CDNI
Internet-Draft
Intended status: Informational
Expires: November 23, 2012

R. van Brandenburg
O. van Deventer
TNO
F. Le Faucheur
K. Leung
Cisco Systems
May 22, 2012

Models for adaptive-streaming-aware CDN Interconnection
draft-brandenburg-cdni-has-01

Abstract

This documents presents thoughts on the potential impact of supporting HTTP Adaptive Streaming technologies in CDN Interconnection scenarios. Our intent is to spur discussion on how the different CDNI interfaces could, and should, deal with content delivered using adaptive streaming technologies and to facilitate working group decisions.

Table of Contents

1. Introduction

HTTP Adaptive Streaming (HAS) 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 RTP/RTSP, RTMP and MMS. This document presents a discussion on what the impact of these different characteristics is to the CDNI interfaces and what HAS-specific optimizations may be required or may be desirable. The scope of this document in its current form is explicitly not to propose any specific solution, but merely to present the available options so that the WG can make an informed decision on which way to go.

1.1. Terminology

This document uses the terminology defined in [I-D.ietf-cdni-problem-statement].

In addition, the following terms are used throughout this document:

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. It is the object of a request routing operation in a CDN. 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: Not-chunked content that is the basis for a segmentation of 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 Media Presentation Description (MPD) file, is a file that list the way the content has been chunked (possibly for multiple encodings) and 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 Smooth Streaming (HSS) and Adobe's HTTP Dynamic Streaming (HDS), and 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 chapter will look 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 will describe the various methods to store HAS content and Section 2.2 will list three methods that are used to address HAS content in a CDN.

2.1. Segmentation versus Fragmentation

All HAS implementations are based around a concept referred to as chunking: the concept of having a server split content up in numerous fixed duration chunks, which 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 independent 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 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 which allow for storing this type chunked content, Fragmented MP4 is most commonly used. With fragmentation, a specific chunk is addressable by subfixing the common file URL with an identifier uniquely identifying the chunk one is interested in, either by timestamp, by byterange, 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, both from 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, either in the form of segments or in the form of 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 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 encodings, and thus Chunk Collections, the content is available in.

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 which, at each instant in time, do know at which servers each file resides. The CDNI interfaces designed in 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 only relate 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 apply that are used for specifying chunk locations.

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 where the Manifest File has been acquired from. 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 a manifest 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 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 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

The downside of using relative URLs is that it forces a CDN to deliver all segments belonging to a given content item with the same surrogate that delivered the Manifest File for that content item. 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 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 which will redirect the client to a surrogate (Absolute URL with Redirection) or point directly to a surrogate hosting the requested content (Absolute URL 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 includes some arguments identifying the requested segment.

The advantage of using Absolute URLs with Redirection is that it allows for maximum flexibility (since chunks can be distributed across surrogates and CDN 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 adds 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 URL without Redirection

In the case of the Absolute URL 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 URLs without Redirection is the following:

http://surrogate.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), as well as the server that is expected to deliver the segment (in this case surrogate.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 Absolute URLs without Redirection 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).

3. Possible HAS Optimizations

In the previous chapter, some of the unique properties of HAS have been discussed. Furthermore, some of the CDN-specific design decisions with regards to addressing chunks have been detailed. In this chapter, the impact of supporting HAS in CDN Interconnection scenarios will be 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 in way that 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 chapter will briefly describe each of the following problem areas together with a number of different options for dealing with them. Section 3.1 will discuss the problem of how to deal with file management of groups of files, or Content Collections. Section 3.2 will deal with a related topic: how to do content acquisition of Content Collections between the uCDN and 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. Section 3.6 finally, describes different scenarios for dealing with the removal of HAS content from CDNs.

3.1. File Management and Content Collections

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 (segment, 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, which are stored independently from each other. An advantage of this

approach is that 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.

This section will 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.

3.1.1. Option 1.1: No HAS awareness

This first 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 explicitely made aware of the relationship between chunks and it 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 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 contains byte range requests, in which case the dCDN might be able to acquire fragmented content as a single file (see Section 3.2.2).

Effect on CDN 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. 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), or merge the different chunks together and place them in the same container (e.g. fragmented MP4).

The downside of this is that in order to do so, the dCDN needs to be fully HAS aware.

Effect on CDN 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 dCDN to store fragmented content as a single file, reducing file management overhead

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

3.2. Content Acquisition of Content Collections

In the previous section the relationship between file management and HAS in a CDNI scenario has been discussed. This section will discuss a related topic, which is content acquisition between two CDNs.

3.2.1. Option 2.1: No HAS awareness

This first option assumes no HAS awareness in both the involved CDNs and the CDNI Interfaces. Just as with Option 1.1 discussed in the previous section 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 in cases with fragmented content where the uCDN uses Manifest Files which 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 CDN Interfaces:

o None

Advantages/Drawbacks:

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

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

3.2.2. Option 2.2: Allow single file acquisition of fragmented content

As discussed in Section 3.2.1, there is one (fairly rare) in cases where fragmented content can be acquired as a single file without any HAS awareness and that is when fragmented content is used and where a Manifest File includes byte range request. 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 single file and individual chunks to serve).

Effect on CDN 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 for more efficient content acquisition in all HAS-specific supported forms

- Requires full HAS awareness on part of dCDN

- Requires significant CDNI Metadata Interface extensions

3.3. Request Routing of HAS content

In this section the effect HAS content has on request routing will be 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 without

Redirection and Absolute URLs with 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 Content Provider. Some Content Providers 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 Content Provider, a CDN may either host the Manifest Files as they were created by the Content Provider, or modify the Manifest File to adapt it to its particular architecture (e.g. by changing relative URLs to Absolute URLs which point to the CDN Request Routing function).

3.3.1. Option 3.1: No HAS awareness

This first 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 have 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 Collections 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 Content Provider 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 Content Provider 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 that chunk to the dCDN. In that case it will probably redirect the client to the Request Routing function of the dCDN (assuming 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 redirections in the case of iterative redirection, or result in one redirection plus one CDNI Request Routing/Redirection Interface request/response).

With no HAS awareness, Relative URLs might or might not work depending on the HAS client implementation 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. The question here is whether the HAS client uses the IP address or the hostname in its chunk request to the dCDN surrogate. In case it uses the exact same hostname with which it requested the manifest file (and which is the result of a request routing operation) than everything works properly (because the dCDN can easily associate the CDNI metadata for the corresponding chuck and therefor serve the corresponding request). However, in situations where the client uses the IP address of the surrogate or does a reverse DNS look-up, the dCDN surrogate may not be able to associate CDNI metadata with the chucnk request and therefore may not be able to serve it. One exception to this is in the scenario where the PATH or QUERY part of the chunk request URL is enough to uniquely identify the particular chunk, in which case the dCDN Surrigate can associate CDNI metadata and serve the request

Since using Absolute URLs without Redirection inherently require a HAS aware CDN, they also cannot be used in this case. The reason for this is that with Absolute URLs without Redirection, 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 CDN Interfaces:

- None

Advantages/Drawbacks:

+ Supports Absolute URLs with Redirection

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

- Not possible to use Absolute URLs without Redirection

- Support for Relative URLs suffers from some brittleness. Makes assumptions on client-side implementation of the HAS client or on structure of PATH or QUERY

- Creates significant signaling overhead in case Absolute URLs with Redirection are used (inter-CDN request redirection for each chunk)

3.3.2. Option 3.2: Manifest File rewriting by uCDN

While Option 3.1 does allow for 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 modify the Manifest File to be able to remove itself from the request routing chain for chunks being referenced in the Manifest File. As described in Section 3.3.1, 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, the uCDN is able to remove this first step, and have the Manifest File point directly to the dCDN request routing function.

In order for the uCDN to be able to do this, it needs the location of the dCDN request routing function (or even better: the location of the dCDN surrogate). The simplest way to obtain this information is to use the CDNI Request Routing Interface in one of two ways. The first way would be to have the uCDN 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 comes in at the uCDN request routing node. The uCDN would then modify the manifest file and deliver the manifest file to the client. A second way to do it would be for the modification of the manifest file to only happen once, when the first client for that particular Content Collection (and redirected to that particular dCDN) sends a Manifest File request. The advantage of the first method is that it maximizes effiency and flexibility by allowing the dCDN to 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). The advantage of the second method is that the uCDN only has to modify the Manifest File once.

Effect on CDN Interfaces:

- CDNI Request Routing Interface: Allow uCDN to query dCDN for the location of its request routing function (is this covered by the existing RR interface?)

- uCDN: Allow for modification of manifest file

Advantages/Drawbacks:

+ Possible to significantly decrease signalling overhead when using Absolute URLs.

+ (Optional) Possible to have uCDN modify manifest with locations of surrogates in dCDN (turning Absolute URLs with Redirection in Absolute URLs without Redirection)

+ Minimal changes to CDNI Interfaces (no HAS awareness)

+ Does not require HAS awareness in dCDN

- Requires high level of HAS awareness in uCDN (for modifying manifest files)

3.3.3. Option 3.3: Two-step Manifest File rewriting

One of the possibilities with Option 3.3 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 sensitive information to the uCDN. In that case they can only provide the uCDN with the location of their request routing function and thereby not be able to use 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 minimal 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 that it 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 CDN Interfaces (apart from those listed under Option 3.3):

- o CDNI Metadata Interface: Add necessary fields for conveying HAS specific information (e.g. the files that make up the Content Collection) to the dCDN.

- o dCDN: Allow for modification of manifest file

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

+ Allows dCDN to use Absolute URLs without Redirection without having to convey sensitive information to the uCDN

- Requires high level of HAS awareness in dCDN (for modifying manifest files)

- Requires adding HAS-specific information to the CDNI Metadata Interface

3.4. Logging

3.4.1. HAS Considerations for CDNI Logging

As stated in [I-D.ietf-cdni-problem-statement], "the CDNI Logging interface enables details of logs or events to be exchanged between interconnected CDNs".

As discussed in [I-D.draft-bertrand-cdni-logging], the CDNI logging information can be used fort multiple purposes including maintenance/ debugging by uCDN, accounting (e.g. in view of billing or settlement), 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 log records would typically be generated on a per-chunk-delivery basis instead of per-content-item-delivery basis. This impacts the scale of every processing step in the Logging Process (see Section 8 of [I-D.draft-bertrand-cdni-logging]), including:

a. Logging Generation and Storing of logs on CDN elements (Surrogate, Request Routers,..)

b. Logging Aggregation within a CDN

c. Logging Manipulation (including Logging Protection, Logging Filtering , Logging Update and Rectification)

d. (Where needed) Logging CDNI Reformatting (e.g. reformatting from CDN-specific format to the CDNI Logging Interface format for export by dCDN to uCDN)

e. Logging exchange via CDNI Logging Interface

f. (Where needed) Logging Re-Reformatting (e.g. reformatting from CDNI Logging Interface format into log-consuming specific 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 tool to debug)

Note that there may be multiple instances of step [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 step [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, step [a], [b] and [d] cannot operate on a per-HAS-session basis and also need to operate on a per-chunk basis).

3.4.2. Candidate Approaches

The following sub-sections discusses the main candidate approaches identified so far for CDNI in terms of dealing with HAS with respect to Logging.

3.4.2.1. Option 4.1: "Do-Nothing" Approach

In this approach, 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]. This approach requires that step [a], [b] , [c], [d] and [e] be performed on per-chunk basis. This approach allows [g] to be performed either on a per-chunk basis (assuming step [e] maintains per-chunk records) or on a more "summarized" manner such as per-HAS-Session basis assuming step [e] summarizes per-chunk records into per-HAS-session records).

Effect on CDN Interfaces:

o None

Effect on uCDN and dCDN:

- 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 (possibly required) for some Log consuming applications (e.g. debugging)

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

+ No extension needed on CDNI interfaces

- High volume of logging information to be handled (storing & processing) at every step of the Logging process from [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). While the high volume of logging information is a potential concern, we are seeking expert input on whether it is a real practical issue, and if yes, then in what timeframe/ assumptions.

3.4.2.2. Option 4.2: "CDNI Metadata Content Collection ID" Approach

In this approach, a "Content Collection ID" (CCID) field is distributed through the CDNI Metadata Interface and the same CCID value is associated with every chunk of the same Content Collection. The objective of this field is to facilitate summarization of per-chunk records at step [f] into something along the lines of per-HAS-session logs, at least for the Log Consuming application that do not require per-chunk detailed information (for example billing).

[Editor's Note: would there be value in adding a little more info in the metadata such as which HAS-scheme is used?]

Effect on CDN Interfaces:

- One additional metadata field (CCID) in CDNI Metadata Interface

Effect on uCDN and dCDN:

- 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 (possibly required) for some Log consuming applications (e.g. debugging)

+ Easier integration (at least in the short term) into existing Logging tools since those 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 lower volume of logging information in step [g] by log consuming applications that do not need per-chunk record details (e.g. billing)

- High volume of logging information to be handled (storing & processing) at every step of the Logging process from [a] to [f]. While the high volume of logging information is a potential concern, we are seeking input on whether it is a real practical issue, and if yes in what timeframe/assumptions

3.4.2.3. Option 4.3: "CDNI Metadata Content Collection ID With dCDN Summarization" Approach

In this approach, a "Content Collection ID" (CCID) field is distributed through the CDNI Metadata Interface and the same CCID value is associated with every chunk of the same Content Collection. In this approach, a summarization of per-chunk records is performed at step [d] (or in earlier steps) taking advantage of the CCID, so that a reduced volume of logging information is to be handled in steps [e] to [g] of the logging process (and is optionally also possible in steps [a] to [c]). The objective of this approach is to reduce the volume of logging information early in the Logging process

Regarding the summarization performed at step [d] (or in earlier steps), there is a continuum in terms of trade-off between level of summarization of per-chunk records and information loss. For example, it appears possible to perform a summarization that results is significant gains with limited information loss, perhaps using summarized logs along the lines of the Event-Based Logging format discussed in section 3.2.2 of [I-D.draft-lefaucheur-cdni-logging-delivery]. Alternatively, it may be possible to perform a summarization that results in very significant gains with significant information loss, perhaps using summarized logs along the lines of the Summary-Based Logging format discussed in section 3.2.3 of

[I-D.draft-lefaucheur-cdni-logging-delivery].

Effect on CDN Interfaces:

- One additional metadata field (CCID) in CDNI Metadata Interface

- Summarized logging information in CDNI Logging Information

Effect on uCDN and dCDN:

- None

Advantages/Drawbacks:

+ Lower volume of logging information to be handled (storing & processing) at every step of the Logging process from [e] to [g], and optionally from [a] to [d] also

+ Small extensions to CDNI interfaces needed

- 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 had ego be selected. While full level of detail may not be needed for some Log consuming applications (e.g. billing), the full level of detail is likely valuable (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 are all capable of handling per-chunk records and may not be capable of handling CDNI summarized records

3.4.2.4. Option 4.4: "CDNI Logging Interface Compression" Approach

In this approach, a loss-less compression technique is applied to the sets of Logging records (e.g. Logging files) for transfer on the IETF 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 CDN Interfaces:

- One additional compression mechanism to be included in the CDNI Logging Interface

Effect on uCDN and dCDN:

- 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 (possibly required) for some Log consuming applications (e.g. debugging)

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

+ Small extension to CDNI interfaces needed

+ Reduced volume of logging information in step [e]

- High volume of logging information to be handled (storing & processing) at every step of the Logging process from [a] to [g], except [e]. While the high volume of logging information is a potential concern, we are seeking expert input on whether it is a real practical issue, and if yes, then in what timeframe/ assumptions

Input is sought on expected compression gains achievable in practice over sets of logs containing per-chunk records.

3.4.2.5. Option 4.5: "Full HAS awareness" Approach

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 Surrogates leverage the HAS information distributed through the CDNI metadata and their HAS-awareness to generate summarized logging information in the very first place. The objective of that approach is to operate on lower volume of logging information right from the very first step of the Logging process.

The summarized HAS logs generated by the Surrogates in this approach are similar to those discussed in the section " "CDNI Metadata Content Collection ID With dCDN Summarization" Approach" and the same trade-offs between information loss and summarization gain apply.

Effect on CDN Interfaces:

- One significant extension of the CDNI Metadata Interface 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

Effect on uCDN and dCDN:

- Full HAS-awareness by dCDN Surrogates

Advantages/Drawbacks:

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

+ Accurate generation of summarized logs because of HAS awareness on Surrogate

- Very significant extensions to CDNI interfaces needed including per HAS-protocol specific support

- Very significant additional requirement for HAS awareness on dCDN

- 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 had ego be selected. While full level of detail may not be needed for some Log consuming applications (e.g. billing), the full level of detail is likely valuable (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 are all capable of handling per-chunk records and may not be capable of handling CDNI summarized records

Input is sought on expected compression gains achievable in practice over sets of logs containing per-chunk records.

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 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 protected URL portion, and 2) authenticator value that confirms the integrity of the URL and authenticates the URL creator. 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 authenticator (i.e. sign the URL) and validate the authenticator. This may or may not be the same key.

There are two types of keys: asymmetric keys and symmetric key. 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 obtain the key. Distribution for the symmetric key requires security to prevent others from taking it. Public key can be distributed freely while private key is kept by the URL signer.

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 delivery node which validates that the URL before delivering the content.

3.5.1. URL Signing in CDNI

For CDNI, URL Signing is based on the Upstream CDN and Downstream CDN as entities that sign and validate the URL, respectively. HTTP-based request routing changes the URL. Thus, each redirection requires the URL to be re-signed. The alternative is protection of only the invariant portion of the URL to avoid re-signing by the transit CDN. DNS-based request routing maintains the same URL. In essence, there are three cases: 1) The URL is validated, rewritten, and re-signed at each request routing hop, 2) The URL is changed but only the invariant portion of URL is validated, and 3) The URL remains the same and validated by the delivery CDN surrogate.

The Downstream CDN needs to obtain the key for validating the URL. When asymmetric keys are used, the public key can be retrieved from the authorization parameters embedded in the URL. The Downstream CDN can validate the URL with the public key, regardless of the relationship with the URL signer (i.e. Downstream CDN has or does not have a direct relationship with the Upstream CDN that signed the URL). When symmetric key is used, the Downstream CDN needs to obtain the key in a secure method that is out of scope. For cascaded CDNs, a common key is distributed for validation of unaltered URL to support DNS-based request routing. Alternatively, a shared key is distributed between adjacent CDNs to support rewritten URL that is used in HTTP-based request routing.

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 Downstream CDN should inform the Upstream CDN 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 Downstream CDN 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. URL Signing has no impact on the Control interface.

3.5.2. Option 5.1: No HAS awareness

For HTTP Adaptive Streaming, the Manifest File contains the Relative Locator, Absolute Locator without Redirection, or Absolute Locator with Redirection for specifying the chunk location. The Authoritative CDN performs URL signing for the Manifest File and chunks. The delivery CDN surrogate has to obtain the key to validate the URL. Although the Manifest file and chunk are treated the same by the CDN that is not HAS aware, there are some implications for URL Signing based on the method used to reference the chunk location in the Manifest file.

For Absolute URL without Redirection, the Authoritative CDN signs the chunk URL which is associated with the delivery CDN surrogate. Since the entire URL is set and does not change during request routing (i.e. DNS-based redirection only and no HTTP-based redirection), there are no issues for signing and validating the URL.

For Relative URL, the Authoritative CDN does not know the URL that will be ultimately used by a Downstream CDN to deliver the chunk. This uncertainty makes it impossible to accurately sign the URL in the Manifest File. URL Basically, URL Signing using this reference method, "as is" for entire URL protection, is not supported. However, instead of signing the entire URL, the Authoritative CDN signs the Relative URL (i.e. invariant portion of the URL) and conveys the protected portion in the authorization parameters embedded in the chunk URL. This approach works the same way as Absolute URL, except the HOST part and (part of) the PATH part of the URL are not signed and validated. The tradeoff is flexibility vs security. The advantage is that the content can be authorized to be delivered from any host with arbitrary directory path. The drawback is lack of protection for the entire URL. One aspect may outweigh the other depending on the needs of the Authoritative CDN.

For Absolute URL with Redirection, the Authoritative CDN signs the chunk URL with the intent that the targeted Request Routing function will validate and redirect the URL with a new URL signature. This

method is same as the Absolute URL without Redirection from URL Signing perspective because the signed URL is the same URL that is validated. Only difference is that the request routing function needs to redirect the request with a new URL which may be signed or not.

Effect on CDN Interfaces:

- oURL Signing is enabled on the CDNI Request Routing (i.e. Capabilities advertisement), Metadata, and Logging interfaces

Advantages/Drawbacks:

+ No HAS awareness necessary in CDNs, no changes to CDNI interface necessary

- URL Signing for every chunk may be considered as processing overhead

3.5.3. Option 5.2: HAS-awareness with Authorization Token

Up to this point, the Manifest File and chunks are treated as normal file in the HTTP delivery. There are no specific changes needed for the CDN to support HAS. However, if CDN is aware of HAS, then the Manifest File and chunk can be treated differently and appropriately.

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 with possibly 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 Downstream CDN 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 authorization token or session based encryption. This section covers the former and next section covers the latter.

After validating the URL and detecting that the requested content is a Manifest File, the delivery CDN surrogate creates a state and sets a HTTP cookie with authorization token for the HTTP session. When a request for a chunk arrives, the surrogate confirms that the HTTP cookie value contains the correct authorization token. If so, the chunk is delivered due to transitive authorization property.

Effect on CDN Interfaces:

- o URL Signing is enabled on the CDNI Request Routing (i.e. Capabilities advertisement), Metadata, and Logging interfaces

Advantages/Drawbacks:

+ Not necessary to validate every chunk URL

+ Less exposure to attacks on the URL Signing keys

- Surrogate needs to maintain state for the authorization token

3.5.4. Option 5.3: HAS-awareness with Session Based Encryption

After validating the URL and detecting that the requested content is a Manifest File, the delivery CDN surrogate manipulates the Manifest File by adding a reference to the key server for protection of specific chunks before it delivers the content. Also, the surrogate sets a HTTP cookie for the HTTP session. When a request for a chunk arrives, the surrogate identifies that the HTTP cookie value is for the same session. If so, the chunk is encrypted with the symmetric key obtained from the key server and then is delivered to the user.

Effect on CDN Interfaces:

- o URL Signing is enabled on the CDNI Request Routing (i.e. Capabilities advertisement), Metadata, and Logging interfaces

Advantages/Drawbacks:

+ Not necessary to validate every chunk URL

+ Less exposure to attacks on the URL Signing keys

+ Confidentiality for the chunk

- Surrogate needs to change the Manifest File

- Surrogate needs to maintain state for the HTTP session

- Key server is required to distribute key to surrogate and user

- Encryption and decryption processing

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 which it wants to remove (e.g. in the form of a URL). Due to the fact that HAS content consists of large groups of files however, things might be more complex. Section 3.1 describes a number of different scenarios for doing file management on these groups of files, while Section 3.2 list the options for performing Content Acquisition on these Content Collections. This section will present 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 a large signalling overhead between the uCDN and dCDN.

Effect on CDN Interfaces:

o None

Advantages/Drawbacks (apart from those 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 which creates large signalling 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 purge identifiers in the metadata of a particular file, it is possible to virtually group together different files making up a Content Collection. A purge identifier can take the form of a random number which is communicated as part of the CDNI Metadata Interface and which is the same for all files making up a particular Content Item. If a uCDN wants to request the dCDN to remove a Content Collection, it can send a purge request containing this purge identifier. The dCDN can then remove all files that contain the shared identifier.

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

[Editor's Note: Could the Purge Identifier introduced in this section

be related to the Content Collection Identifier introduced in Section 3.4.2.2? Should they be the same identifier?]

Effect on CDN Interfaces:

- o CDNI Metadata Interface: Add metadata field for indicating Purge Identifier
- o CDNI Control Interface: Add functionality to be able to send content purge requests containing Purge Identifiers

Advantages/Drawbacks:

- \+ Allows for efficient purging of content from a dCDN
- \+ Does not require HAS awareness on part of dCDN

4. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

5. Security Considerations

TBD.

6. References

6.1. Normative References

[I-D.ietf-cdni-problem-statement]
Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement, draft-ietf-cdni-problem-statement-03", January 2012.

[I-D.ietf-cdni-use-cases]
Bertrand, G., Ed., Stephan, E., Watson, G., Burbridge, T., Eardley, P., and K. Ma, "Use Cases for Content Delivery Network Interconnection, draft-ietf-cdni-use-cases-03", January 2012.

6.2. Informative References

[I-D.draft-bertrand-cdni-logging]
Bertrand, G., Ed. and E. Stephan, "CDNI Logging Interface".

[I-D.draft-lefaucheur-cdni-logging-delivery]
Le Faucheur, F., Viveganandhan, M., and K. Leung, "CDNI Logging Formats for HTTP and HTTP Adaptive Streaming Deliveries".

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
Greenside, 400 Avenue de Roumanille
Sophia Antipolis 06410
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