Independent Submission
Request for Comments: 6983
Category: Informational
ISSN: 2070-1721

R. van Brandenburg
O. van Deventer
TNO
F. Le Faucheur
K. Leung
Cisco Systems

July 2013

Models for HTTP-Adaptive-Streaming-Aware Content Distribution Network Interconnection (CDNI)

Abstract

This document presents thoughts on the potential impact of supporting HTTP Adaptive Streaming (HAS) technologies in Content Distribution Network Interconnection (CDNI) scenarios. The intent is to present the authors' analysis of the CDNI-HAS problem space and discuss different options put forward by the authors (and by others during informal discussions) on how to deal with HAS in the context of CDNI. This document has been used as input information during the CDNI working group process for making a decision regarding support for HAS.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC6707] defines the problem space for Content Distribution Network Interconnection (CDNI) and the associated CDNI interfaces. This includes support, through interconnected Content Delivery Networks (CDNs), of content delivery to End Users using HTTP progressive download and HTTP Adaptive Streaming (HAS).

HTTP Adaptive Streaming is an umbrella term for various HTTP-based streaming technologies that allow a client to adaptively switch between multiple bitrates, depending on current network conditions. A defining aspect of HAS is that, since it is based on HTTP, it is a pull-based mechanism, with a client actively requesting content segments instead of the content being pushed to the client by a server. Due to this pull-based nature, media servers delivering content using HAS often show different characteristics when compared with media servers delivering content using traditional streaming methods such as the Real-time Transport Protocol / Real Time Streaming Protocol (RTP/RTSP), the Real Time Messaging Protocol (RTMP), and the Multimedia Messaging Service (MMS).

This document presents a discussion of the impact of the HAS operation on the CDNI interfaces, and what HAS-specific optimizations may be required or may be desirable. The scope of this document is to present the authors’ analysis of the CDNI-HAS problem space and discuss different options put forward by the authors (and by others during informal discussions) on how to deal with HAS in the context of CDNI. The document concludes by presenting the authors’ recommendations on how the CDNI WG should deal with HAS in its initial charter, with a focus on ’making it work’ instead of including ’nice-to-have’ optimizations that might delay the development of the CDNI WG deliverables identified in its initial charter.

It should be noted that the document is not a WG document but has been used as input during the WG process for making its decision regarding support for HAS. We expect the analysis presented in the document to be useful in the future if and when the WG recharters and wants to reassess the level of HAS optimizations to be supported in CDNI scenarios.

1.1. Terminology

This document uses the terminology defined in [RFC6707] and [CDNI-FRAMEWORK].

For convenience, the definitions of HAS-related terms are restated here:

Content Item: A uniquely addressable content element in a CDN. A content item is defined by the fact that it has its own Content Metadata associated with it. An example of a content item is a video file/stream, an audio file/stream, or an image file.

Chunk: A fixed-length element that is the result of a segmentation or fragmentation operation and that is independently addressable.

Fragment: A specific form of chunk (see Section 2.1). A fragment is stored as part of a larger file that includes all chunks that are part of the chunk collection.

Segment: A specific form of chunk (see Section 2.1). A segment is stored as a single file from a file-system perspective.

Original Content: Non-chunked content that is the basis for a segmentation or fragmentation operation. Based on Original Content, multiple alternative representations (using different encoding methods, supporting different resolutions, and/or targeting different bitrates) may be derived, each of which may be fragmented or segmented.

Chunk Collection: The set of all chunks that are the result of a single segmentation or fragmentation operation being performed on a single representation of the Original Content. A chunk collection is described in a Manifest File.

Content Collection: The set of all chunk collections that are derived from the same Original Content. A content collection may consist of multiple chunk collections, each corresponding to a single representation of the Original Content. A content collection may be described by one or more Manifest Files.

Manifest File: A Manifest File, also referred to as a Media Presentation Description (MPD) file, is a file that lists the way the content has been chunked (possibly for multiple encodings), as well as where the various chunks are located (in the case of segments) or how they can be addressed (in the case of fragments).

2. HTTP Adaptive Streaming Aspects Relevant to CDNI

In the last couple of years, a wide variety of HAS-like protocols have emerged. Among them are proprietary solutions such as Apple’s HTTP Live Streaming (HLS), Microsoft’s HTTP Smooth Streaming (HSS), and Adobe’s HTTP Dynamic Streaming (HDS), as well as various standardized solutions such as 3GPP Adaptive HTTP Streaming (AHS) and MPEG Dynamic Adaptive Streaming over HTTP (DASH). While all of these technologies share a common set of features, each has its own defining elements. This section looks at some of the common characteristics and some of the differences between these technologies and how those might be relevant to CDNI. In particular, Section 2.1 describes the various methods to store HAS content, and Section 2.2 lists three methods that are used to address HAS content in a CDN. After these generic HAS aspects are discussed, two special situations that need to be taken into account when discussing HAS are addressed: Section 2.3 discusses the differences between live content and Video on Demand (VoD) content, while Section 2.4 discusses the scenario where multiple streams are combined in a single Manifest File (e.g., for ad insertion purposes).

2.1. Segmentation versus Fragmentation

All HAS implementations are based on a concept referred to as "chunking": the concept of having a server split content up in numerous fixed-duration chunks that are independently decodable. By sequentially requesting and receiving chunks, a client can recreate and play out the content. An advantage of this mechanism is that it allows a client to seamlessly switch between different encodings of the same Original Content at chunk boundaries. Before requesting a particular chunk, a client can choose between multiple alternative encodings of the same chunk, irrespective of the encoding of the chunks it has requested earlier.

While every HAS implementation uses some form of chunking, not all implementations store the resulting chunks in the same way. In general, there are two distinct methods of performing chunking and storing the results: segmentation and fragmentation.

- With segmentation -- which is, for example, mandatory in all versions of Apple’s HLS prior to version 7 -- the chunks, in this case also referred to as segments, are stored completely independently from each other, with each segment being stored as a separate file from a file-system perspective. This means that each segment has its own unique URL with which it can be retrieved.

- With fragmentation (or virtual segmentation) -- which is, for example, used in Microsoft's HTTP Smooth Streaming -- all chunks, or fragments, belonging to the same chunk collection are stored together as part of a single file. While there are a number of container formats that allow for storing this type of chunked content, fragmented MP4 is most commonly used. With fragmentation, a specific chunk is addressable by suffixing, to the common file URL, an identifier uniquely identifying the chunk that one is interested in, either by timestamp, by byte range, or in some other way.

While one can argue about the merits of each of these two different methods of handling chunks, both have their advantages and drawbacks in a CDN environment. For example, fragmentation is often regarded as a method that introduces less overhead, from both a storage and processing perspective. Segmentation, on the other hand, is regarded as being more flexible and easier to cache. In practice, current HAS implementations increasingly support both methods.

2.2. Addressing Chunks

In order for a client to request chunks, in the form of either segments or fragments, it needs to know how the content has been chunked and where to find the chunks. For this purpose, most HAS protocols use a concept that is often referred to as a Manifest File (also known as a Media Presentation Description, or MPD), i.e., a file that lists the way the content has been chunked and where the various chunks are located (in the case of segments) or how they can be addressed (in the case of fragments). A Manifest File or set of Manifest Files may also identify the different representations, and thus chunk collections, available for the content.

In general, a HAS client will first request and receive a Manifest File, and then, after parsing the information in the Manifest File, proceed with sequentially requesting the chunks listed in the Manifest File. Each HAS implementation has its own Manifest File format, and even within a particular format there are different methods available to specify the location of a chunk.

Of course, managing the location of files is a core aspect of every CDN, and each CDN will have its own method of doing so. Some CDNs may be purely cache-based, with no higher-level knowledge of where each file resides at each instant in time. Other CDNs may have dedicated management nodes that, at each instant in time, do know at which servers each file resides. The CDNI interfaces designed by the CDNI WG will probably need to be agnostic to these kinds of CDN-internal architecture decisions. In the case of HAS, there is a strict relationship between the location of the content in the CDN (in this case chunks) and the content itself (the locations specified in the Manifest File). It is therefore useful to have an understanding of the different methods in use in CDNs today for specifying chunk locations in Manifest Files. The different methods for doing so are described in Sections 2.2.1 to 2.2.3.

Although these sections are especially relevant for segmented content due to its inherent distributed nature, the discussed methods are also applicable to fragmented content. Furthermore, it should be noted that the methods detailed below for specifying locations of content items in Manifest Files do not relate only to temporally segmented content (e.g., segments and fragments) but are also relevant in situations where content is made available in multiple representations (e.g., in different qualities, encoding methods, resolutions, and/or bitrates). In this case, the content consists of multiple chunk collections, which may be described by either a single Manifest File or multiple interrelated Manifest Files. In the latter case, there may be a high-level Manifest File describing the various available bitrates, with URLs pointing to separate Manifest Files describing the details of each specific bitrate. For specifying the locations of the other Manifest Files, the same methods that are used for specifying chunk locations also apply.

One final note relates to the delivery of the Manifest Files themselves. While in most situations the delivery of both the Manifest File and the chunks is handled by the CDN, there are scenarios imaginable in which the Manifest File is delivered by, for example, the Content Service Provider (CSP), and the Manifest File is therefore not visible to the CDN.

2.2.1. Relative URLs

One method for specifying chunk locations in a Manifest File is through the use of relative URLs. A relative URL is a URL that does not include the HOST part of a URL but only includes (part of) the PATH part of a URL. In practice, a relative URL is used by the client as being relative to the location from which the Manifest File has been acquired. In these cases, a relative URL will take the form of a string that has to be appended to the location of the Manifest File to get the location of a specific chunk. This means that in the case where a Manifest File with relative URLs is used, all chunks will be delivered by the same Surrogate that delivered the Manifest File. A relative URL will therefore not include a hostname.

For example, in the case where a Manifest File has been requested (and received) from:

http://surrogate.server.cdn.example.com/content_1/manifest.xml

a relative URL pointing to a specific segment referenced in the Manifest File might be:

segments/segment1_1.ts

which means that the client should take the location of the Manifest File and append the relative URL. In this case, the segment would then be requested from http://surrogate.server.cdn.example.com/content_1/segments/segment1_1.ts.

One drawback of using relative URLs is that it forces a CDN relying on HTTP-based request routing to deliver all segments belonging to a given content item with the same Surrogate that delivered the Manifest File for that content item, which results in limited flexibility. Another drawback is that relative URLs do not allow for fallback URLs; should the Surrogate that delivered the Manifest File break down, the client is no longer able to request chunks. The advantage of relative URLs is that it is very easy to transfer content between different Surrogates and even CDNs.

2.2.2. Absolute URLs with Redirection

Another method for specifying locations of chunks (or other Manifest Files) in a Manifest File is through the use of an absolute URL. An absolute URL contains a fully formed URL (i.e., the client does not have to calculate the URL as in the case of the relative URL but can use the URL from the Manifest File directly).

In the context of Manifest Files, there are two types of absolute URLs imaginable: absolute URLs with redirection and absolute URLs without redirection. The two methods differ in whether the URL points to a request routing node that will redirect the client to a Surrogate (absolute URLs with redirection) or point directly to a Surrogate hosting the requested content (absolute URLs without redirection).

In the case of absolute URLs with redirection, a request for a chunk is handled by the Request Routing system of a CDN just as if it were a standalone (non-HAS) content request, which might include looking up the Surrogate (and/or CDN) best suited for delivering the requested chunk to the particular user and sending an HTTP redirect to the user with the URL pointing to the requested chunk on the specified Surrogate (and/or CDN), or a DNS response pointing to the specific Surrogate.

An example of an absolute URL with redirection might look as follows:

```
http://requestrouting.cdn.example.com/
content_request?content=content_1&segment=segment1_1.ts
```

As can be seen from this example URL, the URL includes a pointer to a general CDN Request Routing function and some arguments identifying the requested segment.

The advantage of using absolute URLs with redirection is that they allow for maximum flexibility (since chunks can be distributed across Surrogates and CDNs in any imaginable way) without having to modify the Manifest File every time one or more chunks are moved (as is the case when absolute URLs without redirection are used). The downside of this method is that it can add significant load to a CDN Request Routing system, since it has to perform a redirect every time a client requests a new chunk.

2.2.3. Absolute URLs without Redirection

In the case of absolute URLs without redirection, the URL points directly to the specific chunk on the actual Surrogate that will deliver the requested chunk to the client. In other words, there will be no HTTP redirection operation taking place between the client requesting the chunk and the chunk being delivered to the client by the Surrogate.

An example of an absolute URL without redirection is the following:

```
http://surrogate1.cdn.example.com/content_1/segments/segment1_1.ts
```

As can be seen from this example URL, the URL includes both the identifier of the requested segment (in this case segment1_1.ts) and the server that is expected to deliver the segment (in this case surrogate1.cdn.example.com). With this, the client has enough information to directly request the specific segment from the specified Surrogate.

The advantage of using absolute URLs without redirection is that it allows more flexibility compared to using relative URLs (since segments do not necessarily have to be delivered by the same server) while not requiring per-segment redirection (which would add significant load to the node doing the redirection). The drawback of this method is that it requires a modification of the Manifest File every time content is moved to a different location (either within a CDN or across CDNs).

2.3. Live Content versus VoD Content

Though the formats and addresses of Manifest Files and chunk files do not typically differ significantly between live and Video-on-Demand (VoD) content, the time at which the Manifest Files and chunk files become available does differ significantly. For live content, chunk files and their corresponding Manifest Files are created and delivered in real time. This poses a number of potential issues for HAS optimization:

- With live content, chunk files are made available in real time. This limits the applicability of bundling for content acquisition purposes. Pre-positioning may still be employed; however, any significant latency in the pre-positioning may diminish the value of pre-positioning if a client requests the chunk prior to pre-positioning or if the pre-positioning request is serviced after the chunk playout time has passed.

- In the case of live content, Manifest Files must be updated for each chunk and therefore must be retrieved by the client prior to each chunk request. Any optimization schemes based on Manifest Files must therefore be prepared to optimize on a per-segment request basis. Manifest Files may also be polled multiple times prior to the actual availability of the next chunk.

- Since live Manifest Files are updated as new chunks become available, the cacheability of Manifest Files is limited. Though timestamping and reasonable Time-to-Live (TTL) settings can improve delivery performance, timely replication and delivery of updated Manifest Files are critical to ensuring uninterrupted playback.

- Manifest Files are typically updated after the corresponding chunk is available for delivery, to prevent premature requests for chunks that are not yet available. HAS optimization approaches that employ dynamic Manifest File generation must be synchronized with chunk creation to prevent playback errors.

2.4. Stream Splicing

Stream splicing is used to create media mashups, combining content from multiple sources. A common example in which content resides outside the CDNs is with advertisement insertion, for both VoD and live streams. Manifest Files that contain absolute URLs with redirection may contain chunk or nested Manifest File URLs that point to content not delivered via any of the interconnected CDNs.

Furthermore, client and downstream proxy devices may depend on non-URL information provided in the Manifest File (e.g., comments or custom tags) for performing stream splicing. This often occurs outside the scope of the interconnected CDNs. HAS optimization schemes that employ dynamic Manifest File generation or rewriting must be cognizant of chunk URLs, nested Manifest File URLs, and other metadata that should not be modified or removed. Improper modification of these URLs or other metadata may cause playback interruptions and in the case of unplayed advertisements may result in loss of revenue for CSPs.

3. Possible HAS Optimizations

In the previous section, some of the unique properties of HAS were discussed. Furthermore, some of the CDN-specific design decisions with regards to addressing chunks have been detailed. In this section, the impact of supporting HAS in CDNI scenarios is discussed.

There are a number of topics, or problem areas, that are of particular interest when considering the combination of HAS and CDNI. For each of these problem areas, it holds that there are a number of different ways in which the CDNI interfaces can deal with them. In general, it can be said that each problem area can either be solved in a way that minimizes the amount of HAS-specific changes to the CDNI interfaces or maximizes the flexibility and efficiency with which the CDNI interfaces can deliver HAS content. The goal for the CDNI WG should probably be to try to find the middle ground between these two extremes and try to come up with solutions that optimize the balance between efficiency and additional complexity.

In order to allow the WG to make this decision, this section briefly describes each of the following problem areas, together with a number of different options for dealing with them. Section 3.1 discusses the problem of how to deal with file management of groups of files, or content collections. Section 3.2 deals with a related topic: how to do content acquisition of content collections between the Upstream CDN (uCDN) and Downstream CDN (dCDN). After that, Section 3.3 describes the various options for the request routing of HAS content, particularly related to Manifest Files. Section 3.4 talks about a number of possible optimizations for the logging of HAS content, while Section 3.5 discusses the options regarding URL signing. Finally, Section 3.6 describes different scenarios for dealing with the removal of HAS content from CDNs.

3.1. File Management and Content Collections

3.1.1. General Remarks

One of the unique properties of HAS content is that it does not consist of a single file or stream but of multiple interrelated files (segments, fragments, and/or Manifest Files). In this document, this group of files is also referred to as a content collection. Another important aspect is the difference between segments and fragments (see Section 2.1).

Irrespective of whether segments or fragments are used, different CDNs might handle content collections differently from a file management perspective. For example, some CDNs might handle all files belonging to a content collection as individual files that are stored independently from each other. An advantage of this approach is that it makes it easy to cache individual chunks. Other CDNs might store all fragments belonging to a content collection in a bundle, as if they were a single file (e.g., by using a fragmented MP4 container). The advantage of this approach is that it reduces file management overhead.

The following subsections look at the various ways with which the CDNI interfaces might deal with these differences in handling content collections from a file management perspective. The different options can be distinguished based on the level of HAS awareness they require on the part of the different CDNs and the CDNI interfaces.

3.1.2. Candidate Approaches

3.1.2.1. Option 1.1: Do Nothing

This option assumes no HAS awareness in both the involved CDNs and the CDNI interfaces. This means that the uCDN uses individual files, and the dCDN is not explicitly made aware of the relationship between chunks and doesn’t know which files are part of the same content collection. In practice, this scenario would mean that the file management method used by the uCDN is simply imposed on the dCDN as well.

This scenario also means that it is not possible for the dCDN to use any form of file bundling, such as the single-file mechanism, which can be used to store fragmented content as a single file (see

Section 2.1). The one exception to this rule is the situation where the content is fragmented and the Manifest Files on the uCDN contain byte range requests, in which case the dCDN might be able to acquire fragmented content as a single file (see Section 3.2.2.2).

Effect on CDNI interfaces:

o None

Advantages/Drawbacks:

+ No HAS awareness necessary in CDNs; no changes to CDNI interfaces necessary

- The dCDN is forced to store chunks as individual files

3.1.2.2. Option 1.2: Allow Single-File Storage of Fragmented Content

In some cases, the dCDN might prefer to store fragmented content as a single file on its Surrogates to reduce file management overhead. In order to do so, it needs to be able to either acquire the content as a single file (see Section 3.2.2.2) or to merge the different chunks together and place them in the same container (e.g., fragmented MP4). The downside of this method is that in order to do so, the dCDN needs to be fully HAS aware.

Effect on CDNI interfaces:

o CDNI Metadata interface: Add fields for indicating the particular type of HAS (e.g., MPEG DASH or HLS) that is used and whether segments or fragments are used

o CDNI Metadata interface: Add field for indicating the name and type of the Manifest File(s)

Advantages/Drawbacks:

+ Allows the dCDN to store fragmented content as a single file, reducing file management overhead

- Complex operation, requiring the dCDN to be fully HAS aware

3.1.2.3. Option 1.3: Access Correlation Hint

An intermediary approach between the two extremes detailed in the previous two sections is one that uses an 'Access Correlation Hint'. This hint, which is added to the CDNI Metadata of all chunks of a particular content collection, indicates that those files are likely to be requested in a short time window from each other. This information can help a dCDN to implement local file storage optimizations for VoD items (e.g., by bundling all files with the same Access Correlation Hint value in a single bundle/file), thereby reducing the number of files it has to manage while not requiring any HAS awareness.

Effect on CDNI interfaces:

- CDNI Metadata interface: Add field for indicating Access Correlation Hint

Advantages/Drawbacks:

+ Allows the dCDN to perform file management optimization

+ Does not require any HAS awareness

+ Very small impact on CDNI interfaces

- Expected benefit compared with Option 1.1 is small

3.1.3. Recommendations

Based on the listed pros and cons, the authors recommend that the WG go for Option 1.1 (do nothing). The likely benefits of going for Option 1.3 are not believed to be significant enough to warrant changing the CDNI Metadata interface. Although Option 1.2 would bring definite benefits for HAS-aware dCDNs, going for this option would require significant CDNI extensions that would impact the WG’s milestones. The authors therefore don’t recommend including it in the current work but mark it as a possible candidate for rechartering once the initial CDNI solution is completed.

3.2. Content Acquisition of Content Collections

3.2.1. General Remarks

In the previous section, the relationship between file management and HAS in a CDNI scenario was discussed. This section discusses a related topic: content acquisition between two CDNs.

With regards to content acquisition, it is important to note the difference between CDNs that do dynamic acquisition of content and CDNs that perform content pre-positioning. In the case of dynamic acquisition, a CDN only requests a particular content item when a cache miss occurs. In the case of pre-positioning, a CDN proactively places content items on the nodes on which it expects traffic for that particular content item. For each of these types of CDNs, there might be a benefit in being HAS aware. For example, in the case of dynamic acquisition, being HAS aware means that after a cache miss for a given chunk occurs, that node might not only acquire the requested chunk but might also acquire some related chunks that are expected to be requested in the near future. In the case of pre-positioning, similar benefits can be had.

3.2.2. Candidate Approaches

3.2.2.1. Option 2.1: No HAS Awareness

This option assumes no HAS awareness in both the involved CDNs and the CDNI interfaces. Just as with Option 1.1, discussed earlier with regards to file management, having no HAS awareness means that the dCDN is not aware of the relationship between chunks. In the case of content acquisition, this means that each and every file belonging to a content collection will have to be individually acquired from the uCDN by the dCDN. The exception to the rule is cases with fragmented content where the uCDN uses Manifest Files that contain byte range requests. In this case, the dCDN can simply omit the byte range identifier and acquire the complete file.

The advantage of this approach is that it is highly flexible. If a client only requests a small portion of the chunks belonging to a particular content collection, the dCDN only has to acquire those chunks from the uCDN, saving both bandwidth and storage capacity.

The downside of acquiring content on a per-chunk basis is that it creates more transaction overhead between the dCDN and uCDN, compared to a method in which entire content collections can be acquired as part of one transaction.

Effect on CDNI interfaces:

- None

Advantages/Drawbacks:

+ Per-chunk content acquisition allows for a high level of flexibility between the dCDN and uCDN

- Per-chunk content acquisition creates more transaction overhead between the dCDN and uCDN

3.2.2.2. Option 2.2: Allow Single-File Acquisition of Fragmented Content

As discussed in Section 3.2.2.1, there is one (fairly rare) case where fragmented content can be acquired as a single file without any HAS awareness, and that is when fragmented content is used and where the Manifest File specifies byte range requests. This section discusses how to perform single-file acquisition in the other (very common) cases. To do so, the dCDN would have to have full HAS awareness (at least to the extent of being able to map between a single file and individual chunks to serve).

Effect on CDNI interfaces:

- CDNI Metadata interface: Add fields for indicating the particular type of HAS (e.g., MPEG DASH or HLS) that is used and whether segments or fragments are used

- CDNI Metadata interface: Add field for indicating the name and type of the Manifest File(s)

Advantages/Drawbacks:

+ Allows for more efficient content acquisition in all HAS-specific supported forms

- Requires full HAS awareness on the part of the dCDN

- Requires significant CDNI Metadata interface extensions

3.2.3. Recommendations

Based on the listed pros and cons, the authors recommend that the WG go for Option 2.1, since it is sufficient to 'make HAS work'. While Option 2.2 would bring benefits to the acquisition of large content collections, it would require significant CDNI extensions that would impact the WG's milestones. Option 2.2 might be a candidate to include in possible rechartering once the initial CDNI solution is completed.

3.3. Request Routing of HAS Content

3.3.1. General Remarks

In this section, the effect HAS content has on request routing is identified. Of particular interest in this case are the different types of Manifest Files that might be used. In Section 2.2, three different methods for identifying and addressing chunks from within a

Manifest File were described: relative URLs, absolute URLs with redirection, and absolute URLs without redirection. Of course, not every current CDN will use and/or support all three methods. Some CDNs may only use one of the three methods, while others may support two or all three.

An important factor in deciding which chunk-addressing method is used is the CSP. Some CSPs may have a strong preference for a particular method and deliver the Manifest Files to the CDN in a particular way. Depending on the CDN and the agreement it has with the CSP, a CDN may either host the Manifest Files as they were created by the CSP or modify the Manifest File to adapt it to its particular architecture (e.g., by changing relative URLs to absolute URLs that point to the CDN Request Routing function).

3.3.2. Candidate Approaches

3.3.2.1. Option 3.1: No HAS Awareness

This option assumes no HAS awareness in both the involved CDNs and the CDNI interfaces. This scenario also assumes that neither the dCDN nor the uCDN has the ability to actively manipulate Manifest Files. As was also discussed with regards to file management and content acquisition, having no HAS awareness means that each file constituting a content collection is handled on an individual basis, with the dCDN unaware of any relationship between files.

The only chunk-addressing method that works without question in this case is absolute URLs with redirection. In other words, the CSP that ingested the content into the uCDN created a Manifest File with each chunk location pointing to the Request Routing function of the uCDN. Alternatively, the CSP may have ingested the Manifest File containing relative URLs, and the uCDN ingestion function has translated these to absolute URLs pointing to the Request Routing function.

In this "absolute URL with redirection" case, the uCDN can simply have the Manifest File be delivered by the dCDN as if it were a regular file. Once the client parses the Manifest File, it will request any subsequent chunks from the uCDN Request Routing function. That function can then decide to outsource the delivery of those chunks to the dCDN. Depending on whether HTTP-based (recursive or iterative) or DNS-based request routing is used, the uCDN Request Routing function will then either directly or indirectly redirect the client to the Request Routing function of the dCDN (assuming that it does not have the necessary information to redirect the client directly to a Surrogate in the dCDN).

The drawback of this method is that it creates a large amount of request routing overhead for both the uCDN and dCDN. For each chunk, the full inter-CDN Request Routing process is invoked (which can result in two HTTP redirections in the case of iterative redirection, or one HTTP redirection plus one CDNI Request Routing Redirection interface request/response). Even in the case where DNS-based redirection is used, there might be significant overhead involved, since both the dCDN and uCDN Request Routing functions might have to perform database lookups and query each other. While with DNS this overhead might be reduced by using DNS’s inherent caching mechanism, this will have significant impact on the accuracy of the redirect.

With no HAS awareness, relative URLs might or might not work, depending on the type of relative URL that is used. When a uCDN delegates the delivery of a Manifest File containing relative URLs to a dCDN, the client goes directly to the dCDN Surrogate from which it has received the Manifest File for every subsequent chunk. As long as the relative URL is not path-absolute (see [RFC3986]), this approach will work fine.

Since using absolute URLs without redirection inherently requires a HAS-aware CDN, absolute URLs without redirection cannot be used in this case because the URLs in the Manifest File will point directly to a Surrogate in the uCDN. Since this scenario assumes no HAS awareness on the part of the dCDN or uCDN, it is impossible for either of these CDNs to rewrite the Manifest File and thus allow the client to either go to a Surrogate in the dCDN or to a Request Routing function.

Effect on CDNI interfaces:

o None

Advantages/Drawbacks:

+ Supports absolute URLs with redirection

+ Supports relative URLs

+ Does not require HAS awareness and/or changes to the CDNI interfaces

- Not possible to use absolute URLs without redirection

- Creates significant signaling overhead in cases where absolute URLs with redirection are used (inter-CDN request redirection for each chunk)

3.3.2.2. Option 3.2: Manifest File Rewriting by uCDN

While Option 3.1 does allow absolute URLs with redirection to be used, it does so in a way that creates a high level of request routing overhead for both the dCDN and the uCDN. This option presents a solution to significantly reduce this overhead.

In this scenario, the uCDN is able to rewrite the Manifest File (or generate a new one) to be able to remove itself from the request routing chain for chunks being referenced in the Manifest File. As described in Section 3.3.2.1, in the case of no HAS awareness, the client will go to the uCDN Request Routing function for each chunk request. This Request Routing function can then redirect the client to the dCDN Request Routing function. By rewriting the Manifest File (or generating a new one), the uCDN is able to remove this first step and have the Manifest File point directly to the dCDN Request Routing function.

A key advantage of this solution is that it does not directly have an impact on the CDNI interfaces and is therefore transparent to these interfaces. It is a CDN-internal function that a uCDN can perform autonomously by using information configured for regular CDNI operation or received from the dCDN as part of the regular communication using the CDNI Request Routing Redirection interface.

More specifically, in order for the uCDN to rewrite the Manifest File, the minimum information needed is the location of the dCDN Request Routing function (or, alternatively, the location of the dCDN delivering Surrogate). This information can be available from configuration or can be derived from the regular CDNI Request Routing Redirection interface. For example, the uCDN may ask the dCDN for the location of its request routing node (through the CDNI Request Routing Redirection interface) every time a request for a Manifest File is received and processed by the uCDN Request Routing function. The uCDN would then modify the Manifest File and deliver the Manifest File to the client. One advantage of this method is that it maximizes efficiency and flexibility by allowing the dCDN to optionally respond with the locations of its Surrogates instead of the location of its Request Routing function (and effectively turning the URLs into absolute URLs without redirection). There are many variations on this approach, such as where the modification of the Manifest File is only performed once (or once per period of time) by the uCDN Request Routing function, when the first client for that particular content collection (and redirected to that particular dCDN) sends a Manifest File request. The advantage of such a variation is that the uCDN only has to modify the Manifest File once (or once per time period). The drawback of this variation is that the dCDN is no longer in a position to influence the request routing decision across individual content requests.

It should be noted that there are a number of things to take into account when changing a Manifest File (see, for example, Sections 2.3 and 2.4 on live HAS content and ad insertion). Furthermore, some CSPs might have issues with a CDN changing Manifest Files. However, in this option the Manifest File manipulation is only being performed by the uCDN, which can be expected to be aware of these limitations if it wants to perform Manifest File manipulation, since it is in its own best interest that its customer’s content gets delivered in the proper way and since there is a direct commercial and technical relationship between the uCDN (the Authoritative CDN in this scenario) and its customer (the CSP). Should the CSP want to limit Manifest File manipulation, it can simply arrange this with the uCDN bilaterally.

Effect on CDNI interfaces:

- None

Advantages/Drawbacks:

+ Possible to significantly decrease signaling overhead when using absolute URLs

+ (Optional) Possible to have the uCDN rewrite the Manifest File with locations of Surrogates in the dCDN (turning absolute URLs with redirection into absolute URLs without redirection)

+ No changes to CDNI interfaces

+ Does not require HAS awareness in the dCDN

- Requires a high level of HAS awareness in the uCDN (for modifying Manifest Files)

3.3.2.3. Option 3.3: Two-Step Manifest File Rewriting

One of the possibilities with Option 3.2 is allowing the dCDN to provide the locations of a specific Surrogate to the uCDN, so that the uCDN can fit the Manifest File with absolute URLs without redirection and the client can request chunks directly from a dCDN Surrogate. However, some dCDNs might not be willing to provide this information to the uCDN. In that case, they can only provide the uCDN with the location of their Request Routing function, thereby preventing the use of absolute URLs without redirection.

One method for solving this limitation is allowing two-step Manifest File manipulation. In the first step, the uCDN would perform its own modification and place the locations of the dCDN Request Routing function in the Manifest File. Then, once a request for the Manifest File comes in at the dCDN Request Routing function, it would perform a second modification in which it replaces the URLs in the Manifest Files with the URLs of its Surrogates. This way, the dCDN can still profit from having limited request routing traffic while not having to share sensitive Surrogate information with the uCDN.

The downside of this approach is that it not only assumes HAS awareness in the dCDN but also requires some HAS-specific additions to the CDNI Metadata interface. In order for the dCDN to be able to change the Manifest File, it has to have some information about the structure of the content. Specifically, it needs to have information about which chunks make up the content collection.

Effect on CDNI interfaces (apart from those already listed under Option 3.2):

- CDNI Metadata interface: Add necessary fields for conveying HAS-specific information (e.g., the files that make up the content collection) to the dCDN
- CDNI Metadata interface: Allow dCDN to modify Manifest File

Advantages/Drawbacks (apart from those already listed under Option 3.2):

+ Allows the dCDN to use absolute URLs without redirection, without having to convey sensitive information to the uCDN

- Requires a high level of HAS awareness in the dCDN (for modifying Manifest Files)

- Requires adding HAS-specific and Manifest File manipulation-specific information to the CDNI Metadata interface

3.3.3. Recommendations

Based on the listed pros and cons, the authors recommend going for Option 3.1, with Option 3.2 as an optional feature that may be supported as a CDN-internal behavior by a uCDN. While Option 3.1 allows for HAS content to be delivered using the CDNI interfaces, it does so with some limitations regarding supported Manifest Files and, in some cases, with a large amount of signaling overhead. Option 3.2 can solve most of these limitations and presents a significant reduction in request routing overhead. Since Option 3.2 does not require any changes to the CDNI interfaces but only changes the way the uCDN uses the existing interfaces, supporting it is not expected to result in a significant delay of the WG’s milestones. The authors recommend that the WG not include Option 3.3, since it raises some questions of potential brittleness and including it would result in a significant delay of the WG’s milestones.

3.4. Logging

3.4.1. General Remarks

As stated in [RFC6707], the CDNI Logging interface enables details of logs or events to be exchanged between interconnected CDNs.

As discussed in [CDNI-LOGGING], the CDNI logging information can be used for multiple purposes, including maintenance/debugging by a uCDN, accounting (e.g., for billing or settlement purposes), reporting and management of end-user experience (e.g., to the CSP), analytics (e.g., by the CSP), and control of content distribution policy enforcement (e.g., by the CSP).

The key consideration for HAS with respect to logging is the potential increase of the number of log records by two to three orders of magnitude, as compared to regular HTTP delivery of a video, since by default log records would typically be generated on a per-chunk-delivery basis instead of a per-content-item-delivery basis. This impacts the scale of every processing step in the logging process (see [CDNI-LOGGING]), including:

a. Logging information generation and storing on CDN elements (Surrogate, Request Routers, ...)

b. Logging information aggregation within a CDN

c. Logging information manipulation (including information protection, filtering, update, and rectification)

d. (Where needed) CDNI reformatting of logging information (e.g., reformatting from a CDN-specific format to the CDNI Logging interface format for export by the dCDN to the uCDN)

e. Logging exchange via the CDNI Logging interface

f. (Where needed) Logging re-reformatting (e.g., reformatting from the CDNI Logging interface format into a log-consuming application)

g. Logging consumption/processing (e.g., feed logs into uCDN accounting application, feed logs into uCDN reporting system to provide per-CSP views, feed logs into debugging tools)

Note that there may be multiple instances of steps [f] and [g] running in parallel.

While the CDNI Logging interface is only used to perform step [e], we note that its format directly affects steps [d] and [f] and that its format also indirectly affects the other steps (for example, if the CDNI Logging interface requires per-chunk log records, steps [a], [b], and [d] cannot operate on a per-HAS-session basis, and they also need to operate on a per-chunk basis).

This section discusses the main candidate approaches identified for CDNI in terms of dealing with HAS with respect to logging.

3.4.2. Candidate Approaches

3.4.2.1. Option 4.1: Do Nothing

In this approach, nothing is done specifically for HAS, so each HAS-chunk delivery is considered, for CDNI logging, as a standalone content delivery. In particular, a separate log record for each HAS-chunk delivery is included in the CDNI Logging interface in step [e] (as defined in Section 3.4.1). This approach requires that steps [a], [b], [c], [d], and [f] also be performed on a per-chunk basis. This approach allows step [g] to be performed either on a per-chunk basis (assuming that step [f] maintains per-chunk records) or in a more "summarized" manner, such as on a per-HAS-session basis (assuming that step [f] summarizes per-chunk records into per-HAS-session records).

Effect on CDNI interfaces:

- None

Advantages/Drawbacks:

+ No information loss (i.e., all details of each individual chunk delivery are preserved). While this full level of detail may not be needed for some log-consuming applications (e.g., billing), this full level of detail is likely valuable (and possibly required) for some log-consuming applications (e.g., debugging)

- + Easier integration (at least in the short term) into existing logging tools, since those tools are all capable of handling per-chunk records

- + No extension needed on CDNI interfaces

- - High volume of logging information to be handled (storing and processing) at every step of the logging process, from steps [a] to [g] (while summarization in step [f] is conceivable, it may be difficult to achieve in practice without any hints for correlation in the log records)

An interesting question is whether a dCDN could use the CDNI Logging interface specified for the "do nothing" approach to report summarized "per-session" log information in the case where the dCDN performs such summarization. The high-level idea would be that when a dCDN performs HAS log summarization, for its own purposes anyway, this dCDN could include in the CDNI Logging interface one or more log entries for a HAS session (instead of one entry per HAS chunk) that summarize the deliveries of many/all HAS chunks for a session. However, the authors feel that when considering the details of this idea, it is not achievable without explicit agreement between the uCDN and dCDN about how to perform/interpret such summarization. For example, when a HAS session switches between representations, the uCDN and dCDN would have to agree on things such as:

- o whether the session will be represented by a single log entry (which therefore cannot convey the distribution across representations), or multiple log entries, such as one entry per contiguous period at a given representation (which therefore would be generally very difficult to correlate back into a single session)

- o what the single URI included in the log entry would correspond to (for example, the Manifest File, top-level playlist, or next-level playlist, ...)

The authors feel that since explicit agreement is needed between the uCDN and dCDN on how to perform/interpret the summarization, this method can only work if it is specified as part of the CDNI Logging interface, in which case it would effectively boil down to Option 4.4 (full HAS awareness / per-session logs) as defined below.

We note that support by CDNI of a mechanism (independent of HAS) allowing the customization of the fields to be reported in log entries by the dCDN to the uCDN would mitigate concerns related to the scaling of HAS logging, because it ensures that only the necessary subset of fields is actually stored, reported, and processed.

3.4.2.2. Option 4.2: CDNI Metadata Content Collection ID

In this approach, a "Content Collection IDentifier (CCID)" field is distributed through the CDNI Metadata interface, and the same CCID value is associated through the CDNI Metadata interface with every chunk of the same content collection. The CCID value needs to be such that it allows, in combination with the content URI, unique identification of a content collection. When the CCID is distributed, and CCID logging is requested from the dCDN, the dCDN Surrogates are to store the CCID value in the corresponding log entries. The objective of this field is to facilitate optional summarization of per-chunk records at step [f] into something along the lines of per-HAS-session logs, at least for the log-consuming applications that do not require per-chunk detailed information (for example, billing).

We note that if the dCDN happens to have sufficient HAS awareness to be able to generate a "Session IDentifier (Session-ID)", optionally including such a Session-ID (in addition to the CCID) in the per-chunk log record would further facilitate optional summarization at step [f]. The Session-ID value to be included in a log record by the delivering CDN is such that

- different per-chunk log records with the same Session-ID value must correspond to the same user session (i.e., delivery of the same content to the same End User at a given point in time).

- log records for different chunks of the same user session (i.e., delivery of the same content to the same End User at a given point in time) should be provided with the same Session-ID value. While undesirable, there may be situations where the delivering CDN uses more than one Session-ID value for different per-chunk log records of a given session -- for example, in scenarios of fail-over or load balancing across multiple Surrogates and where the delivering CDN does not implement mechanisms to synchronize Session-IDs across Surrogates.

Effect on CDNI interfaces:

- CDNI Metadata interface: One additional metadata field (CCID) in the CDNI Metadata interface. We note that a similar content collection ID is discussed for the handling of other aspects of HAS and observe that further thought is needed to determine whether such a CCID should be shared for multiple purposes or should be independent.

- CDNI Logging interface: Two additional fields (CCID and Session-ID) in CDNI logging records.

Advantages/Drawbacks:

+ No information loss (i.e., all details of each individual chunk delivery are preserved). While this full level of detail may not be needed for some log-consuming applications (e.g., billing), this full level of detail is likely valuable (and possibly required) for some log-consuming applications (e.g., debugging)

+ Easier integration (at least in the short term) into existing logging tools, since those tools are all capable of handling per-chunk records

+ Very minor extension to CDNI interfaces needed

+ Facilitated summarization of records related to a HAS session in step [f] and therefore ability to operate on a lower volume of logging information in step [g] by log-consuming applications that do not need per-chunk record details (e.g., billing) or that need per-session information (e.g., analytics)

- High volume of logging information to be handled (storing and processing) at every step of the logging process, from steps [a] to [f]

3.4.2.3. Option 4.3: CDNI Logging Interface Compression

In this approach, a lossless compression technique is applied to the sets of logging records (e.g., logging files) for transfer on the CDNI Logging interface. The objective of this approach is to reduce the volume of information to be stored and transferred in step [e].

Effect on CDNI interfaces:

- One compression mechanism to be included in the CDNI Logging interface

Advantages/Drawbacks:

+ No information loss (i.e., all details of each individual chunk delivery are preserved). While this full level of detail may not be needed for some log-consuming applications (e.g., billing), this full level of detail is likely valuable (and possibly required) for some log-consuming applications (e.g., debugging)

+ Easier integration (at least in the short term) into existing logging tools, since those tools are all capable of handling per-chunk records

+ Small extension to CDNI interfaces needed

+ Reduced volume of logging information in step [e]

+ Compression likely to also be applicable to logs for non-HAS content

- High volume of logging information to be handled (storing and processing) at every step of the logging process, from steps [a] to [g], except step [e].

3.4.2.4. Option 4.4: Full HAS Awareness/Per-Session Logs

In this approach, HAS awareness is assumed across the CDNs interconnected via CDNI, and the necessary information to describe the HAS relationship across all chunks of the same content collection is distributed through the CDNI Metadata interface. In this approach, the dCDN leverages the HAS information distributed through the CDNI Metadata and their HAS awareness, to do one of the following:

- directly generate summarized logging information at logging information generation time (which has the benefit of operating on a lower volume of logging information as early as possible in the successive steps of the logging process), or
- (if per-chunk logs are generated) accurately correlate and summarize per-chunk logs into per-session logs for exchange over the CDNI Logging interface

Effect on CDNI interfaces:

- CDNI Metadata interface: Significant extension to convey HAS relationship across chunks of a content collection. Note that this extension requires specific support for every HAS protocol to be supported over the CDNI mesh
- CDNI Logging interface: Extension to specify summarized per-session logs

Advantages/Drawbacks:

+ Lower volume of logging information to be handled (storing and processing) at every step of the logging process, from steps [a] to [g]

+ Accurate generation of summarized logs because of HAS awareness in the dCDN (for example, where the Surrogate is also serving the Manifest File(s) for a content collection, the Surrogate may be able to extract definitive information about the relationship between all chunks)

- Very significant extensions to CDNI interfaces needed, including specific support for available HAS protocols
- Very significant additional requirement for HAS awareness on the dCDN and for this HAS awareness to be consistent with the defined CDNI logging summarization

- Some information loss (i.e., all details of each individual chunk delivery are not preserved). The actual information loss depends on the summarization approach selected (typically, the lower the information loss, the lower the summarization gain), so the right "sweet spot" would have to be selected. While a full level of detail may not be needed for some log-consuming applications (e.g., billing), such a full level of detail is likely valuable (and possibly required) for some log-consuming applications (e.g., debugging)

- Less easy integration (at least in the short term) into existing logging tools, since those tools are all capable of handling per-chunk records but may not be capable of handling CDNI summarized records

- Challenges in defining behavior (and achieving summarization gain) in the presence of load balancing of a given HAS session across multiple Surrogates (in the same dCDN or a different dCDN)

3.4.3. Recommendations

Because of its benefits (in particular simplicity, universal support by CDNs, and support by all log-consuming applications), the authors recommend that per-chunk logging as described in Section 3.4.2.1 (Option 4.1) be supported by the CDNI Logging interface as a "High Priority" (as defined in [CDNI-REQUIREMENTS]) and be a mandatory capability of CDNs implementing CDNI.

Because of its very low complexity and its benefits in facilitating some useful scenarios (e.g., per-session analytics), we recommend that the CCID mechanisms and Session-ID mechanism as described in Section 3.4.2.2 (Option 4.2) be supported by the CDNI Metadata interface and the CDNI Logging interface as a "Medium Priority" (as defined in [CDNI-REQUIREMENTS]) and be an optional capability of CDNs implementing CDNI.

The authors also recommend that

(i) the ability of the uCDN to request inclusion of the CCID and Session-ID fields (in log entries provided by the dCDN) be supported by the relevant CDNI interfaces

(ii) the ability of the dCDN to include the CCID and Session-ID fields in CDNI log entries (when the dCDN is capable of doing so) be indicated in the CDNI Logging interface (in line with the "customizable" log format expected to be defined independently of HAS)

(iii) items (i) and (ii) be supported as a "Medium Priority" (as defined in [CDNI-REQUIREMENTS]) and be an optional capability of CDNs implementing CDNI

When performing dCDN selection, a uCDN may want to take into account whether a given dCDN is capable of reporting the CCID and Session-ID. Thus, the authors recommend that the ability of a dCDN to advertise its support of the optional CCID and Session-ID capability be supported by the CDNI Footprint & Capabilities Advertisement interface as a "Medium Priority" (as defined in [CDNI-REQUIREMENTS]).

The authors also recommend that a generic mechanism (independent of HAS) be supported that allows the customization of the fields to be reported in logs by CDNs over the CDNI Logging interface -- because of the reduction of the logging information volume exchanged across CDNs that it allows by removing information that is not of interest to the other CDN.

Because the following can be achieved with very little complexity and can provide some clear storage/communication compression benefits, the authors recommend that, in line with the concept of Option 4.3, some existing very common compression techniques (e.g., gzip) be supported by the CDNI Logging interface as a "Medium Priority" (as defined in [CDNI-REQUIREMENTS]) and be an optional capability of CDNs implementing CDNI.

Because of its complexity, the time it would take to understand the trade-offs of candidate summarization approaches, and the time it would take to specify the corresponding support in the CDNI Logging interface, the authors recommend that the log summarization discussed in Section 3.4.2.4 (Option 4.4) not be supported by the CDNI Logging interface at this stage but that it be kept as a candidate topic of great interest for a rechartering of the CDNI WG once the first set of deliverables is produced. At that time, we suggest investigating the notion of complementing a "push style" CDNI Logging interface that would support summarization via an on-demand "pull type" interface that would in turn allow a uCDN to request the subset of the detailed logging information that it may need but that is lost in the summarized pushed information.

The authors note that while a CDN only needs to adhere to the CDNI Logging interface on its external interfaces and can perform logging in a different format within the CDN, any possible CDNI logging approach effectively places some constraints on the dCDN logging format. For example, to support the "do nothing" approach, a CDN needs to perform and retain per-chunk logs. As another example, to support the "full HAS awareness/per-session logs" approach, the dCDN cannot use a logging format that summarizes data in a way that is incompatible with the summarization specified for CDNI logging (e.g., summarizes data into a smaller set of information than what is specified for CDNI logging). However, the authors feel that such constraints are (i) inevitable, (ii) outweighed by the benefits of a standardized logging interface, and (iii) acceptable because, in the case of incompatible summarization, most or all CDNs are capable of reverting to per-chunk logging as per the "do nothing" approach that we recommend as the base mandatory approach.

3.5. URL Signing

URL signing is an authorization method for content delivery. This is based on embedding the HTTP URL with information that can be validated to ensure that the request has legitimate access to the content. There are two parts: 1) parameters that convey authorization restrictions (e.g., source IP address and time period) and/or a protected URL portion, and 2) a message digest that confirms the integrity of the URL and authenticates the entity that creates the URL. The authorization parameters can be anything agreed upon between the entity that creates the URL and the entity that validates the URL. A key is used to generate the message digest (i.e., sign the URL) and validate the message digest. The two functions may or may not use the same key.

There are two types of keys used for URL signing: asymmetric keys and symmetric keys. Asymmetric keys always have a key pair made up of a public key and private key. The private key and public key are used for signing and validating the URL, respectively. A symmetric key is the same key that is used for both functions. Regardless of the type of key, the entity that validates the URL has to obtain the key. Distribution of the symmetric key requires security to prevent others from taking it. A public key can be distributed freely, while a private key is kept by the URL signer. The method for key distribution is out of scope for this document.

URL signing operates in the following way. A signed URL is provided by the content owner (i.e., URL signer) to the user during website navigation. When the user selects the URL, the HTTP request is sent to the CDN, which validates that URL before delivering the content.

3.5.1. HAS Implications

The authorization lifetime for URL signing is affected by HAS. The expiration time in the authorization parameters of URL signing limits the period that the content referenced by the URL can be accessed. This works for URLs that directly access the media content, but for HAS content the Manifest File contains another layer of URLs that reference the chunks. The chunk URL that is embedded in the content may be requested some undetermined amount of time later. The time period between access to the Manifest File and chunk retrieval may vary significantly. The type of content (i.e., live or VoD) impacts this time variance as well. This property of HAS content needs to be addressed for URL signing.

3.5.2. CDNI Considerations

For CDNI, the two types of request routing are DNS-based and HTTP-based. The use of symmetric vs. asymmetric keys for URL signing has implications for the trust model between the CSP and CDNs and for the key distribution method that can be used.

DNS-based request routing does not change the URL. In the case of a symmetric key, the CSP and the Authoritative CDN have a business relationship that allows them to share a key (or multiple keys) for URL signing. When the user requests content from the Authoritative CDN, the URL is signed by the CSP. The Authoritative CDN (as a uCDN) redirects the request to a dCDN via DNS. There may be more than one level of redirection to reach the delivering CDN. The user would obtain the IP address from DNS and send the HTTP request to the delivering CDN, which needs to validate the URL. This requires that the key be distributed from the Authoritative CDN to the delivering CDN. This may be problematic when the key is exposed to a delivering CDN that does not have a relationship with the CSP. The combination of DNS-based request routing and symmetric key function is a generic issue for URL signing and not specific to HAS content. In the case of asymmetric keys, the CSP signs the URL with its private key. The delivering CDN validates the URL with the associated public key.

HTTP-based request routing changes the URL during the redirection procedure. In the case of a symmetric key, the CSP signs the original URL with the same key used by the Authoritative CDN to validate the URL. The Authoritative CDN (as a uCDN) redirects the request to the dCDN. The new URL is signed by the uCDN with the same key used by the dCDN to validate that URL. The key used by the uCDN to validate the original URL is expected to be different than the key used to sign the new URL. In the case of asymmetric keys, the CSP signs the original URL with its private key. The Authoritative CDN validates that URL with the CSP's public key. The Authoritative CDN redirects the request to the dCDN. The new URL is signed by the uCDN with its private key. The dCDN validates that URL with the uCDN's public key. There may be more than one level of redirection to reach the delivering CDN. The URL signing operation described previously applies at each level between the uCDN and dCDN for both symmetric keys and asymmetric keys.

URL signing requires support in most of the CDNI interfaces. The CDNI Metadata interface should specify the content that is subject to URL signing and provide information to perform the function. The dCDN should inform the uCDN that it supports URL signing in the asynchronous capabilities information advertisement as part of the Request Routing interface. This allows the CDN selection function in request routing to choose the dCDN with URL signing capability when the CDNI Metadata of the content requires this authorization method. The logging interface provides information on the authorization method (e.g., URL signing) and related authorization parameters used for content delivery. Having the information in the URL is not sufficient to know that the Surrogate enforced the authorization. URL signing has no impact on the control interface.

3.5.3. Option 5.1: Do Nothing

This approach means that the CSP can only perform URL signing for the top-level Manifest File. The top-level Manifest File contains chunk URLs or lower-level Manifest File URLs, which are not modified (i.e., no URL signing for the embedded URLs). In essence, the lower-level Manifest Files and chunks are delivered without content access authorization.

Effect on CDNI interfaces:

- None

Advantages/Drawbacks:

+ Top-level Manifest File access is protected

+ The uCDN and dCDN do not need to be aware of HAS content

- Lower-level Manifest Files and chunks are not protected, making this approach unqualified for content access authorization

3.5.4. Option 5.2: Flexible URL Signing by CSP

In addition to URL signing for the top-level Manifest File, the CSP performs flexible URL signing for the lower-level Manifest Files and chunks. For each HAS session, the top-level Manifest File contains signed chunk URLs or signed lower-level Manifest File URLs for the specific session. The lower-level Manifest File contains session-based signed chunk URLs. The CSP generates the Manifest Files dynamically for the session. The chunk (segment/fragment) is delivered with content access authorization using flexible URL signing, which protects the invariant portion of the URL. A "segment" URL (e.g., HLS) is individually signed for the invariant

URL portion (relative URL) or the entire URL (absolute URL without redirection) in the Manifest File. A "fragment" URL (e.g., HTTP Smooth Streaming) is signed for the invariant portion of the template URL in the Manifest File. More details are provided later in this section. The URL signing expiration time for the chunk needs to be long enough to play the video. There are implications related to signing the URLs in the Manifest File. For live content, the Manifest Files are requested at a high frequency. For VoD content, the Manifest File may be quite large. URL signing can add more computational load and delivery latency in high-volume cases.

For HAS content, the Manifest File contains the relative URL, absolute URL without redirection, or absolute URL with redirection for specifying the chunk location. Signing the chunk URL requires that the CSP know the portion of the URL that remains when the content is requested from the delivering CDN Surrogate.

For absolute URLs without redirection, the CSP knows that the chunk URL is explicitly linked with the delivering CDN Surrogate and can sign the URL based on that information. Since the entire URL is set and does not change, the Surrogate can validate the URL. The CSP and the delivering CDN are expected to have a business relationship in this case, and so either symmetric keys or asymmetric keys can be used for URL signing.

For relative URLs, the URL of the Manifest File provides the root location. The method of request routing affects the URL used to ultimately request the chunk from the delivering CDN Surrogate. For DNS, the original URL does not change. This allows the CSP to sign the chunk URL based on the Manifest File URL and the relative URL. For HTTP, the URL changes during redirection. In this case, the CSP does not know the redirected URL that will be used to request the Manifest File. This uncertainty makes it impossible to accurately sign the chunk URLs in the Manifest File. Basically, URL signing using this reference method "as is" for protection of the entire URL is not supported. However, instead of signing the entire URL, the CSP signs the relative URL (i.e., the invariant portion of the URL) and conveys the protected portion in the authorization parameters embedded in the chunk URL. This approach works in the same way as absolute URLs without redirection, except that the HOST part and (part of) the PATH part of the URL are not signed and validated. The security level should remain the same, as content access authorization ensures that the user that requested the content has the proper credentials. This scheme does not seem to compromise the authorization model, since the resource is still protected by the authorization parameters and message digest. Further evaluation of security might be helpful.

For absolute URLs with redirection, the method of request routing affects the URL used to ultimately request the chunk from the delivering CDN Surrogate. This case has the same conditions as those indicated above for the relative URL. The difference is that the URL is for the chunk instead of the Manifest File. For DNS, the chunk URL does not change and can be signed by the CSP. For HTTP, the URL used to deliver the chunk is unknown to the CSP. In this case, the CSP cannot sign the URL, and this method of reference for the chunk is not supported.

Effect on CDNI interfaces:

- Requires the ability to exclude the variant portion of the URL in the signing process. (NOTE: Is this issue specific to URL signing support for HAS content and not CDNI?)

Advantages/Drawbacks:

+ The Manifest File and chunks are protected

+ The uCDN and dCDN do not need to be aware of HAS content

+ DNS-based request routing with asymmetric keys and HTTP-based request routing for relative URLs and absolute URLs without redirection work

- The CSP has to generate Manifest Files with session-based signed URLs and becomes involved in content access authorization for every HAS session

- Manifest Files are not cacheable

- DNS-based request routing with symmetric keys may be problematic due to the need for transitive trust between the CSP and delivering CDN

- HTTP-based request routing for absolute URLs with redirection does not work, because the URL used by the delivering CDN Surrogate is unknown to the CSP

3.5.5. Option 5.3: Flexible URL Signing by uCDN

This is similar to the previous section, with the exception that the uCDN performs flexible URL signing for the lower-level Manifest Files and chunks. URL signing for the top-level Manifest File is still provided by the CSP.

Effect on CDNI interfaces:

- Requires the ability to exclude the variant portion of the URL in the signing process. (NOTE: Is this issue specific to URL signing support for HAS content and not CDNI?)

Advantages/Drawbacks:

+ The Manifest File and chunks are protected

+ The CSP does not need to be involved in content access authorization for every HAS session

+ The dCDN does not need to be aware of HAS content

+ DNS-based request routing with asymmetric keys and HTTP-based request routing for relative URLs and absolute URLs without redirection work

- The uCDN has to generate Manifest Files with session-based signed URLs and becomes involved in content access authorization for every HAS session

- Manifest Files are not cacheable

- The Manifest File needs to be distributed through the uCDN

- DNS-based request routing with symmetric keys may be problematic due to the need for transitive trust between the uCDN and non-adjacent delivering CDN

- HTTP-based request routing for absolute URLs with redirection does not work, because the URL used by the delivering CDN Surrogate is unknown to the uCDN

3.5.6. Option 5.4: Authorization Group ID and HTTP Cookie

Based on the Authorization Group ID metadata, the CDN validates the URL signing or validates the HTTP cookie for request of content in the group. The CSP performs URL signing for the top-level Manifest File. The top-level Manifest File contains lower-level Manifest File

URLs or chunk URLs. The lower-level Manifest Files and chunks are delivered with content access authorization using an HTTP cookie that contains session state associated with authorization of the top-level Manifest File. The Group ID metadata is used to associate the related content (i.e., Manifest Files and chunks). It also specifies content (e.g., regexp method) that needs to be validated by either URL signing or an HTTP cookie. Note that the creator of the metadata is HAS aware. The duration of the chunk access may be included in the URL signing of the top-level Manifest File and set in the cookie. Alternatively, the access control duration could be provided by the CDNI Metadata interface.

Effect on CDNI interfaces:

- CDNI Metadata interface: Authorization Group ID metadata identifies the content that is subject to validation of URL signing or validation of an HTTP cookie associated with the URL signing

- CDNI Logging interface: Report the authorization method used to validate the request for content delivery

Advantages/Drawbacks:

+ The Manifest File and chunks are protected

+ The CDN does not need to be aware of HAS content

+ The CSP does not need to change the Manifest Files

- Authorization Group ID metadata is required (i.e., CDNI Metadata interface enhancement)

- Requires the use of an HTTP cookie, which may not be acceptable in some environments (e.g., where some targeted User Agents do not support HTTP cookies)

- The Manifest File has to be delivered by the Surrogate

3.5.7. Option 5.5: HAS Awareness with HTTP Cookie in CDN

The CDN is aware of HAS content and uses URL signing and HTTP cookies for content access authorization. URL signing is fundamentally about authorizing access to a content item or its specific content collections (representations) for a specific user during a time period, possibly also using some other criteria. A chunk is an instance of the sets of chunks referenced by the Manifest File for the content item or its specific content collections. This relationship means that once the dCDN has authorized the Manifest File, it can assume that the associated chunks are implicitly authorized. The new function for the CDN is to link the Manifest File with the chunks for the HTTP session. This can be accomplished by using an HTTP cookie for the HAS session.

After validating the URL and detecting that the requested content is a top-level Manifest File, the delivering CDN Surrogate sets an HTTP cookie with a signed session token for the HTTP session. When a request for a lower-level Manifest File or chunk arrives, the Surrogate confirms that the HTTP cookie value contains the correct session token. If so, the lower-level Manifest File or chunk is delivered in accordance with the transitive authorization mechanism. The duration of the chunk access may be included in the URL signing of the top-level Manifest File and set in the cookie. The details of the operation are left to be determined later.

Effect on CDNI interfaces:

- CDNI Metadata interface: New metadata identifies the content that is subject to validation of URL signing and information in the cookie for the type of HAS content

- Request Routing interface: The dCDN should inform the uCDN that it supports URL signing for known HAS content types in the asynchronous capabilities information advertisement. This allows the CDN selection function in request routing to choose the appropriate dCDN when the CDNI Metadata identifies the content

- CDNI Logging interface: Report the authorization method used to validate the request for content delivery

Advantages/Drawbacks:

+ The Manifest File and chunks are protected

+ The CSP does not need to change the Manifest Files

- Requires full HAS awareness on the part of the uCDN and dCDN

- Requires extensions to CDNI interfaces

- Requires the use of an HTTP cookie, which may not be acceptable in some environments (e.g., where some targeted User Agents do not support HTTP cookies)

- The Manifest File has to be delivered by the Surrogate

3.5.8. Option 5.6: HAS Awareness with Manifest File in CDN

The CDN is aware of HAS content and uses URL signing for content access authorization of Manifest Files and chunks. The CDN generates or rewrites the Manifest Files and learns about the chunks based on the Manifest File. The embedded URLs in the Manifest File are signed by the CDN. The duration of the chunk access may be included in the URL signing. The details of the operation are left to be determined later. Since this approach is based on signing the URLs in the Manifest File, the implications for live and VoD content mentioned in Section 3.5.4 apply.

Effect on CDNI interfaces:

- CDNI Metadata interface: New metadata identifies the content that is subject to validation of URL signing and information in the cookie for the type of HAS content
- Request Routing interface: The dCDN should inform the uCDN that it supports URL signing for known HAS content types in the asynchronous capabilities information advertisement. This allows the CDN selection function in request routing to choose the appropriate dCDN when the CDNI Metadata identifies the content
- CDNI Logging interface: Report the authorization method used to validate the request for content delivery

Advantages/Drawbacks:

+ The Manifest File and chunks are protected

+ The CSP does not need to change the Manifest Files

- Requires full HAS awareness on the part of the uCDN and dCDN
- Requires extensions to CDNI interfaces
- Requires the CDN to generate or rewrite the Manifest File
- The Manifest File has to be delivered by the Surrogate

3.5.9. Recommendations

The authors consider Option 5.1 (do nothing) unsuitable for access control of HAS content.

Where the HTTP cookie mechanism is supported by the targeted User Agents and the security requirements can be addressed through the proper use of HTTP cookies, the authors recommend using Option 5.4 (Authorization Group ID and HTTP cookie) and therefore that Option 5.4 be supported by the CDNI solution. This method does not require Manifest File manipulation, as Manifest File manipulation may be a significant obstacle to deployment. Otherwise, the authors recommend that Option 5.2 (flexible URL signing by the CSP) or Option 5.3 (flexible URL signing by the uCDN) be used and therefore that flexible URL signing be supported by the CDNI solution. Options 5.2 and 5.3 protect all the content, do not require that the dCDN be aware of HAS, do not impact CDNI interfaces, support all different types of devices, and support the common cases of request routing for HAS content (i.e., DNS-based request routing with asymmetric keys and HTTP-based request routing for relative URLs).

Options 5.5 and 5.6 (HAS awareness in CDNs using HTTP cookies or Manifest Files) have some advantages that should be considered for future support (e.g., a CDN that is aware of HAS content can manage the content more efficiently in a broader context). Content distribution, storage, delivery, deletion, access authorization, etc. can all benefit. Including HAS awareness as part of the current CDNI charter, however, would almost certainly delay the CDNI WG’s milestones, and the authors therefore do not recommend it right now.

3.6. Content Purge

At some point in time, a uCDN might want to remove content from a dCDN. With regular content, this process can be relatively straightforward; a uCDN will typically send the request for content removal to the dCDN, including a reference to the content that it wants to remove (e.g., in the form of a URL). However, due to the fact that HAS content consists of large groups of files, things might be more complex. Section 3.1 described a number of different scenarios for doing file management on these groups of files, while Section 3.2 listed the options for performing content acquisition on these content collections. This section presents the options for requesting a content purge for the removal of a content collection from a dCDN.

3.6.1. Option 6.1: No HAS Awareness

The most straightforward way to signal content purge requests is to just send a single purge request for every file that makes up the content collection. While this method is very simple and does not require HAS awareness, it obviously creates signaling overhead between the uCDN and dCDN, since a reference is to be provided for each content chunk to be purged.

Effect on CDNI interfaces:

- None

Advantages/Drawbacks (apart from those already listed under Option 3.3):

+ Does not require changes to the CDNI interfaces or HAS awareness

- Requires individual purge request for every file making up a content collection (or, alternatively, requires the ability to convey references to all the chunks making up a content collection inside a purge request), which creates signaling overhead

3.6.2. Option 6.2: Purge Identifiers

There exists a potentially more efficient method for performing content removal of large numbers of files simultaneously. By including a "Purge IDentifier (Purge-ID)" in the metadata of a particular file, it is possible to virtually group together different files making up a content collection. A Purge-ID can take the form of an arbitrary number or string that is communicated as part of the CDNI Metadata interface, and that is the same for all files making up a particular content item but different across different content items. If a uCDN wants to request that the dCDN remove a content collection, it can send a purge request containing this Purge-ID. The dCDN can then remove all files that share the corresponding Purge-ID.

The advantage of this method is that it is relatively simple to use by both the dCDN and uCDN and requires only limited additions to the CDNI Metadata interface and CDNI Control interface.

The Purge-ID is similar to the CCID discussed in Section 3.4.2.2 for handling HAS logging, and we note that further thought is needed to determine whether the CCID and Purge-ID should be collapsed into a single element or remain separate elements.

Effect on CDNI interfaces:

- CDNI Metadata interface: Add metadata field for indicating Purge-ID
- CDNI Control interface: Add functionality to convey a Purge-ID in purge requests

Advantages/Drawbacks:

+ Allows for efficient purging of content from a dCDN

+ Does not require HAS awareness on the part of a dCDN

3.6.3. Recommendations

Based on the listed pros and cons, the authors recommend that the WG have mandatory support for Option 1.1 (do nothing). In addition, because of its very low complexity and its benefit in facilitating low-overhead purge of large numbers of content items simultaneously, the authors recommend that Purge-IDs (Option 6.2; see Section 3.6.2) be supported as an optional feature by the CDNI Metadata interface and the CDNI Control interface.

3.7. Other Issues

This section includes some HAS-specific issues that came up during the discussion of this document and that do not fall under any of the categories discussed in the previous sections.

- As described in Section 2.2, a Manifest File might be delivered by either a CDN or the CSP and thereby be invisible to the CDN delivering the chunks. Obviously, the decision of whether the CDN or CSP delivers the Manifest File is made between the uCDN and CSP, and the dCDN has no choice in the matter. However, some dCDNs might only want to offer their services in the cases where they have access to the Manifest File (e.g., because their internal architecture is based on the knowledge inside the Manifest File). For these cases, it might be useful to include a field in the CDNI Capability Advertisement to allow dCDNs to advertise the fact that they require access to the Manifest File.

4. Security Considerations

This document does not discuss security issues related to HTTP or HAS delivery, as these topics are expected to be discussed in the CDNI WG documents, including [CDNI-FRAMEWORK].

5. Acknowledgements

The authors would like to thank Kevin Ma, Stef van der Ziel, Bhaskar Bhupalam, Mahesh Viveganandhan, Larry Peterson, Ben Niven-Jenkins, and Matt Caulfield for their valuable contributions to this document.

6. References

6.1. Normative References

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

6.2. Informative References

[CDNI-FRAMEWORK]
Peterson, L., Ed., and B. Davie, "Framework for CDN Interconnection", Work in Progress, February 2013.

[CDNI-LOGGING]
Bertrand, G., Ed., Stephan, E., Peterkofsky, R., Le Faucheur, F., and P. Grochocki, "CDNI Logging Interface", Work in Progress, October 2012.

[CDNI-REQUIREMENTS]

Leung, K., Ed., and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", Work in Progress, July 2013.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

Authors' Addresses

Ray van Brandenburg
TNO
Brassersplein 2
Delft 2612CT
the Netherlands

Phone: +31-88-866-7000
EMail: ray.vanbrandenburg@tno.nl

Oskar van Deventer
TNO
Brassersplein 2
Delft 2612CT
the Netherlands

Phone: +31-88-866-7000
EMail: oskar.vandeventer@tno.nl

Francois Le Faucheur
Cisco Systems
E.Space Park - Batiment D
6254 Allee des Ormes - BP 1200
06254 Mougins cedex
France

Phone: +33 4 97 23 26 19
EMail: flefauch@cisco.com

Kent Leung
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408-526-5030
EMail: kleung@cisco.com