CDNI
Internet-Draft
Intended status: Informational
Expires: August 27, 2012

R. van Brandenburg
O. van Deventer
TNO

February 24, 2012

Models for adaptive-streaming-aware CDN Interconnection draft-brandenburg-cdni-has-00

Abstract

This documents presents thougths 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 should deal with content delivered using adaptive streaming technologies.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 27, 2012.

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 session-less 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 session-less 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. The scope of this document is explicitely not to define solutions, but merely to identify different methods of handling HAS in a CDN and the potential problems when using HAS in a CDNI context. The issues identified in this document may be used as input for defining HAS-specific requirements for the CDNI interfaces.

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 adressable 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. For a discussion on what may constitute a Content Item with regards to HAS, see section 3.1.1.

Chunk: A fixed length element that is the result of a segmentation or fragmentation operation being performed on a single encoding of the Original Content. A chunk is independently, and uniquely, addressable. Depending on the way a Chunk is stored, it may also be referred to as a Segment or Fragment.

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: Unchunked content that is the basis for a segmentation of fragmentation operation. Based on Original Content, multiple alternative encodings, resolutions or 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 encoding 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 being a single encoding, or variant, 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 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 (MSS) and Adobe's HTTP Dynamic Streaming (HDS), and various standardized solutions such as 3GPP AHS (AHS) and MPEG DASH (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 differences between these technologies and how these differences 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 length 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. Before requesting a particular chunk, a client can choose between mulitple alternatives of the same chunk, irrespective of the encoding of the chuncks it has requested earlier.

NOTE: The set of all chunks belonging to a single encoding, and thus the result of a single chunking operation, will from now on be referred to as a Chunk Collection. The set of all encodings of the same Original Content will be referred to as a Content Collection. A Content Collection can therefore consist of multiple Chunk Collections.

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 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 seperate 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 adressable 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 efficient with regards to caching. 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): 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 qualities, encodings, or variants. 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 seperate 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. Full Locator

One method for specifying locations of chunks (or other manifest files) in a manifest file is through the use of a Full Locator. A Full Locator takes the form of an URL and is defined by the fact that it directly points to the specific chunk on the actual the server that is expected to deliver the requested chunk to the client.

An example of a Full Locator is the following:

http://deliverynode.server.cdn.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 deliverynode.server.cdn.com). With this, the client has enough information to directly request the specific segment from the specified delivery node.

2.2.2. Relative Locator

Another method for specifying chunk locations in a manifest file is through the use of Relative Locator. A Relative Locator is a pointer that is relative to the location where the manifest file has been acquired from. In most cases a Relative Locator 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 a Relative Locator is used, all chunks will be delivered by the same delivery node that delivered the manifest file. A Relative Locator will therefore not include a hostname.

For example, in the case a manifest file has been requested (and received) from http://deliverynode.server.cdn.com/content_1/manifest.xml, a Relative Locator 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 Locator. In this case, the segment would then be requested from http://deliverynode.server.cdn.com/content_1/segments/segment1_1.ts

2.2.3. Chunk Request Routing

A final method for specifying chunk locations in a manifest file is through the use of request routing. In this case, chunks are handled by the request routing system of a CDN just as if they were 'normal' content items. A chunk request comes in at a central request routing node and is handled just as if it were a regular content request, which might include looking up the delivery node 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 delivery node.

An example of an URL pointing to a redirection mechanism might look as follows:

> http://requestrouting.cdn.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.

3. Impact of HAS on CDNI

In the previous chapter, some of the unique properties of HAS have been discussed. Furthermore, some of the CDN-specific design decision with regards to addressing chunks have been detailed. In this chapter, the impact of supporting HAS in CDNI will be discussed. The scope of this chapter is explicitely not to define solutions, but merely to identify potential problems and issues that need to be agreed on. For this purpose, each subsection will pose a number of open questions that will need to be answered by the CDNI WG. At a later stage, the answers to these questions may be used to solicit HAS-related requirements for the CDNI Interfaces.

The chapter is divided into three subsections. The first subsection, 3.1, will discuss the impact supporting HAS has on the general CDNI architecture, use cases and requirements. The other four subsections, 3.2 to 3.5, will discuss the impact of HAS on each of the four CDNI Interfaces.

3.1. General

This section will discuss the impact supporting HTTP Adaptive Streaming has on the general CDNI architecture, use cases and requirements.

3.1.1. On the definition of a Content Item in CDNI

[I-D.ietf-cdni-problem-statement] defines content as

> Content: Any form of digital data. One important form of content with additional constraints on distribution and delivery is continuous media (i.e. where there is a timing relationship between source and sink).

This very broad definition of content is useful for generalizing the CDNI interfaces in a way that allows them to be agnostic to the type of content that is delivered. However, what a Content Item in a CDN

constitutes may become relevant in the context of HAS if one considers a Content Item to be the element with which Content Metadata is associated, and the element that is the object of a CDN(I) request routing operation.

An example of a Content Item in the general sense is a video file or stream, such as a TV show or movie, or an audio file, such as an MP3. In a simple case, a single MP3 is a single Content Item. In a more complex case, a particular piece of content is made available in multiple resolutions, languages or qualities. A video item, for example, might be made available in three different resolutions. In these cases, it depends on the datamodel of a particular CDN (or a particular Content Provider) how it defines a Content Item. In some CDNs, all three video files might be seen as seperate Content Items, each with their own set of Content Metadata. In other CDNs, the three alternative encodings of the same content are seen as a single Content Item, with a single set of Content Metadata describing that the content consists of three different versions. The CDNI Interfaces defined in the CDNI WG are affected by these kinds differences in the ways different CDNs work and might need to be agnostic to these kinds of CDN-internal (or even Content Provider-related) decisions.

For content delivered using HAS, there is an even wider variety in the way different CDNs might interpret the definition of a Content Item. For example, CDNs using the Relative Locator method (see section 2.2.2) in their manifest files, might define all chunks that are part of the same Content Collection, and therefore referenced in the same (set of) manifest file(s), as a single Content Item for the purposes of Content Metadata and request routing. Other CDNs might define all chunks related to a single encoding of a particular video item, and thus part of the same Chunk Collection, as a single Content Item, thereby having multiple inter-related Content Items which are part of the same 'parent' Content Item. Yet another group of CDNs, especially those using the Chunk Request Routing method (see Section 2.2.3), might define every individual chunk as a seperate Content Item, with a seperate set of metadata describing each chunk.

In order for the CDNI WG to realise a standardized method of dealing with metadata, logging and request routing, it will be important to first have a common understanding of the term Content Item, and what it constitutes, especially with regard to HAS. One option would be to not impose a specific model, but allow the CDNI interfaces to support all the different definitions of Content Item (i.e. from considering each chunk to be a Content Item to considering all chunks originating from the same Original Content, and thus part of the same Content Collection, to be a single content item).

- Should the CDNI Interfaces be agnostic to the definition of a Content Item in a particular CDN?

And if this is not the case, then

- What constitutes a Content Item for the purposes of associating metadata and request routing?

- How does the definition of a Content Item relate to Chunks, Chunk Collections and Content Collection?

If the WG decides to not impose a specific definition of what constitutes a Content Item, it is for further study whether it is required for all parties involved in the delivery of a single video item, CSP, uCDN and dCDN, to use the same definition. For example, is it possible for the uCDN to define a complete Content Collection as a single Content Item, but for the dCDN which delivers the actual content to see each chunk as a seperate Content Item and handle the metadata accordingly?

- Is it necessary for all CDNs involved in the delivery of a single video item to use the same definition of a Content Item (e.g. can the uCDN define a Content Collection as a Content Item and the dCDN define a chunk as a Content Item)?

3.1.2. General CDNI-HAS Requirements

This section is a placeholder for HAS-specific CDNI requirements that are not related to a specific CDNI interface.

3.2. Impact on Request Routing Interface

This section will discuss the impact supporting HTTP Adaptive Streaming has on the CDNI Request Routing Interface.

3.2.1. Dealing with manifest files

In section 2.2, three different methods for identifying and addressing chunks from within a manifest file were described: Full Locators, Relative Locators and Chunk Request Routing. Of course not every current CDN will use and/or support all three methods. Some CDNs may only support one of the three methods, while others may support two or all three.

The question is whether all CDNs involved in the delivery of a single video item need to support the same method. Is it for example possible for a dCDN to use Chunck Request Routing while the uCDN uses Full Locators? This question boils down to the more fundamental

question of who manages manifest file. Should a dCDN be allowed to change a manifest file? Or even create a new one?

- Should the CDNI Interfaces be agnostic to the way chunks are identified in the manifest file and requested by the client?

- Should it be possible for a uCDN and dCDN to use two different methods of addressing chunks in manifest files?

And, related to this

- Should a dCDN be able to adapt a manifest file (i.e. is a manifest file part of the content, and therefore by definition not adaptable, or is it part of the delivery method) ?

If the CDNI WG decides that the anwer to these questions is negative, this will probably mean that the only supported method for Chunk Adressing is using Relative Locators. For both Full Locators as well as Chunk Request Routing, it is necessary for the delivering CDN to change the URLs specified in the manifest file.

3.2.2. HAS Request Routing

One of the essential questions relating to HAS and request routing is whether CDNI request routing is handled on a per chunk level, a per Content Collection level, or somewhere in between. This question is tightly related to the definition of a Content Item, discussed in section 3.1.1. If a Content Item is the object of request routing, then it depends on the definition of a Content Item what type of content element is being request routed.

- Should the CDNI Interfaces specify what element of a HAS stream is being request routed (i.e. should the CDNI interfaces support per-chunk request routing) ?

While having a seperate request routing operation for every chunk will probably not be very efficient, only allowing for entire Content Collections to be request routed is very limiting. For example, it might be efficient for a dCDN targeted at mobile devices to only host (and thus be able to deliver) the lower resolution encodings of a given video item. In this case it probably wouldn't make sense to force the mobile CDN to host all resolutions (including the very high ones with multi-channel audio) that are hosted by the uCDN since there will be no clients accessing the high resolution content through the mobile CDN.

- Should it be possible for a dCDN to host (and deliver) only a subset of all the chunks, or chunk collections, of a given Content Collection?

3.2.3. Dividing content over multiple nodes

An aspect that is related to the issues discussed in sections 3.2.1 and 3.2.2, is that of dividing content over multiple delivery nodes. In non-cache-based CDNs, where the location of content on delivery nodes is managed by a centralized process and where content is often pre-positioned, different Chunk Collections belonging to the same Content Collection, or even different chunks belonging to the same Chunk Collection, may be distributed over different delivery nodes. For example, the most popular resolutions of a particular Content Collection may be hosted on more delivery nodes than the less popular resolutions. In order to allow for this kind of internal CDN optimization it is necessary that the dCDN is able adapt the manifest file.

- Should it be possible for a dCDN to distribute the chunks, or Chunk Collections, constituting a given Content Collection over its delivery nodes?

3.2.4. HAS Requirements for the CDNI Request Routing Interface

This section is a placeholder for HAS-specific requirements for the CDNI Request Routing interface.

3.3. Impact on Metadata Interface

This section will discuss the impact supporting HTTP Adaptive Streaming has on the CDNI Metadata Interface.

3.3.1. HAS-specific Metadata

In section 3.1.1, the impact of HAS on the definition of a Content Item, and what constitutes a Content Item was discussed. More specifically, it was discussed how different CDNs might see chunks or chunk collections as Content Items or as parts of Content Items. This question also has an effect on the way the CDNI Metadata Interface. If one defines a Content Item as the CDN element with which Content Metadata is associated, this raises the question how Content Metadata is associated with HAS content. For example, is there specific metadata element associated with each chunk, with each chunk collection, with each content collection, or is there a specific form of metadata for HAS content?

- Is Content Metadata associated with Content Collections, Chunk Collections or chunks?

- Is it necessary to extend the CDNI Metadata model with HAS-specific extensions?

3.3.2. HAS Requirements for the CDNI Metadata Interface

This section is a placeholder for HAS-specific requirements for the CDNI Metadata interface.

3.4. Impact on Logging Interface

This section will discuss the impact supporting HTTP Adaptive Streaming has on the CDNI Logging Interface.

3.4.1. Log processing

One aspect of using HAS in a CDN context that has been getting a lot of attention is logging. In contrast to other streaming solutions which are either session-based (e.g. RTP) or using a single file (e.g. Progressive Download), the chunked nature of HAS means that regular logging methods that simply log each content request will generate extremely large log files with a seperate entry for each chunk being accessed. Apart from the large file size of these log files, a further problem is that due to the distributed nature of these log entries it can be difficult to trace them back to a specific number of clients or users, which makes reporting difficult.

For this reason, some CDNs use dedicated log processing software which accumulates, processes and aggregates log files. This log processing software is a further element where different CDNs might have different approaches. For example, some CDNs might do the log processing at a low-level, such as real-time in the delivery nodes. Other CDNs might process the log files in batches at a centralized location, such as once every hour or every day.

For the CDNI Logging interface, it will be necessary to realise a common understanding on what type of logging information is passed between the uCDn and the dCDN in the case a HAS-like protocol is used for delivery. Ideally, this interface is agnostic to the CDN-internal method of log processing. However, this might not be possible. For example, it can be argued that for transparency reasons, it is necessary for the dCDN to communicate the raw log files back to the uCDN. On the other hand, this creates a very large overhead in the inter-CDN communication, with log files for popular content possibly exceeding the file size of the content itself. Another method would be to require the dCDN to aggregate the log

files before reporting back to the uCDN, however this would require the dCDN to have specific log processing software.

- Should the CDNI Logging Interface define a specific log-file format to be used for HAS content?

3.4.2. HAS Requirements for the CDNI Logging Interface

This section is a placeholder for HAS-specific requirements for the CDNI Logging interface.

3.5. Impact on Control Interface

This section will discuss the impact supporting HTTP Adaptive Streaming has on the CDNI Control Interface.

NOTE: At this point the impact of HAS on the CDNI Control Interface has not yet been determined.

3.5.1. HAS Requirements for the CDNI Control Interface

This section is a placeholder for HAS-specific requirements for the CDNI Control interface.

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

[Anchor] "".

Authors' Addresses

Ray van Brandenburg
TNO
Brassersplein 2
Delft 2612CT
the Netherlands

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

M. Oskar van Deventer
TNO
Brassersplein 2
Delft 2612CT
the Netherlands

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