

Manual

Managing CDN DASH Streaming System

2018.01.15

Presenter : Linh Van Ma

Outline

1. Introduction

2. System Working Flow

3. Testing Scenarios

4. Implemented Functions

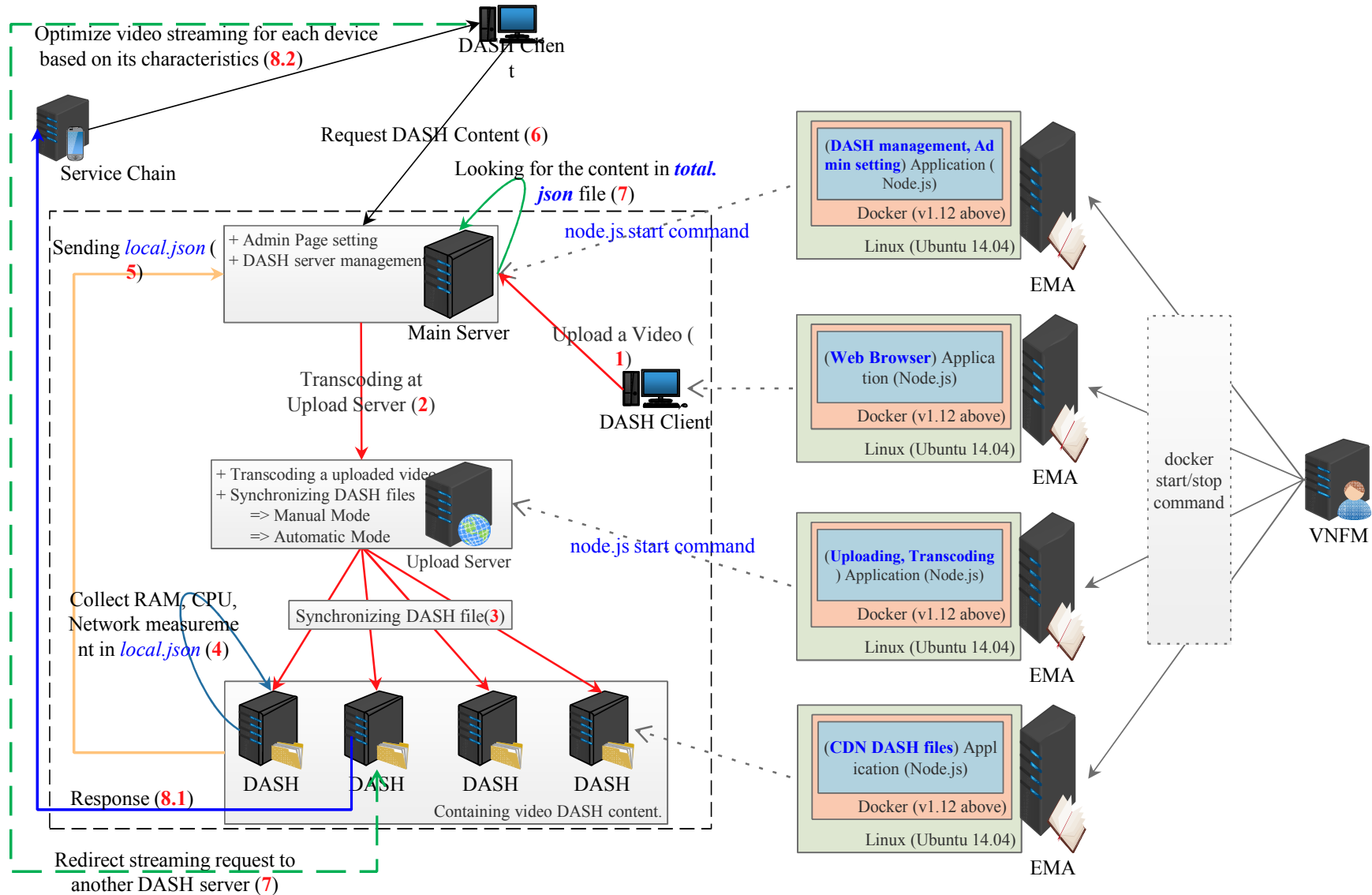
- **Functions:** [1] Server Management. [2] Service Management. [3] Upload server Management. [4] Client Management.
- **Streaming Processes:** [1] Fragmenting Uploaded Videos. [2] Dynamic Adaptive Streaming.

5. Node.js Implementation

1. Introduction (What is it?)

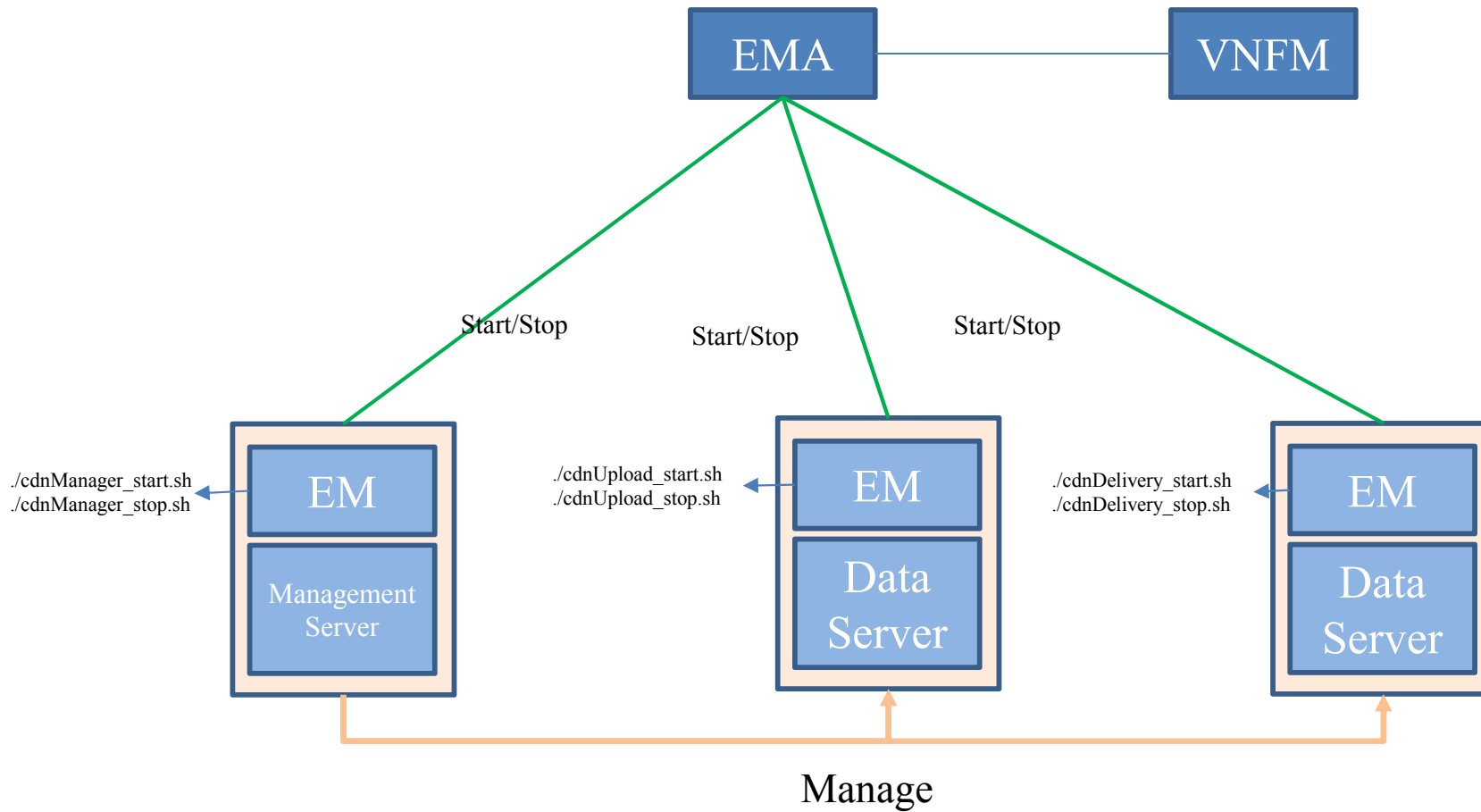
- 1) This system aims to support dynamic adaptive streaming over HTTP (DASH) that enables high quality streaming of media content over the Internet delivered from conventional HTTP web servers.
 - The client selects the segment with the highest bit rate possible that can be downloaded in time for playback **without causing stalls or re-buffering events** in the playback.
 - An MPEG-DASH client can **seamlessly adapt to changing network conditions** and provide high quality playback with fewer stalls or re-buffering events.
- 2) This system is also built to support CDN streaming service.
 - Each delivery server (CDN) is managed by a main server.
 - The main server listens streaming request from clients and redirect to CDNs.
- 3) The system is implemented on Node.js, it can also run on Docker cooperating with FNCP (Future Network Computing Platform).

1. Introduction (How does it works?)



1. Introduction

VNF DataCenter System Overview



2. System Working Flow

2.1 Server Managed Function

- 1) This function is implemented on the main server where we manage CDNs.
 - Managing local performance of CDNs.
 - Managing network state
 - Managing geography

 - ✓ Sorting CDNs and ordering them in an order.

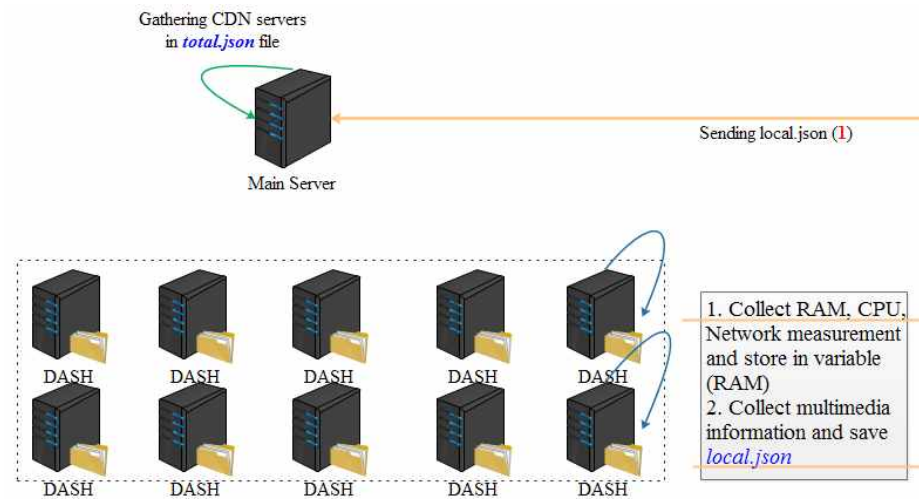
- 2) Each CDN is managed by inputting its information, such as IP address, CDN type whether **upload** or **delivery** in a web form.
 - CDN information is saved in mysql server Ubuntu.
 - We can also delete, edit CDN information once the CDN information was stored in the database.

2. System Working Flow

2.1 Server Managed Function

- 1) How does the main server collect data from CDNs?
 - Each CDN periodically collects its performance information and stores in a variable (temporary memory, RAM). Three seconds for local performance, three hundred seconds (five minutes) for network measurement.
 - Each CDN periodically check and store multimedia information in a **local.json** file. (10 seconds)

 - Each time when the main server receives local performancesort, network, geography, it processes collected information from CDNs and sorts in an order then saves in **server_rank.json**.
 - Each time when the main server receives **multimedia information** from a CDN, it stores the information in **ip_list.json** file, then gathers data from all CDNs in **total.json** using both **server_rank.json** and **ip_list.json**.



2. System Working Flow

2.1 Server Managed Function

1) Which information save in the JSON files?

- `local.json` stores an array of multimedia information such as one video information is given as the following.

```
{  
  "videoName": "(Paddy_Sun)_Sunflower__Paddy_Sun",  
  "stream": "http://168.131.39.38:8001//home/nonsense/Desktop/test/cdnnodejs/uploads/_/Paddy_Su/(Paddy_Sun)_Sunflower__Paddy_Sun.mpd",  
  "stFile": ["(Paddy_Sun)_Sunflo_output144", "(Paddy_Sun)_Sunflo_output240", "(Paddy_Sun)_Sunflo_output360"],  
  "Filecount": 659,  
  "listResol": "[108x144] [180x240] [270x360] ",  
  "maxRel": "270x360"  
}
```

- `total.json` file saves information as the following: {ip, metric, localFiles: `local.json`}.

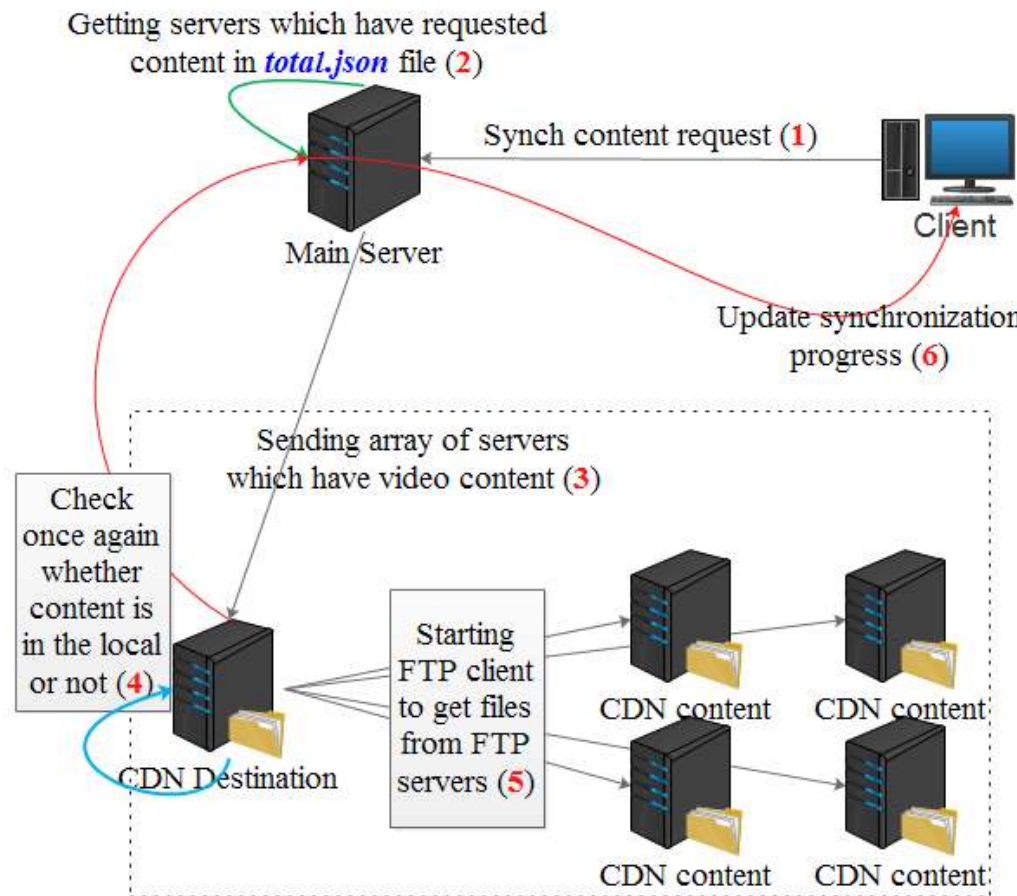
2. System Working Flow

2.2 Upload Server Managed Function (What is it?)

- 1) This function is to manage content synchronization between CDNs.
- 2) It has two synchronization mode which are automatic mode and manual mode.
- 3) Automatic mode
 - The main sever finds the best serving server for current coming request from a client then automatically syncs the video requested content to the best server from servers which have the content, if the best server does contain the video.
- 4) Manual Mode
 - In the manual mode, the list of currently available videos appears on the left.
 - The selected videos on the left will be synced to the chosen IP on the right by pressing the sync button.

2. System Working Flow

2.2 Upload Server Managed Function (How does it work?)



2. System Working Flow

2.2 Upload Server Managed Function (How does it work?)

1. We establish two communication channels
 - ✓ Communication channel between client and the main server (*using socket.io node.js*).
 - ✓ Communication channel between the main server and CDNs, between one CDN with other CDNs (*using tcp socket node.js*).
 - There is one communication between clients and upload servers will be talked later.
 - Each CDN server is started as streaming server as well as FPT server to share files, it also can be a FTP client to get files.

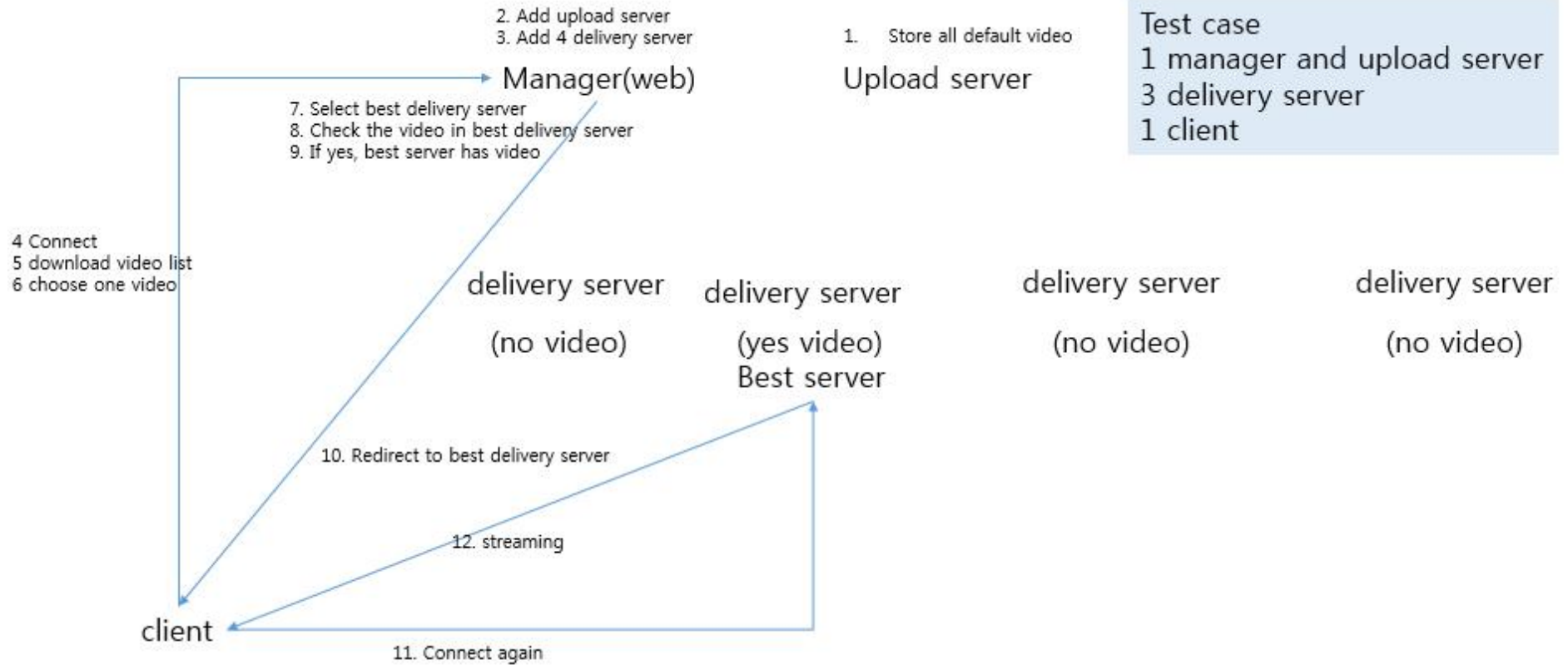
2. System Working Flow

2.2 Upload Server Managed Function (How does it work?)

1. First, a client web admin send a synchronization request using JSON format {IP, Array of checked videos} using socket.io browser communication.
2. Secondly, the main server gets a list of servers which has a video content among the received array video.
3. Thirdly, it sends the gotten array (video array which contains server array).
4. Fourthly, The server with IP checks whether a video is available in local or not.
5. Fifthly, The server CDN with IP starting FTP client to get files from other CDN FTP server which have a video content.
6. Sixthly, the IP server also updates synchronization progress to the web admin.

3. Testing

3.1 Test Case



3. Testing

3.2 Setting up Testing Environment

- Creating a **bridge** which connects all components (Manager, Delivery, Upload, Client)

```
# docker network create --driver=bridge network1 --subnet=10.100.0.0/24
```

- Starting testing CDN system

- Starting Manage Server

```
# docker run --network=network1 -it --name EM_Manager johnpekl/cdnmanager  
# ./cdnManager_start.sh (Manager_IP: 10.100.0.2)
```

- Starting Upload Server

```
# docker run --network=network1 -it --name EM_Upload johnpekl/cdnupload  
# ./cdnUpload_start.sh (Upload_IP: 10.100.0.3)
```

- Starting Delivery Servers

```
# docker run --network=network1 -it --name EM_Delivery1 johnpekl/cdndelivery  
# ./cdnDelivery_start.sh (EM_Delivery1_IP: 10.100.0.4)  
# docker run --network=network1 -it --name EM_Delivery2 johnpekl/cdndelivery  
# ./cdnDelivery_start.sh (EM_Delivery2_IP: 10.100.0.5)  
# docker run --network=network1 -it --name EM_Delivery3 johnpekl/cdndelivery  
# ./cdnDelivery_start.sh (EM_Delivery3_IP: 10.100.0.6)
```

- Starting Client

```
# docker run --network=network1 -dt --rm --name EM_WebClient -v /dev/shm:/dev/shm --privileged johnpekl/cdnclient  
&& docker exec -it EM_WebClient /bin/bash  
Client_IP: 10.100.0.7
```

3. Testing

3.3 Preparing for Client VNC (Virtual Network Computing)

➤ We can start Web-Client interface with difference options

✓ Starting and listening with chosen ports

```
# docker run --network=network1 -it --rm -p 6080:80 --name EM_WebClient -v /dev/shm:/dev/shm --privileged johnpekl/cdnclient
```

✓ Starting and listening with default HTTP port (80)

```
# docker run --network=network1 -dt --rm --name EM_WebClient -v /dev/shm:/dev/shm --privileged johnpekl/cdnclient
```

✓ Starting and allowing TCPDump inside client container

```
# docker run --network=network1 -it --rm -p 6080:80 --name EM_WebClient -v /dev/shm:/dev/shm --cap-add=NET_ADMIN johnpekl/cdnclient
```

✓ Starting and allowing typing command on terminal

```
# docker run --network=network1 -dt --rm --name EM_WebClient -v /dev/shm:/dev/shm --privileged johnpekl/cdnclient && docker exec -it EM_WebClient /bin/bash
```

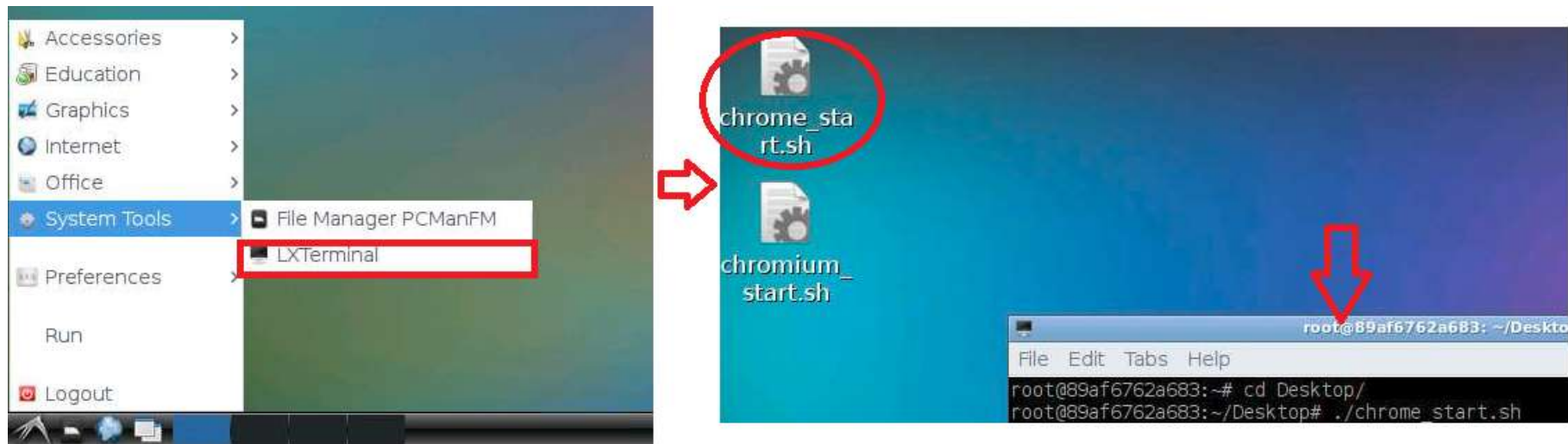
✓ **Note:** “/dev/shm:/dev/shm *–privileged*” allows running Google-chrome browser with *–no-sandbox* option inside client container, which supports displaying our built web-interface correctly since it is based on HTML5 (bootstrap).

3. Testing

3.3 Preparing for Client VNC (Virtual Network Computing)

- Web-client interface VNC
 - It was built based on “*docker-ubuntu-vnc-desktop*”, <https://github.com/fcwu/docker-ubuntu-vnc-desktop>
 - We added Google-chrome and Chromium to display our HTML5-based web interface
 - After starting the client container and connecting to the VNC by typing *10.100.0.7* on the Google-chrome browser, open LXTerminal with the following commands as shown in the figure below

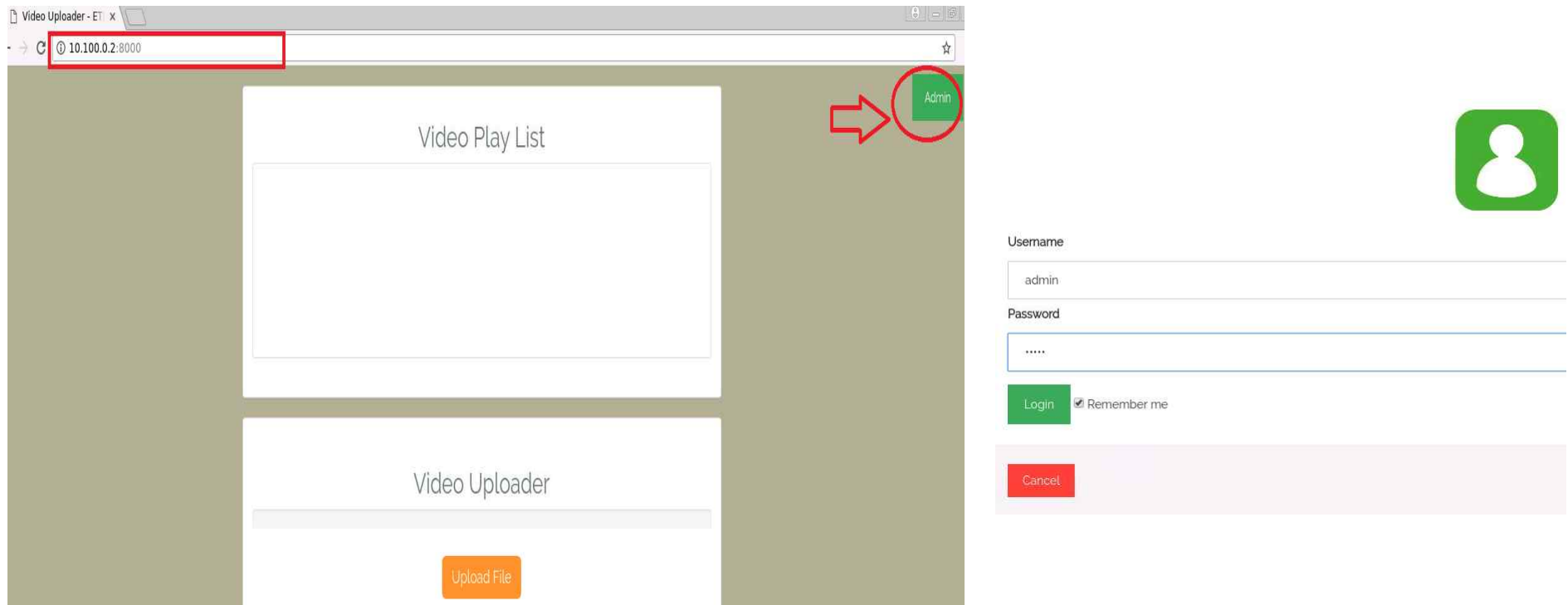
```
# cd Desktop/  
# ./chrome_start.sh
```



3. Testing

3.4 Connect All CDNs

- Typing *10.100.0.2:8000* which is the address the main server is listening
- Click **Admin** to login manager interface (User: *admin*, Pass: *admin* or it can be random values)



4. Functions

4.0 Main Streaming Web Interface (How does it look?)

The screenshot displays a web interface with two main sections: "Video Play List" and "Video Uploader".

Video Play List: A list of four video entries is shown in a white box with a thin border. The entries are: [Video: 1 | (Paddy_Sun)_Sunflower__Paddy_Sun], [Video: 2 | Britt_Nicole__Ready_or_Not_Lyrics_ft], [Video: 3 | TWICE], and [Video: 4 | TobyMac__Steal_My_Show_(Lyrics)].

Video Uploader: This section features a blue progress bar at the top labeled "Done". Below it is an orange "Upload File" button. A red cloud-shaped callout with the word "Click" and an upward arrow points to the button. Below the button, the file name "File: Mandisa__Overcomer.mp4" is listed. A list of encoding progress percentages follows: "Encoding 144p: 63.01%", "Encoding 240p: 51.34%", "Encoding 360p: 36.91%", "Encoding 480p: 27.88%", and "Encoding 720p: 15.03%". A red arrow points from the text "Fragmenting progress at upload server" to the "Encoding 480p" line. At the bottom of this section, it says "Fragmenting: waiting for completion of the encoding".

In the top right corner of the interface, there is a green "Admin" button. At the bottom of the page, the ETRI logo and text "ETRI Electronics and Telecommunications Research Institute. All rights reserved." are visible.

4. Functions

4.1 Server Management (Connect All CDNs)

- Adding CDNs with its type, note, name and IP address



Server List

SERVER NAME	TYPE	IP ADDRESS	NOTE	#CLIENTS	Edit	Delete
upload	upload	10.100.0.3	up-ser	0		
d1	delivery	10.100.0.4	first de	0		
d2	delivery	10.100.0.5	second d	0		
d3	delivery	10.100.0.6	third de	0		



4. Functions

4.2 Service Management (How does it look?)

- 1) This function manages streaming service of a CDN by checking how many videos are storing in the local CDN.
 - We can also delete a video or all of available video on the local CDN.
 - If we hover mouse over a video, it will play as long as the mouse points over the video.

The screenshot shows a web dashboard with a navigation bar containing: Server Management, CDN Metric Management, Service Management (active), Upload Server Management, and Clients Management. A 'Log out' button is in the top right corner. Below the navigation bar is a 'Server List' table:

SERVER NAME	TYPE	IP ADDRESS	NOTE
asdasd	delivery	168.131.39.35	asdasdasd
sdasdasd	upload	168.131.39.38	sadasd

A red box highlights the second row of the table, and a red arrow points to it with the word 'Click' in a speech bubble. Below the table is a 'Video List at a Selected Server [168.131.39.38]' section with a 'Delete All Video' button. The video list contains 9 items, each with a video thumbnail, title, '100% Video' status, and a 'Delete' button:

- (Paddy_Sun)_Sun
- Britt_Nicole__
- Funny_Cat_Vines
- GalacticRose__
- Motivational_sh
- SPRING_2017__Sh
- TWICE
- TobyMac__Steal
- Whatsapp_Status

At the bottom of the dashboard, there is a footer: ETRI Electronics and Telecommunications Research Institute, All rights reserved.

4. Functions

4.3 Upload Server Management (How does it look?)



Content Synchronization Method

Synchronization Method

Automatic

Manual

Manual Synchronization from the Upload Server

Content on Upload Server List				
#	File name	Max Resolution	#Resolution [Width x Height]	Check
1	(Paddy_S	270x360	[108x144] [180x240] [270x360]	<input type="checkbox"/>
2	Britt_Ni	270x480	[80x144] [134x240] [202x360] [270x480]	<input type="checkbox"/>
3	TWICE	404x720	[80x144] [134x240] [202x360] [270x480] [404x720]	<input type="checkbox"/>
4	TobyMac_	270x480	[80x144] [134x240] [202x360] [270x480]	<input type="checkbox"/>

DASH Server List				
#	Name	IP Address	Sync	Progress
1	asdasd	168.131.39.35	<input type="button" value="Sync"/>	0%

Check (in a red cloud bubble) points to the 'Check' column of the first table.

Sync checked videos to this server (in red text) points to the 'Sync' button of the second table.

1. In the automatic mode, an automatic algorithm is implemented to synchronize file without consideration of admin.
2. In the manual mode, the list of currently available videos appears on the left.
3. The selected videos on the left will be synced to the chosen IP on the right by pressing the sync button.

4. Functions

4.4 Client Management

1. It manages clients which have video streaming with the CDN system

Log out

Server Management Content Management Upload Server Management Clients Management

Current Connected Client List

CLIENT IP	CURRENT STREAMING VIDEO	START TIME
10.100.0.1	[10.100.0.3] => (Paddy_Sun)_Sunflower__Paddy_Sun	오전 7:08:26

CDN Server Selection [10.100.0.1]

#	IP Address	RTT (ms)	Distance (Km)	Select
1	10.100.0.3	347	783	<input type="checkbox"/>
2	10.100.0.4	471	559	<input type="checkbox"/>
3	10.100.0.5	627	765	<input type="checkbox"/>
4	10.100.0.6	781	426	<input type="checkbox"/>

Auto Manual

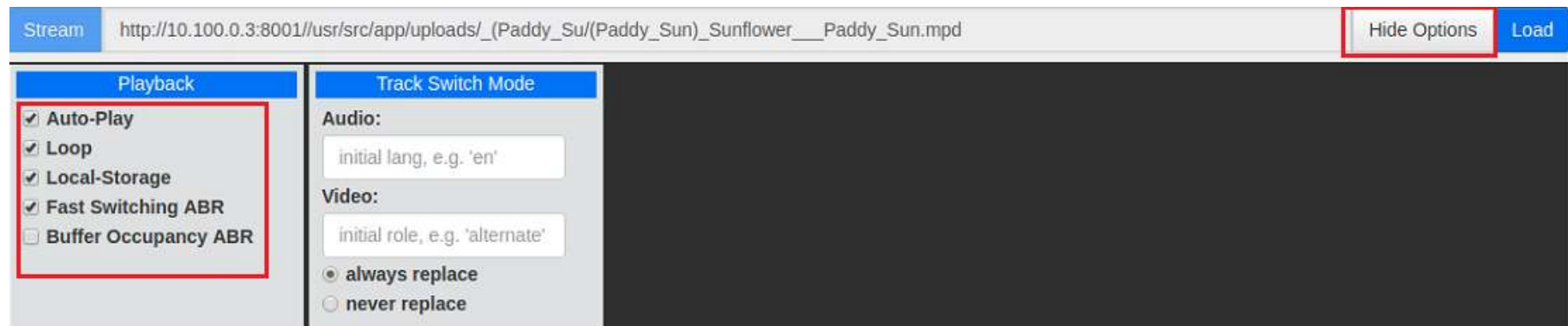
4. Functions

4.5 Streaming Web Interface

4. Functions

4.5 Streaming Web Interface

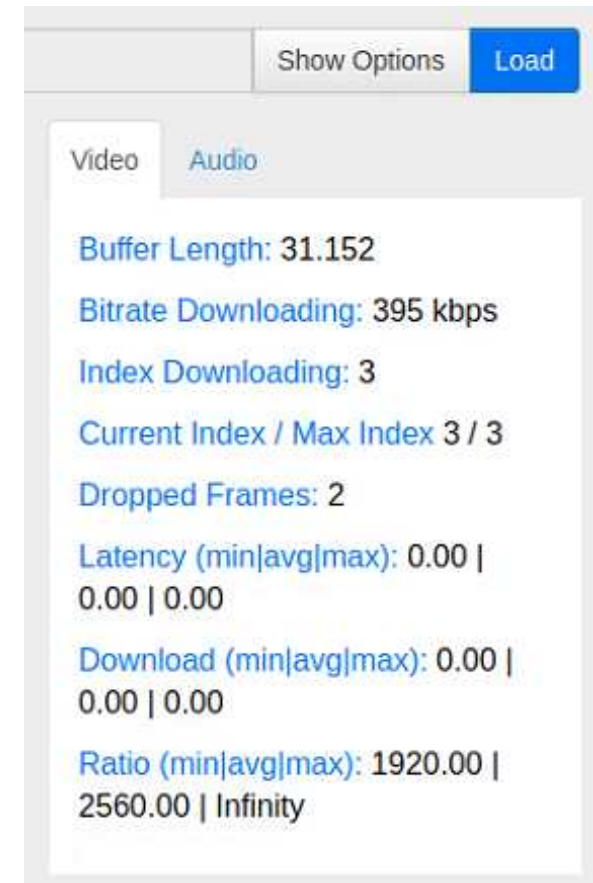
1. **Auto-Play**: Enables automatic startup of the media once the media is loaded
2. **Loop**: Enables looping of the media once playback has completed
3. **Local-Storage**: Enables local storage of player state (last bitrate, a/v or text track etc). This is then used when the next time media is played.
4. **Fast Switching ABR (Adaptive Bitrate)**: Enables faster ABR switching (time to render). Only when the new quality is higher than the current.
5. **Buffer Occupancy ABR**: BOLA (Buffer Occupancy based Lyapunov Algorithm) is an ABR ruleset. When enabled, it will disable the default heuristics in Dash.js that depend strictly on average and real-time throughput measurements



4. Functions

4.5 Streaming Web Interface (Right Panel Parameters)

1. **Buffer Length:** The length of the forward buffer, in seconds.
2. **Bitrate Downloading:** The bitrate of the representation being downloaded.
3. **Index Downloading:** The representation index being downloaded and appended to the buffer.
4. **Current Index / Max Index:** The representation index being rendered.
5. **Dropped Frames:** The absolute count of frames dropped by the rendering pipeline since play commenced.
6. **Latency (min|avg|max):** The minimum, average and maximum latency over the last 4 requested segments. Latency is the time in seconds from request of segment to receipt of first byte.
7. **Download (min|avg|max):** The minimum, average and maximum download time for the last 4 requested segments. Download time is the time in seconds from first byte being received to the last byte.
8. **Ratio (min|avg|max):** The minimum, average and maximum ratio of the segment playback time to total download time over the last 4 segments



The screenshot shows a web interface with a 'Show Options' button and a blue 'Load' button. Below these are two tabs: 'Video' (selected) and 'Audio'. The 'Video' tab displays the following parameters:

- Buffer Length: 31.152
- Bitrate Downloading: 395 kbps
- Index Downloading: 3
- Current Index / Max Index 3 / 3
- Dropped Frames: 2
- Latency (min|avg|max): 0.00 | 0.00 | 0.00
- Download (min|avg|max): 0.00 | 0.00 | 0.00
- Ratio (min|avg|max): 1920.00 | 2560.00 | Infinity

5. Node.js Implementation

5.1 Introduction of Node.js

- **Asynchronous Programming**: Node.js uses a module architecture to simplify the creation of complex applications.
- Every function in Node.js is **asynchronous**. Therefore, everything that would normally block the thread is instead executed in the background.
 - ✓ This is the most important thing to remember about Node.js. For example, if you are reading a file on the file system, you have to specify a callback function that is executed when the read operation has completed.
- Node.js is **only an environment** - meaning that you have to do everything yourself. There is not a default HTTP server, or any server for that matter.
 - ✓ This can be overwhelming for new users, but the payoff is a high performing web app. One script handles all communication with the clients. This considerably reduces the number of resources used by the application.

5. Node.js Implementation

5.2 Program Structure

Not use, but was about use for lorenwest/node-monitor

Contain DASH streaming files (*.m4s)

Logic Program System Behaviour

Main Application (Starting Script, Program starts from here)

Name	Date modified	Type	Size
config	8/8/2017 8:23 PM	File folder	
node_modules	12/8/2017 1:19 AM	File folder	
public	7/24/2017 1:53 AM	File folder	
smmcConnection	10/15/2017 12:39 AM	File folder	
smmcDASHStreaming	9/21/2017 12:04 AM	File folder	
smmcDatabase	8/31/2017 10:03 PM	File folder	
smmcFTPGetFile	9/20/2017 11:40 PM	File folder	
smmcMonitor	10/19/2017 11:35 PM	File folder	
smmcServerInfors	10/19/2017 11:32 PM	File folder	
smmcUploadVideoTest	9/20/2017 11:42 PM	File folder	
smmcVideoProcessUpload	9/20/2017 10:56 PM	File folder	
uploads	9/21/2017 12:29 AM	File folder	
views	9/19/2017 4:24 PM	File folder	
app	12/6/2017 4:37 PM	JS File	4 KB
LICENSE	11/23/2016 4:28 PM	File	2 KB
package.json	12/8/2017 1:19 AM	JSON File	2 KB
README	7/28/2017 11:25 PM	MD File	8 KB

5. Node.js Implementation

5.2 Program Structure

Once the package is in node_modules, you can use it in your code.

Contain DASH Player client, and some other bootstrap, jquery

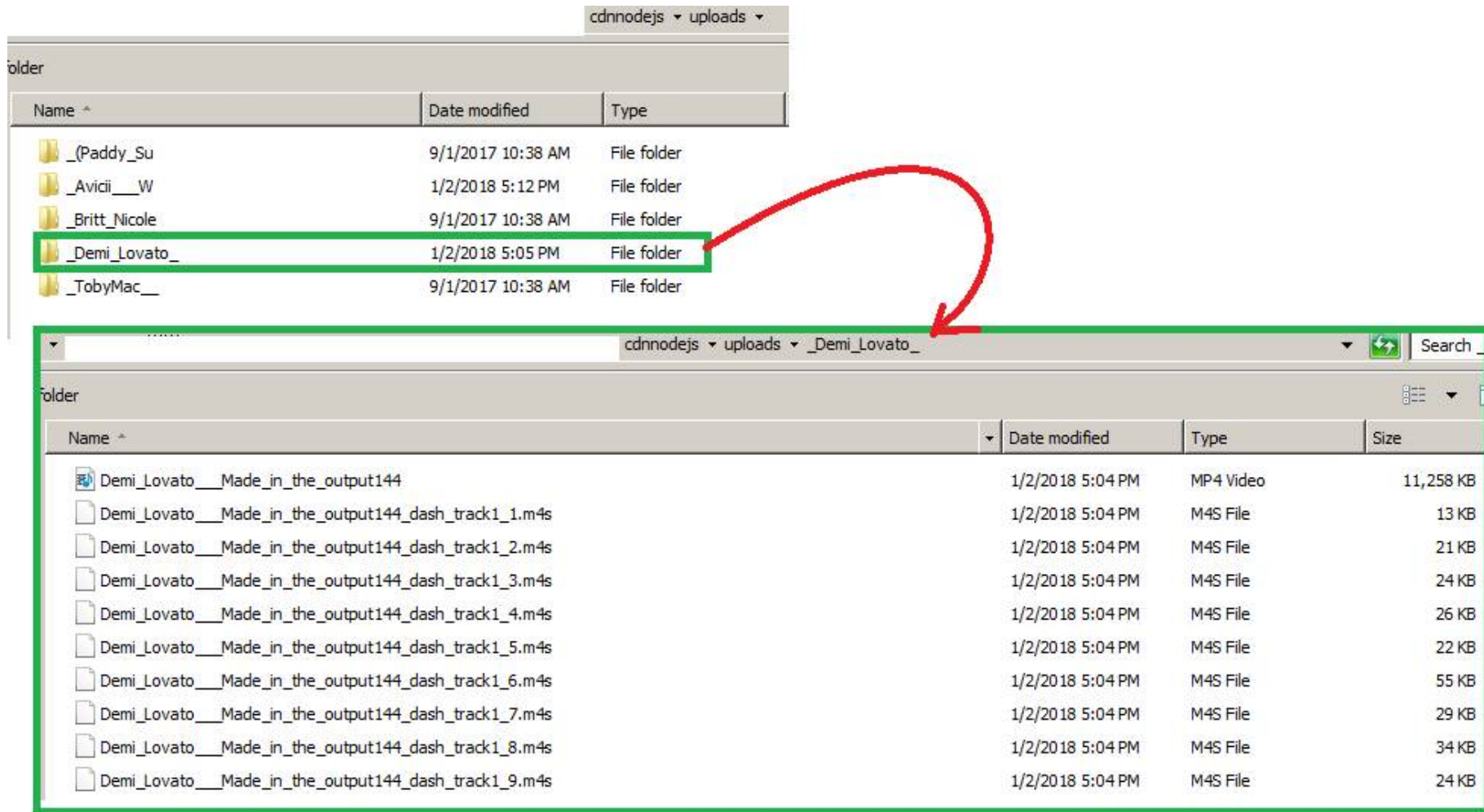
Contain html of the first page (website)

all you need to know about what's required for node.js program

Name	Date modified	Type	Size
config	8/8/2017 8:23 PM	File folder	
node_modules	12/8/2017 1:19 AM	File folder	
public	7/24/2017 1:53 AM	File folder	
smmcConnection	10/15/2017 12:39 AM	File folder	
smmcDASHStreaming	9/21/2017 12:04 AM	File folder	
smmcDatabase	8/31/2017 10:03 PM	File folder	
smmcFTPGetFile	9/20/2017 11:40 PM	File folder	
smmcMonitor	10/19/2017 11:35 PM	File folder	
smmcServerInfors	10/19/2017 11:32 PM	File folder	
smmcUploadVideoTest	9/20/2017 11:42 PM	File folder	
smmcVideoProcessUpload	9/20/2017 10:56 PM	File folder	
uploads	1/9/2018 11:43 AM	File folder	
views	9/19/2017 4:24 PM	File folder	
app	12/6/2017 4:37 PM	JS File	4 KB
LICENSE	11/23/2016 4:28 PM	File	2 KB
package.json	12/8/2017 1:19 AM	JSON File	2 KB
README	7/28/2017 11:25 PM	MD File	8 KB

5. Node.js Implementation

5.2 Program Structure (“Upload” folder)



5. Node.js Implementation

5.3 Logic Program (“smmc...”)

- smmcConnection: Handling connections (socket.io and tcpConnection) from web browsers and from delivery/upload servers to main server.
- smmcDASHStreaming: Handling DASH streaming request from DASH Player (web browser).
- smmcDatabase: Interact with database “dashservers” containing information of delivery, upload servers which managed by main server.
- smmcFTPGetFile: Synchronize DASH files from upload to delivery servers.
- smmcMonitor: Manage CDN (delivery, upload server) and main server, it almost contains starting scripts of CDN and main server based on input command. It mostly interacts with smmcConnection, main script of each side (either CDN or main server) is started from here separately.
- smmcServerInfors: Contain delete file scripts, local file information script periodically checking and storing information in JSON file, and JSON file containing local files information in /list folder.
- smmcVideoProcessUpload: Contain script to decode uploaded videos to different resolutions, and segment them into DASH, small segment with length from 2-10 seconds with .m4s file extension.

5. Node.js Implementation

5.4 Starting Node.JS script

- We defined three different group of functions for main, upload and delivery servers.
- Main server starts with: `node app.js -main`
- Delivery server starts with: `node app.js -delivery`
- Upload server starts with: `node app.js -upload`

5. Node.js Implementation

5.4 Starting Node.JS script (“app.js”)

```
var express = require('express');
var app = express();
var path = require('path');
var fs = require('fs');
var server = require('http').createServer(app);
var io = require('socket.io').listen(server);
var cors = require('cors');
```

Reuse published modules

```
var videoInfors = require('./smmcServerInfors/serverInfors.js');
var nodejsdb = require('./smmcDatabase/nodejsdb.js');
var fragmentingFile = require('./smmcVideoProcessUpload/fragmentingFile.js');

var monitorCDNs = require('./smmcMonitor/monitorCDNs.js');
var startCDNs = require('./smmcMonitor/startCDNs.js');

var typeCDN = ['delivery', 'upload'];
var TAG = 'app.js: ';
```

Implemented new modules

```
process.argv.splice(1).forEach(function(val, index, array) {
  if (val === '-main') {
    monitorCDNs.mainMonitor(io, app);
    nodejsdb.createDatabase();

    server.listen(8000, function() {
      console.log(TAG, 'The web manage interface is listening on port 8000');
    });
  } else if (val === '-delivery') {
    startCDNs.startCDNServers(io, typeCDN[0]);
    videoInfors.local_checking(typeCDN[0]);
  } else if (val === '-upload') {
    app.use(cors({
      credentials: true
    }));
    fragmentingFile.fragmentingOnUploadServer(app, io);
    startCDNs.startCDNServers(io, typeCDN[1]);
    videoInfors.local_checking(typeCDN[1]);
    socketBrowserServerComm();

    server.listen(8000, function() {
      console.log(TAG, 'The upload server is listening on port 8000');
    });
    console.log(TAG, 'Upload server has been started');
  }
});
```

Starting socket.io interacting with browser, and TCP connection interacting with CDN (delivery, upload server)

Initiating mySql connection interacting with database of CDNs

Listening socket.io connection from browser (Browser interface manages all actions between main and CDNs)

Starting CDNs (delivery and upload servers) to interact and be managed by the main server
Periodically check local information and store in JSON file

Allowing main server direct upload link to upload server

Fragmenting uploaded video to DASH content

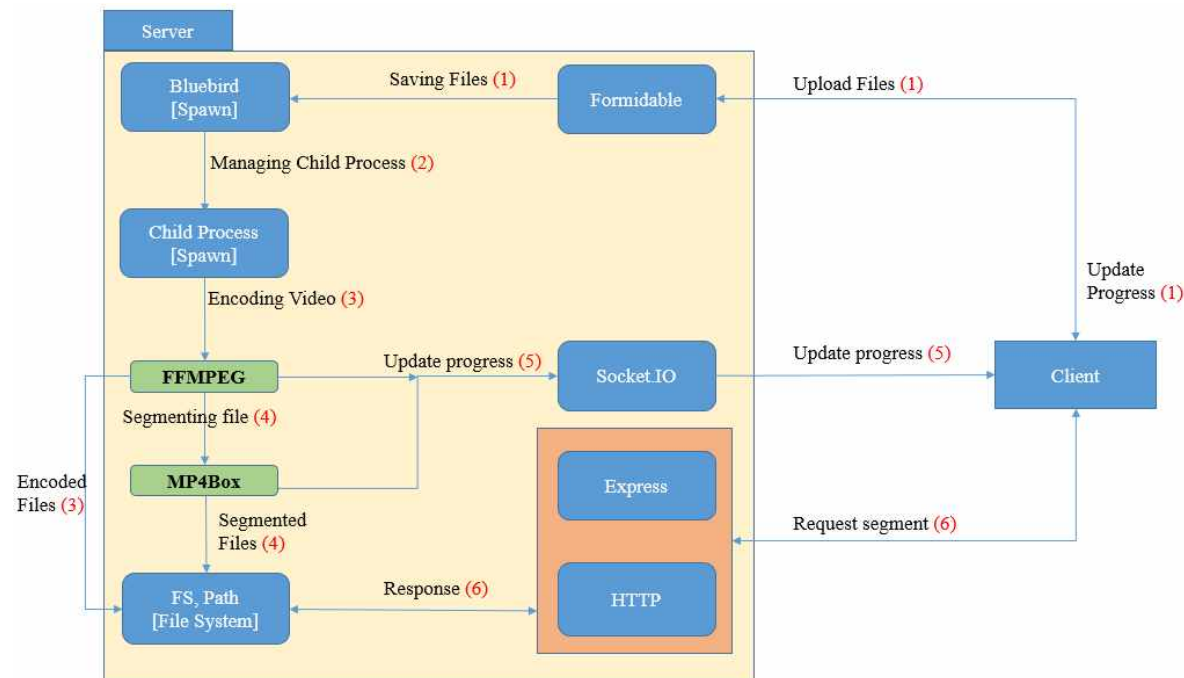
Socket.io Listening

});

5. Node.js Implementation

5.5 Processing Uploaded Video (“*smmcVideoProcessUpload*” folder)

- ❖ Clients upload videos to server and it is processed to (Dynamic Adaptive Streaming over HTTP) DASH content.

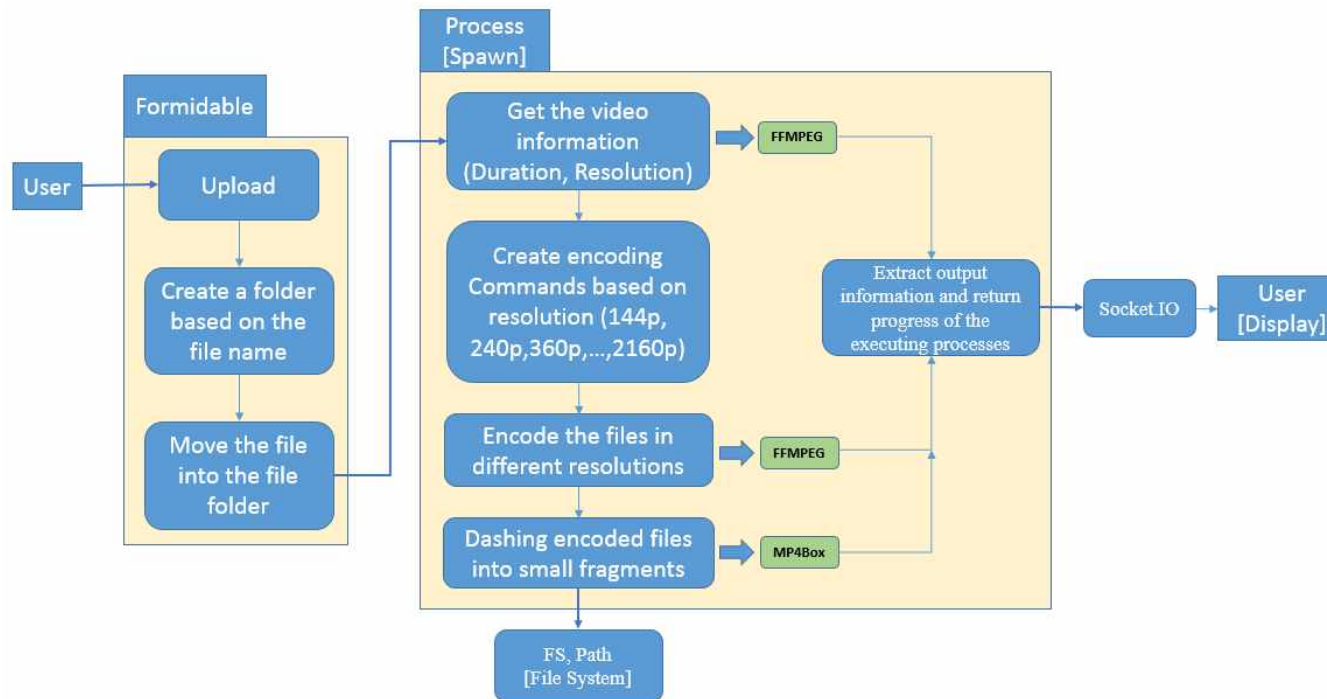


Transcoding Content between Clients and Servers

5. Node.js Implementation

5.5 Processing Uploaded Video (“*smmcVideoProcessUpload*” folder)

- ❖ The progress of transcoding will be sent to clients in real-time.

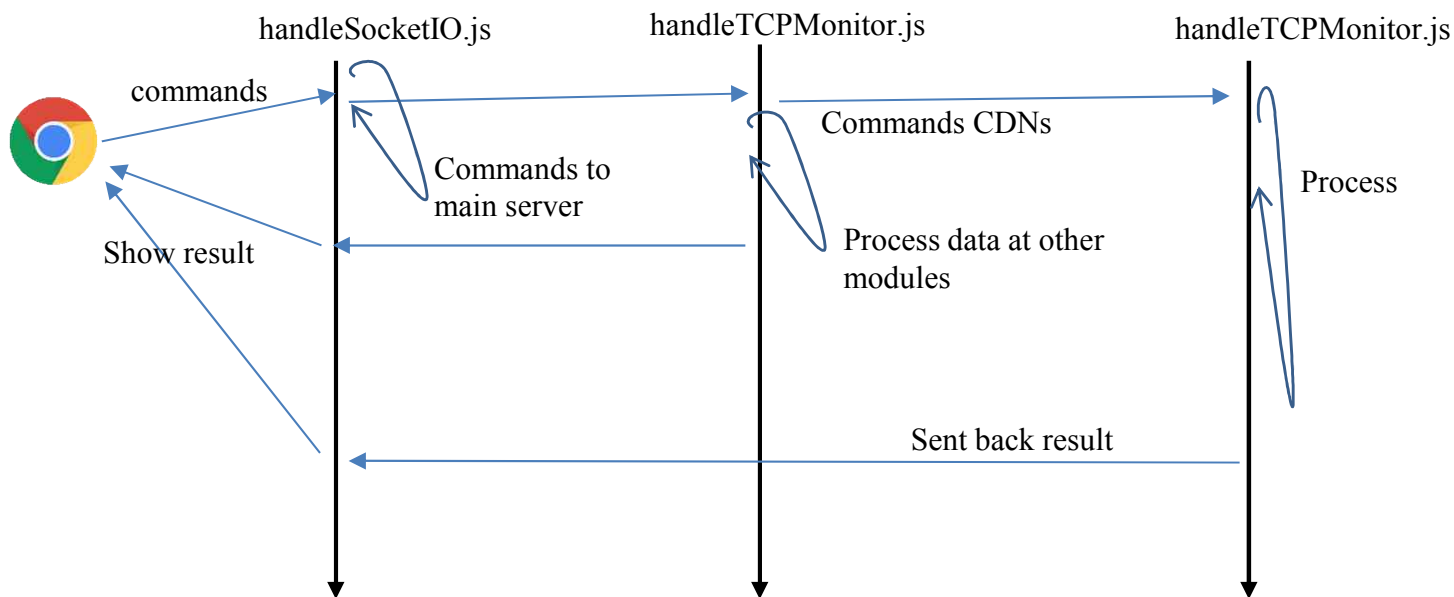


Transcoding Content between Clients and Servers

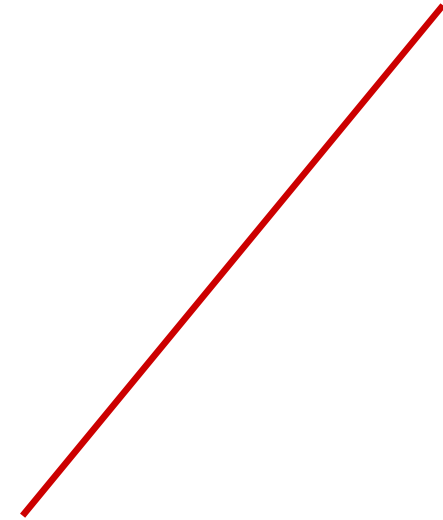
5. Node.js Implementation

5.6 Handle connection

- ❖ All actions are managed by the main server. All command, interaction must be passed through the main server.



Q & A



Thank You

