Live video streaming in the FIFA World Cup

Juanjo Ramos
Hudl


Hi, I'm Troy McClure.
ReactiveX
An API for asynchronous programming
with observable streams

FIFA.com / EN

FOOTBALL TECHNOLOGY

INNOVATIONS STANDARDS RESOURCE HUB BLOG CONTACT

INNOVATIONS / EPTS

HUDL SELECTED AS REPLAY SOLUTION PROVIDER FOR THE FIFA WOMEN'S WORLD CUP FRANCE 2019™

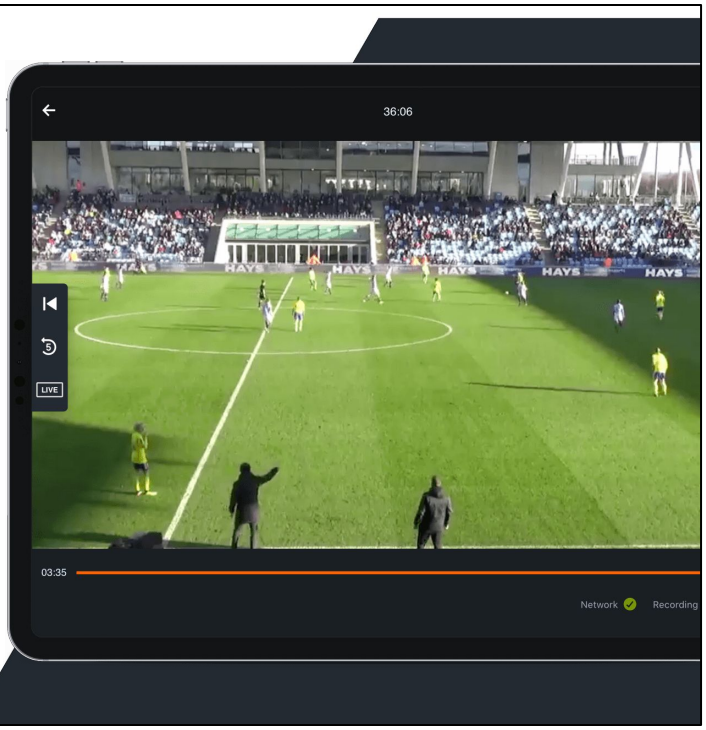
FIFA continues to develop in-game technology solutions.

For the upcoming FIFA Women's World Cup France 2019™ (7 June – 7 July), Hudl has been selected as Replay Solution provider.

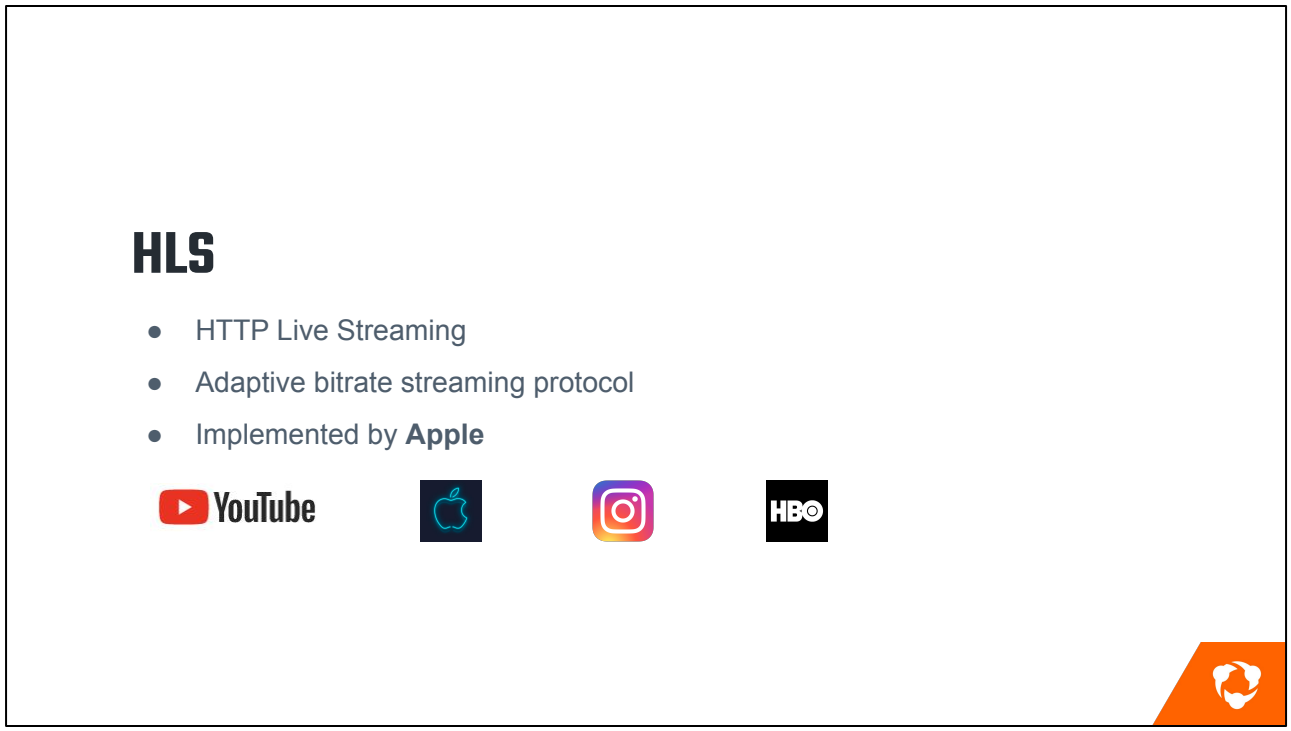
Hudl Replay software will be used by team analysts to support their tactical decisions as well as the medical representatives to evaluate and assess potential injuries.

Hudl Replay

- **iOS** app (iPad only)
- **Live** video streaming client
- Server on the same (V/W)**LAN**
- It uses **HLS**



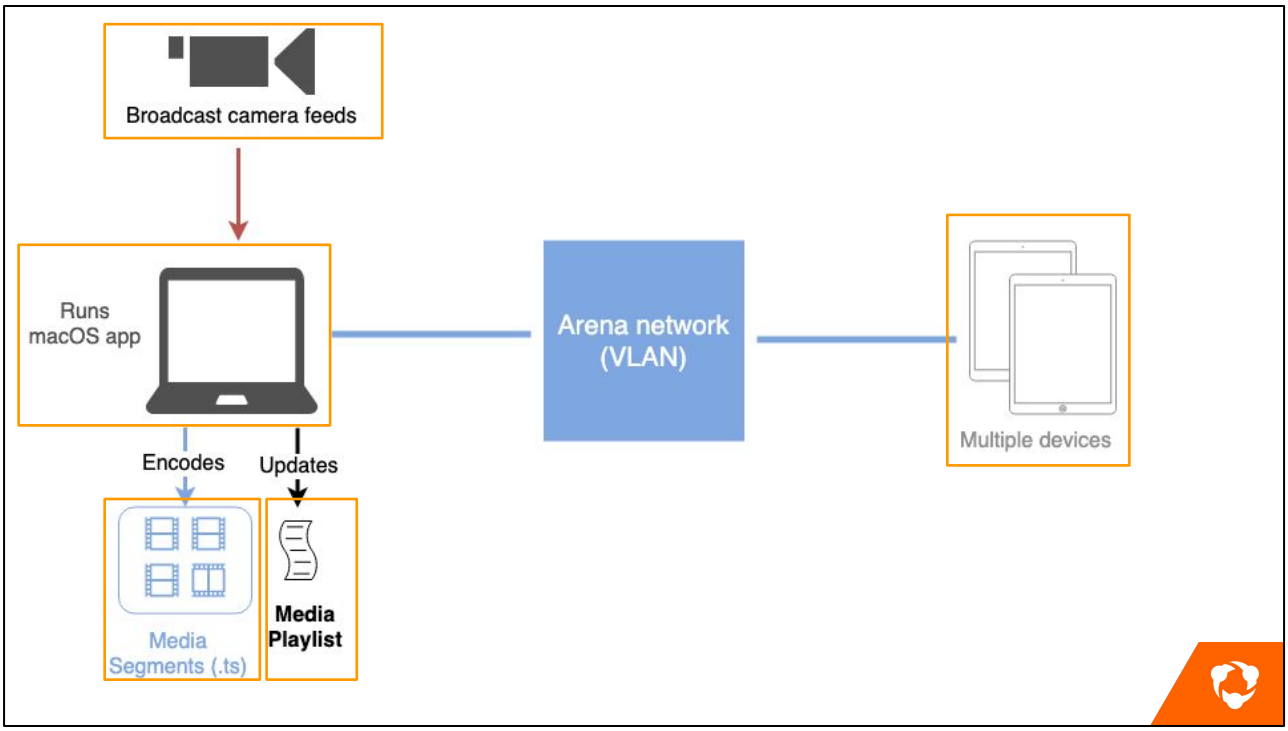
What's HLS and what do you do with it?



- HLS stands for HTTP Live Streaming,
  - So, it is a protocol that streams content over HTTP and therefore it gets all the HTTP benefits
  - Content - video
  - It can be used to stream live content like last Apple's keynote or video on demand like an episode from Game of Thrones.
- Adaptive bitrate streaming protocol
  - Watching video. Quality drops due to internet.
  - After some time it goes back to a high resolution.

    - That’s adaptive bitrate part. It allows the players to switch between streams of different quality (or bitrate) of the same content.
- Implemented by Apple
- Used everywhere. These apps are just an example.

HLS in action



- Hudl Replay setup.
- Server and client on the same network. Normally Internet
- Many common elements with any HLS setup
- Video input. Camera recording, server hosting the video
- HLS server. In our case, macOS running an app that does multiple things
- Server does 2 important things:
    - Encodes the media segments. Video input cannot be consumed as it is being recorded.
        - Encoder creates small chunks of video that last normally between 2 and 10s

        - Those chunks of video, that is, the media segments are the ones that are going to be distributed and played by the client
    - Playlist file. Playing text file with all the information client needs to play the HLS stream
- At the other end we have the clients that play the HLS stream, the video

(Master) Playlist

- 2 types.
- First one

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-START:TIME-OFFSET=-60


#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="English",DEFAULT=YES,AUTOSELECT=YES,FORCED=NO,LANGUAGE="en",URI="https://apple-events.akamaized.
net/hls/live/681800/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/cc/eng.m3u8"
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs-b",NAME="English",DEFAULT=YES,AUTOSELECT=YES,FORCED=NO,LANGUAGE="en",URI="https://apple-events.akamaized
.net/hls/live/681800-b/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/cc/eng.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=1343570,AVERAGE-BANDWIDTH=1390400,CODECS="avc1.4d401e,mp4a.40.2",RESOLUTION=640x360,FRAME-RATE=29.970,SUBTITLES="subs"
https://apple-events.akamaized.net/hls/live/681800/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/1200/1200.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=567560,AVERAGE-BANDWIDTH=660000,CODECS="avc1.4d401e",RESOLUTION=640x360,URI="https://apple-events.akamaized.
net/hls/live/681800/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/1200/1200_I-Frame.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=1343570,AVERAGE-BANDWIDTH=1390400,CODECS="avc1.4d401e,mp4a.40.2",RESOLUTION=640x360,FRAME-RATE=29.970,SUBTITLES="subs-
b"
https://apple-events.akamaized.net/hls/live/681800-b/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/1200/1200.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=567560,AVERAGE-BANDWIDTH=660000,CODECS="avc1.4d401e",RESOLUTION=640x360,URI="https://apple-events.akamaized.
net/hls/live/681800-b/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/1200/1200_I-Frame.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=725540,AVERAGE-BANDWIDTH=730400,CODECS="avc1.4d401e,mp4a.40.2",RESOLUTION=640x360,FRAME-RATE=29.970,SUBTITLES="subs"
https://apple-events.akamaized.net/hls/live/681800/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/0600/0600.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=283780,AVERAGE-BANDWIDTH=330000,CODECS="avc1.4d401e",RESOLUTION=640x360,URI="https://apple-events.akamaized.
net/hls/live/681800/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/0600/0600_I-Frame.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=725540,AVERAGE-BANDWIDTH=730400,CODECS="avc1.4d401e,mp4a.40.2",RESOLUTION=640x360,FRAME-RATE=29.970,SUBTITLES="subs-b"
https://apple-events.akamaized.net/hls/live/681800-b/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/0600/0600.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=283780,AVERAGE-BANDWIDTH=330000,CODECS="avc1.4d401e",RESOLUTION=640x360,URI="https://apple-events.akamaized.
net/hls/live/681800-b/0208kmksrrgukmmpvlwqmnzbuhaylttzxoazcamnfmmnni/master/0600/0600_I-Frame.m3u8"
```

- From Apple's live Keynote
- Allows for adaptive bitrate.
- **Different encodings for the same content**
- Different video qualities for the same content
- Resolution, bandwidth, frame rate that allows the client
- Each of this encodings point to a Media Playlist which is this m3u8 files

Media Playlist

```
#EXTM3U
#EXT-X-PLAYLIST-TYPE:EVENT
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-ALLOW-CACHE:YES
#EXTINF:2.000000
0000_00000.ts
#EXTINF:2.000000
0000_00001.ts
#EXTINF:2.000000
0000_00002.ts
#EXTINF:2.000000
0000_00003.ts
#EXTINF:2.000000
0000_00004.ts
#EXTINF:2.000000
0000_00005.ts
#EXTINF:2.000000
0000_00006.ts
#EXTINF:2.000000
0000_00007.ts
#EXTINF:2.000000
0000_00008.ts
```

1. EVENT or VOD

2. Segment duration
3. Segment URI

Server - Client lag >= ~2.5 x segment_duration

Random Access

- **All info to play a selected encoding or resolution**
- Metadata about the playlist and info about the segments
- Important metadata: Playlist type
- Other important information: The media segments, the small chunks of video.
- For each segment
  - Duration
  - URI
- 2 important things about HLS:
  - Lag
  - Cool thing: Random access. Client can play any point in video w/o having any previous video

```
#EXTM3U
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-ALLOW-CACHE:YES
#EXTINF:2.000000
0000_00000.ts
#EXTINF:2.000000
0000_00001.ts
#EXTINF:2.000000
0000_00002.ts
#EXTINF:2.000000
0000_00003.ts
#EXTINF:2.000000
0000_00004.ts
#EXTINF:2.000000
0000_00005.ts
#EXTINF:2.000000
0000_00006.ts
#EXTINF:2.000000
0000_00007.ts
#EXT-X-ENDLIST
```

1. VOD
2. Segment duration
3. Segment URI
4. End of list

- **Hudl Replay only uses Media Playlists. We do not use Master Playlists**

Hudl Replay in action

- Unfortunately, not allowed to show content from the World Cup
- If watched highlights, Mounted on a tripod

- Normally assistant coaches
- Connected via ethernet adaptor
- Reviewing the game
- When they want to show something

- Unplug it
- Walk to the head coach or the player they want to show the video to
- Locker room
- **TL;DR It is important that the app shows the video even when the iPad is disconnected from the network**

Our problem

- I’m going to demo a very common workflow of our app
- I’m using a recording of Youtube but imagine this is a live video
- The same problem happens in in all HLS clients

- Assistant coach Spanish National team
- World Cup final. Another day in the office
- The game is 0-0 despite many chances
- Spoiler alert: Your team is about to score
- You celebrate because you’re human
- How can you help to keep the score?
- Think about a play that happened 40 minutes ago when Robben almost score
- You want to show the video to the Head Coach so the team can defend better
- Pick up the iPad, disconnect it from the network, airplane mode
- Walk towards the head coach to show it

- Scrub back
- Perma load look like a fool. No video
- Player may even stall and end up with a black screen

- Just to be clear, I’m not saying that Youtube or any other HLS client app is bad. They’re great apps and they comply 100% with the HLS protocol
- The reason why that happens is because that use case is not supported by HLS

Why does that happen?

1. Return requested segments in a timely fashion
2. Live video ⇒ Playlist file must be updated frequently

- Like any server
- Live video, new segments, that is, new video should be continuously created and therefore the playlist updated

Content always available

- All benefits from HLS
- Extend HLS protocol to ensure at least previously watched video is available

What didn't work

01 Remote server only
02 Offline playback
03 Live offline playback

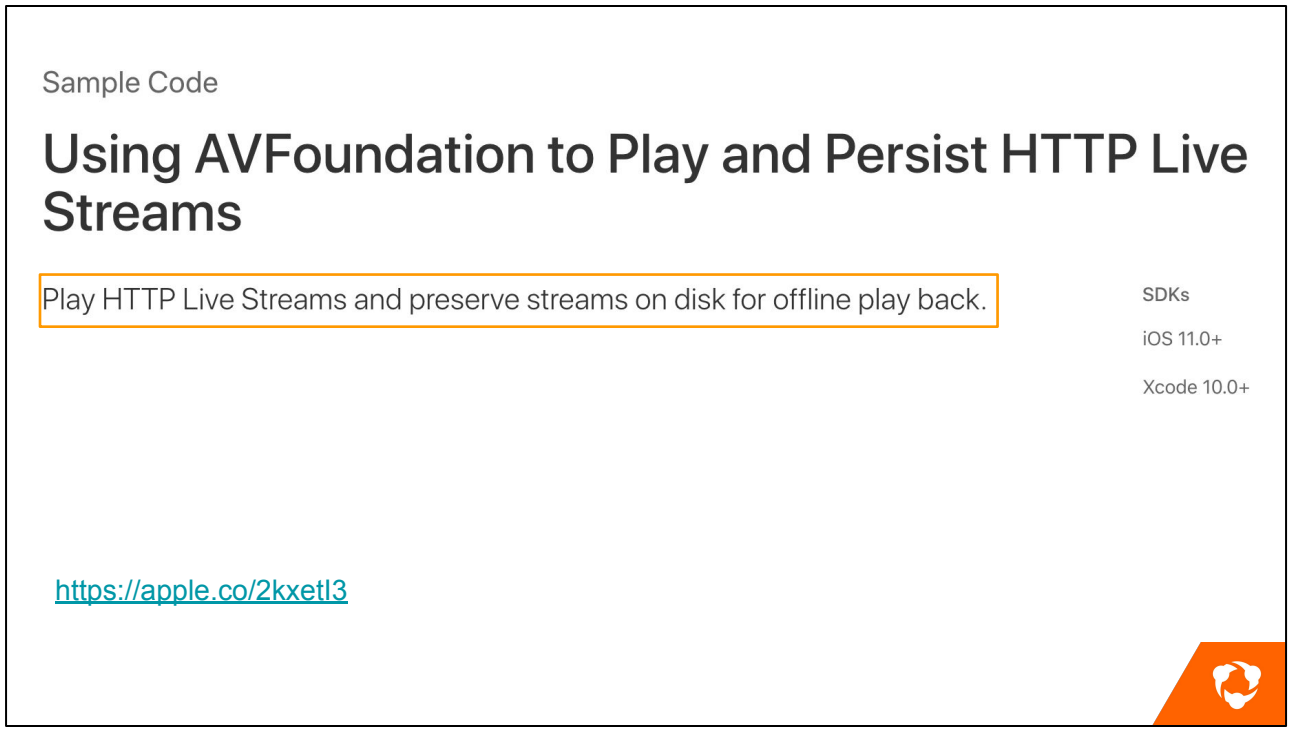
1. Remote server only

The first thing we tried and this is what lead us to check other apps was to just play the remote HLS stream

```swift
import UIKit
import AVFoundation
import AVKit

class ViewController: AVPlayerViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        let urlString = "https://mnmedias.api.telequebec.tv/m3u8/29880.m3u8"
        let url = URL(string: urlString)!

        let asset = AVURLAsset(url: url)
        let playerItem = AVPlayerItem(asset: asset)
        let player = AVPlayer(playerItem: playerItem)
        self.player = player
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        player?.play()
    }
}

```

- This is all the code you need to play an HLS playlist
- You get your URL with your playlist.
- Wrap that into an AVURLAsset
- Wrap the asset into an AVPlayerItem
- Create a player by passing that item
- Call play() and you’re good to go
- This code will launch an AVPlayerViewController playing the playlist at the specified URL
- It doesn’t work. It has the same problem as the Youtube app.

2. Offline playback



- It turns out that within Apple's documentation there's this sample code that also comes with great documentation.
- It says: "Play HTTP Live Streams and preserve streams on disk for offline play back."
- Er… hello? Cheers Apple, that is exactly what we needed
- It plays the live stream and when the iPad is offline we played the stream from disk

```swift
func setupAssetDownload(url: URL) -> AVURLAsset {
    // Create new background session configuration.
    let configuration = URLSessionConfiguration.background(withIdentifier: "Replay-test")

    // Create a new AVAssetDownloadURLSession with background configuration, delegate, and queue
    let downloadSession = AVAssetDownloadURLSession(configuration: configuration,
                                                    assetDownloadDelegate: self,
                                                    delegateQueue: OperationQueue.main)

    let asset = AVURLAsset(url: url)

    // Create new AVAssetDownloadTask for the desired asset
    let downloadTask = downloadSession.makeAssetDownloadTask(asset: asset,
                                                             assetTitle: "Demo",
                                                             assetArtworkData: nil,
                                                             options: nil)
    // Start task and begin download
    downloadTask?.resume()

    return downloadTask!.urlAsset
}
```

- On top of that the code is super simple.
- The main thing that changes wrt to the previous example is how to determine the **AVURLAsset** passed to the player.
- Before it was just a URL. Here we wrap the AVURLAsset in a AVAssetDownloadTask so it can be downloaded while played.

```
86      func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: Error?) {
87          guard let error = error else {
88              print("The download failed with unknown error")
89              return
90          }
91          print(error.localizedDescription)                                   Thread 1: breakpoint 5.1
92      }
```

OfflineDownloader 〉 Thread 1 〉 0 OfflineDownloader.urlSession(_:task:didCompleteWithError:)

```
Printing description of error:
▿ Optional<Error>
  - some : Error Domain=AVFoundationErrorDomain Code=-11800 "The operation could not be completed"
      UserInfo={NSLocalizedDescription=The operation could not be completed,
      _NSURLErrorFailingURLSessionTaskErrorKey=BackgroundAVAssetDownloadTask <CA4C2515-DDE8-4166-844D-FAE898D611DF>.<1>,
      _NSURLErrorRelatedURLSessionTaskErrorKey=(
    "BackgroundAVAssetDownloadTask <CA4C2515-DDE8-4166-844D-FAE898D611DF>.<1>"
), NSLocalizedFailureReason=An unknown error occurred (-16655)}
```

- Yout get this error

“We’ll think about it”

Apple AVFoundation Engineer

Sample Code

Using AVFoundation to Play and Persist HTTP Live Streams

Play HTTP Live Streams and preserve streams on disk for offline play back.

SDKs

iOS 11.0+

Xcode 10.0+

NOTE

- You cannot save a live HLS stream while it is in progress. If you try to save a live HLS stream, the system throws an exception. Only Video On Demand (VOD) streams support offline playback.

- **Next thing I know is that they updated the documentation page to include this.**

3. Live offline playback

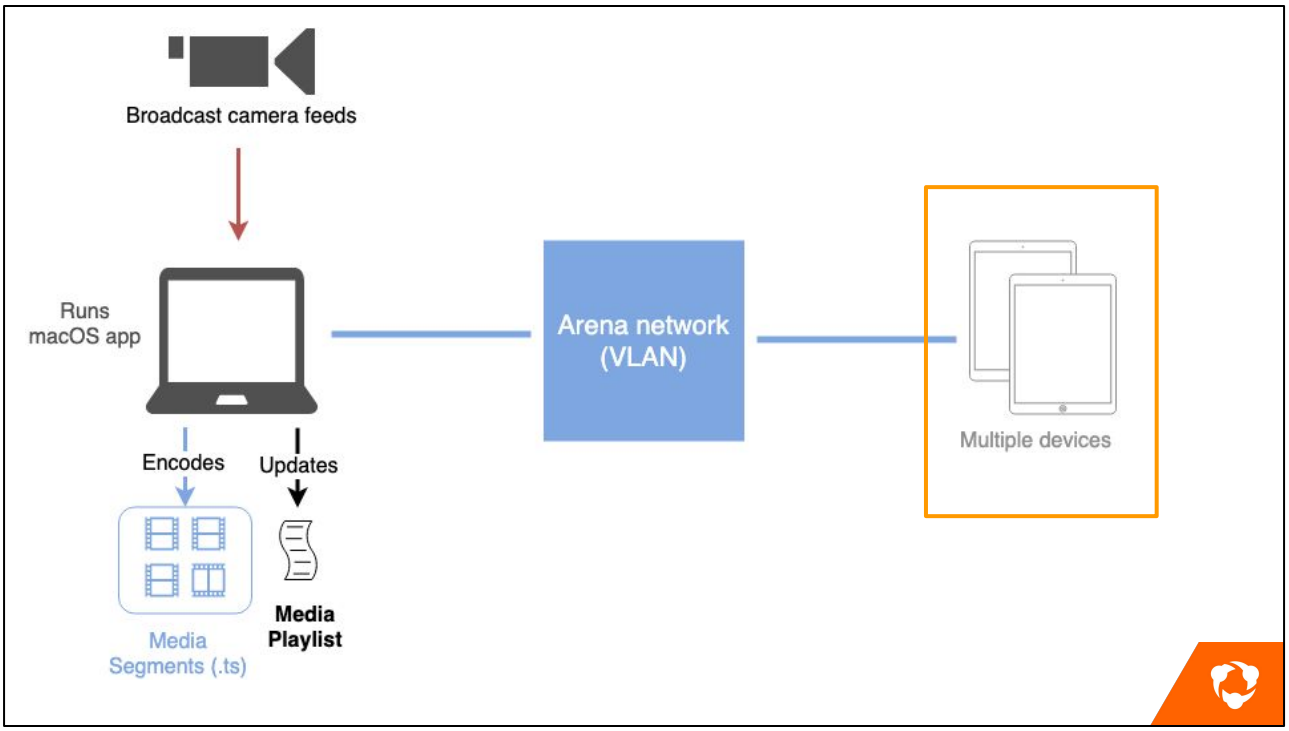
- High complexity
- Suboptimal scrubbing experience
- Getting the worst of both worlds

- In this approach we tried to use the remote server when the iPad was connected to the network
- Local offline playback when it wasn’t

What did work: Custom offline playback

Always play local content

- The key was to abstract the player about the existence of a remote server



- Nothing changes on the server side
  - It continues to capture video, encode media segment and update the media playlist
- Hudl Replay is an iOS app so we did everything. We did not change anything on the server when implementing Hudl Replay.
- All the logic we added was in the iOS app

Why does that happen?

1. Return requested segments in a timely fashion
2. Live video ⇒ Playlist file must be updated frequently

Problems to solve

1. How to download and store media segments
2. How to play local media segments (.ts files)
3. How to ensure the player never perma loads

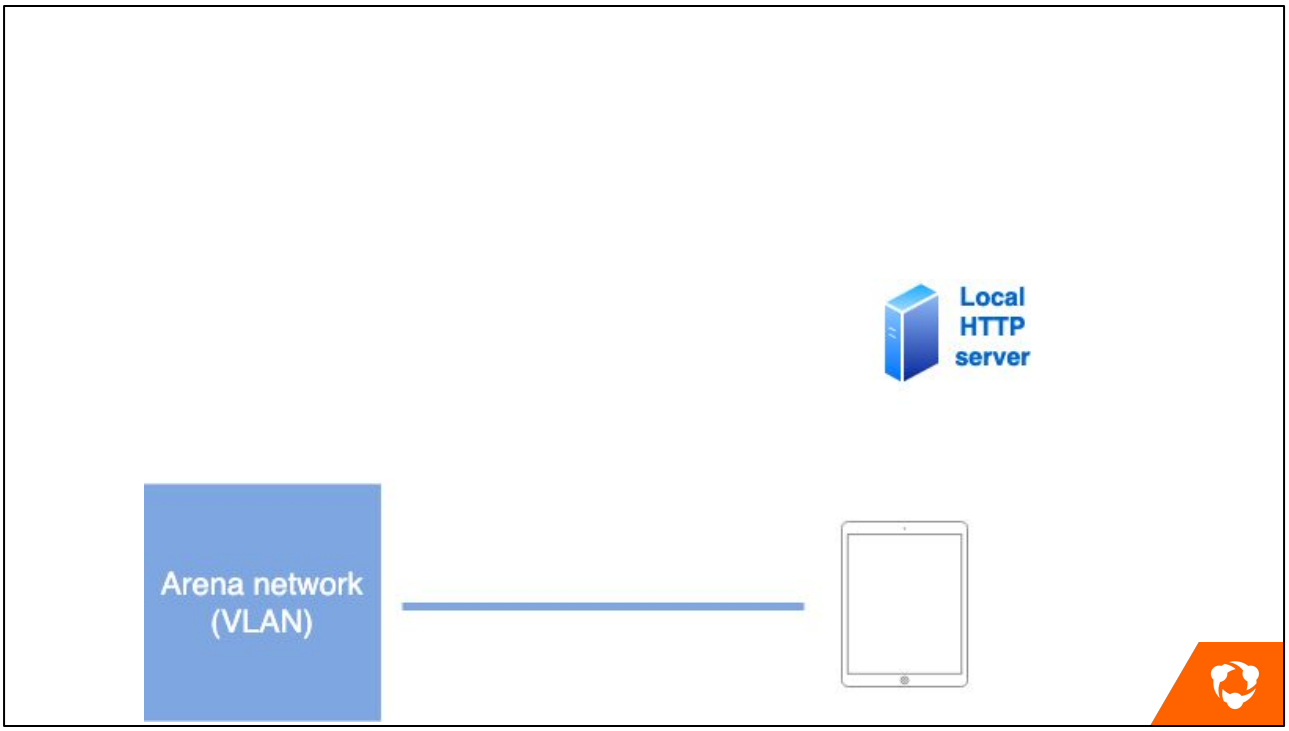
- We want to download and store media segments so they can be played locally. That way we do not depend on a network connection to play them
- That come with the problem of playing “ts” files locally. Ts is the format we use, and most of the people use, for our media segments, the small chunks of video, and iOS cannot play those files directly
- The real problem to solve is that the player never perma loads and content is always available to the user

Play .ts files

- iOS can’t play ts files directly.

```
19      override func viewDidLoad() {
20          super.viewDidLoad()
21          let path = Bundle.main.path(forResource: "video", ofType:"ts")!
22          let url = URL(fileURLWithPath: path)
23
24          let urlAsset = AVURLAsset(url: url)
25          let playerItem = AVPlayerItem(asset: urlAsset)
26          let player = AVPlayer(playerItem: playerItem)
27          self.player = player
28      }
29
30      override func viewDidAppear(_ animated: Bool) {
31          super.viewDidAppear(animated)
32
33          self.view.addSubview(customView)
34
35          self.player?.play()
36      }
```

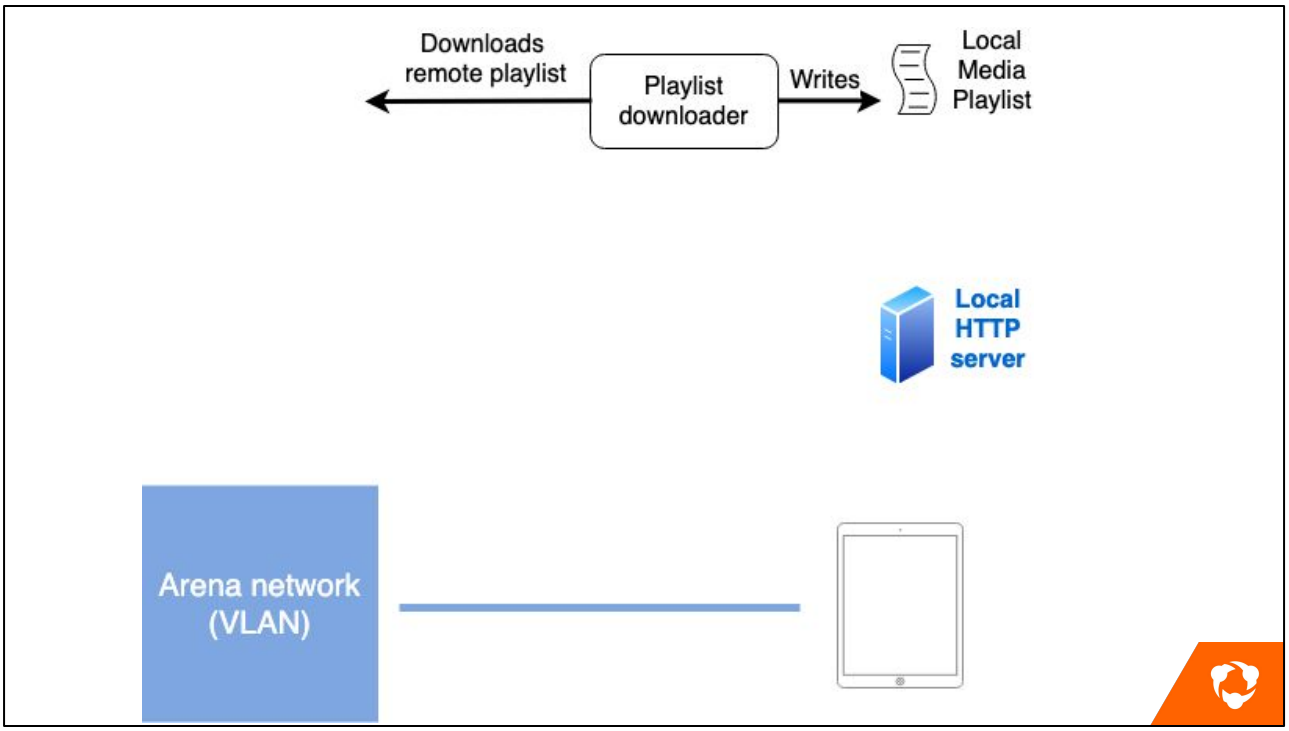
- This doesn’t work.
- The player won’t play anything
- But, iOS can play ts files when they’re been served over an HLS server



So that's what we did. The application has a local http server that acts as the HLS server from the player's perspective

Local http server

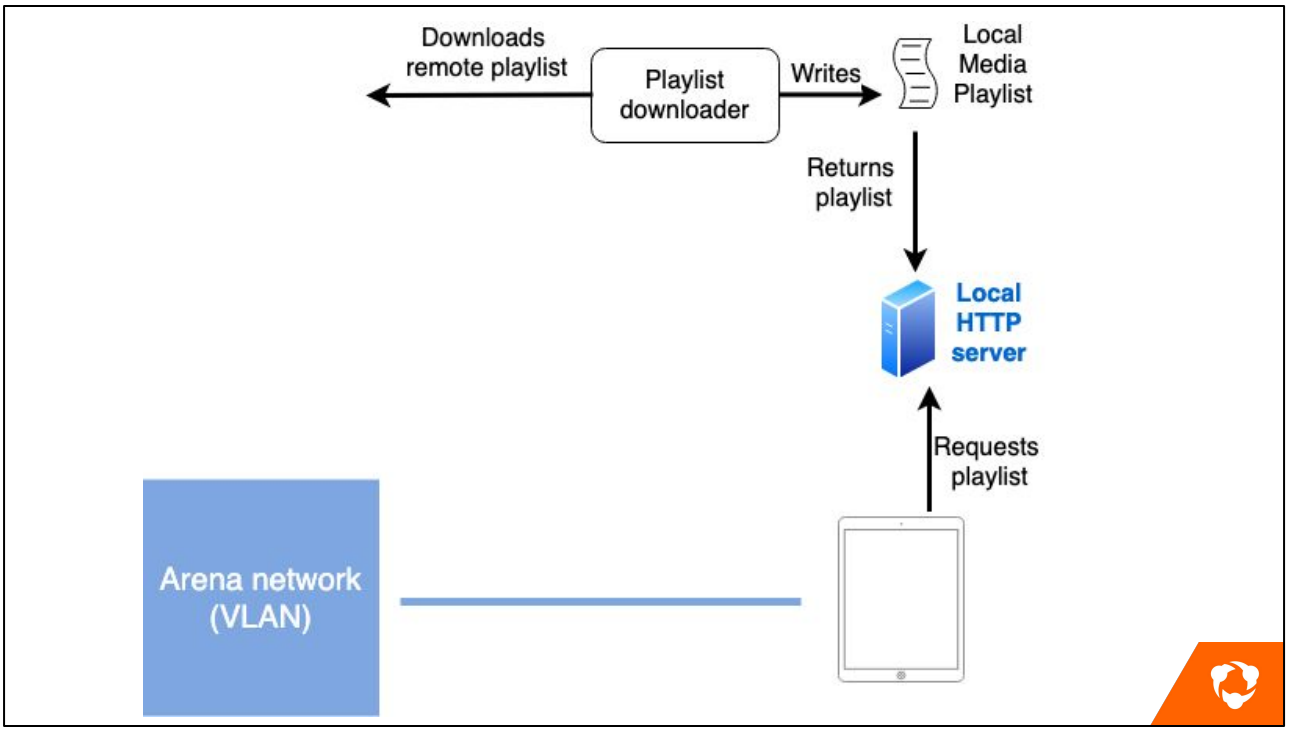
- The key is that from the player's perspective, everything runs locally.
- Both the playlist file and the media segments are served from the local server
- The local server then proxies the player's requests to another services



- The first request the player is going to do is the playlist file
- So, we need to ensure that the playlist file is ready by the time the players requests it
- What we do is to download it from the remote server and write it in the local filesystem so it is accessible for the local http server
- Because it is live video, the playlist will be updated as the server encodes more media segments.
- So we need to continuously download the updates the remote server is producing

```swift
override func viewDidLoad() {
    super.viewDidLoad()
    let urlString = "https://localhost:8080/local-playlist.m3u8"
    let url = URL(string: urlString)!

    let urlAsset = AVURLAsset(url: url)
    let playerItem = AVPlayerItem(asset: urlAsset)
    let player = AVPlayer(playerItem: playerItem)
    self.player = player
}

override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)

    self.view.addSubview(customView)

    self.player?.play()
}
```

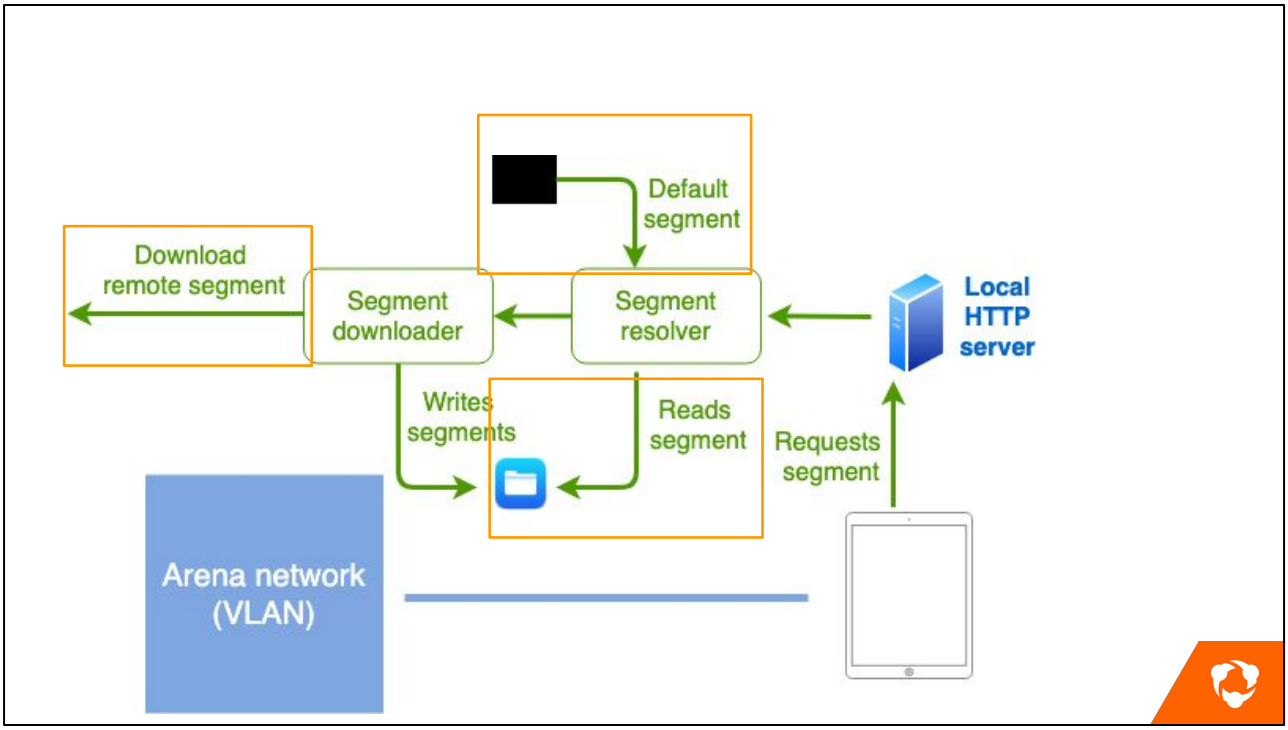
- The way we start the player looks like something like this.
- We pass a URL pointing to localhost



- When the player starts, it requests the playlist
- When the device asks for the playlist, the local http server, not the remote one returns it

Download media segments

- Next thing is going to do is request the media segments, that is, the video

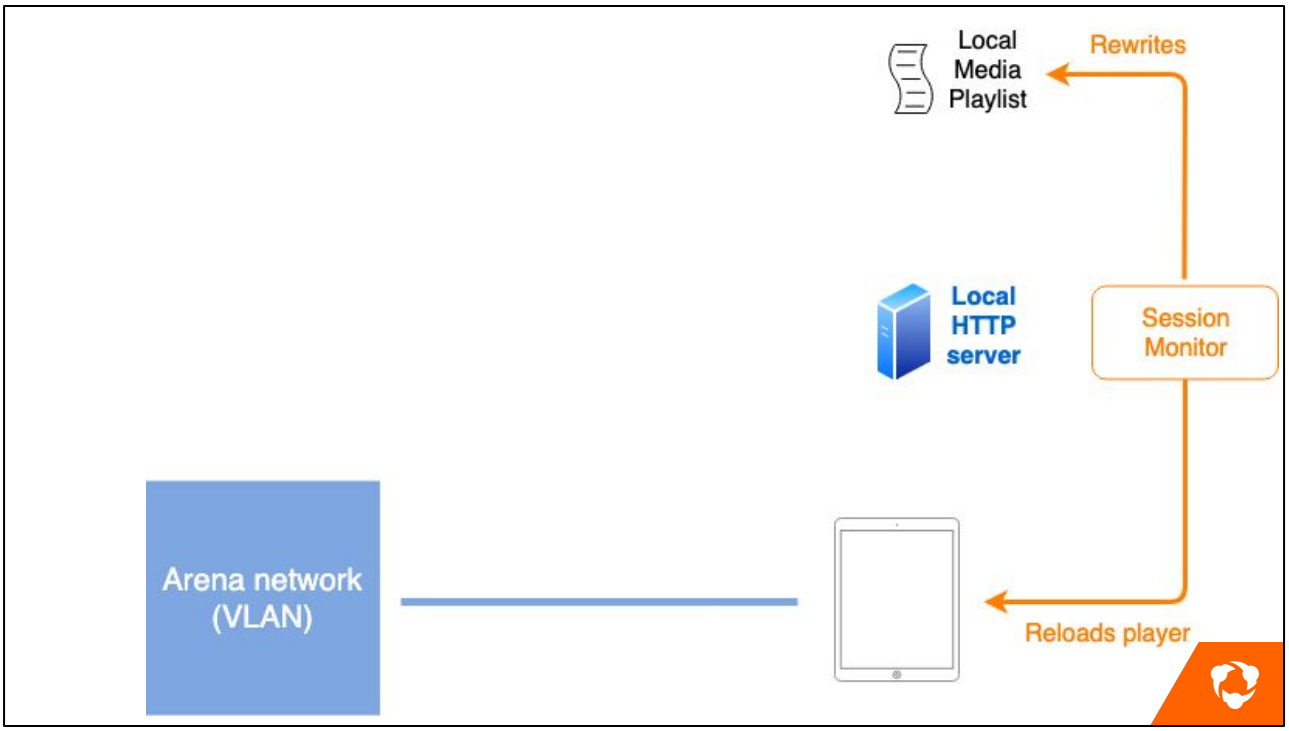

Default segment
Download remote segment
Segment downloader
Segment resolver
Local HTTP server
Writes segments
Reads segment
Requests segment
Arena network (VLAN)

Why does that happen?

1. Return requested segments in a timely fashion ✔️
2. Live video ⇒ Playlist file must be updated frequently

No permaloading

- We need to ensure that for live video, playlist file updates frequently



- Session Monitor is basically a network monitor.
- When it detects the local playlist cannot be updated frequently, it rewrites it and reloads the player so changes take effect
- Let’s look in detail at how that rewrite looks like

```
#EXTM3U
#EXT-X-PLAYLIST-TYPE:EVENT
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-ALLOW-CACHE:YES
#EXTINF:2.000000
0000_00000.ts
#EXTINF:2.000000
0000_00001.ts
#EXTINF:2.000000
0000_00002.ts
#EXTINF:2.000000
0000_00003.ts
#EXTINF:2.000000
0000_00004.ts
#EXTINF:2.000000
0000_00005.ts
…
…
...
```

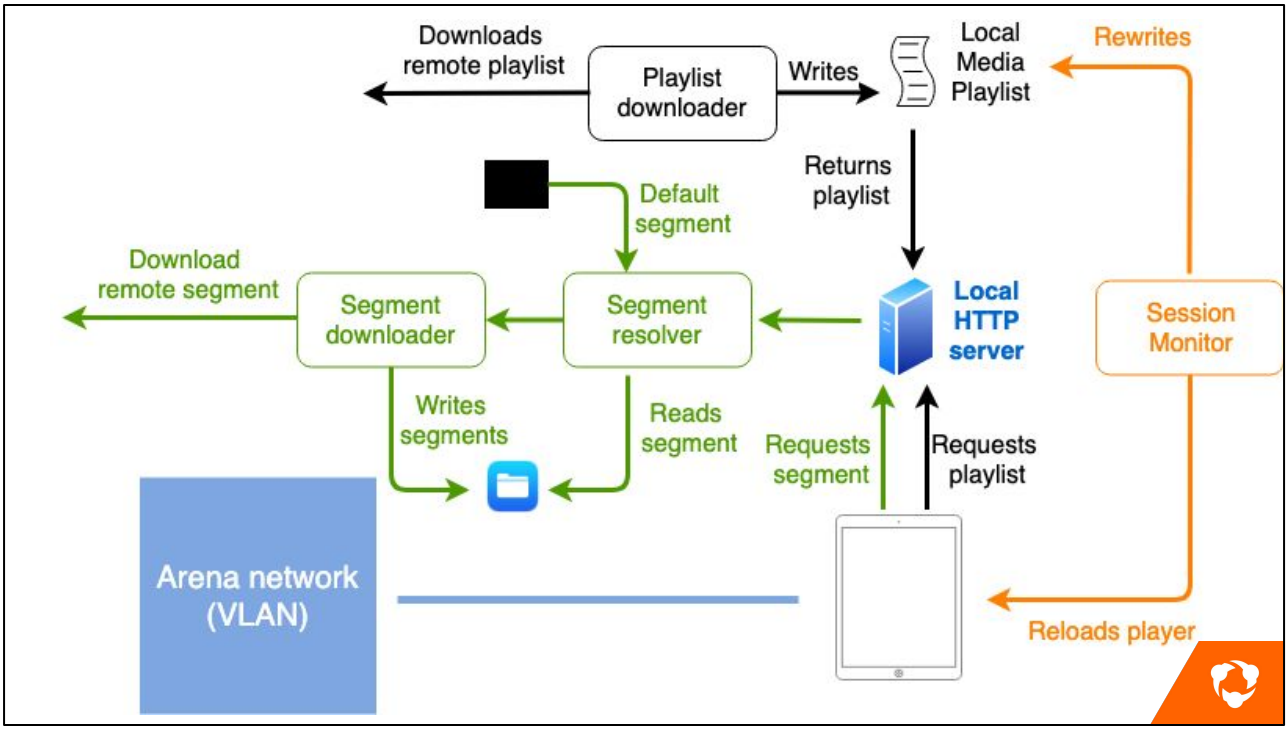
- The problem is that for live video, or playlists of Event type, the playlist file needs to be updated frequently.
- If the iPad is disconnected from the network we cannot update the playlist frequently.
- So, the only thing we can do is to change the playlist type

```
#EXTM3U
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-ALLOW-CACHE:YES
#EXTINF:2.000000
0000_00000.ts
#EXTINF:2.000000
0000_00001.ts
#EXTINF:2.000000
0000_00002.ts
#EXTINF:2.000000
0000_00003.ts
#EXTINF:2.000000
0000_00004.ts
#EXTINF:2.000000
0000_00005.ts
…
…
…
#EXT-X-ENDLIST
```

- We change the playlist type to VOD or Video On Demand
- That is done by changing the playlist type value and adding an ENDLIST tag at the very end of the file
- That doesn't require the playlist to be updated frequently because it's a stream with a concrete start and end
- By the time you start playing it, your player knows when it starts, when it ends, the duration, everything.
- **Session monitor does the inverse operation**

Why does that happen?

1. Return requested segments in a timely fashion ✔️
2. ~~Live video ⇒ Playlist file must be updated frequently~~


Downloads
remote playlist
Playlist
downloader
Writes
Local
Media
Playlist
Rewrites
Returns
playlist
Default
segment
Download
remote segment
Segment
downloader
Segment
resolver
Local
HTTP
server
Session
Monitor
Writes
segments
Reads
segment
Requests
segment
Requests
playlist
Arena network
(VLAN)
Reloads player

- So now if you, as the assistant coach of Spain, want to show the Head Coach the play where Robben almost scored so the team can defend it better, you can.

Juanjo Ramos

@JuanjoRamos82

Thank you!

Q&A