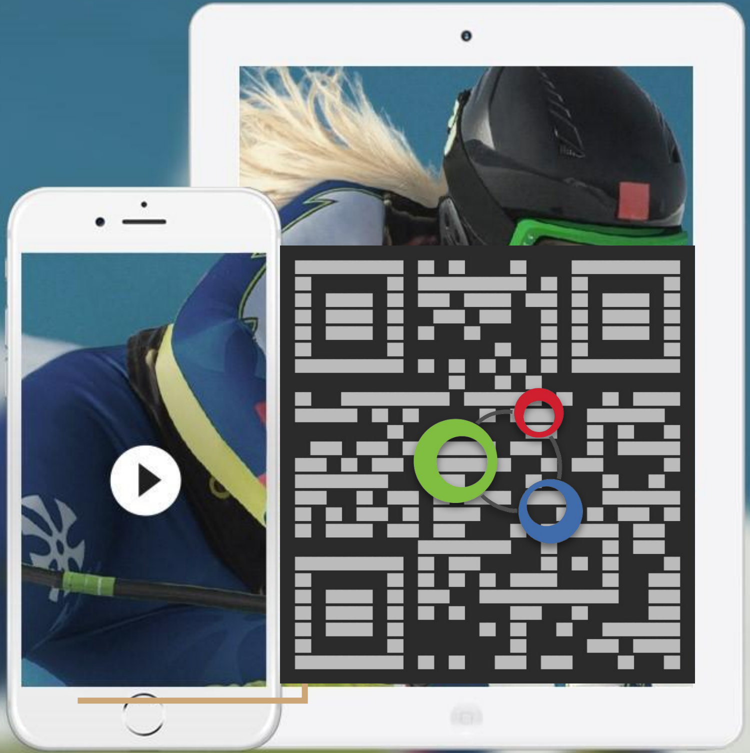
# Low Latency Live Streaming Apple LLHLS / CMAF

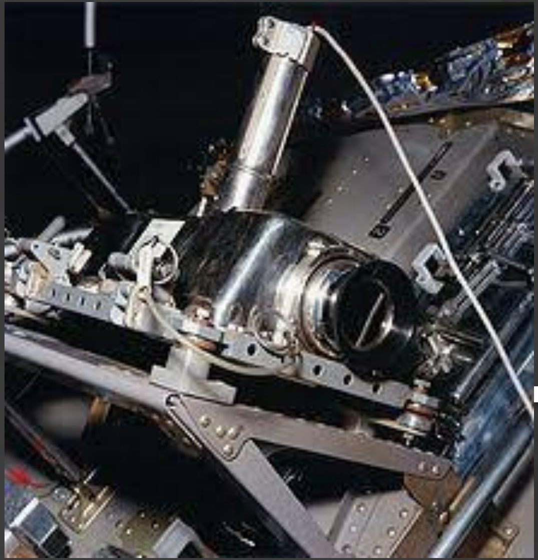
Kevin Staunton-Lambert
Solutions Architect
R&D (July 2019)
@kevleyski



switchmedia
Effortless Video
www.switch.tv

# what we mean by live latency… July, 1969

Delay between getting light into glass of the camera onto the glass of the viewers



**3s time delay**

(10fps@300)

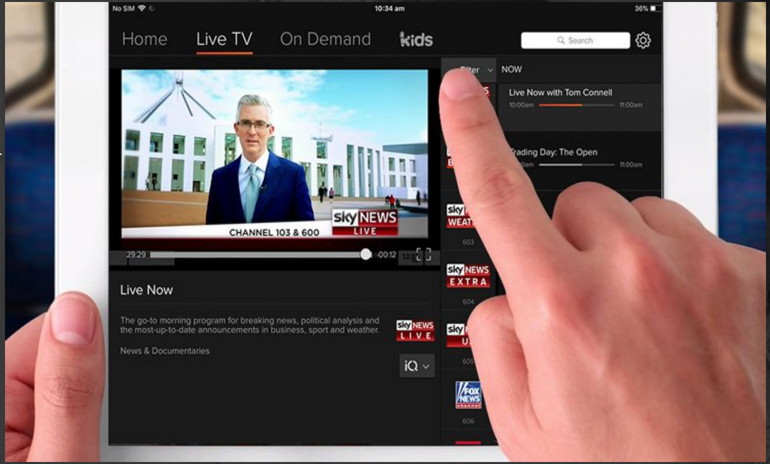

#JusticeForHoneysuckle

# **what** we mean by live latency… July, **2019**

switchmedia
switch.tv

Delay between getting light into glass of the camera onto the glass of the viewers


**42s time delay**
(25fps@1080)




**#50YearsOfProgress**

switchmedia
switch.tv

# why do we care?

Social Media/eSports - shared emotion gets lost+kinda weird

Bizzare Hangouts/Adult interactive entertainment “VR issues”

Live Sports spoilers, hearing a ‘no try’ given that I’m still mentally helping the ref to decide upon - live score alerts

Questionable fairness in interactive betting/online auctions

News/Webinar - pretty awkward Q&A audience interactions

… and the problems get worse with 4K live streaming

# **who** are the usual suspects…

switchmedia
switch.tv

Camera live output - raw *vs* compressed video (e.g. JVC 1.6s)

Outside broadcast uplinks - satellite hops *vs* optic fibre

Adaptive Bitrate transcodes - slow as your best rendition

Media file packagers, CDN file upload and propagation

Decoders (player apps, MSE SW vs HW, DRM CENC/EME)

External factors like SSAI Ad Injectors, live profanity buffers

# **how** can we fix it?

switchmedia
switch.tv

## Throw **$** at the problem…

- *Buy* better cameras - reduce compression/buffer time
- *Buy* faster hardware encoder GPU/FPGA + SSD storage
- *Buy* more live encoders (transcoders) + hw encryption
- *Pay* for better uplink/downlink/CDN bandwidth
- *Pay* to support the best solutions for *every* player type

# **how** else can we reduce live latency?

Some emerging technologies that help make things better...

- Improving codecs - better balance between hardware encoding/decoding - e.g. AV1 rav1e/dav1d
- More efficient media packaging, CMAF+DASH, LHLS
- Elastic cloud compute - scaling up/down to demand
- Better transport options, reusing connections, FEC
- Various proprietary solutions

## **what** can you do to reduce your live latency…

switchmedia
switch.tv

First, a quick recap on the evolution and some common techniques which contribute to the emerging technologies to reduce streaming media latency today…

# RTMP Real Time Messaging Protocol

switchmedia
switch.tv

A great solution where latency is largely coming from the camera encoder to **push** compressed video to a distribution

But RTMP is (almost always) P2P Unicast TCP/IP, so...

Need more relay ports? get more/bigger servers!

Want to load balance/bypass firewalls? Maybe RTMPT (HTTP)

Ok for eSports/live betting with small actual paying punters meanwhile those just watching along in higher latency (if legally permitted) … scaling up to Grand Final Footy? Nope!

… also, **Adobe Flash/FLV** player in your app ain't gonna fly

switchmedia
switch.tv

# **UDP** Near zero latency (1980’s fire and pray)

Still an excellent way to transfer data over packet networks, NICs can be optimised for media broadcast via **multicast** IP, plus you get better control over fifo_size/pkt_size etc

Typically you push to destination IP or route via multicast IP

But this method is particularly **error prone** - expect media glitches due to packet loss and packet reordering over net

UDP does not scale well over Internet distribution, at least not cheaply via regular HTTP CDNs

… and hey, you’ll need an app for all this too

# **WebSockets** raw media pipes via TCP

switchmedia
switch.tv

Can achieve really excellent Ultra low latency (<200ms), sending raw live video into a WebSocket *directly* into web browser - TCP covers your errors, but can (and does) buffer

Renders video using hardware (MSE) via <video> tag

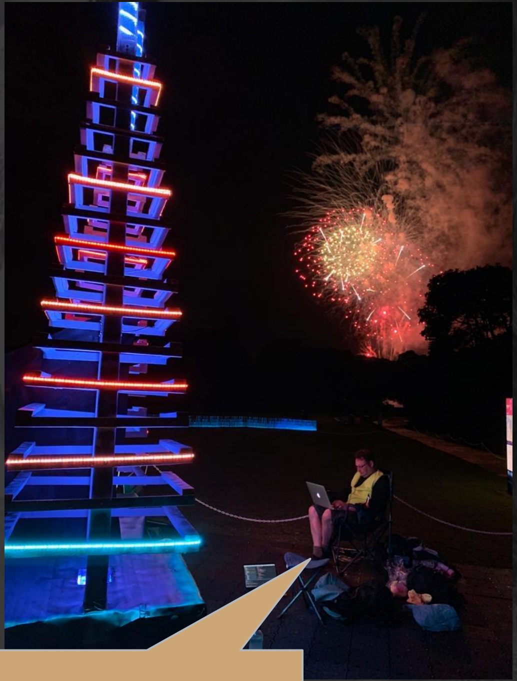
But also HTML5 <canvas> which means WebGL 3D surfaces and cool GLES3 fragment shader effects/overlays etc

Also easy to integrate, WebSockets are well supported in all common programming languages too

# **WebSockets** demo - live at Vivid Sydney 2019!

switchmedia
switch.tv

Anyone see these odd things in the Royal Botanical Gardens?

I’m clearly too busy

# **WebSockets** demo - explained

switchmedia
switch.tv

The SwitchMedia Light Teleportal exchanges video over WebSockets into WebGL
But latency was just a bit *too* good for folks to see themselves in the one directional queue (40,000 people on a warm night)
So, we slowed it down with some software encoding - world's first high latency encoding over ultra low latency maybe?!
Ultimately, basic client/server model doesn't scale too well
Ties up TCP port per client, also no feasible CDN options meaning a very busy and expensive origin server and proxies

# **WebRTC** (Web Real-Time Communication)



W3C (standard) WebRTC, similar in nature to WebSockets so ultra low latency, but many caveats to get it/keep it going

To handle external delivery via Internet it needs proxies to work (ICE/TURN/STUN) to establish/maintain connectivity

Depending on connection can be UDP or TCP, fun stuff like NAT traversal and SDP session management implies some operational headaches and so expect ongoing support costs

# QUIC (Quick UDP Internet Connections)



Google's UDP based delivery (e.g. YouTube Live)

Utilises SPDY (now HTTP/2) multiplexed connections which allows sharing a pipeline avoiding overheads with opening/reopening sockets and so reduces TCP handshake latency

Elephant in the room, led to a browser fallout war, e.g. no support for QUIC scheduled for Mozilla Firefox or Safari

But, Microsoft says it's currently working on it for EDGE

# **SRT** (Secure Reliable Transport via UDP)



UDP with end-to-end (AES/DVB) encryption + a big bonus…

Includes forward error correction (FEC) which is utilised in traditional digital broadcast. Some additional data is added so the receiver can auto patch up common errors (NAK packets). Algorithm also automatically retries to refetch segments again if time frame allows incase FEC above fails

… open source protocol but no takers into the MSE

(more an FFmpeg/Gstreamer/VLC) RedBee claim **3.5s**

*(* kudos to Haivision and Wowza for open source )*

## RIST (Reliable Internet Stream Transport via UDP)

switchmedia
switch.tv

RTP low latency solution from the Video Services Forum

Encoder (sender) and Decoder (receiver) send packets via relays message server, which resends on not acknowledged packets (NACT/RTCP) based on number of retries set by the decoder - this effectively self recovers the bitstream from typical UDP packet loss

# **IPFS** (InterPlanetary File System)

IPFS is peer-to-peer and works by converting files into blocks

These are then hash referenced around the web (ala torrent)

For media stream this works reasonably well because IPFS gateway supports byte-range requests

4K VoD example, but tech can be applied to live:

https://streams.switchmedia.asia/streama/HV402/.mp4

switchmedia
switch.tv

# **HLS** Legacy (HTTP Live Streaming)

switchmedia
switch.tv

Adaptive Segment file based delivery via TCP/HTTP1.1

- MPEG2 Transport Stream (TS) files - often muxed sometimes separated audio e.g. separate AAC files
- Recommended segment length 6s (used to be 10s)
- Players request 3 (possibly AES encrypted) segments and decode them before playback can begin
- Overall latency typically more than **30 secs** ... so not great

# **HLS** Legacy (limitations around reduced latency)

switchmedia
switch.tv

Shorter segments files *do* help but with side effects...

More files uploaded on the CDN = additional charges
Greater overheads as less compression + TCP connections
Less data + transport delays leads to more buffering which means longer playlists which undoes some of your good work
SwitchMedia tests give lowest watchable latency around **9s**

*Apple's player seems to be most robust to errors but is closed source - hls.js ok but is MSE browser only, ExoPlayer not great, others also fail often*

## MPEG-DASH (Dynamic Adaptive Streaming over HTTP)

switchmedia
switch.tv

Provides better flexibility over HLS where live streams are now organised into AdaptationSets, also we separate segment timing info away from the media, add in presentation availability windows and can handle multi-DRM and importantly it's standards based (HLS remains an RFC) mediaRanges supported since conception allowing more efficient byte-range transfer over HTTP, whilst HLS more recently has this feature, it's a clumsier after thought

## **MPEG-DASH** (Dynamic Adaptive Streaming over HTTP)

switchmedia
switch.tv

Other benefits over HLS is supporting multiple linear live events with HLS discontinuities is a royal pain in the bum, DASH to the rescue with multi-periods
(and just maybe dynamic x-links will *actually* be supported in all players some day)
Also a large difference to early HLS is use of fMP4/ISOBMFF where MPEG-TS has no native MSE support in the browser*
*(* actually DASH supports TS but seldom implemented outside of HbbTV/DVB-DASH)*

# **ISO BMFF** Base Media File Format (MP4)

switchmedia
switch.tv

Standardised what is commonly known as MP4 container files

All data put in well defined ‘boxes’ (aka QuickTime ‘atoms’)

- Movie Box (moov) and Movie Fragment Box (moof)
- Media Data Boxes (mdat)
- ftyp, styp etc...

Separation of media initialisation *vs* segments removes *much duplication* of metadata about the media - e.g. *moov* box contains codec info - every TS segments duplicates such data

# fMP4 Fragmented ISOBMFF

switchmedia
switch.tv

Large ISOBMFF files can be further partitioned into fragment files which are then easy - and so quicker/more efficient to

- generate (add new boxes whilst old ones being read)
- manipulate (easy to filter, skip over and go back to)
- serve (easy to pass boxes around via byte range requests)

Supported by *both* HLS *and* DASH means less bandwidth needed ⇨ faster upload times ⇨ lower CDN and storage costs and a slightly better overall bitstream compression

# h2 HTTP/2 Sending less, sharing more and PUSH



HTTP/1.1 limits 6 open connections, so juggling often needed

h2 removes this limit and also brings in HPACK which compresses the headers by indexing them (huffman) which reduces duplicates, combined with gzip of manifests reduces over bandwidth with streaming media
Dependency and waiting
When sending files, piggybacking by pushing other files sharing the same TCP/HTTP connection reduces overhead

switchmedia
switch.tv

# new low-latency streaming technologies

Some emerging technologies

- Apple Low-Latency HLS (Pantos update June 2019)
- Community LHLS
- Ultra Low Latency CMAF (ULL-CMAF)
- Peer-to-Peer WebRTC
- Broadcast WebRTC
- HTTP/3
- Some other proprietary solutions

# **WebRTC** Peer2Peer and Broadcast WebRTC

switchmedia
switch.tv

There exist some interesting scaling opportunities to be had via peer sharing where the origin shares content into a browser and it becomes the new origin - others piggyback on that player instance and in turn become origin nodes.
Broadcast WebRTC is coming but my guess is never likely to be ‘super bowl’ event ready either, presumably the costs saved in scalable CDN distribution vs support costs to relay
SwitchMedia tests show WebRTC is prone to errors (jitter)

# Salsify Codec/network integration

switchmedia
switch.tv

Works by integrating the video codec with the network transport protocol allowing it to respond quickly to changing network conditions and avoid provoking packet drops and queueing delays.
Optimizes the compressed length and transmission time of each frame, based on a current estimate of the network's capacity vs frame rate or bit rate. … but ultimately needs a codec or a way into the MSE

# **CMAF** Common Media Application Format

switchmedia
switch.tv

Standard from *Apple*, *Microsoft* and *Akamai* that enforces fMP4 and encoding profiles used across **HLS** and **DASH**

Media structure becomes consistent, video must fit within a smaller profiles/level set of AV1/ HEVC (H.265)/AVC (H.264)
Audio is *never* multiplexed content (always separate files) and must be AAC-LC/HE-AAC at given rates
Subtitles must be TTML/WebVTT
Encryption (whilst optional) must be AES ctr … and so on

# ULL-CMAF Low Latency Chunking

switchmedia
switch.tv

Here we also **chunk** the fMP4 fragments further by adding meta-data about media byte ranges into the MPD or M3U8

Byte range info allows players to take smaller playable bites at the segment file still being generated rather than waiting then chewing on the whole segment before starting playback

Chunked HTTP transfer reduces overheads whilst allowing the encoder to append new ‘boxes’ to same segment file

# **Apple LLHLS** Low-Latency HLS



Apple announced Low-Latency HLS last month at WWDC19

Version 9 of HLS brings in some new tags but remains **backwards compatible** for all earlier players - in that the new tags will be ignored (as per specification)

It works by adding partial media segment files into the mix, this can be CMAF fMP4, but Apple continues to supports TS files in the form of partial MPEG2 TS transport segments too

# Apple LLHLS Partial Segment Tags #EXT-X-PART

switchmedia
switch.tv

## Partial TS segments are described in media playlist like this

INDEPENDENT=YES tells the player part 0 of segment **787** has an IDR (starts with an independent I-Frame)

```
#	0,INDEPENDENT=YES,URI="filePart787.0.ts"
#	URI="filePart787.1.ts"
.
#EXT-X-PART:DURATION=0.20000,INDEPENDENT=YES,URI="filePart787.15.ts"
#EXT-X-PART:DURATION=0	URI="filePart787.16.ts"
#EXT-	="filePart787.17.ts"
#EXT-	="filePart787.18.ts"
#EXT-X-PART:DURATION=0.20000,INDEPENDENT=YES,URI="filePart787.19.ts"
#EXT-X-PRELOAD-HINT:TYPE=PART,URI="filePart787.20.mp4"
#EXTINF:3.96667,
fileSequence787.ts
```

part 15 of segment **787** also starts with an independent I-Frame

Note, remains backwards compatible by also always including the entire segment **787** in the usual HLS way

# **Apple LLHLS** Partial Segment TS Files

switchmedia
switch.tv

There's nothing particularly special about a partial TS, literally split on TS 188 packet syncbyte and can be split outside GOP

`fileSequence787.ts` *(is same as)*

```
cat filePart787.0.ts
    filePart787.1.ts
     ...
    filePart787.19.ts > fileSequence787.ts
```

First part always 0 of arbitrary 20 parts, requesting part 21 = part 0 of next seg (according to spec, Apple's LHLS demo tools are buggy ;-)

# Apple LLHLS HTTP/2 PUSH ?_HLS_push=1

switchmedia
switch.tv

HTTP/2 is now a requirement to push the segments file along with the playlist as when it becomes ready
Piggybacking the segments this way significantly reduces the overhead of establishing repeated TLS / TCP sessions

Playlists are also always compressed under HTTP/2 - streams with a long DVR window (large review scrub back buffer) this compression also reduces latency to download it

# Apple LLHLS Preload #EXT-X-PRELOAD-HINT

switchmedia
switch.tv

`#EXT-X-PRELOAD-HINT`

Lets the server tell the player client the upcoming (parital) segment that is not yet actually available

This avoids the requirement for HTTP/2 push/blocking mechanism having to wait for the segment to get ready where the client can request it until it's actually available

# Apple LLHLS TLS 1.3

switchmedia
switch.tv

Transport Layer Security 1.3 is also a requirement, ultimately it has less handshake overheads

TLS false start (tolerates receiving TLS records on the transport connection early, before the protocol has reached the state to process them)
Zero Round Trip Time (0-RTT - resumed connection when certificate has been used before)

# Apple LLHLS #EXT-X-SERVER-CONTROL

switchmedia
switch.tv

```
#EXT-X-SERVER-CONTROL:CAN-BLOCK-RELOAD=YES,CAN-SKIP-UNTIL=24,PART-HOLD-BACK=0.610
```

This tells the player that the server has the following capabilities…

CAN-BLOCK-RELOAD=YES: Mandatory, simply means I have ?_HLS… support

CAN-SKIP-UNTIL=<seconds> I'll give you 24 seconds back on ?_HLS_skip=YES

PART-HOLD-BACK=<seconds>: Indicates the recommended **live edge** time when playing. This must be at least 3 x PART-TARGET - we have 20 parts per 4 second segment, so 0.2 seconds per parts, so *hold player back for* (3 * 0.2) < 0.61 seconds

# Apple LLHLS Delta Updates ?_HLS_skip=YES

switchmedia
switch.tv

```
#EXT-X-SERVER-CONTROL:CAN-BLOCK-RELOAD=YES,CAN-SKIP-UNTIL=24,PART-HOLD-BACK=0.610
```

Playlist optimised by only sending what changed in a given time window - here the server tells the player I'll give you the next 24 seconds from when you next call ?_HLS_skip=YES

Typically delta changes fit in single MTU making it more efficient to load the playlists

Large DVR windows (review buffer) become highly compressed and much faster to parse thus reducing latency

# **Apple LLHLS** Blocking Playlist Reload ?_HLS_msn=

switchmedia
switch.tv

When requesting live media playlist, wait until the first segment is also ready and give me back both at same time (saving additional unnecessary HTTPS/TCP round trips)

```
GET https://lowlatency.switch.tv/lhls/2M/lowLatencyHLS.php?_HLS_msn=23058
```

*...blocking/waiting until filePart23058.x + fileSequence23058 becomes available...*

```
#EXT-X-PART:DURATION=0.20000,URI="filePart23058.0.ts"

#EXT-X-PART:DURATION=0.20000,URI="filePart23058.19.ts"
#EXTINF:3.96667,
fileSequence23058.ts
```

# Apple LLHLS Rendition Reports ?_HLS_report

switchmedia
switch.tv

```
#EXT-X-RENDITION-REPORT
```

Adds metadata to other media renditions to make switching between ABR faster
_HLS_report=<path> points to the Media Playlist of the specified rendition. (either relative to the URI of the Media Playlist request being made, or an absolute path on the same server. Multiple report parameters are allowed for different paths.

# Apple LLHLS tools

switchmedia
switch.tv

## tsrecompressor

"produces and encodes a continuous stream of audio and video, either programmatically (a bip-bop image derived from the system clock) or by capturing video from the system camera and microphone. It encodes the stream at several different bit rates and multicasts them as MPEG-2 Transport Streams to local UDP ports."

## mediastreamsegmenter *(updated)*

"tool listens for its input stream on a local port and packages it for HLS. It writes a single live Media Playlist with its corresponding Media Segments (including Partial Segments in Low-Latency mode). It can also perform segment encryption. It writes its output to the local filesystem or to a WebDAV endpoint."

# **Apple LLHLS** demo (load balanced origin 2.01s)

switchmedia
switch.tv

Example Apple low latency live streaming on tvOS13
For those with iOS13 Beta 2/3 Safari you can also goto

SwitchMedia OpenResty Origin:

https://lowlatency.switch.tv

Local Clock https://lowlatency.switch.tv**/time**

Disclaimer: Roger Pantos' demo of this with Sydney went pretty awry
This is all brand new stuff, when writing these slides low-latency HLS wasn't even available in iOS & iPadOS 13 beta 1, and no Safari doesn't have it either even today

# **Apple LLHLS** demo - Fastly CDN **3.15s**

switchmedia
switch.tv

Seconds tick over exactly on the middle 3 appearing so 30.148 - 27.0 = 3.15s
Fastly CDN:
https://alhls-switchmedia.global.ssl.fastly.net/lhls/master.m3u8

# Apple LLHLS demo - explained

switchmedia
switch.tv

https://lowlatency.s Playlist Delta Request Part1678.18.ts

https://lowlatency.switch.tv/lhls/2M/lowLatencyHLS.php?_HLS_skip=YES

https://lowlatency.switch.tv/lhls/2M/filePart1678.19.ts

https://lowlatency.switch.tv/lhls/2M/filePart1678.20.ts — Last partial segment of 1678

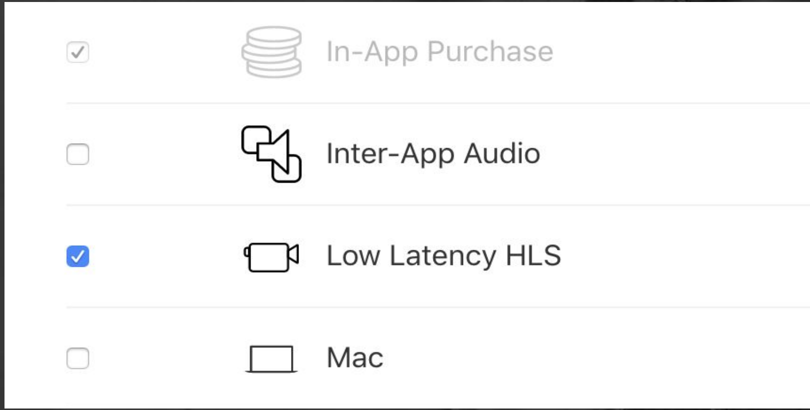
https://lowlatency.switch.tv/lhls/2M/filePart1679.1.ts — First partial segment of 1679

https://lowlatency.switch.tv/lhls/2M/filePart1679.2.ts

# **Apple LLHLS** Low Latency AVPlayer apps edits

switchmedia
switch.tv

Certificates, Identifiers & Profiles
new Low Latency HLS capability
(not enterprise yet)




(in Xcode need to add)

```
<plist version="1.0">
  <dict>
    <key>com.apple.developer.coremedia.hls.low-latency</key><true/>
```

AVPlayerItem now has some new properties to request how far from **live edge** we *ought* to be stay in vs what we *want* to try and stay within - too small and buffering is more likely for example - too long and you are not lowest latency

switchmedia
switch.tv

# **Community LHLS** (a nod to the original LHLS)

Community lead initiative (Periscope, JW, Twitch and others)
Apple's 'not invented here' and their draconian AppStore requirements probably will more or less kill this as a standard
But avoids the CDN cache busting complication where Apple reserves ?_HLS query string - also CMAF/MSE better approach
Quick demo (Akamai/JW) https://lowlatency.switch.tv/**clhls**

Just maybe a best-of-breed solution may result from all this
… but my guess is that's some pretty wishful thinking

# WebAssembly (on the CDN edge)

CDNs can help reduce latency too when streaming large scale
Apple Low-Latency manifests and partial segments can be generated at the local point-of-presence
Fastly is an example of a CDN supporting compute on the edge where WebAssembly can be used to process manifests and potentially process input TS streams a chunk it - generating virtual files on-the-fly without uploading them
Link to SwitchMedia slides on this topic:
https://goo.gl/**2ahsEY**

switchmedia
switch.tv

# other excellent low latency solutions out there

switchmedia
switch.tv

## NetInsight Sye / nanoStream / Bambuser / Phenix etc

Great results, often MPEG-TS over UDP solutions, so easy to manipulate/multicast but often there are other things to check...

- Might be unencrypted only (probably means bitstream is manipulated)
- May require proprietary CDN to operate at speed
- Might need ports opening for the client
- May require their player to be integrated (so ongoing support, and Apple likes to screw with us from time to time by pushing back on this)
- Sometimes camera encoding is also a major factor in performance
- Closed sources typically hampers debugging production issues

# h3 (next gen) HTTP/3

switchmedia
switch.tv

HTTP/3 will include *best-of-breed* of parts from QUIC, SRT, RIST and WebRTC/ObjectRTC

Combined with CMAF packaging using agreed common standards makes it viable for others to then adopt moving forwards

… but then standards always take time to establish

# **players** supporting low latency today (mid July)

switchmedia
switch.tv

Chromecast MPL and CAF (Shaka) - no partial segments yet
HbbTV no - recent moves to HTML5/MSE this will improve
hls.js - community LHLS, slightly stunned by Apple right now
DASH.IF - CMAF supported, low latency mode option
AVPlayer - iOS13 beta 2, AppleTV tvOS13 beta, **Safari soon**
Android ExoPlayer media range yes, LHLS in progress
Roku/TelstraTV - partials not supported, slow to keep up
Bitmovin, THEO - nothing public yet, closed source (JW actively support hls.js)

switchmedia
switch.tv

# Conclusion

In a nutshell things also revolve around TCP vs UDP and codec efficiency around how errors are tolerated (and so hidden)

TCP imposes overheads around setting up and maintaining said connections - but with bonus it’s generally error free

UDP is crazy error prone, so basically the race is on find the most acceptable workaround for a/v bitstreams

# Conclusion

switchmedia
switch.tv

Partial segments - CMAF and Partial TS - over HTTP/3 seems to be the accepted trend around packaging and transporting the bitstream, the common themes being...

- Cross-platform, all player set-tops, SmartTVs, all browsers
- Scalable, runs over regular HTTP/CDN technology
- Supports CENC/EME (AES encrypted bitstreams)
- Open Source (no patents/royalties) is VVC dead on arrival?

# Conclusion

switchmedia
switch.tv

Good news is Apple, Microsoft and Google all in agreement
e.g. MSE support finally arrived in iPadOS 13 last week

Basically there is an obvious cost/benefit trade off
Lowering your latency ⇨ more expensive infrastructure
Monetising (live ad insertion) ⇨ more expensive encoders

… and, how bothered is the end viewer really anyway? If they were *that* bothered they'd have just gone to the game!? :-)

# Thanks!

## Slides are here:

https://tinyurl.com/**yyr2rz8m**

AV1 https://goo.gl/**pGnNgJ**

WebAssembly https://goo.gl/**2ahsEY**


**Twitter/LinkedIn: kevleyski**

**Phone**: +61 2 8096 8277

**Email**: klambert@switch.tv

**Website**: www.switch.tv

**Address**: Suite 121, Jones Bay Wharf,

26-32 Pirrama Rd, Pyrmont, Sydney, NSW, Australia


switchmedia

Effortless Video