1.ts
9 #EXTINF:8.341667,
10 space2.ts
11 #EXTINF:8.341667,
12 space3.ts
13 #EXTINF:8.341667,
14 space4.ts
15 #EXTINF:8.341667,
16 space5.ts
17 #EXTINF:8.341667,
18 space6.ts
19 #EXTINF:8.341667,
20 space7.ts
21 #EXTINF:8.341667,
22 space8.ts
23 #EXTINF:8.341667,
24 space9.ts
25 #EXTINF:8.341667,
26 space10.ts
27 #EXTINF:8.341667,
28 space11.ts
29 #EXTINF:6.973633,
30 space12.ts
31 #EXTINF:2.836167,
32 space13.ts
33 #EXT-X-ENDLIST
```


HLS - Transcoding

```
ffmpeg -i space.mp4 -hls_list_size 0 -f hls space.m3u8
```

- Use ffmpeg to convert to HLS
- "-f hls" to specify the output format as HLS
- "-hls_list_size 0" to keep all ts files in the index
 - By default, ffmpeg will only keep the last 5 ts files in the index file
 - This is good if you are live-streaming (This is the HTTP Live Streaming protocol after all)
 - Since our use case is hosting Video on Demand (VOD), we want to keep every ts file in the index
 - Setting the list size to 0 means the size is not limited

MPEG-DASH

- Divide the video into multiple files that can use a variety of formats (m4s by default in ffmpeg output which is mp4 encoded)
- One .mpd (Media Presentation Description) index file containing tons of information in an XML format
- Hosts all files and set the video source as the index file
- Browser reads the index file to find an init file and naming convention for content files to request

chunk-stream0-00001.m4s
chunk-stream0-00002.m4s
chunk-stream0-00003.m4s
chunk-stream0-00004.m4s
chunk-stream0-00005.m4s
chunk-stream0-00006.m4s
chunk-stream0-00007.m4s
chunk-stream0-00008.m4s
chunk-stream0-00009.m4s
chunk-stream0-00010.m4s
chunk-stream0-00011.m4s
chunk-stream0-00012.m4s
chunk-stream0-00013.m4s
chunk-stream0-00014.m4s
init-stream0.m4s
space.mpd

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="urn:mpeg:dash:schema:mpd:2011"
4   xmlns:xlink="http://www.w3.org/1999/xlink"
5   xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 http://dashif.org/docs/DASH-IF-DP-1502/
6   profiles="urn:mpeg:dash:profile:isoff-live:2011"
7   type="static"
8   mediaPresentationDuration="PT1M48.3S"
9   maxSegmentDuration="PT5.0S"
10  minBufferTime="PT13.9S">
11  <ProgramInformation>
12  </ProgramInformation>
13  <ServiceDescription id="0">
14  </ServiceDescription>
15  <Period id="0" start="PT0.0S">
16    <AdaptationSet id="0" contentType="video" startWithSAP="1">
17      <Representation id="0" mimeType="video/mp4" codec="H.264" >
18        <SegmentTemplate timescale="30000" initialization="init-stream0.m4s" >
19          <SegmentTimeline>
20            <S t="0" d="203203" />
21            <S d="250250" r="10" />
22            <S d="209209" />
23            <S d="85085" />
24          </SegmentTimeline>
25        </SegmentTemplate>
26      </Representation>
27    </AdaptationSet>
28  </Period>
29 </MPD>
```


MPEG-DASH

- Full mpd file from the example

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="urn:mpeg:dash:schema:mpd:2011"
4   xmlns:xlink="http://www.w3.org/1999/xlink"
5   xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd"
6   profiles="urn:mpeg:dash:profile:isoff-live:2011"
7   type="static"
8   mediaPresentationDuration="PT1M48.3S"
9   maxSegmentDuration="PT5.0S"
10  minBufferTime="PT13.9S">
11  <ProgramInformation>
12  </ProgramInformation>
13  <ServiceDescription id="0">
14  </ServiceDescription>
15  <Period id="0" start="PT0.0S">
16    <AdaptationSet id="0" contentType="video" startWithSAP="1" segmentAlignment="true" bitstreamSwitching="true" frameRate="30000/1001" maxWidth="1920" maxHeight="1080" par="16:9" lang="eng">
17      <Representation id="0" mimeType="video/mp4" codecs="avc1.640028" bandwidth="2403076" width="1920" height="1080" sar="1:1">
18        <SegmentTemplate timescale="30000" initialization="init-stream$RepresentationID$.m4s" media="chunk-stream$RepresentationID$-$Number%05d$.m4s" startNumber="1">
19          <SegmentTimeline>
20            <S t="0" d="203203" />
21            <S d="250250" r="10" />
22            <S d="209209" />
23            <S d="85085" />
24          </SegmentTimeline>
25        </SegmentTemplate>
26      </Representation>
27    </AdaptationSet>
28  </Period>
29 </MPD>
```


MPEG-DASH - Transcoding

```
ffmpeg -i space.mp4 -f dash space.mpd
```

- Use ffmpeg to convert to MPEG-DASH
- "-f dash" to specify the output format as MPEG-DASH
- Creates an mpd with the name you provide
 - All other files follow a default naming convention
 - May want to create a new directory for each video to avoid naming conflicts

Adaptive Bit-Rate Streaming

The Problem

- You host a video on your web app
 - You even use HLS or MPEG-DASH to segment the video into 2-10 seconds chunks
- You want high quality so you host chunks in 4K@60Hz
 - Can require ~25Mb/s bandwidth to stream
- And someone visits your site using eduroam on a bad day..
 - The video buffers, stutters, or doesn't play at all
- We need a solution that:
 - Allows users with slow connections to enjoy your content
 - Delivers high quality to users with high-speed Internet

Adaptive Bit-Rate

- Instead of hosting the a single video at a single resolution
 - Host multiple versions of the same video at different resolutions
 - Each resolution requires a different bit rate to stream
- User visits your page
 - Their browser adapts to the current download bandwidth available
 - Stream the highest bit rate video that fits the bandwidth

Adaptive Bit-Rate

- Using HLS or MPEG-DASH
 - Create chunks at several different resolutions/bit rates
 - Add information about all resolutions in the index file
- With the video segmented into ~2-10 second chunks
 - Easy for the browser to switch between resolutions

Adaptive Bit-Rate

- The browser can adapt the requested bit-rate based on current conditions
- Limited interruption for the user, though quality can change over time



Adaptive Bit-Rate - HLS

- Using HLS, m3u8 index files can be nested
- Convert your video into multiple HLS resolutions
- Combine them into a single index file with references to the others
- This example contains references to 3 different resolutions

```
≡ party.m3u8
≡ party480p.m3u8
≡ party720p.m3u8
≡ party1080p.m3u8
```

```
1 #EXTM3U
2 #EXT-X-STREAM-INF:BANDWIDTH=5000000,RESOLUTION=1920x1080
3 party1080p.m3u8
4 #EXT-X-STREAM-INF:BANDWIDTH=2500000,RESOLUTION=1280x720
5 party720p.m3u8
6 #EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=854x480
7 party480p.m3u8
```


Adaptive Bit-Rate - MPEG-DASH

- The mpd file can contain multiple resolutions
- The media is represented in multiple layers
 - Period
 - Adaptation Set
 - Representation

```
15 <Period id="0" start="PT0.0S">
16   <AdaptationSet id="0" contentType="video" startWithSAP="1" segmentAlignment="true" bitstreamSwitching="true" frameRate="30/1" maxWidth="640" maxHeight="1138" par="9:16" lang="und">
17     <Representation id="0" mimeType="video/mp4" codecs="avc1.64001f" bandwidth="480684" width="540" height="960" sar="1:1"...>
25     <Representation id="2" mimeType="video/mp4" codecs="avc1.64001f" bandwidth="1000000" width="640" height="1138" sar="5121:5120"...>
33   </AdaptationSet>
34   <AdaptationSet id="1" contentType="audio" startWithSAP="1" segmentAlignment="true" bitstreamSwitching="true" lang="und">
35     <Representation id="1" mimeType="audio/mp4" codecs="mp4a.40.2" bandwidth="128000" audioSamplingRate="44100"...>
45   </AdaptationSet>
46 </Period>
```


Adaptive Bit-Rate

- Your task [For AO3]:
 - Programmatically convert an uploaded mp4 into hls with multiple resolutions
 - Serve those videos by setting the source of a video to the index file for that video
- Hint: Get ffmpeg to do all the work for you
 - You should not manually write any of these files
 - Do research to find the command(s)/flags you need to send to ffmpeg to do the job

Video Players

- Most built-in video players do not support HLS or MPEG-DASH
- You cannot rely on the browser having a player for either of these formats (eg. During grading we will not use a browser with built-in support for HLS or MPEG-DASH)
- We must use a 3rd party video player or library
 - Several players available (eg. dash.js)
 - Examples in the following slides will use video.js

Video Players

- This example downloads the css and js for video.js from a CDN
- Uses "class" and "data-setup" attributes on a video element to tell the library to do its thing
- You now have a video player that supports both HLS and MPEG-DASH

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <link href="https://vjs.zencdn.net/8.10.0/video-js.css" rel="stylesheet"/>
5      <title>CSE312 Video Example</title>
6  </head>
7  <body>
8
9      <video class="video-js" width="300" controls autoplay data-setup="{}">
10         <source src="main.m3u8"/>
11         Your browser does not support video playback
12     </video>
13
14     <script src="https://vjs.zencdn.net/8.10.0/video.min.js"></script>
15 </body>
16 </html>
```

```
<video class="video-js" width="300" controls autoplay data-setup="{}">
    <source src="output.mpd"/>
    Your browser does not support video playback
</video>
```


Video Players

- Your video player will now have a consistent look across all browsers
- Don't have to worry about what formats each browser supports
- You support any format supported by video.js

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <link href="https://vjs.zencdn.net/8.10.0/video-js.css" rel="stylesheet"/>
5      <title>CSE312 Video Example</title>
6  </head>
7  <body>
8
9      <video class="video-js" width="300" controls autoplay data-setup="{}">
10         <source src="main.m3u8"/>
11         Your browser does not support video playback
12     </video>
13
14     <script src="https://vjs.zencdn.net/8.10.0/video.min.js"></script>
15 </body>
16 </html>
```

