MINIMIZING MANIFEST OVERHEAD

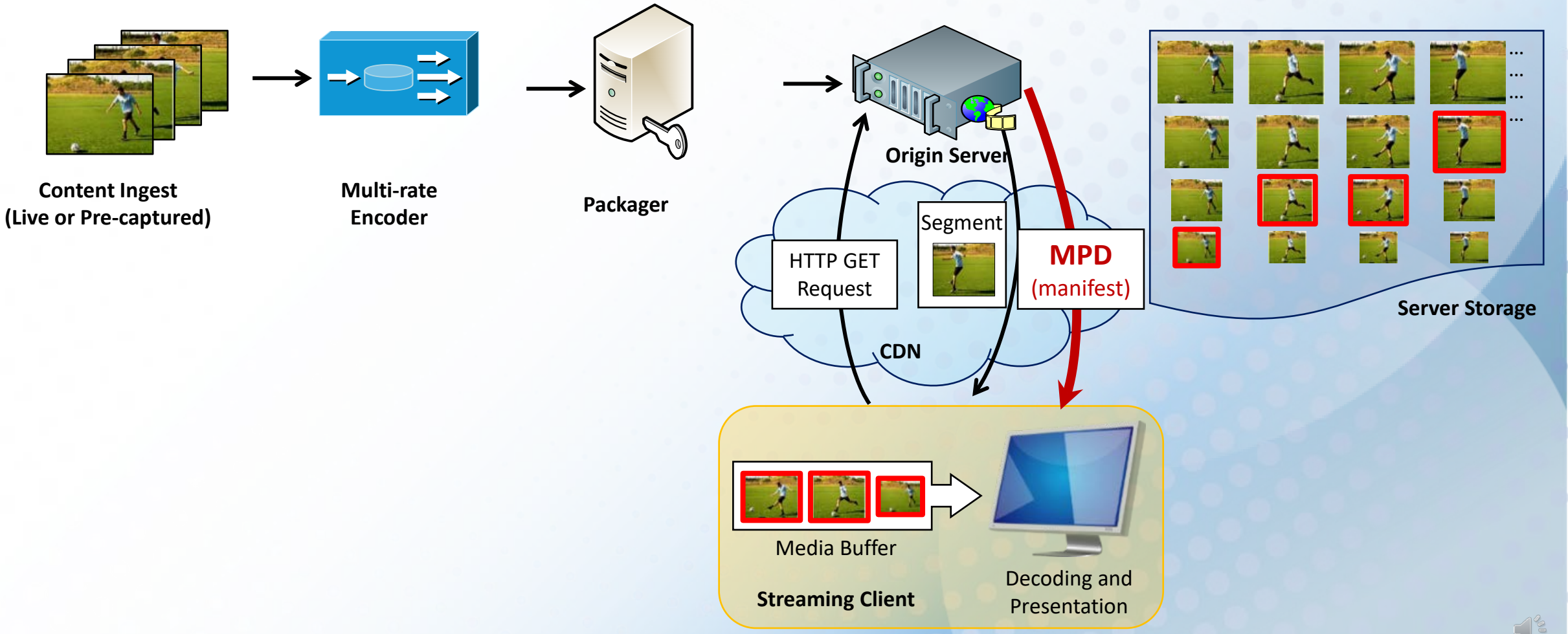
Alex Giladi

Fellow

Comcast

December 2, 2020

ADAPTIVE STREAMING SESSION



MANIFEST OVERHEAD IN DASH

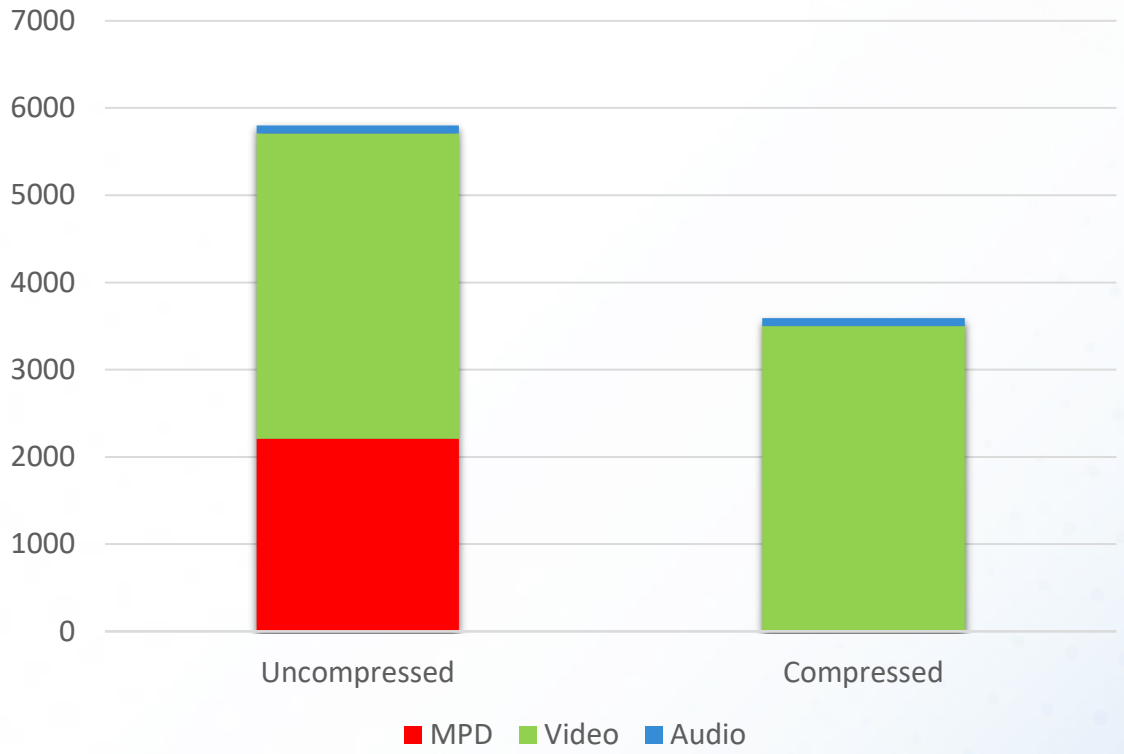
MPD OVERHEAD

- Bandwidth taken by MPD downloads
- Number of HTTP GET requests for MPD
- Startup delay due to first MPD download and parsing
- Memory and processing overhead

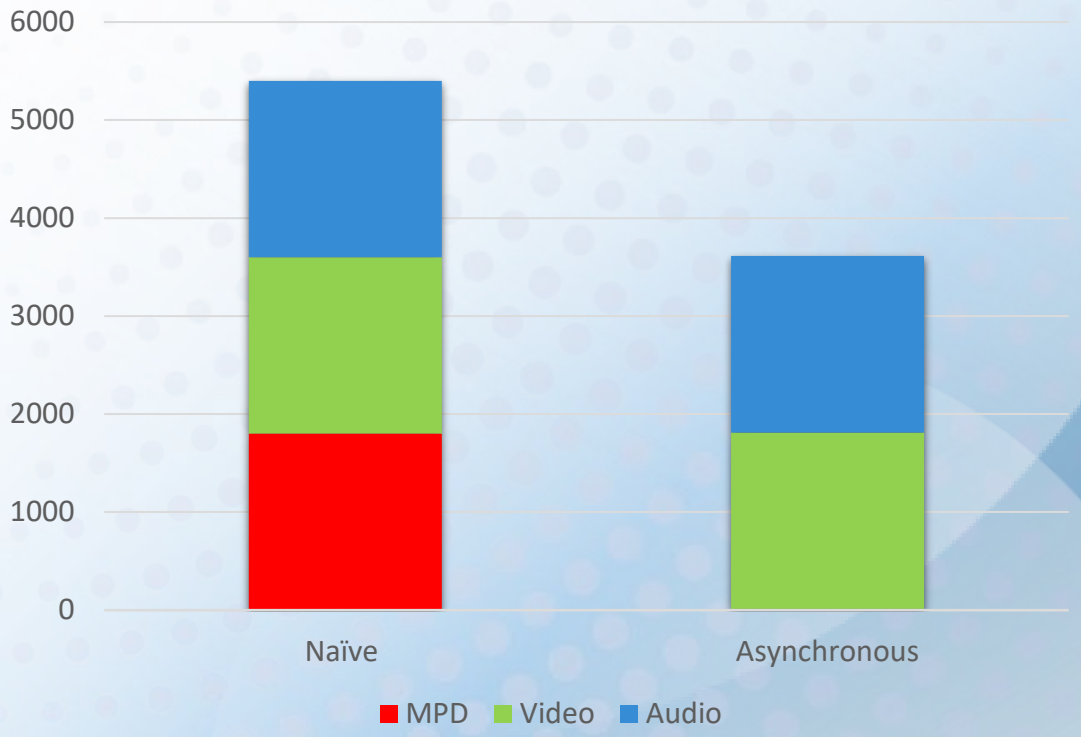
SIGNIFICANT ISSUE FOR LINEAR AND DVR SERVICES

- HLS and many DASH implementations request a new manifest for every new segment
- Typical DASH segments are ~2-sec – 1800 MPD requests per hour
- Advertisement and multi-DRM often result in increased overhead

CAN WE REDUCE THE MANIFEST OVERHEAD?



Bandwidth reduction



Request rate reduction

COMCAST

CONTENTS

REDUCING MPD TRAFFIC OVERHEAD

- Generic: HTTP compression
- DASH-specific
  - MPD patches
  - Asynchronous MPD updates

REDUCING MPD FOOTPRINT

- Stay concise
- Gap signaling

ON-THE-WIRE COMPRESSION

HTTP COMPRESSION

- Ubiquitously supported part of HTTP/1.1 and later
- Compresses body of the HTTP response
  - Only when the request indicates that a specific algorithm will be accepted in lieu of “normal” response

COMPRESSION ALGORITHMS

- gzip and deflate – “classic” compression algorithms mandated by HTTP/1.1
- brotli – better-performing newer algorithm
  - Widely supported IETF standard, but not required for HTTP/1.1 compliance
  - Very efficient in compressing highly redundant XML documents.

MPD	**Periods**	**DRM**	**DAI**	**Uncompressed MPD** (bytes)	**w/Gzip** (bytes)	**w/Brotli** (bytes)
A	1	3	N	61904	7096	5598
B	18	3	Y	326933	37322	5589
C	30	2	Y	425867	17266	6583
D	9	3	Y	552219	16885	10975

DASH MPD PATCHING

ON-THE-WIRE COMPRESSION IS NOT ENOUGH

- Exceptionally efficient in reducing size of a given document
- Application-agnostic – ignores application-level redundancies
- Memory and processing footprint unchanged

COMPRESS MPD SEQUENCES

- Consecutive MPDs most often differ due to addition or/and removal of segments
- These differences typically modify just a few lines within an MPD

DASH MPD PATCH: HOW IT WORKS

"XML DIFF" BETWEEN CONSECUTIVE MPD'S 

- Concept dates to 3GP DASH, introduced in its current form in the 5th edition of DASH
- Operates on XML elements and attributes
  - Uses XPath-based XML Patch framework (specified in IETF RFC 5261)
  - 3 directives: add, remove, replace
- Contains elements of version control to maintain consistency
  - Apply changes only if the base MPD versions (@publishTime and @id) match
  - Version update is a mandatory step in patching
- DASH client decides whether to download a full MPD or a patch

```
<Patch
    mpdId="42" originalPublishTime="2020-05-13T05:34:58Z"
    publishTime="2020-05-13T05:34:28.60Z">
    <replace sel="/MPD/@publishTime">2020-05-13T05:34:28.60Z</replace>
    <replace sel="/MPD/PatchLocation[0]">
        <PatchLocation>live-stream/patch.mpd?publishTime=2020-05-13T05%3A34%3A28.60Z</PatchLocation>
    </replace>
    <add sel="/MPD/Period[@id='1588435200']/AdaptationSet[@id='1']/SegmentTemplate/SegmentTimeline">
        <S d="360360" r="1" t="5494659049"/>
    </add>
    <add sel="/MPD/Period[@id='1588435200']/AdaptationSet[@id='2']/SegmentTemplate/SegmentTimeline">
        <S d="360960" t="5494660288"/>
    </add>
</Patch>
```

Version control

Change set

DASH MPD PATCH: PERFORMANCE

PATCHES ARE TINY

- Less than 1KB for 1 video and 1 audio adaptation set, w/segments added and removed
- Less than 10 XML elements – negligible memory and processing overhead
- Patches compress well -- ~300 bytes

ORDER OF MAGNITUDE REDUCTION IN MPD TRAFFIC

- 99.9% decrease in traffic – far more than HTTP Brotli transfer coding
- Simulation conditions
  - MPD requested every 2 sec
  - 12 periods per hour, other changes are segment additions/removals

MPD	Naïve	Naïve + Brotli	Patch	Patch + Brotli
A	241.80	21.87	3.05	1.54
B	1227.09	21.82	3.62	1.54
C	1663.55	25.71	2.50	1.43
D	2157.113	42.87	4.12	1.55

MPD traffic (kbps)

PREDICTIVE TEMPLATES AND TIMELINE EXTENSION

WHAT IS A DASH TEMPLATE?

- “Glorified printf”: `for (i=1; i<100; i++) printf(“segment%d.mp4”, i);`
  - $Number$ -- incrementing segment number
  - $Time$ stands for the earliest presentation time of a segment
- Segment URL can be derived before the segment is encoded
  - The client calculates the time window when this segment can be successfully downloaded

HOW CAN YOU GUESS THE NEXT URL?

- $Number$: increment by one
- $Time$: increment by the duration of the current segment
- Segment duration can be derived from the segment itself w/o an extra MPD request
  - “Timeline extension”, described in DASH-IF IOP

ASYNCHRONOUS MPD UPDATES

TIMELINE EXTENSION IS NOT A "SILVER BULLET"

- Work as long as everything goes as planned – "the turkey problem"
  - What if a new period (e.g. an ad) started?
  - What if there is a segment gap?

TRIGGER MPD UPDATE WHEN NEEDED

- MPD Validity Expiration event
  - Inband event (`emsg`), embedded into the beginning of a media segment
- Indicates when the current MPD becomes invalid
- Used in conjunction with predictive templates to trigger an MPD update
  - Never request an MPD (or patch) on a segment addition/removal – only on a "material" change
- Comes at a cost of tight coupling between segment and MPD generation

TIMELINE EXTENSION + ASYNC UPDATES: PERFORMANCE

REDUCTION IN TRAFFIC

- Most of the time MPD update bis not needed
- Orders of magnitude reduction in traffic
- Orders of magnitude reduction in number of HTTP GET requests for MPDs

MPD	Naïve		Asynchronous updates		
	Traffic (kbps)	GET requests	Uncompressed (kbps)	Brotli (kbps)	GET requests
A	241.80	1800	1.62	0.15	12
B	1277.09	1800	8.51	0.15	12
C	1663.55	1800	11.08	0.17	12
D	2157.11	1800	14.38	0.29	12

MPD traffic in kbps and GET req/hr

MEMORY AND PROCESSING OVERHEADS

COMPRESSION ALONE IS NOT ENOUGH

- Compression near-eliminates the network and CDN aspects of the overhead
- Clients and packagers process and store uncompressed MPD in memory
- Footprint proportional to number of XML elements and MPD size

BE CONCISE

- Never put SegmentTimeline element at Representation level
- Anything pertaining to all representations should *only* appear at AdaptationSet level
  - Examples: InbandEventStream
- Don’t use attributes with their default values – you can omit them entirely
  - Example: scanType is always “progressive” by default – don’t use it
- Use short namespace if external namespaces are frequently used
  - Consider declaring external namespace once at MPD level rather than at element level
  - Example: SCTE 35, Microsoft PlayReady, …

GAP SIGNALING

SEGMENTS CAN BE LOST

- Segments can be lost due to transcoder, input, or network failures (“gap”)
- Gap is naïvely interpreted as loss and subsequent “addition” of representation
  - Single Period split into three
  - >100 periods observed on misbehaving feeds – problematic for players

EXPLICIT GAP SIGNALING

- Gap can be indicated as explicitly in the MPD
  - FailoverContent element specifies the time range for which no are segments available
- Major reduction in number of XML elements
  - No need to describe every single representation all over again
  - No duplication of inlined information
- Simulation
  - Single-period MPD (MPD A)
  - Single segment lost in 720p representation

	Naïve	Gaps
XML elements	365	126
Size (bytes)	184483	62081

MPD A footprint with single segment loss

CONTENT PROTECTION REFERENCING

DRM INFORMATION CAN TAKE >70% OF MPD

- License acquisition information is carried in ContentProtection descriptors in MPD
  - Separate descriptor element for each DRM system in each adaptation set in each period
  - License acquisition information typically carried as base64-coded `pssh`
  - E.g. 72 identical copies of `pssh` in a 9-period MPD per each DRM

REFERENCE DESCRIPTORS

- Carry single descriptor per license at MPD level, reference it from adaptation sets
  - 1 “full” descriptor, and 72 one-liner references to it
- Major reduction in memory footprint

MPD	**DRMs**	Periods	Original		ContentProtection referencing	
			Uncompressed	Brotli	Uncompressed	Brotli
A	3	1	61904	5598	18170	5596
B	3	18	326933	5589	88010	5568
D	3	9	552219	10975	155650	11051

MPD size (bytes)

SUMMARY

MPD TRAFFIC OVERHEADS CAN BE REDUCED BY >99.99%

- HTTP compression
- MPD patches
- Asynchronous MPD updates

MPD SIZE REDUCTION

- Being concise is critical
- New tools in DASH
  - Gap signaling
  - ContentProtection referencing

WHAT'S NEXT?

- Systems headers are highly redundant
  - `moof` box
  - Non-VCL NAL units (e.g. CEA 708)
- Relevant for low-latency and low-rate representations

TRY THIS ~~AT HOME~~ IN YOUR DASH DEPLOYMENT!

COMCAST

Q&A

- sli.do
- Slack Alex Giladi at video-dev