

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 25, 2013

G. Bertrand, Ed.
E. Stephan
France Telecom - Orange
R. Peterkofsky
Skytide, Inc.
F. Le Faucheur
Cisco Systems
P. Grochocki
Orange Polska
October 22, 2012

CDNI Logging Interface
draft-bertrand-cdni-logging-02

Abstract

This memo specifies the Logging interface between a downstream CDN (dCDN) and an upstream CDN (uCDN) that are interconnected as per the CDN Interconnection (CDNI) framework. First, it describes a reference model for CDNI logging. Then, it specifies the actual protocol for CDNI logging information exchange covering the information elements as well as the transport of those.

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 April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1.	Introduction	5
1.1.	Terminology	6
1.2.	Abbreviations	8
2.	CDNI Logging Reference Model	8
2.1.	CDNI Logging interactions	8
2.2.	Overall Logging Chain	13
2.2.1.	Logging Generation and During-Generation Aggregation	15
2.2.2.	Logging Collection	15
2.2.3.	Logging Filtering	16
2.2.4.	Logging Rectification and Post-Generation Aggregation	16
2.2.5.	Log-Consuming Applications	17
2.2.5.1.	Maintenance/Debugging	17
2.2.5.2.	Accounting	17
2.2.5.3.	Analytics and Reporting	18
2.2.5.4.	Security	18
2.2.5.5.	Legal Logging Duties	18
2.2.5.6.	Notions common to multiple Log Consuming Applications	18
3.	CDNI Logging Information Structure and Transport	20
4.	CDNI Logging Fields	22
4.1.	Generic Fields	22
4.1.1.	Semantics of Generic CDNI Logging Fields	22
4.1.2.	Syntax of Generic CDNI Logging Fields	24
4.2.	Logging Fields for Content Delivery	25
4.2.1.	Semantics for Delivery CDNI Logging Fields	25
4.2.2.	Syntax for Delivery CDNI Logging Fields	26
4.3.	Logging Fields for Content Acquisition	26
4.3.1.	Semantics for Acquisition CDNI Logging Fields	27
4.3.2.	Syntax for Acquisition CDNI Logging Fields	27
4.4.	Logging Fields for Control	27
4.5.	Logging Fields for Other Operations	27
5.	CDNI Logging Records	28
5.1.	Content Delivery	28
5.2.	Content Acquisition	29
5.2.1.	Logging Records Provided by dCDN to uCDN	29
5.2.2.	Logging Records Provided by uCDN to dCDN	29
5.3.	Content Invalidation and Purging	30
5.4.	Logging Extensibility	30
6.	CDNI Logging File Format	30
6.1.	Logging Files	31
6.2.	File Format	31
6.2.1.	Headers	31
6.2.2.	Body (Logging Records) Format	32
6.2.3.	Footer Format	33

7. CDNI Logging File Transport Protocol	33
8. Logging Control	33
9. Open Issues	34
10. IANA Considerations	35
11. Security Considerations	35
11.1. Privacy	35
11.2. Non Repudiation	35
12. Acknowledgments	35
13. References	35
13.1. Normative References	35
13.2. Informative References	36
Appendix A. Examples Log Format	37
A.1. W3C Common Log File (CLF) Format	37
A.2. W3C Extended Log File (ELF) Format	38
A.3. National Center for Supercomputing Applications (NCSA) Common Log Format	39
A.4. NCSA Combined Log Format	39
A.5. NCSA Separate Log Format	39
A.6. Squid 2.0 Native Log Format for Access Logs	40
Appendix B. Requirements	41
B.1. Additional Requirements	41
B.2. Compliancy with Requirements draft	42
Appendix C. CDNI WG's position on candidate protocols for Logging Transport	42
C.1. CDNI WG's position on Syslog	42
C.2. CDNI WG's position on SNMP	42
Authors' Addresses	42

1. Introduction

This memo specifies the Logging interface between a downstream CDN (dCDN) and an upstream CDN (uCDN). First, it describes a reference model for CDNI logging. Then, it specifies the actual protocol for CDNI logging information exchange covering the information elements as well as the transport of those.

The reader should be familiar with the work of the CDNI WG:

- o CDNI problem statement [RFC6707] and framework [I-D.ietf-cdni-framework] identify a Logging interface,
- o Section 7 of [I-D.ietf-cdni-requirements] specifies a set of requirements for Logging,
- o [I-D.ietf-cdni-use-cases] outlines real world use-cases for interconnecting CDNs. These use cases require the exchange of Logging information between the dCDN and the uCDN.

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

The present document describes:

- o The CDNI Logging reference model (Section 2),
- o The CDNI Logging information structure and Transport (Section 3),
- o The CDNI Logging Fields (Section 4),
- o The CDNI Logging Records (Section 5),
- o The CDNI Logging File format (Section 6),
- o The CDNI Logging File Transport Protocol (Section 7),
- o and, finally, the description of the CDNI Logging Control that is to be supported by the CDNI Control Interface Section 8.

In the Appendices, the document provides:

- o A list of identified requirements (Appendix B.1), which should be considered for inclusion in [I-D.ietf-cdni-requirements],

1.1. Terminology

In this document, the first letter of each CDNI-specific term is capitalized. We adopt the terminology described in [RFC6707] and [I-D.ietf-cdni-framework], and extend it with the additional terms defined below.

For clarity, we use the word "Log" only for referring to internal CDN logs and we use the word "Logging" for any inter-CDN information exchange and processing operations related to the CDNI Logging interface. Log and Logging formats may be different.

Log: CDN internal information collection and processing operations.

Logging: Inter-CDN information exchange and processing operations.

CDNI Logging Field: an atomic element of information that can be included in a CDNI Logging Record. The time an event/task started, the IP address of an End user to whom content was delivered, and the URI of the content delivered are examples of CDNI logging fields.

CDNI Logging Record: an information record providing information about a specific event. This comprises a collection of CDNI Logging Fields.

Separator Character: a specific character used to enable the parsing of Logging Records. This character separates the Logging Fields that compose a Logging Record.

Logging File: a file containing Logging Records and additional information for easing the processing of the Logging Records.

CDN Reporting: the process of providing the relevant information that will be used to create a formatted content delivery report provided to the CSP in deferred time. Such information typically includes aggregated data that can cover a large period of time (e.g., from hours to several months). Uses of Reporting include the collection of charging data related to CDN services and the computation of Key Performance Indicators (KPIs).

CDN Monitoring: the process of providing content delivery information in real-time. Monitoring typically includes data in real time to provide visibility of the deliveries in progress, for service operation purposes. It presents a view of the global health of the services as well as information on usage and performance, for network services supervision and operation management. In particular, monitoring data can be used to generate alarms.

End-User experience management: study of Logging data using statistical analysis to discover, understand, and predict user behavior patterns.

Class-of-requests: A Class-of-requests identifies a set of content Requests, related to a specific CSP, received from clients in a given footprint and sharing common properties. These properties include:

- o Any header, URL parameter, query parameter of an HTTP (or RTMP) content request
- o Any header, or sub-domain of the FQDN of a DNS lookup request

Examples:

- o Class-of-Requests = all the requests that include the HTTP header "User-Agent: Mozilla/5.0" related to CSP "http://*.cdn.example.com" from AS3215
- o Class-of-Requests = all the DNS requests from anywhere and related to CSP "cdn*.example.com"

Delivery Service: A Delivery Service is defined by a set of Class-of-Requests and a list of parameters that apply to all these Class-of-Requests (logging format, delivery quality/capabilities requirements...)

Service Agreement: A service agreement is defined by a uCDN identifier, a dCDN identifier, a set of Delivery Services and a list of parameters that apply to the Service Agreement.

Once a Service Agreement is agreed between the administrative entities managing the CDNs to be interconnected, the upstream CDN and the downstream CDN of the CDNI interconnection must be configured according to this agreed Service Agreement. For instance, a given uCDN (uCDN1) may request a given dCDN (dCDN1) to configure one Delivery Service for handling requests for HTTP Adaptive streaming videos delegated by uCDN1 and related to a specific CSP (CSP1) and another one for handling requests for static pictures delegated by uCDN1 and related to CSP1. These Delivery services would belong to the Service Agreement between uCDN1 and dCDN1 for CSP1. In this simple example, uCDN1 may request dCDN1 to include Delivery Service information in its CDNI Logging, to help uCDN1 to provide relevant reports to CSP1.

1.2. Abbreviations

- o API: Application Programming Interface
- o CCID: Content Collection Identifier
- o CDN: Content Delivery Network
- o CDNP: Content Delivery Network Provider
- o CoDR: Content Delivery Record
- o CSP: Content Service Provider
- o DASH: Dynamic Adaptive Streaming over HTTP
- o dCDN: downstream CDN
- o FTP: File Transfer Protocol
- o HAS: HTTP Adaptive Streaming
- o KPI: Key Performance Indicator
- o PVR: Personal Video Recorder
- o SID: Session Identifier
- o SFTP: SSH File Transfer Protocol
- o SNMP: Simple Network Management Protocol
- o uCDN: upstream CDN

2. CDNI Logging Reference Model

2.1. CDNI Logging interactions

The CDNI logging reference model between a given uCDN and a given dCDN involves the following interactions:

- o control by the uCDN of the logging to be performed by the dCDN (e.g. control of which logging fields are to be communicated to the uCDN for a given task performed by the dCDN, control of which types of events are to be logged). This is supported by the CDNI Control interface.

- o generation and collection by the dCDN of logging information related to the completion of any task performed by the dCDN on behalf of the uCDN (e.g. delivery of the content to an end user) or related to events happening in the dCDN that are relevant to the uCDN (e.g. failures or unavailability in dCDN). This takes place within the dCDN and does not directly involve CDNI interfaces.
- o communication by the dCDN to the uCDN of the logging information collected by the dCDN relevant to the uCDN. This is supported by the CDNI Logging interface. For example, the uCDN may use this logging information to charge the CSP, to perform analytics and monitoring for operational reasons, to provide analytics and monitoring views on its content delivery to the CSP, or to perform troubleshooting.
- o control by the dCDN of the logging to be performed by the uCDN on behalf of the dCDN. This is supported by the CDNI Control interface.
- o generation and collection by the uCDN of logging information related to the completion of any task performed by the uCDN on behalf of the dCDN (e.g. serving of content by uCDN to dCDN for acquisition purposes by dCDN) or related to events happening in the uCDN that are relevant to the dCDN. This takes place within the uCDN and does not directly involve CDNI interfaces.
- o communication by the uCDN to the dCDN of the logging information collected by the uCDN relevant to the dCDN. This is supported by the CDNI Logging interface. For example, the dCDN may use this logging information for security auditing or content acquisition troubleshooting.

Figure 1 provides an example of CDNI Logging interactions in a particular scenario where 4 CDNs are involved in the delivery of content from a given CSP: the uCDN has a CDNI interconnection with dCDN1 and dCDN2. In turn, dCDN2 has a CDNI interconnection with dCDN3. uCDN, dCDN1, dCDN2 and dCDN3 deliver content for the CSP. In this example, the CDNI Logging interface enables the uCDN to obtain logging information from all the dCDNs involved in the delivery. In the example, uCDN uses the Logging data:

- o to analyze the performance of the delivery operated by the dCDNs and to adjust its operations (e.g., request routing) as appropriate
- o to provide reporting (non-real time) and monitoring (real time) information to CSP.

For instance, uCDN merges Logging data, extracts relevant KPIs, and presents a formatted report to CSP, in addition to a bill for the content delivered by uCDN itself or its dCDNs on his behalf. uCDN may also provide Logging data as raw log files to CSP, so that CSP can use its own Logging analysis tools.

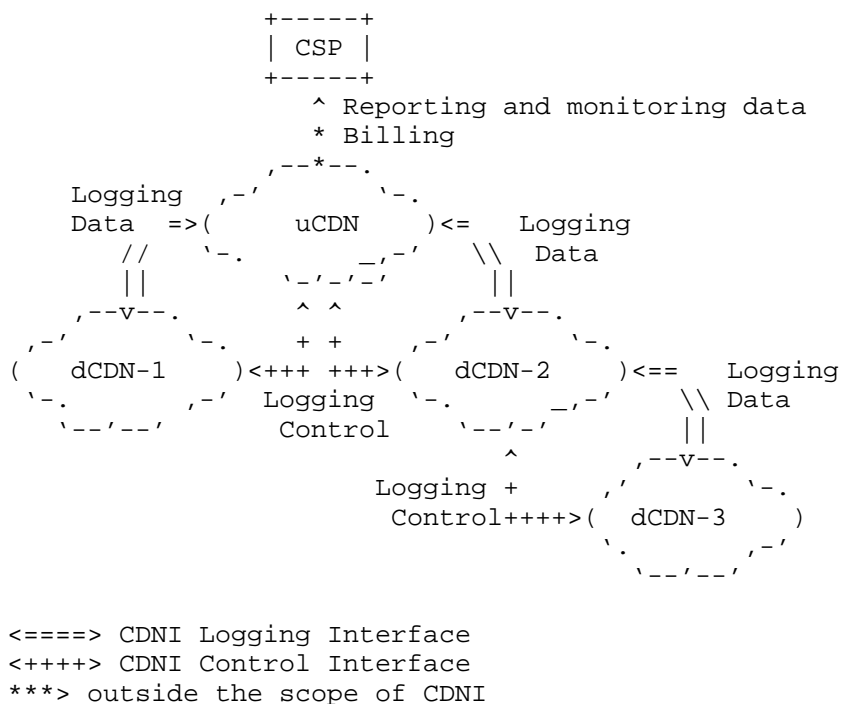


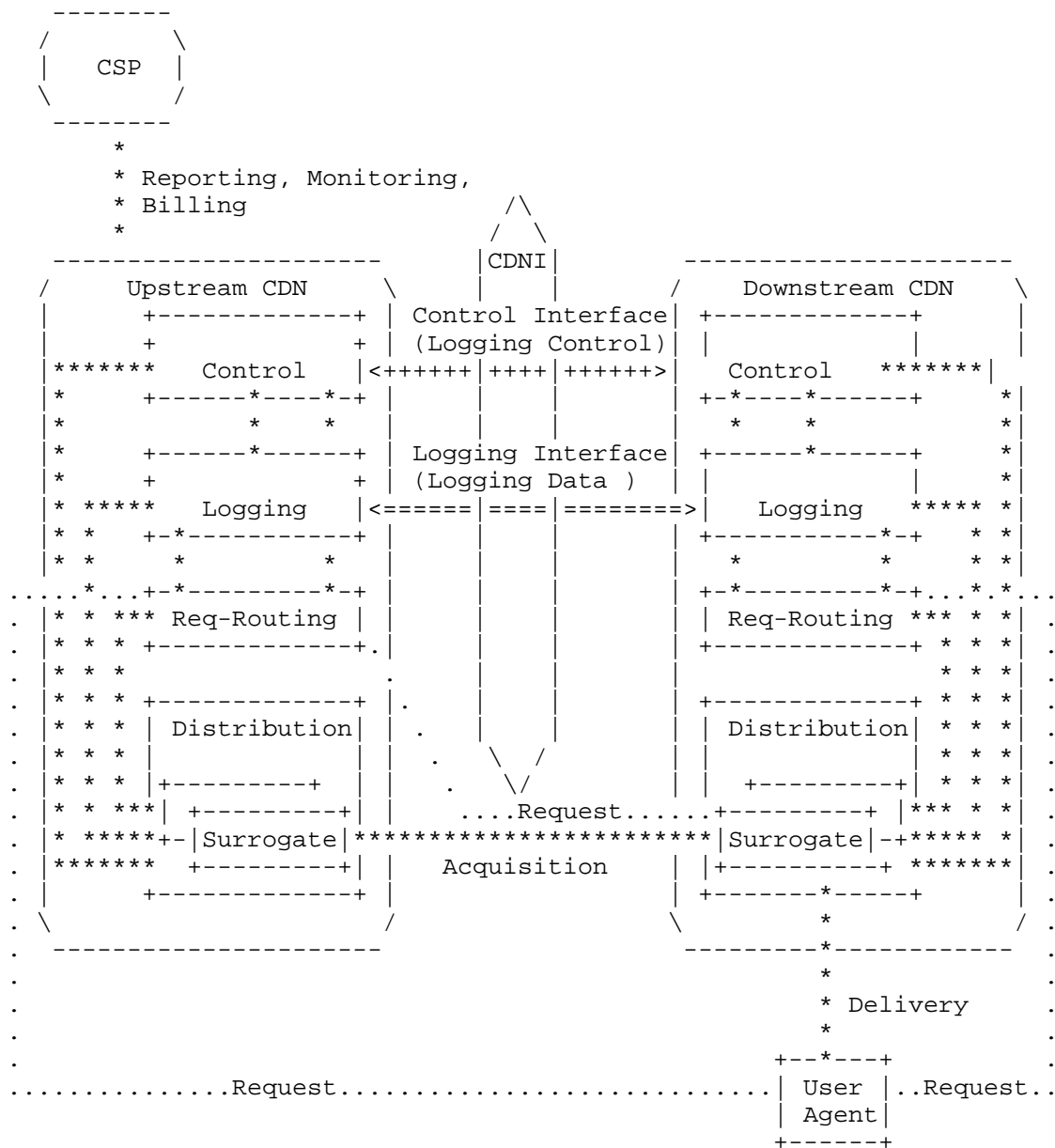
Figure 1: Interactions in CDNI Logging Reference Model

A dCDN (e.g. dCDN-2) integrates the relevant logging data obtained from its dCDNs (e.g. dCDN-3) in the logging data that it provides to the uCDN, so that the uCDN ultimately obtains all logging information relevant to a CSP for which it acts as the authoritative CDN.

Note that the format of Logging data that a CDN provides over the CDNI interface might be different from the one that the CDN uses internally. In this case, the CDN needs to reformat the Logging data before it provides this data to the other CDN over the CDNI Logging interface. Similarly, a CDN might reformat the Logging data that it receives over the CDNI Logging interface before injecting it into its log-consuming applications or before providing some of this logging information to the CSP. Such reformatting operations introduce

latency in the logging distribution chain and introduce a processing burden. Therefore, there are benefits in specifying CDNI Logging format that are as close as possible from the CDN Log formats commonly used in CDNs today.

Figure 2 maps the CDNI Logging interactions discussed above onto the CDNI Reference Model defined in [RFC6707].



<====> CDNI Logging Interface

<++++> CDNI Control Interface

**** interfaces outside the scope of CDNI

.... interfaces outside the scope of CDNI

Figure 2: Mapping of CDNI Logging interactions on the CDNI Reference Model

As illustrated in Figure 2, the Logging Control (including signaling of which logging fields are to be communicated across CDNs for a given task) occurs over the Control Interface level. The rationale for using the Control interface for Logging Control (instead of for instance using the Metadata interface) includes:

- o the Logging Control interactions typically define fairly static information for initializing and controlling the Logging interface, which matches the role of the Control Interface as described in [I-D.ietf-cdni-framework] and [RFC6707].
- o the Logging Control information (specifying the Logging information format and scope is primarily intended to be consumed by the (typically fairly centralized) logical entity responsible for collecting intra-CDN logs, processing, filtering those and then exporting the relevant subset of logs/fields to the other CDNs.
- o the surrogates within a given CDN are typically not expected to need to be aware of the specific set of fields or set of events that have been requested by various interconnected CDNs. Rather the surrogates are likely to perform some generic logging for all services regardless of the peculiarities of every CDNI agreement. Processing (e.g. filtering, format adaptation) of the generic logging information generated by the Surrogates is expected to take place to ensure that each interconnected CDN receives the specific set of fields and logs it has requested through Logging Control. Therefore there is no need to ensure that the Logging control information be easily distributable through the CDNs right down to surrogates.
- o the Control interface is expected to support the capability to apply control at the granularity of content sets (e.g. for content Purge) which is required for Logging Control since it is expected that a CDN may require different sets of logging fields and events for different sets of content (e.g. because it only needs to perform coarse billing for a given CSP while it needs to provide detailed analytics for another CSP).

2.2. Overall Logging Chain

This section discusses the overall logging chain within and across CDNs to clarify how CDN Logging information is expected to fit in this overall chain. Figure 3 illustrates the overall logging chain within the dCDN, across CDNs using the CDNI Logging interface and

within the uCDN. For readability, the Figure only considers logging information flowing from the dCDN to the uCDN. Note that the logging chain illustrated in the Figure is obviously only indicative and varies in specific environments. For example, there may be more or less instantiations of each entity (ie there may be 4 Log consuming applications in a given CDN. As another example, there may be one instance of Rectification process per Log Consuming Application instead of a shared one.

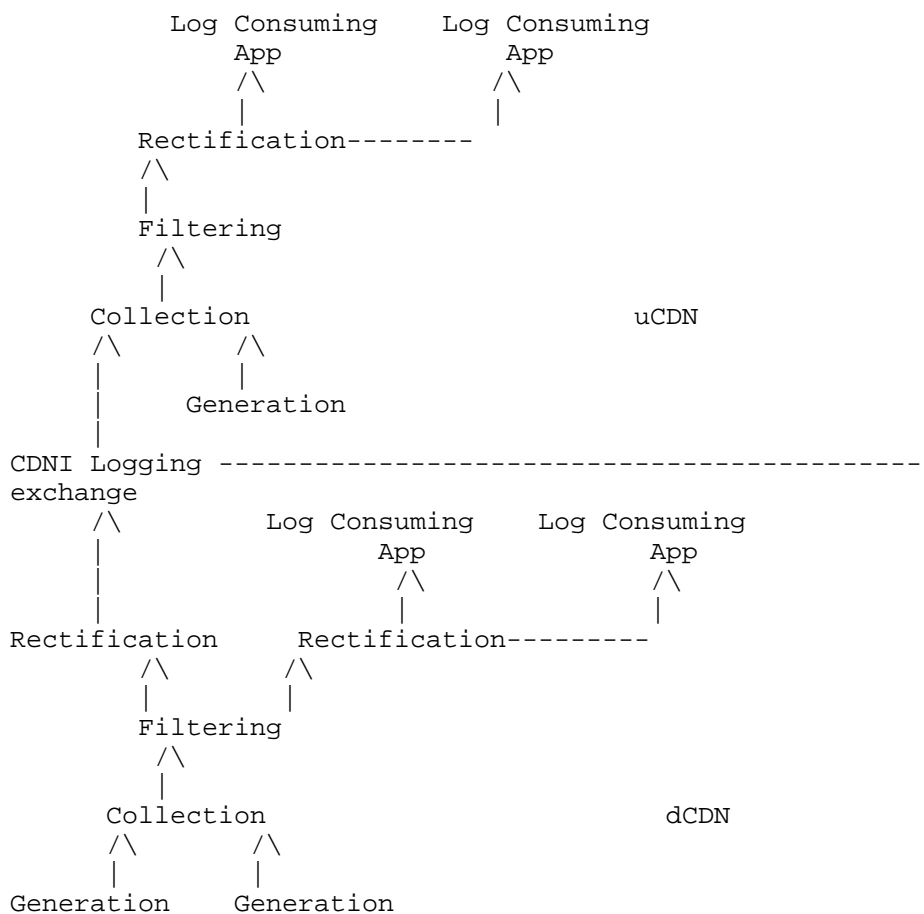


Figure 3: CDNI Logging in the overall Logging Chain

The following subsections describe each of the processes potentially involved in the logging chain of Figure 3.

2.2.1. Logging Generation and During-Generation Aggregation

CDNs typically generate logging information for all significant task completions, events, and failures. Logs are typically generated by many devices in the CDN including the surrogates, the request routing system, and the control system.

The amount of Logging information generated can be huge. Therefore, during contract negotiations, interconnected CDNs often agree on a Logging retention duration, and optionally, on a maximum size of the Logging data that the dCDN must keep. If this size is exceeded, the dCDN must alert the uCDN but may not keep more Logs for the considered time period. In addition, CDNs may aggregate logs and transmit only summaries for some categories of operations instead of the full Logging data. Note that such aggregation leads to an information loss, which may be problematic for some usages of Logging (e.g., debugging).

[I-D.brandenburg-cdni-has] discusses logging for HTTP Adaptive Streaming (HAS). In accordance with the recommendations articulated there, it is expected that a surrogate will generate separate logging information for delivery of each chunk of HAS content. This ensures that separate logging information can then be provided to interconnected CDNs over the CDNI Logging interface. Still in line with the recommendations of [I-D.brandenburg-cdni-has], the logging information for per-chunk delivery may include some information (a Content Collection Identifier and a Session Identifier as discussed in Section 4.1.1) intended to facilitate subsequent post-generation aggregation of per-chunk logs into per-session logs. Note that a CDN may also elect to generate aggregate per-session logs when performing HAS delivery, but this needs to be in addition to, and not instead of, the per-chunk delivery logs. We note that this may be revisited in future versions of this document.

2.2.2. Logging Collection

This is the process that continuously collects logs generated by the log-generating entities within a CDN.

In a CDNI environment, in addition to collecting logging information from log-generating entities within the local CDN, the Collection process also collects logging information provided by another CDN, or other CDNs, through the CDNI Logging interface. This is illustrated in Figure 3 where we see that the Collection process of the uCDN collects logging information from log-generating entities within the uCDN as well as logging information coming through CDNI Logging exchange with the dCDN through the CDNI Logging interface.

2.2.3. Logging Filtering

A CDN may require to only present different subset of the whole logging information collected to various log-consuming applications. This is achieved by the Filtering process.

In particular, the Filtering process can also filter the right subset of information that needs to be provided to a given interconnected CDN. For example, the filtering process in the dCDN can be used to ensure that only the logging information related to tasks performed on behalf of a given uCDN are made available to that uCDN (thereby filtering all the logging information related to deliveries by the dCDN of content for its own CSPs). Similarly, the Filtering process may filter or partially mask some fields, for example, to protect End Users' privacy when communicating CDNI Logging information to another CDN. Filtering of logging information prior to communication of this information to other CDNs via the CDNI Logging interface requires that the downstream CDN can recognize the set of log records that relate to each interconnected CDN.

The CDN will also filter some internal scope information such as information related to its internal alarms (security, failures, load, etc).

In some use cases described in [I-D.ietf-cdni-use-cases], the interconnected CDNs do not want to disclose details on their internal topology. The filtering process can then also filter confidential data on the dCDNs' topology (number of servers, location, etc.). In particular, information about the requests served by every Surrogate may be confidential. Therefore, the Logging information must be protected so that data such as Surrogates' hostnames is not disclosed to the uCDN. In the "Inter-Affiliates Interconnection" use case, this information may be disclosed to the uCDN because both the dCDN and the uCDN are operated by entities of the same group.

2.2.4. Logging Rectification and Post-Generation Aggregation

If Logging is generated periodically, it is important that the sessions that start in one Logging period and end in another are correctly reported. If they are reported in the starting period, then the Logging of this period will be available only after the end of the session, which delays the Logging generation.

A Logging rectification/update mechanism could be useful to reach a good trade-off between the Logging generation delay and the Logging accuracy. Depending on the selected Logging protocol(s), such mechanism may be invaluable for real time Logging, which must be provided rapidly and cannot wait for the end of operations in

progress.

In the presence of HAS, some log-consuming applications can benefit from aggregate per-session logs. For example, for analytics, per-session logs allow display of session-related trends which are much more meaningful for some types of analysis than chunk-related trends. In the case where the log-generating entities have generated during-generation aggregate logs, those can be used by the applications. In the case where aggregate logs have not been generated, the Rectification process can be extended with a Post-Generation Aggregation process that generates per-session logs from the per-chunk logs, possibly leveraging the information included in the per-chunk logs for that purpose (Content Collection Identifier and a Session Identifier). However, in accordance with [I-D.brandenburg-cdni-has], this document does not define exchange of such aggregate logs on the CDNI Logging interface. We note that this may be revisited in future versions of this document.

2.2.5. Log-Consuming Applications

2.2.5.1. Maintenance/Debugging

Logging is useful to permit the detection (and limit the risk) of content delivery failures. In particular, Logging facilitates the resolution of configuration issues.

To detect faults, Logging must enable the reporting of any CDN operation success and failure, such as request redirection, content acquisition, etc. The uCDN can summarize such information into KPIs. For instance, Logging format should allow the computation of the number of times during a given epoch that content delivery related to a specific service succeeds/fails.

Logging enables the CDN providers to identify and troubleshoot performance degradations. In particular, Logging enables the communication of traffic data (e.g., the amount of traffic that has been forwarded by a dCDN on behalf of an uCDN over a given period of time), which is particularly useful for CDN and network planning operations.

2.2.5.2. Accounting

Logging is essential for accounting, to permit inter-CDN billing and CSP billing by uCDNs. For instance, Logging enables the uCDN to check the total amount of traffic delivered by every dCDN and for every Delivery Service, as well as, the associated bandwidth usage (e.g., peak, 95th percentile), and the maximum number of simultaneous sessions over a given period of time.

2.2.5.3. Analytics and Reporting

The goal of analytics is to gather any relevant information to track audience, analyze user behavior, and monitor the performance and quality of content delivery. For instance, Logging enables the CDN providers to report on content consumption (e.g., delivered sessions per content) in a specific geographic area.

The goal of reporting is to gather any relevant information to monitor the performance and quality of content delivery and allow detection of delivery issues. For instance, reporting could track the average delivery throughput experienced by End Users in a given region for a specific CSP or content set over a period of time.

2.2.5.4. Security

The goal of security is to prevent and monitor unauthorized access, misuse, modification, and denial of access of a service. A set of information is logged for security purposes. In particular, a record of access to content is usually collected to permit the CSP to detect infringements of content delivery policies and other abnormal End User behaviors.

2.2.5.5. Legal Logging Duties

Depending on the country considered, the CDNs may have to retain specific Logging information during a legal retention period, to comply with judicial requirements.

2.2.5.6. Notions common to multiple Log Consuming Applications

2.2.5.6.1. Logging Information Views

Within a given log-consuming application, different views may be provided to different users depending on privacy, business, and scalability constraints.

For example, an analytics tool run by the uCDN can provide one view to an uCDN operator that exploits all the logging information available to the uCDN, while the tool may provide a different view to each CSP exploiting only the logging information related to the content of the given CSP.

As another example, maintenance and debugging tools may provide different views to different CDN operators, based on their operational role.

2.2.5.6.2. Key Performance Indicators (KPIs)

This section presents, for explanatory purposes, a non-exhaustive list of Key Performance Indicators (KPIs) that can be extracted/produced from logs.

Multiple log-consuming applications, such as analytics, monitoring, and maintenance applications, often compute and track such KPIs.

In a CDNI environment, depending on the situation, these KPIs may be computed by the uCDN or by the dCDN. But it is usually the uCDN that computes KPIs, because uCDN and dCDN may have different definitions of the KPIs and the computation of some KPIs requires a vision of all the deliveries performed by the uCDN and all its dCDNs.

Here is a list of important examples of KPIs:

- o Number of delivery requests received from End Users in a given region for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Percentage of delivery successes/failures among the aforementioned requests
- o Number of failures listed by failure type (e.g., HTTP error code) for requests received from End Users in a given region and for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Number and cause of premature delivery termination for End Users in a given region and for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Maximum and mean number of simultaneous sessions established by End Users in a given region, for a given Delivery Service, and during a given period of time (e.g., hour/day/week/month)
- o Volume of traffic delivered for sessions established by End Users in a given region, for a given Delivery Service, and during a given period of time (e.g., hour/day/week/month)
- o Maximum, mean, and minimum delivery throughput for sessions established by End Users in a given region, for a given Delivery Service, and during a given period of time (e.g., hour/day/week/month)
- o Cache-hit and byte-hit ratios for requests received from End Users in a given region for each piece of content, during a given period

of time (e.g., hour/day/week/month)

- o Top 10 of the most popularly requested content (during a given day/week/month),
- o Terminal type (mobile, PC, STB, if this information can be acquired from the browser type header, for example).

Additional KPIs can be computed from other sources of information than the Logging -- for instance, data collected by a content portal or by specific client-side APIs. Such KPIs are out of scope for the present memo.

The KPIs used depend strongly on the considered log-consuming application -- the CDN operator may be interested in different metrics than the CSP is. In particular, CDN operators are often interested in delivery and acquisition performance KPIs, information related to Surrogates' performance, caching information to evaluate the cache-hit ratio, information about the delivered file size to compute the volume of content delivered during peak hour, etc.

Some of the KPIs, for instance those providing an instantaneous vision of the active sessions for a given CSP's content, are useful especially if they are provided in real time. By contrast, some other KPIs, such as those averaged over a long period of time, can be provided in non-real time.

3. CDNI Logging Information Structure and Transport

As defined in Section 1.1 a CDNI logging field is as an atomic logging information element and a CDNI Logging Record is a collection of CDNI Logging Fields containing all logging information corresponding to a single logging event.

This document defines non-real time transport of CDNI Logging information over the CDNI interface. For such non-real time transport, this document defines a third level of structure, the CDNI Logging File, that is a collection of CDNI Logging Records. This structure is described in Figure 4. This document then specifies how to transport such CDNI Files across interconnected CDNs. We observe that this approach can be tuned in a real deployment to achieve near-real time exchange of CDNI Logging information, e.g. by increasing the frequency of logging file creation and distribution throughout the Logging chain, but it is not expected that this approach can support real time transport (e.g. sub-second) of CDNI logging information.

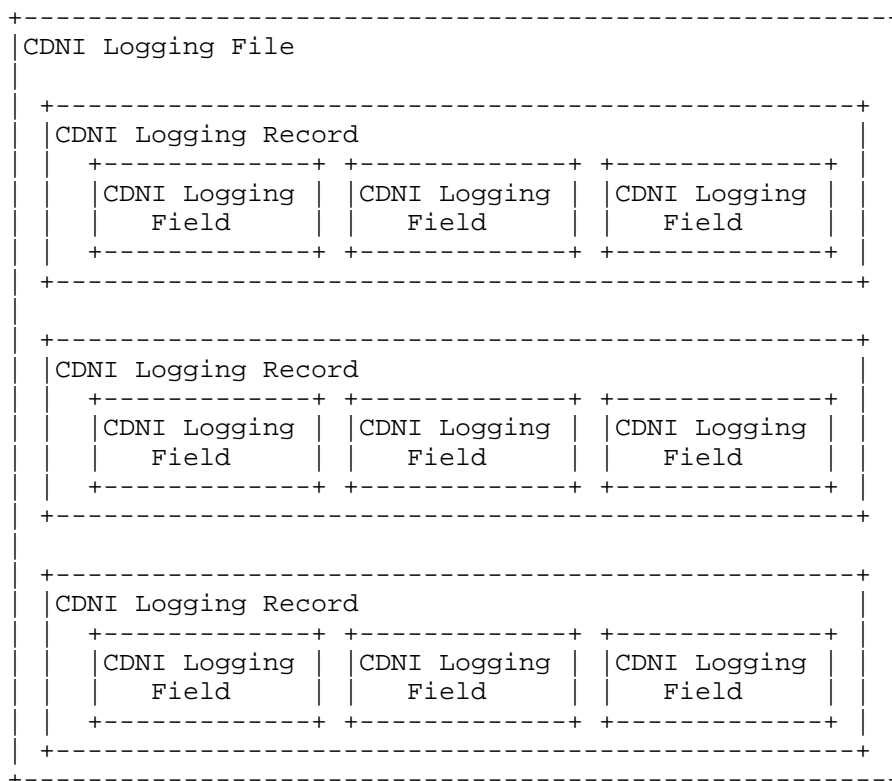


Figure 4: Structure of Logging Files

It is expected that future version of this document will also specify real time transport of CDNI Logging information over the CDNI interface. We note that this might involve direct transport of CDNI Logging Records without prior grouping into a file structure to avoid the latency associated with creating and transporting such a file structure throughout the logging chain.

The semantics and encoding of the CDNI Logging fields are specified in Section 4. The semantics and encoding of CDNI Records are specified in Section 5. The CDNI Logging File format is specified in Section 6. The protocol for transport of CDNI Logging File is specified in Section 7.

4. CDNI Logging Fields

Existing CDNs Logging functions collect and consolidate logs performed by their Surrogates. Surrogates usually store the logs using a format derived from Web servers' and caching proxies' log standards such as W3C, NCSA [ELF] [CLF], or Squid format [squid]. In practice, these formats are adapted to cope with CDN specifics. Appendix A presents examples of commonly used log formats.

4.1. Generic Fields

This section specifies a set of generic CDNI Logging Fields that are expected to be found in multiple types of CDNI Logging records.

4.1.1. Semantics of Generic CDNI Logging Fields

The semantics of the generic CDNI Logging Fields are specified in Table 1.

Name	Description
Start-time	A start date and time associated with a logged event; for instance, the time at which a Surrogate received a content delivery request or the time at which an origin server received a content acquisition request.
End-time	An end date and time associated with a logged event. For instance, the time at which a Surrogate completed the handling of a content delivery request (e.g., end of delivery or error).
Duration	The duration of an operation in milliseconds. For instance, this field could be used to provide the time it took the Surrogate to send the requested file to the End-User or the time it took the Surrogate to acquire the file on a cache-miss event. In the case where Start-time, End-time, and Duration appear in a Logging Record, the Duration is to be interpreted as a total activity time related to the logged operation.
Client-IP	The IP address of the User Agent that issued the logged request or of a proxy, for instance "203.0.113.1".
Client-port	The source port of the logged request (e.g., 9542)
Destination-IP	The IP address of the host that received the logged request (e.g., 192.0.2.2).

Destination-port	The destination port of the logged request (e.g., 80).
Operation	The kind of operation that is logged; for instance, Acquisition, Delivery, or Purging.
URI_full	The full requested URL (e.g., "http://nodel.peer-a.op-b.net/cdn.csp.com/movies/potter.avi?param=11&user=toto"). When HTTP request redirection is used, this URI includes the SurrogateFQDN. If the association of requests to Surrogates is confidential, the dCDN can present only URI_part to uCDN.
URI_part	The requested URL path (e.g., /cdn.csp.com/movies/potter.avi?param=11&user=toto if the full request URL was "http://nodel.peer-a.op-b.net/cdn.csp.com/movies/potter.avi?param=11&user=toto"). The URI without host-name typically includes the "CDN domain" (ex.cdn.csp.com) - cf. [I-D.ietf-cdni-framework]: it enables the identification of the CSP service agreed between the CSP and the CDNP operating the uCDN.
Protocol	The protocol and protocol version of the message that triggered the Logging entry (e.g., HTTP/1.1).
Request-method	The protocol method of the request message that triggered the Logging entry.
Status	The protocol method of the reply message related to the Logging entry
Bytes-Sent	The number of bytes at application-layer protocol-level (e.g., HTTP) of the reply message related to the Logging entry. It includes the size of the response headers.
Headers-Sent	The number of bytes corresponding to response headers at application-layer protocol-level (e.g., HTTP) of the reply message related to the Logging entry.
Bytes-received	The number of bytes (headers + body) of the message that triggered the Logging entry.
Referrer	The value of the Referrer header in an HTTP request.
User-Agent	The value of the User Agent header in an HTTP request.
Cookie	The value of the Cookie header in an HTTP request.
Byte-Range	[Ed. note: to be defined]
Cache-control	The value of the cache-control header in an HTTP answer. This header is particularly important for content acquisition logs.
Record-digest	A digest of the Logging Record; it enables detecting corrupted Logging Records.

CCID	A Content Collection Identifier (CCID) eases the correlation of several Logging Records related to a Content Collection (e.g., a movie split in chunks).
SID	A Session Identifier (SID) eases the correlation (and aggregation) of several Logging Records related to a session. The SID is especially relevant for summarizing HAS Logging information [I-D.brandenburg-cdni-has].

Table 1: Semantics of Generic CDNI Logging Fields

NB: we define three fields related to the timing of logged operations: Start-time, End-time, and Duration. Start-time is typically useful for human readers (e.g., while debugging), however, some servers log the operation's End-time which corresponds to the time of log record generation. In absence of Logging summarization, only two of these three fields are required to obtain relevant timing information on the operation. However, when some kind of Logging aggregation/summarization is used, it can be advantageous to keep the three fields: for instance, in the case of HAS, keeping the three fields permits computing an average delivery bitrate from a single Logging Record aggregating information on the delivery of multiple consecutive video chunks.

Multiple header fields, in addition to the ones explicitly listed in the table could be reproduced in the Logging records.

Note that uCDN may want to filter Logging data by user (and not by IP address) to provide more relevant information to the CSP. In such case, a user may be identified as a combination of several pieces of information such as the client IP and User Agent or through the SID.

The URI_full provides information on the Surrogate that provided the content. This information can be relevant, for instance, for the Inter-Affiliates use case described in [I-D.ietf-cdni-use-cases]. However, in some cases it may be considered as confidential and the dCDN may provide URI_part instead.

4.1.2. Syntax of Generic CDNI Logging Fields

Table 2 illustrates the definition of the information elements. It provides examples using Apache log format strings [apache] when they exist.

[Ed Note, this should be replaced with actual selected format for CDNI]

[Ed. note: specify for all Logging Fields the type (e.g., varchar, int, float, ...) and the maximum size (e.g., varchar(200))]

Name	String	Example
Time	%t	[10/Oct/2000:13:55:36-0700]
Duration	%D	-
Client-IP	%a	203.0.113.45
Operation	-	-
URI_full	%U	-
Protocol	%H	HTTP/1.0
Request method	%m	GET
Status	%>s	200
Bytes Sent	%O	2326
Bytes received	%I	432
Header	\ "%{Referrer}i\" \ "%{User-agent}i\" "	"http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"

Table 2: Examples using Apache format

4.2. Logging Fields for Content Delivery

Beyond the Logging Fields described in previous section, this section defines additional Logging Fields that are specifically related to Content Delivery operations. Note that the uCDN may not transfer the information provided in some of these fields to the CSP, depending on the CSP's interest in the information and on the information's confidentiality level.

4.2.1. Semantics for Delivery CDNI Logging Fields

The semantics of the generic CDNI Logging Fields are specified in Table 3.

Name	Definition
uCDN-ID	An element authenticating the operator of the uCDN as the authority having delegated the request to the dCDN.
Delivering-CDN-ID	An identifier (e.g., an aggregation of an IP address and a FQDN) of the Delivering CDN. The Delivering-CDN-ID might be considered as confidential by the dCDN. In such case, the dCDN could either not provide this field to the uCDN or overwrite the Delivering-CDN-ID with its own identifier.
Cache-bytes	The number of body bytes served from caches. This quantity permits the computation of the byte hit ratio.
Action	The Action describes how a given request was treated locally: through which transport protocol, with or without content revalidation, with a cache hit or cache miss, with fresh or stale content, and (if relevant) with which error. Example with Squid format [squid]: "TCP_REFRESH_FAIL_HIT" means that an expired copy of an object requested through TCP was in the cache. Squid attempted to make an If-Modified-Since request, but it failed. The old (stale) object was delivered to the client.

Table 3: Semantics of the Delivery CDNI Logging Fields

[Ed. note: Other information that could be logged include operations related to the authorization of the requests, URL rewriting rules enforced, the X-FORWARDED-FOR non standard HTTP header...]

4.2.2. Syntax for Delivery CDNI Logging Fields

[Ed Note: To be added]

4.3. Logging Fields for Content Acquisition

This section specifies Logging fields that are specific to Content Acquisition operations.

4.3.1. Semantics for Acquisition CDNI Logging Fields

Table 4 specifies the semantics of the Acquisition specific CDNI Logging Fields.

Name	Definition
dCDN identifier	An element authenticating the operator of the dCDN as the authority requesting the content to the uCDN
Caching_date	Date at which the delivered content was stored in cache
Validity_headers	A copy of all headers related to content validity: no-cache, ETag, Vary, last-modified...
Lookup_duration	Duration of the DNS resolution for resolving the FQDN of (uCDN's or CSP's) origin server.
Delay_to_first_bit	Duration of the operations from the sending of the content acquisition request to the reception of the first bit of the requested content.
Delay_to_last_bit	Duration of the operations from the sending of the content acquisition request to the reception of the last bit of the requested content.

Table 4: Semantics of the Acquisition CDNI Logging Fields

These information elements may be used in Content Acquisition Logging provided by dCDN to uCDN and, potentially, in Content Acquisition Logging provided by uCDN to dCDN.

4.3.2. Syntax for Acquisition CDNI Logging Fields

[Ed Note: To be added]

4.4. Logging Fields for Control

[Ed. note: LOGS RELATED TO KEY EXCHANGES FOR INSTANCE, SECTION TO BE WRITTEN AFTER THE CONTROL INTERFACE IS MORE CLEARLY DEFINED]

4.5. Logging Fields for Other Operations

Logging can be used for debugging. Therefore, all kinds of CDN operations might be logged, depending on the agreement between the dCDN and the uCDN. In particular, operations related to Request

Routing and Metadata can be logged.

5. CDNI Logging Records

[Ed. note: we need to specify the encoding of the file, the separation character, etc...]

This section defines a set of central events that a dCDN should register and publish through the Logging interface.

We classify the logged events depending on the CDN operation to which they relate: Content Delivery, Content Acquisition, Content Invalidation/Purging, etc.

5.1. Content Delivery

Some CSPs pay a lot of attention to the protection of their content (e.g., premium video CSPs). To fulfill the needs of these CSPs, a CDN shall log all the details of the content delivery authorizations. This means that a dCDN must be able to provide Logging detailing the content delivery/content acquisition authorizations and denials as well as information on why the request is authorized/denied.

CSPs and CDN service providers pay a lot of attention to errors related to content delivery. It is therefore of utmost importance that the dCDN provides detailed error information in the Logging data. This information should typically be available even when Logging is aggregated.

The content delivery events triggering the generation of a Logging Record include:

- o Reception of a content request,

The generated Logging Record typically embeds information about:

- o Denial of delivery (error or unauthorized request, e.g., HTTP 401) for a request,
- o Beginning of delivery (authorization) of a requested content,
- o End of an authorized delivery (success),
- o End of an authorized delivery (failure during the delivery, e.g., HTTP 403).

5.2. Content Acquisition

5.2.1. Logging Records Provided by dCDN to uCDN

When the uCDN requires the dCDN to provide Logging for acquisition related events, the events triggering the generation of a Logging Record include:

- o Emission of a content acquisition request (first try or retry) for a cache hit or a cache miss with content revalidation

The generated Logging Record typically embeds information about:

- o Reception of a reply indicating denial of delivery (error or unauthorized request) for a content acquisition request,
- o End of an authorized acquisition (success),
- o End of an authorized acquisition (failure)

Note that a dCDN may acquire content only from the uCDN. In this case, the uCDN can log the dCDN's content acquisition operations itself, and thus, the uCDN may not require the dCDN to log acquisition related events. However, comparing the dCDN and uCDN logs is often useful for debugging and for security auditing.

5.2.2. Logging Records Provided by uCDN to dCDN

When the dCDN requires the uCDN to provide Logging for acquisition related events, the events triggering the generation of a Logging Record include:

- o Reception of a content acquisition request for the considered Delivery Service for a cache hit or a cache miss with content revalidation

The generated Logging Record typically embeds information about:

- o Emission of a reply indicating denial of delivery (error or unauthorized request) for a content acquisition request,
- o End of an authorized acquisition (success),
- o End of an authorized acquisition (failure).

5.3. Content Invalidation and Purging

When the uCDN requests a dCDN to log invalidation/purging events (e.g., for security), the events triggering the generation of a Logging Record include:

- o Reception of a content invalidation/purging request

The generated Logging Record typically embeds information about:

- o Denial of the invalidation/purging request (error or unauthorized request, with details about the causes of the error),
- o Beginning of invalidation/purging (authorization) for a given content purging request,
- o End of an authorized invalidation/purging (success),
- o End of an authorized invalidation/purging (failure).

5.4. Logging Extensibility

Future usages might introduce the need for additional Logging fields. In addition, some use-cases such as an Inter-Affiliate Interconnection [I-D.ietf-cdni-use-cases], might take advantage of extended Logging exchanges. Therefore, it is important to permit CDNs to use additional Logging fields besides the standard ones, if they want. For instance, an "Account-name" identifying the contract enforced by the dCDN for a given request could be provided in extended fields.

The required Logging Records may depend on the considered services. For instance, static file delivery (e.g., pictures) typically does not include any delivery restrictions. By contrast, video delivery typically implies strong content delivery restrictions, as explained in [I-D.ietf-cdni-use-cases], and Logging could include information about the enforcement of these restrictions. Therefore, to ease the support of varied services as well as of future services, the Logging interface should support optional Logging Records.

6. CDNI Logging File Format

Interconnected CDNs may support various Logging formats. However, they must support at least the default Logging File format described here.

6.1. Logging Files

[Ed. Note: How many files (one per type of Delivery Service (e.g., HTTP, WMP) and per type of Event (e.g., Errors, Delivery, Acquisition,...?) and what would be inside... These aspects needs to be detailed...]

6.2. File Format

The Logging file format should be independent from the selected transport protocol, to guarantee a flexible choice of transport protocols. [Ed. note: for the real time Logging exchanges, this might be hard]

All Logging Records in a Logging File must share the same format (same set of Logging Fields, in the same order, with the same semantics, separated by the same Separator Character), to ease the parsing of the Logging data by the CDN that receives the Logging File. The CDN that provides the Logging data is responsible for guaranteeing the consistency of the Logging records' formats, typically via its log filtering and aggregation processes (see Section 2.2.3).

6.2.1. Headers

Logging files must include a header with the information described in Figure 5.

Field	Description	Examples
Format	Identification of CDNI Log format.	standard_cdni_errors_http_v1
Fields	A description of the record format (list of fields).	
Log-ID	Identifier for the CDNI Log file (facilitates detection of duplicate Logs and tracking in case of aggregation).	abcdef1234
Log-Timestamp	Time, in milliseconds, the CDNI Log was generated.	[20/Feb/2012:00:29.510+0200]
Log-Origin	Identifier of the authority (e.g., dCDN or uCDN) providing the Log- -ging	cdn1.cdni.example.com

Figure 5: Logging Headers

All time-related Logging Fields and data in the Logging File headers/ footers must provide a time zone and be at least at millisecond (ms) accuracy. The accuracy must be consistent to permit the computation of KPIs involving operations realized on several CDNs.

[Ed. note: would it make sense to add a kind of "example Logging Record" in the Logging file and associated semantic (e.g. in a structure data format) ?]

6.2.2. Body (Logging Records) Format

[Ed. note: the W3C extended log format is a good base candidate to look at.]

[Ed. note: Records used for real time information and non-real time information could use different formats. In this version, we do not yet tackle the problem of real time logging exchanges]

6.2.3. Footer Format

Logging files must include a footer with the information described in Figure 6.

Field	Description	Examples
Log Digest	Digest of the complete Log (facilitates detection of Log corruption)	

Figure 6: Logging footers

This digest field permits the detection of corrupted Logging files. This can be useful, for instance, if a problem occurs on the filesystem of the dCDN Logging system and leads to a truncation of a logging file. Additional mechanisms to avoid corrupted Logging files are expected to be provided by the Logging transport protocol, cf. Section 7.

7. CDNI Logging File Transport Protocol

As presented in [RFC6707], several protocols already exist that could potentially be used to exchange CDNI Logging between interconnected CDNs.

The offline exchange of non real-time Logging could rely on several protocols. In particular, the dCDN could publish the Logging on a server where the uCDN would retrieve them using a secure protocol (yet to be identified).

[Ed. note: Propose protocol, e.g. SSH File Transfer Protocol (SFTP) [I-D.ietf-secsh-filexfer]. and add call flow]

[Ed note: include options for lossless compression]

8. Logging Control

The CDNI Control interface is responsible for correctly configuring the Logging interface between interconnected CDNs, for every Delivery Service and according to the Logging configuration agreed during business negotiations.

This section will identify the parameters that the CDNI Control interface should manage on uCDN and dCDN for activating, updating, or removing a CDNI Logging configuration for a given Delivery Service.

[Ed. Note: uCDN shall be able to select the type of events that a dCDN should include in the Logging that the latter provides to the uCDN. This will be discussed during business negotiations and the Control must enforce the agreed configuration. The use of multiple levels of Logging granularity such as Syslog's "severity levels" (Emergency, Alert, Critical, ..., Debug) [RFC5424] may help in providing the most relevant amount of information depending on the intended Logging usage, as specified during the Logging format negotiation.]

[Ed. note: the specification all Logging Fields' maximum size (e.g., varchar(200)) might be constrained in some CDNs so need to exchange that information during the configuration]

9. Open Issues

The main remaining tasks on this ID are the following:

- o Detail the Logging Fields' syntax
- o Recommend a Logging File Transport Protocol and detail the call-flows
- o Detail mechanisms for Real-Time Logging

[Ed. Note: The format for Time is still to be agreed on. RFC 5322 (Section 3.3) format could be used or ISO 8601 formatted date and time in UTC (same format as proposed in [draft-caulfield-cdni-metadata-core-00]). Also see RFC5424 Section 6.2.3.]

[Ed. Note: When to log the end of a session when the End-User pauses a video display?]

[Ed. note: (comment from Kevin) how are errors handled ? If the client gets handed a bunch of 403s and 404s, but still gets the content eventually, without triggering an event, are those still logged? For Bytes-Sent, if there were aborted requests, do those get counted as well? Not all client behavior can be correlated with the simplified log]

10. IANA Considerations

This memo includes no request to IANA.

11. Security Considerations

11.1. Privacy

CDNs have the opportunity to collect detailed information about the downloads performed by End-Users. The provision of this information to another CDN introduces End-Users privacy protection concerns.

11.2. Non Repudiation

Logging provides the raw material for charging. It permits the dCDN to bill the uCDN for the content deliveries that the dCDN makes on behalf of the uCDN. It also permits the uCDN to bill the CSP for the content Delivery Service. Therefore, non-repudiation of Logging data is essential.

12. Acknowledgments

The authors would like to thank Sebastien Cubaud, Anne Marrec, Yannick Le Louedec, and Christian Jacquenet for detailed feedback on early versions of this document and for their input on existing Log formats.

The authors would like also to thank Fabio Costa, Yvan Massot, Renaud Edel, and Joel Favier for their input and comments.

Finally, they thank the contributors of the EU FP7 OCEAN project for valuable inputs.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, March 2009.

13.2. Informative References

- [CLF] A. Luotonen, "The Common Log-file Format, W3C (work in progress)", 1995, <<http://www.w3.org/pub/WWW/Daemon/User/Config/Logging.html>>.
- [ELF] Phillip M. Hallam-Baker and Brian Behlendorf, "Extended Log File Format, W3C (work in progress), WD-logfile-960323", <<http://www.w3.org/TR/WD-logfile.html>>.
- [I-D.bertrand-cdni-experiments]
Faucheur, F. and L. Peterson, "Content Distribution Network Interconnection (CDNI) Experiments", draft-bertrand-cdni-experiments-02 (work in progress), February 2012.
- [I-D.brandenburg-cdni-has]
Brandenburg, R., Deventer, O., Faucheur, F., and K. Leung, "Models for adaptive-streaming-aware CDN Interconnection", draft-brandenburg-cdni-has-03 (work in progress), July 2012.
- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-01 (work in progress), July 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.
- [I-D.ietf-cdni-use-cases]
Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-10 (work in progress), August 2012.
- [I-D.ietf-secsh-filexfer]
Galbraith, J. and O. Saarenmaa, "SSH File Transfer Protocol", draft-ietf-secsh-filexfer-13 (work in progress), July 2006.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

[apache] "Apache 2.2 log files documentation", Feb. 2012,
 <<http://httpd.apache.org/docs/current/logs.html>>.

[squid] "Squid Log-Format documentation", Feb. 2012,
 <<http://wiki.squid-cache.org/SquidFaq/SquidLogs>>.

Appendix A. Examples Log Format

This section provides example of log formats implemented in existing CDNs, web servers, and caching proxies.

Web servers (e.g., Apache) maintain at least one log file for logging accesses to content (the Access Log). They can typically be configured to log errors in a separate log file (the Error Log). The log formats can be specified in the server's configuration files. However, webmasters often use standard log formats to ease the log processing with available log analysis tools.

A.1. W3C Common Log File (CLF) Format

The Common Log File (CLF) format defined by the World Wide Web Consortium (W3C) working group is compatible with many log analysis tools and is supported by the main web servers (e.g., Apache) Access Logs.

According to [CLF], the common log-file format is as follows:
 remotehost rfc931 authuser [date] "request" status bytes.

Example (from [apache]): 127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326

The fields are defined as follows [CLF]:

Element	Definition
remotehost	Remote hostname (or IP number if DNS hostname is not available, or if DNSLookup is Off).
rfc931	The remote logname of the user.
authuser	The username that the user employed to authenticate himself.
[date]	Date and time of the request.
"request"	An exact copy of the request line that came from the client.
status	The status code of the HTTP reply returned to the client.

bytes	The content-length of the document transferred.	
+-----+		

Table 5: Information elements in CLF format

A.2. W3C Extended Log File (ELF) Format

The Extended Log File (ELF) format defined by W3C extends the CLF with new fields. This format is supported by Microsoft IIS 4.0 and 5.0.

The supported fields are listed below [ELF].

Element	Definition
date	Date at which transaction completed
time	Time at which transaction completed
time-taken	Time taken for transaction to complete in seconds
bytes	bytes transferred
cached	Records whether a cache hit occurred
ip	IP address and port
dns	DNS name
status	Status code
comment	Comment returned with status code
method	Method
uri	URI
uri-stem	Stem portion alone of URI (omitting query)
uri-query	Query portion alone of URI

Table 6: Information elements in ELF format

Some fields start with a prefix (e.g., "c-", "s-"), which explains which host (client/server/proxy) the field refers to.

- o Prefix Description
- o c- Client
- o s- Server
- o r- Remote
- o cs- Client to Server.
- o sc- Server to Client.

- o sr- Server to Remote Server (used by proxies)
- o rs- Remote Server to Server (used by proxies)

Example: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent) sc-status sc-substatus sc-win32-status time-taken

```
2011-11-23 15:22:01 x.x.x.x GET /file 80 y.y.y.y Mozilla/
5.0+(Windows;+U;+Windows+NT+6.1;+en-US;+rv:1.9.1.6)+Gecko/
20091201+Firefox/3.5.6+GTB6 200 0 0 2137
```

A.3. National Center for Supercomputing Applications (NCSA) Common Log Format

This format for Access Logs offers the following fields:

- o host rfc931 date:time "request" statuscode bytes
- o x.x.x.x userfoo [10/Jan/2010:21:15:05 +0500] "GET /index.html HTTP/1.0" 200 1043

A.4. NCSA Combined Log Format

The NCSA Combined log format is an extension of the NCSA Common log format with three (optional) additional fields: the referral field, the user_agent field, and the cookie field.

- o host rfc931 username date:time request statuscode bytes referrer user_agent cookie
- o Example: x.x.x.x - userfoo [21/Jan/2012:12:13:56 +0500] "GET /index.html HTTP/1.0" 200 1043 "http://www.example.com/" "Mozilla/4.05 [en] (WinNT; I)" "USERID=CustomerA;IMPID=01234"

A.5. NCSA Separate Log Format

The NCSA Separate log format refers to a log format in which the information gathered is separated into three separate files. This way, every entry in the Access Log (in the NCSA Common log format) is complemented with an entry in a Referral log and another one in an Agent log. These three records can be correlated easily thanks to the date:time value. The format of the Referral log is as follows:

- o date:time referrer
- o Example: [21/Jan/2012:12:13:56 +0500]
"http://www.example.com/index.html"

The format of the Agent log is as follows:

- o date:time agent
- o [21/Jan/2012:12:13:56 +0500] "Microsoft Internet Explorer - 5.0"

A.6. Squid 2.0 Native Log Format for Access Logs

Squid [squid] is a popular piece of open-source software for transforming a Linux host into a caching proxy. Variations of Squid log format are supported by some CDNs.

Squid common access log format is as follow: time elapsed remotehost code/status bytes method URL rfc931 peerstatus/peerhost type.

Squid also supports a more detailed native access log format: Timestamp Elapsed Client Action/Code Size Method URI Ident Hierarchy/ From Content

According to Squid 2.0 documentation [squid], these fields are defined as follows:

Element	Definition
time	Unix timestamp as UTC seconds with a millisecond resolution.
duration	The elapsed time in milliseconds the transaction busied the cache.
client address	The client IP address.
bytes	The size is the amount of data delivered to the client, including headers.
request method	The request method to obtain an object.
URL	The requested URL.
rfc931	may contain the ident lookups for the requesting client (turned off by default)
hierarchy code	The hierarchy information provides information on how the request was handled (forwarding it to another cache, or requesting the content to the Origin Server).
type	The content type of the object as seen in the HTTP reply header.

Table 7: Information elements in Squid format

Squid also uses a "store log", which covers the objects currently kept on disk or removed ones, for debugging purposes typically.

Appendix B. Requirements

B.1. Additional Requirements

Section 7 of [I-D.ietf-cdni-requirements], already specifies a set of requirements for Logging (LOG-1 to LOG-16). Some security requirements also affect Logging (e.g., SEC-4).

This section is a placeholder for requirements identified in the work on logging, before they are proposed to the requirements draft authors.

Logging data is sensitive as it provides the raw material for producing bills etc. Therefore, the protocol delivering the Logging data must be reliable to avoid information loss. In addition, the protocol must scale to support the transport of large amounts of Logging data.

CDNs need to trust Logging information, thus, they want to know:

- o who issued the Logging (authentication), and
- o if the Logging has been modified by a third party (integrity).

Logging also contains confidential data, and therefore, it should be protected from eavesdropping.

All these needs translate into security requirements on both the Logging data format and on the Logging protocol.

Finally, this protocol must comply with the requirements identified in [I-D.ietf-cdni-requirements].

[Ed. note: cf. requirements draft: "SEC-4 [MED] The CDNI solution should be able to ensure that the Downstream CDN cannot spoof a transaction log attempting to appear as if it corresponds to a request redirected by a given Upstream CDN when that request has not been redirected by this Upstream CDN. This ensures non-repudiation by the Upstream CDN of transaction logs generated by the Downstream CDN for deliveries performed by the Downstream CDN on behalf of the Upstream CDN."]

B.2. Compliancy with Requirements draft

This section checks that all the identified requirements in the Requirements draft are fulfilled by this document.

[Ed. note: to be written later]

Appendix C. CDNI WG's position on candidate protocols for Logging Transport

This section will be expanded later with the position of the WG considering the alternative candidate protocols for Logging in CDNI.

[Ed. Note: in a later version, this memo will include an analysis of candidate protocols, based upon a set of (basic) requirements, such as reliable transport mode, preservation of the integrity of the information conveyed by the protocol, etc.]

C.1. CDNI WG's position on Syslog

[Ed. note: to be written later]

[Ed. note: add a few sentences to clarify why not directly use syslog... Operational reasons...]

C.2. CDNI WG's position on SNMP

As explained in [RFC6707], "SNMP traps pose scalability concerns and SNMP does not support guaranteed delivery of Traps and therefore could result in log records being lost and the consequent CoDRs and billing records for that content delivery not being produced as well as that content delivery being invisible to any analytics platforms."

Authors' Addresses

Gilles Bertrand (editor)
France Telecom - Orange
38-40 rue du General Leclerc
Issy les Moulineaux, 92130
FR

Phone: +33 1 45 29 89 46
Email: gilles.bertrand@orange.com

Stephan Emile
France Telecom - Orange
2 avenue Pierre Marzin
Lannion F-22307
France

Email: emile.stephan@orange.com

Roy Peterkofsky
Skytide, Inc.
One Kaiser Plaza, Suite 785
Oakland CA 94612
USA

Phone: +01 510 250 4284
Email: roy@skytide.com

Francois Le Faucheur
Cisco Systems
Greenside, 400 Avenue de Roumanille
Sophia Antipolis 06410
FR

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

Pawel Grochocki
Orange Polska
ul. Obrzezna 7
Warsaw 02-691
Poland

Email: pawel.grochocki@orange.com

CDNI
Internet-Draft
Intended status: Informational
Expires: December 30, 2012

G. Chen
China Telecom
M. Li
H. Xia
ZTE Corporation
J. Liang
China Telecom
June 28, 2012

Intra-CDN Provider CDNi Experiment
draft-chen-cdni-intra-cdn-provider-cdni-experiment-00

Abstract

In [I-D.ietf-cdni-use-cases], the Inter-Affiliates CDN Interconnection use case is described. In this scenario, a large CDN Provider may have several autonomous or semi-autonomous subsidiaries that each operates on their own CDN. The CDN Provider needs to make these down-stream CDNs interoperate to provide a consistent service to its customers on the whole collective footprint.

This document illustrates in details the CDNi experiment that has been carried out by China Telecom, and the lessons and experiences to CDNi standardization work.

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 December 30, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. Intra-CDN Provider CDNi Experiments	3
2.1. Experiment Configuration	4
2.2. Logging	7
2.3. UniContentID	7
2.4. Request Routing and Content Acquisition	8
2.4.1. Request Routing and Content Acquisition in Province B	8
2.4.2. Request Routing and Content Acquisition between Province A and Province B	10
2.4.3. Test Results	12
2.5. Control	12
3. Lessons Learned	14
3.1. Simplification of operation procedures	14
3.2. Redirection	14
3.3. UniContentID	14
3.4. Metadata and Logging	15
3.5. Inter-Operator CDN Interconnection	15
4. Security Considerations	15
5. IANA Considerations	15
6. Acknowledgments	16
7. References	16
7.1. Normative References	16
7.2. Informative References	16
Authors' Addresses	17

1. Introduction

As a CDN service provider, China Telecom has established video CDNs in more than ten provinces in China. These video CDNs, provided by different vendors, are relatively independent and only provide services to the end users of their own provinces. Under this circumstance, if a Content Provider (CP) wants to provide services to multiple provinces, it needs to interact with CDNs in other provinces via interfaces which may support different standards.

China Telecom launched the CDN interconnection trial network in 2011 where CDNs from six different vendors (ZTE, Huawei, Cisco, etc.) were used to conduct the interconnection experiment in three provinces. This experiment aims at testing the scenario where the operator provides autonomous services via CDN interconnection in order to provide enhanced user experience. It is noted that a simplification of the interconnection framework and the corresponding procedures would really improve the service and real-time viewing experience.

This experiment is not intended to cover all of the use cases or the scenarios that are within the scope of CDNi work. It simply provides some practical information gathered from the actual network experiment as a reference for the CDNi standardization work. These CDN interconnection implementation experiments cover mostly the intra-operator interconnection scenarios.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document reuses the terminology defined in:

[I-D.draft-ietf-cdni-problem-statement-06],

[I-D.draft-ietf-cdni-requirements-03],

[I-D.draft-ietf-cdni-framework-00], and

[I-D.draft-ietf-cdni-use-cases-08].

2. Intra-CDN Provider CDNi Experiments

2.1. Experiment Configuration

The interconnection of four CDNs in two provinces has been tested in this experiment. Each province has two CDNs interconnected which are provided by different CDN vendors. As depicted in Figure 1, CDN A1 of Province A has contracts with service providers CP1 and CP2, and it acts as the content storage center of the nation. CDN B1 of Province B has contract with service provider CP3. Meanwhile, CDN A1 and CDN B1 are the sub-center CDNs of respective province, while CDN A2 and CDN B2 are the regional CDNs of respective province. CDN A1 and CDN B1 are deployed on the provincial backbone networks, while CDN A2 and CDN B2 are deployed on the MANs. CDN A1 is the upstream CDN of CDN A2. Services are provided to the end users by certain node in regional CDN, which is usually the geographically closest one to the end user. However, if this desired node is overloaded, certain re-routing or load-balancing criteria could be used to choose another node. CDN B1 and CDN B2 have similar deployment. The provincial center CDN A1 and CDN B1 are interconnected with each other. They do not have interconnection with the region CDNs in other provinces than themselves. The regional CDNs of the respective provinces do not interconnect with one another either.

China Telecom's CDN trial network offers two types of services: intra-province service (provided by CP2 and CP3) and inter-province service (provided by CP1). Intra-province service is provided independently within the province without any interconnection with CDNs in other provinces. When inter-province service is provided, content is ingested to the CDN in a single province and then distributed among the CDNs in all other provinces in the trial network. In this experiment the inter-province service is ingested via CDN A1 node and the end users can obtain services through the CDNs that are located in their own provinces.

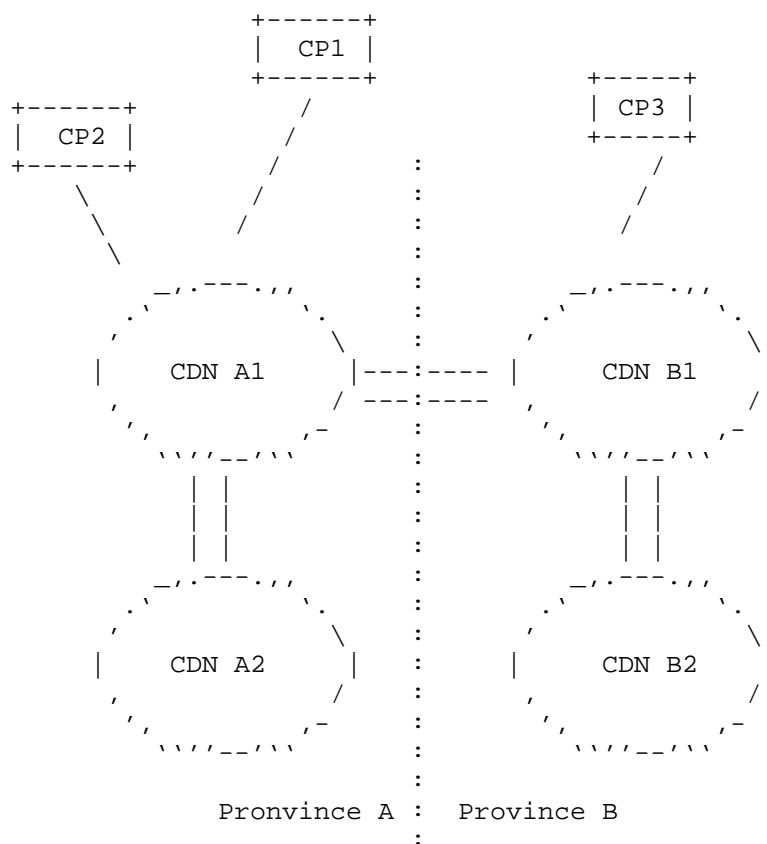


Figure 1 CDNI between Two Different Provinces, each with Two CDNs

The details of the experiment are as presented below:

CP3 has contract with CDN B1 to provide content delivery service, e.g., IPTV service, to the end users in the Province B region. CP3 does not serve end users outside Province B. As an autonomous service by China Telecom, the content of this service is ingested into CDN B1. As its downstream CDN, CDN B1 can delegate the content requests from end users to CDN B2 to perform content delivery.

When CDN B1 receives the content request from EU B of Province B related to service provided by CP3, it redirects this request to CDN B2. If CDN B2 has locally cached the copy of the content requested by EU B (cache hit case), it serves EU B directly by delivering the content to EU B. If CDN B2 does not have the content cached (cache miss case), it acquires the content from CDN B1.

CP1 has contract with CDN A1 to provide content delivery service, e.g., OTT service, to end users within Province A and in the region of Province B. The content for this service is ingested from CDN A1. As for the cross-province routing, since it is within the same operator, static configuration can be used for dCDN selection. In this experiment, the national content storage center CDN A1 configures locally the relationship table of the end user's IP addresses and the loading condition of dCDNs.

When CDN A1 receives a content request from EU B of Province B, it redirects the request to CDN B2 directly after it checks the local configuration table without going through multiple redirection processes like CDN A1->CDN B1->CDN B2. If CDN B2 has locally cached a copy of the content that has been requested by EU B (cache hit case), it serves EU B directly by delivering the content to EU B. If CDN B2 does not have the content cached (cache miss case), it acquires the content from CDN B1. If CDN B1 does not have the content cached either, it acquires the content from CDN A1.

In this experiment, we use a content acquisition method that is different from the current CDNi work. The method is based on Content Identification by using UniContentID that is defined to uniquely identify a content item. A detailed description of this method is presented in Section 2.3.

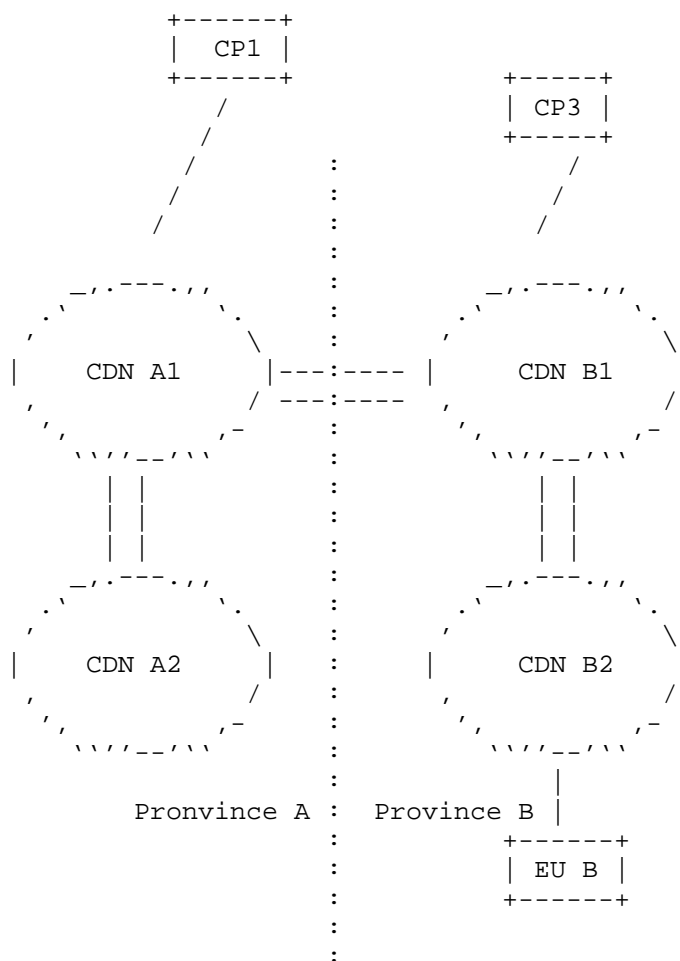


Figure 2 CDNi between Two Different Provinces

2.2. Logging

Since in this experiment CDN interconnection is implemented within the scope of the same operator, charging-related operations via Logging Interface are not required. Therefore, we have neither implemented nor tested any Logging operations.

2.3. UniContentID

In the current IETF CDNi standards, it is required to add the URL of original request and the URL in the process through CDNs in the request routing. When cache is not hit, downstream CDN needs to use the information above to trace to the source. The redirection flows

are complex and the URL becomes very long which makes the implementation very difficult.

In this experiment, the UniContentID as defined in [I-D.draft-chen-cdni-rr-content-acquisition] is used. UniContentID is described by two tuple as (ProviderID, ContentID), e.g. ('iptv.netitv.com', '01234567890123456789012345678900'), which can uniquely identify a content item. We trace the content source according to the configuration table of ProviderID and the corresponding relationship between ProviderID and the IP address of upstream CDN. It is our view that redirection and content acquisition are different routes.

2.4. Request Routing and Content Acquisition

2.4.1. Request Routing and Content Acquisition in Province B

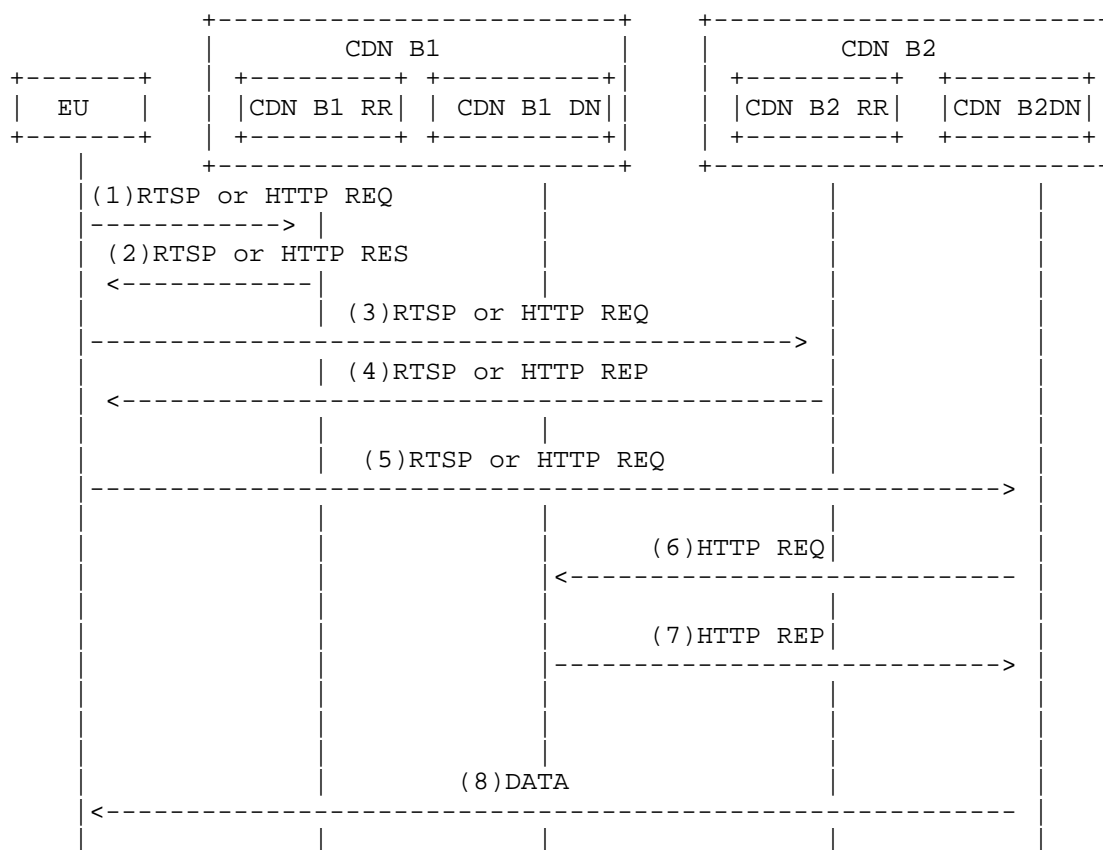


Figure 3 Request Routing and Content A Acquisition in Province B

The Message sequence of Figure 3 is shown below in details.

(1) End-User sends a request to the load balancer of CDN B1, i.e. RR of CDN B1 for the content. The URL includes the parameter of CMSID and Domain (Please refer to [I-D. draft-chen-cdni-rr-content-acquisition] for corresponding definitions).

(2) The RR of CDN B1 chooses an optimal RR of dCDN, i.e. RR of CDN B2 for the End-User according to the load of dCDN and the IP Pool information.

(3) End-User sends a request to the dCDN, i.e. RR of CDN B2 for content acquisition.

(4) RR of CDN B response to the End-User for the information of a delivery node i.e. DN.

(5) End-User sends a content request to the DN of CDN B2, the URL include the information of CMSID and Domain. The DN of CDN B2 analyses the ProviderID according the information of CMSID and Domain and looks up if the content exists in the cache according to the ProviderID and ContentID. If the content is cached, the DN of CDN B2 serves the End-User. Otherwise, it skips to step (6).

(6) The DN of CDN B2 looks up the configuration table and determines, according to the ProviderID, to acquire content uniquely identified by the ProviderID and ContentID from the DN of CDN B1.

(7) The content in the DN of CDN B1 relays to the DN of CDN B2.

(8) The relayed content is served by the DN of CDN B2 to the End-User via playing by downloading.

2.4.2. Request Routing and Content Acquisition between Province A and Province B

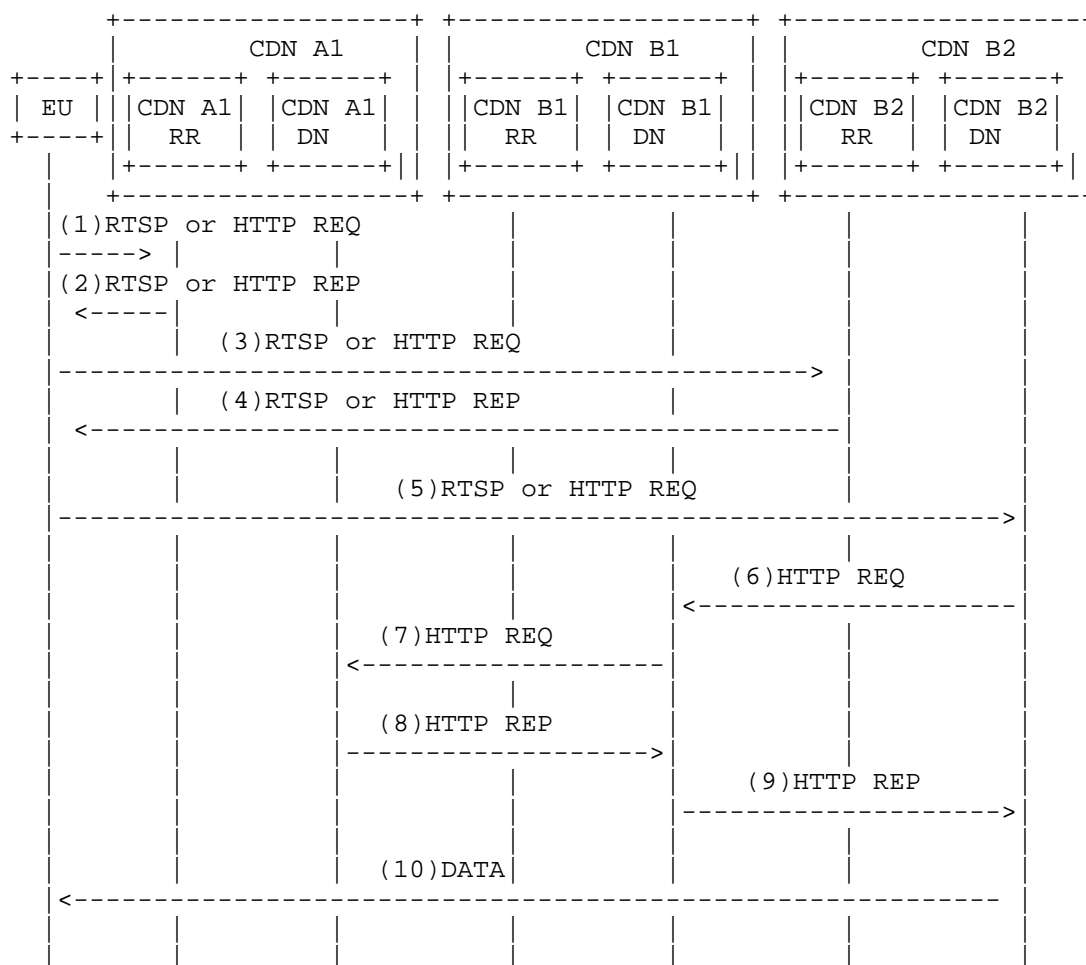


Figure 4 RR and Content Acquisition between Province A and Province B

The Message sequence of Figure 4 is shown below in details.

Step (1)~(5) is similar to step(1)~(5) of Section 2.4.1. Note that the request routing process does not need the participation of CDN B1.

(6) The DN of CDN B2 looks up the configuration table and determines, according to the ProviderID, to acquire content uniquely identified by the ProviderID and ContentID from the DN of CDN B1. If the content is cached in DN of CDN B1, the DN of CDN B1 serves the End-User. Otherwise, it skips to step (7).

(7) The DN of CDN B1 looks up the configuration table and determines, according to the ProviderID, to acquire content from the DN of CDN

A1.

(8) The content in the DN of CDN A1 relays to the DN of CDN B1.

(9) The content in the DN of CDN B1 relays to the DN of CDN B2.

(10) The relayed content is served by the DN of CDN B2 to the End-User via playing for downloading.

2.4.3. Test Results

Based on the experiment model above, we have tested the CDN interconnection scenario in China Telecom's trial network where 1 Gbps video traffic is not hit in the local cache and content acquisition is needed. Performance tests are done by using the tools from Shineck and Spirent. We have made such operations as fast forward, fast rewind and positioning play etc. The testing results show that the response time is less than one second and the Max DF is less than 50msec. Also we conducted the test for a period of six hours for stability tests through complex operation of fast forward, fast rewind, positioning play, etc. The tests results show that the response time is less than one second and call loss is less than 0.1%. During the process of performance tests, the user requests the video on demand and Live contents through set-up box and conducts complex operation such as fast forward, fast rewind, positioning play, etc. The program is smooth during the play. The tests show that the CDN interconnection architecture for intra-operator video service is both feasible and efficient.

2.5. Control

In the IETF CDNi draft [I-D.murray-cdni-triggers], the uCDN controls the content operation, i.e., content adding, content purging and content modifying are performed through the control interface. In this section, we describe a general content control flow by using CDN B1 and CDN B2 as an example which are used in this experiment.

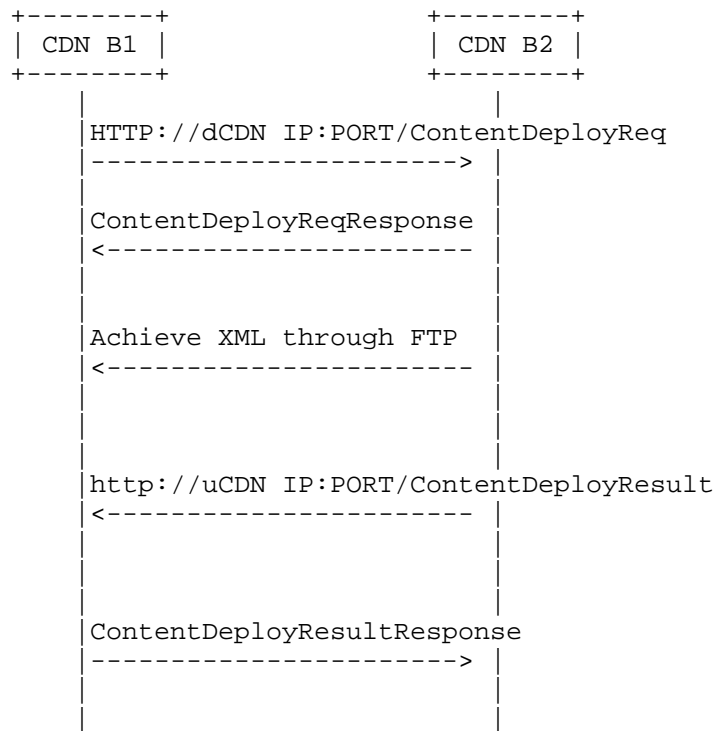


Figure 5 Message Exchange for Content Acquisition

The Message sequence of Figure 5 is shown below in details.

(1) CDN B1 sends a content management requests to the CDN B2 including the content adding, content purging and content modifying. The content object can be live content, video on demand content or TV on demand. The object of ContentDeployReq includes the URL address of XML description of the content object.

(2) CDN B2 checks if the URL FTP address from CDN B1 is OK. If the result is OK, CDN B2 responds positively (success) to the CDN B1.

(3) CDN B2 login in the FTP of CDN B1 to achieve the XML data of content object and executes the operation according to the instruction of CDN B1, i.e., content adding, content purging or content modifying.

(4) CDN B2 responds to the CDN B1 for the operation results, i.e., with either a success or a failure result.

(5) CDN B1 confirms to the CDN B2 for the operation result and registers the operation result.

3. Lessons Learned

The CDNi functionality tested in this experiment is applicable to intra-operator case only. The inter-province service is ingested via CDN in one province and can be used throughout the entire trial network in two provinces. During the initial stage of CDNi standardization, the most practical scenario to be considered is the interconnection of CDNs distributed in different geographical regions within one operator so as to enhance the consistency and continuity of the services provided by operators themselves. Therefore, this experiment is intended to provide some experiences and references on CDNi networking to those operators who have initial requirements for internal CDN interconnection across different geographical regions.

3.1. Simplification of operation procedures

It is demonstrated that relatively simple methods can be used to simplify or optimize the Request Redirection, Content Acquisition, Content Pre-Positioning, Content Addition/Modification/Deletion procedures. This is conducive to operators when they also act as service providers to serve the end users by fulfilling the requirements of real-time service and demanding user experience.

3.2. Redirection

According to the HTTP/DNS-based Request Routing process defined in the current [I-D.ietf-cdni-framework], multiple redirection processes are needed to determine the final CDN node that is suitable to serve the end user. This may be convenient in the case of two-level CDNs. But in case of large-scale CDN networking or complex CDN topology, this would cause serious delay. The method provided in this experiment, i.e., the one based on the local configuration of the relationship table of end users' IP addresses and load situation of dCDNs, the uCDN, as the national content storage center, can quickly acquire and locate the final dCDN that is suitable to serve the end user. By using this technique, the routing selection can be largely simplified especially when operators have large-scale internal networking.

3.3. UniContentID

In current IETF CDNi work [I-D.ietf-cdni-framework], the content acquisition by dCDN from uCDN is achieved via embedding the URL of the original request as well as that of the CDN the request is redirected to during the redirection process. This would result in a very long URL. Limited by the length and format of URL, such approach would cause serious delay and waste of resources. In addition, it would be difficult to implement this, especially under

the complex CDNi topology.

The unique content identification (named UniContentID in this document) can be used to uniquely identify the content item ingested by the content provider. The content source can also be resolved from UniContentID contained in the end user's content request for content acquisition. Due to the uniformity of UniContentID format and its unchangeable nature during the transmission among CDNs, it can all along identify the content requested by the end user even after multiple forwards under complex CDNi topology. Note that these introduce the need for defining a unique content identification for content item in CDNi framework.

3.4. Metadata and Logging

The metadata related to CDNi content delivery would be relatively simple. Such policy information as content ingestion, content acquisition, etc. can be achieved by pre-configuration. It is also due to the scope within one single operator, there is no need for charging-related operations via logging interface, which, as a result, can be simplified or may not even be supported.

3.5. Inter-Operator CDN Interconnection

For CDN interconnection across operators, if the operators are clearly aware of each other's CDN framework (according to their agreements), the method that is used in this experiment can also be utilized as a reference, i.e., for implementing end user's request redirection and content acquisition via maintaining the configuration table, which can also achieve relatively high content delivery efficiency. For those operators who have complicated internal CDN topology or proprietary APIs or CDN topology, it is our view that it requires to seek solutions for dynamic request redirection and content acquisition.

4. Security Considerations

This experiment is carried out within a single operator. No security issues are considered at this stage of the experiment.

5. IANA Considerations

This memo has no IANA Considerations.

6. Acknowledgments

To be added later

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

7.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", April 2012.
- [I-D.ietf-cdni-problem-statement]
Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", May 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", December 2011.
- [I-D.ietf-cdni-use-cases]
Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", June 2012.
- [I-D.murray-cdni-triggers]
Murray, R. and B. Niven-Jenkins, "CDN Interconnect Triggers", February 2012.
- [I-D.narten-iana-considerations-rfc2434bis]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

Authors' Addresses

Ge Chen
China Telecom
109 West Zhongshan Ave
Guangzhou, Tianhe District
China

Phone:
Email: cheng@gsta.com

Mian Li
ZTE Corporation
Nanjing, 210012
China

Phone:
Email: li.mian@zte.com.cn

Hongfei Xia
ZTE Corporation
Nanjing, 210012
China

Phone:
Email: xia.hongfei@zte.com.cn

Jie Liang
China Telecom
109 West Zhongshan Ave
Guangzhou, Tianhe District
China

Phone:
Email: liangj@gsta.com

CDNI WG
Internet-Draft
Intended status: Experimental
Expires: August 3, 2016

G. Chen
China Telecom
M. Li
H. Xia
ZTE Corporation
B. Khasnabish
ZTE (TX) Inc.
J. Liang
China Telecom
January 31, 2016

Intra-CDN Provider CDNi Experiment
draft-chen-cdni-intra-cdn-provider-cdni-experiment-04

Abstract

In [RFC6770], the Inter-Affiliates CDN Interconnection use case is described. In this scenario, a large CDN Provider may have several autonomous or semi-autonomous subsidiaries that each operates on their own CDN. The CDN Provider needs to make these down-stream CDNs interoperate to provide a consistent service to its customers on the whole collective footprint.

This document illustrates in details the CDNi experiment that has been carried out by China Telecom, and the lessons and experiences to CDNi standardization work.

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 3, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. Intra-CDN Provider CDNi Experiments	3
2.1. Experiment Configuration	3
2.2. Logging	7
2.3. UniContentID	7
2.4. Request Routing and Content Acquisition	8
2.4.1. Request Routing and Content Acquisition in Province B	8
2.4.2. Request Routing and Content Acquisition between Province A and Province B	9
2.4.3. Test Results	11
2.5. Control	11
3. Lessons Learned	13
3.1. Simplification of operation procedures	13
3.2. Redirection	13
3.3. UniContentID	13
3.4. Metadata and Logging	14
3.5. Inter-Operator CDN Interconnection	14
4. Security Considerations	14
5. IANA Considerations	14
6. Acknowledgments	15
7. References	15
7.1. Normative References	15
7.2. Informative References	15
Authors' Addresses	16

1. Introduction

As a CDN service provider, China Telecom has established video CDNs in more than ten provinces in China. These video CDNs, provided by different vendors, are relatively independent and only provide services to the end users of their own provinces. Under this circumstance, if a Content Provider (CP) wants to provide services to multiple provinces, it needs to interact with CDNs in other provinces via interfaces which may support different standards.

In 2011, China Telecom launched the CDN interconnection trial network where CDNs from six different vendors (ZTE, Huawei, Cisco, etc.) were used to conduct the interconnection experiment in three provinces. This experiment aims at testing the scenario where the operator provides autonomous services via CDN interconnection in order to provide enhanced user experience. It is noted that a simplification of the interconnection framework and the corresponding procedures would really improve the service and real-time viewing experience.

This experiment is not intended to cover all of the use cases or the scenarios that are within the scope of CDNi work. It simply provides some practical information gathered from the actual network experiment as a reference to the CDNi standardization work. These CDN interconnection implementation experiments cover mostly the intra-operator interconnection scenarios.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document reuses the terminology defined in the following RFCs: [RFC6707], [RFC6770], [RFC7336], and [RFC7337].

2. Intra-CDN Provider CDNi Experiments

2.1. Experiment Configuration

The interconnection of four CDNs in two provinces has been tested in this experiment. Each province has two CDNs interconnected which are provided by different CDN vendors. As depicted in Figure 1, CDN A1 of Province A has contracts with service providers CP1 and CP2, and it acts as the content storage center of the nation. CDN B1 of Province B has contract with service provider CP3. Meanwhile, CDN A1 and CDN B1 are the sub-center CDNs of respective province, while CDN A2 and CDN B2 are the regional CDNs of respective province. CDN A1 and CDN B1 are deployed on the provincial backbone networks, while

CDN A2 and CDN B2 are deployed on the MANs. CDN A1 is the upstream CDN of CDN A2. Services are provided to the end users by certain node in regional CDN, which is usually the geographically closest one to the end user. However, if this desired node is overloaded, certain re-routing or load-balancing criteria could be used to choose another node. CDN B1 and CDN B2 have similar deployment. The provincial center CDN A1 and CDN B1 are interconnected with each other. They do not have interconnection with the region CDNs in other provinces than themselves. The regional CDNs of the respective provinces do not interconnect with one another either.

China Telecom's CDN trial network offers two types of services: intra-province service (provided by CP2 and CP3) and inter-province service (provided by CP1). Intra-province service is provided independently within the province without any interconnection with CDNs in other provinces. When inter-province service is provided, content is ingested to the CDN in a single province and then distributed among the CDNs in all other provinces in the trial network. In this experiment the inter-province service is ingested via CDN A1 node and the end users can obtain services through the CDNs that are located in their own provinces.

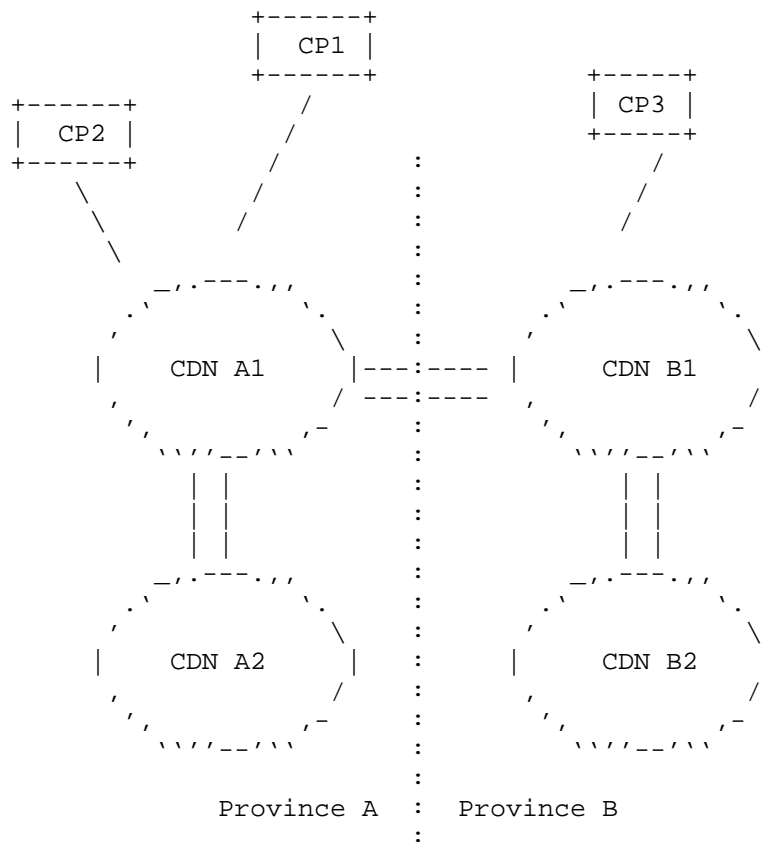


Figure 1 CDNI between Two Different Provinces, each with Two CDNs

The details of the experiment are as presented below:

CP3 has contract with CDN B1 to provide content delivery service, e.g., IPTV service, to the end users in the Province B region. CP3 does not serve end users outside Province B. As an autonomous service by China Telecom, the content of this service is ingested into CDN B1. As its downstream CDN, CDN B1 can delegate the content requests from end users to CDN B2 to perform content delivery.

When CDN B1 receives the content request from EU B of Province B related to service provided by CP3, it redirects this request to CDN B2. If CDN B2 has locally cached the copy of the content requested by EU B (cache hit case), it serves EU B directly by delivering the content to EU B. If CDN B2 does not have the content cached (cache miss case), it acquires the content from CDN B1.

CP1 has contract with CDN A1 to provide content delivery service, e.g., OTT service, to end users within Province A and in the region of Province B. The content for this service is ingested from CDN A1. As for the cross-province routing, since it is within the same operator, static configuration can be used for dCDN selection. In this experiment, the national content storage center CDN A1 configures locally the relationship table of the end user's IP addresses and the loading condition of dCDNs.

When CDN A1 receives a content request from EU B of Province B, it redirects the request to CDN B2 directly after it checks the local configuration table without going through multiple redirection processes like CDN A1->CDN B1->CDN B2. If CDN B2 has locally cached a copy of the content that has been requested by EU B (cache hit case), it serves EU B directly by delivering the content to EU B. If CDN B2 does not have the content cached (cache miss case), it acquires the content from CDN B1. If CDN B1 does not have the content cached either, it acquires the content from CDN A1.

In this experiment, we use a content acquisition method that is different from the current CDNi work. The method is based on Content Identification by using UniContentID that is defined to uniquely identify a content item. A detailed description of this method is presented in Section 2.3.

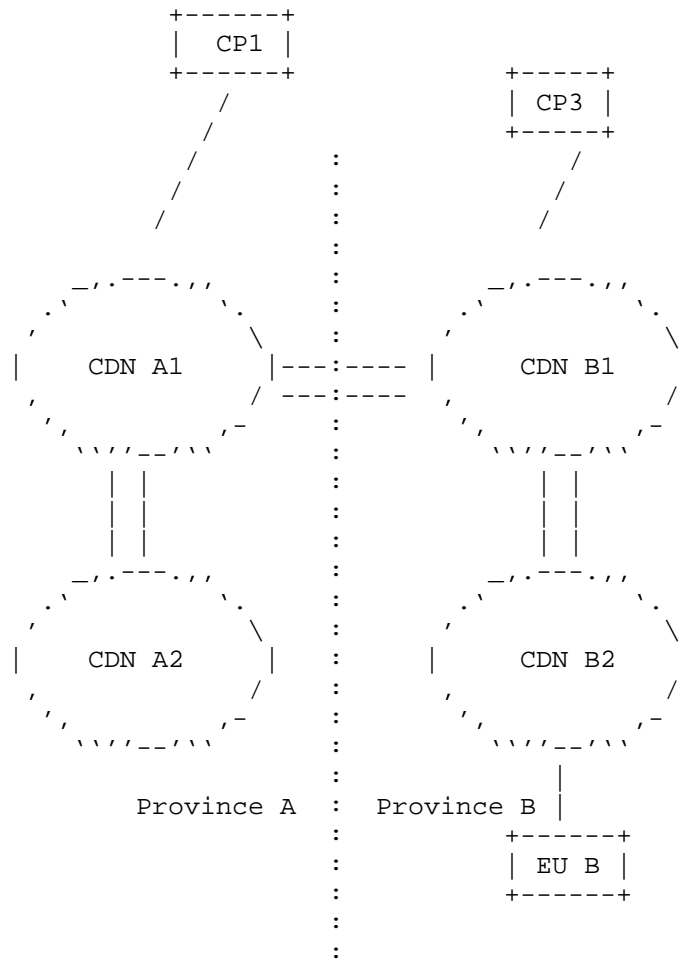


Figure 2 CDNI between Two Different Provinces

2.2. Logging

Since in this experiment CDN interconnection is implemented within the scope of the same operator, charging-related operations via Logging Interface are not required. Therefore, we have neither implemented nor tested any Logging operations.

2.3. UniContentID

In the current IETF CDNi standards, it is required to add the URL of original request and the URL in the process through CDNs in the request routing. When cache is not hit, downstream CDN needs to use the information above to trace the source. The redirection flows are

complex and the URL becomes very long which makes the implementation very difficult.

In this experiment, the UniContentID as defined in [I-D.draft-chen-cdni-rr-content-acquisition] is used. UniContentID is described by two tuple as (ProviderID, ContentID), e.g.

('iptv.netitv.com', '01234567890123456789012345678900'), which can uniquely identify a content item. We trace the content source according to the configuration table of ProviderID and the corresponding relationship between ProviderID and the IP address of upstream CDN. It is our view that redirection and content acquisition are different routes.

2.4. Request Routing and Content Acquisition

2.4.1. Request Routing and Content Acquisition in Province B

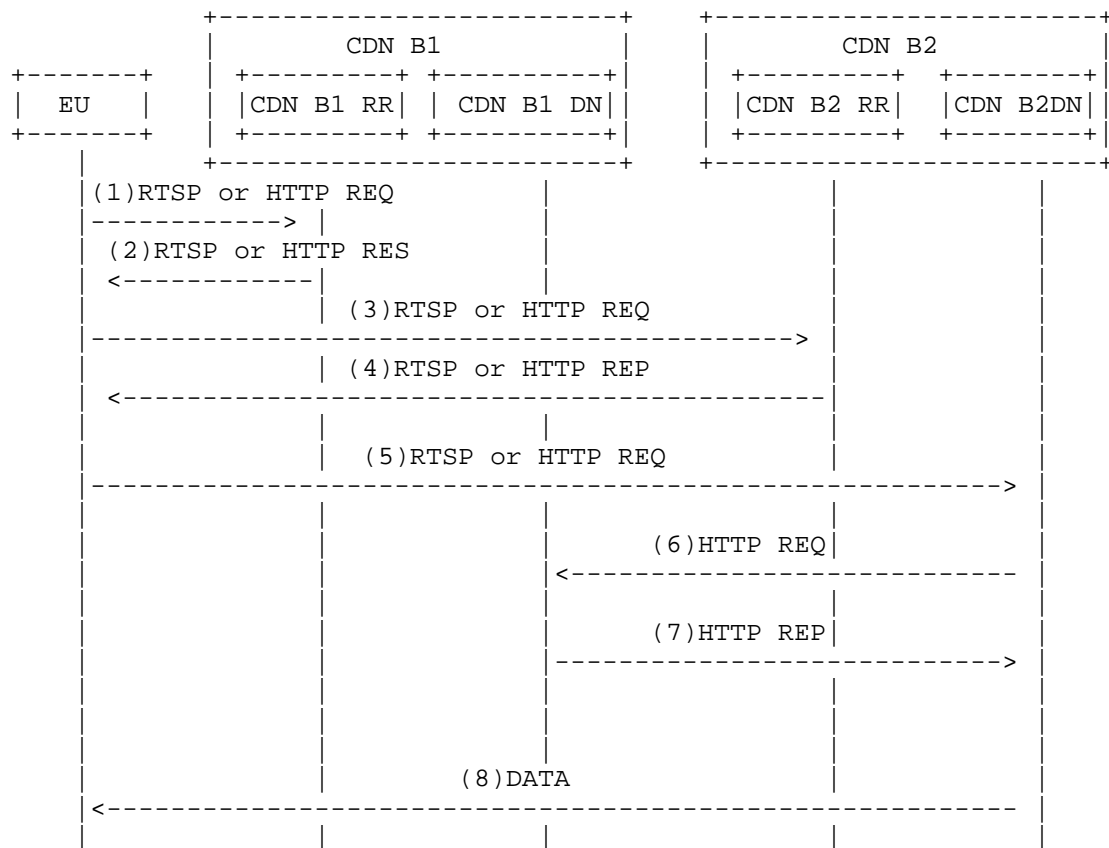


Figure 3 Request Routing and Content A Acquisition in Province B

The Message sequence of Figure 3 is shown below in details.

(1) End-User sends a request to the load balancer of CDN B1, i.e. RR of CDN B1 for the content. The URL includes the parameter of CMSID and Domain (Please refer to [I-D. draft-chen-cdni-rr-content-acquisition] for corresponding definitions).

(2) The RR of CDN B1 chooses an optimal RR of dCDN, i.e. RR of CDN B2 for the End-User according to the load of dCDN and the IP Pool information.

(3) End-User sends a request to the dCDN, i.e. RR of CDN B2 for content acquisition.

(4) RR of CDN B response to the End-User for the information of a delivery node i.e. DN.

(5) End-User sends a content request to the DN of CDN B2, the URL include the information of CMSID and Domain. The DN of CDN B2 analyses the ProviderID according the information of CMSID and Domain and looks up if the content exists in the cache according to the ProviderID and ContentID. If the content is cached, the DN of CDN B2 serves the End-User. Otherwise, it skips to step (6).

(6) The DN of CDN B2 looks up the configuration table and determines, according to the ProviderID, to acquire content uniquely identified by the ProviderID and ContentID from the DN of CDN B1.

(7) The content in the DN of CDN B1 relays to the DN of CDN B2.

(8) The relayed content is served by the DN of CDN B2 to the End-User via playing by downloading.

2.4.2. Request Routing and Content Acquisition between Province A and Province B

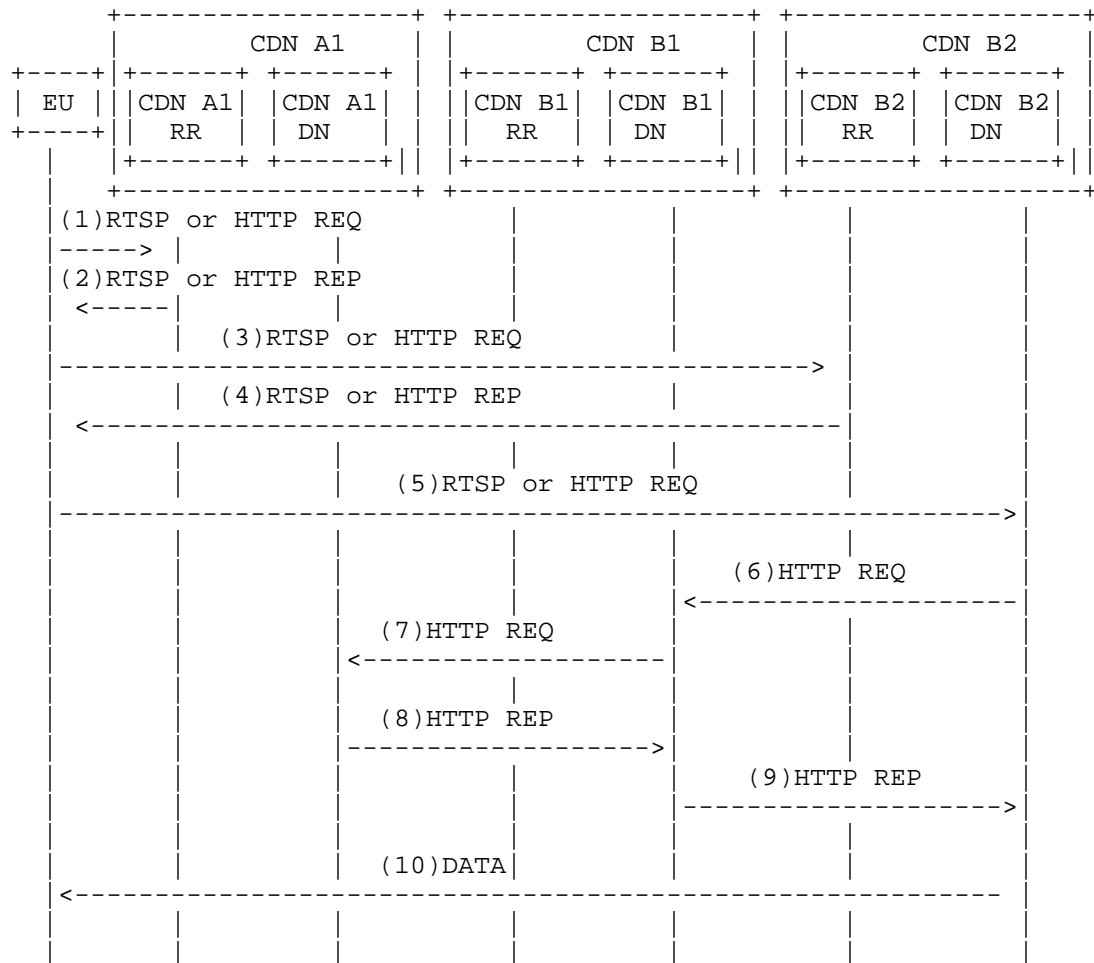


Figure 4 RR and Content Acquisition between Province A and Province B

The Message sequence of Figure 4 is shown below in details.

Step (1)~(5) is similar to step(1)~(5)of Section 2.4.1.Note that the request routing process does not need the participation of CDN B1.

(6) The DN of CDN B2 looks up the configuration table and determines, according to the ProviderID, to acquire content uniquely identified by the ProviderID and ContentID from the DN of CDN B1. If the content is cached in DN of CDN B1, the DN of CDN B1 serves the End-User. Otherwise, it skips to step (7).

(7) The DN of CDN B1 looks up the configuration table and determines, according to the ProviderID, to acquire content from the DN of CDN A1.

(8) The content in the DN of CDN A1 relays to the DN of CDN B1.

(9) The content in the DN of CDN B1 relays to the DN of CDN B2.

(10) The relayed content is served by the DN of CDN B2 to the End-User via playing for downloading.

2.4.3. Test Results

Based on the experiment model above, we have tested the CDN interconnection scenario in China Telecom's trial network where 1 Gbps video traffic is not hit in the local cache and content acquisition is needed. Performance tests are done by using the tools from Shineck and Spirent. We have made such operations as fast forward, fast rewind and positioning play etc. The testing results show that the response time is less than one second and the Max DF is less than 50msec. Also we conducted the test for a period of six hours for stability tests through complex operation of fast forward, fast rewind, positioning play, etc. The tests results show that the response time is less than one second and call loss is less than 0.1%. During the process of performance tests, the user requests the video on demand and Live contents through set-up box and conducts complex operation such as fast forward, fast rewind, positioning play, etc. The program is smooth during the play. The tests show that the CDN interconnection architecture for intra-operator video service is both feasible and efficient.

2.5. Control

In the IETF CDNi draft [I-D.murray-cdni-triggers], the uCDN controls the content operation, i.e., content adding, content purging and content modifying are performed through the control interface. In this section, we describe a general content control flow by using CDN B1 and CDN B2 as an example which are used in this experiment.

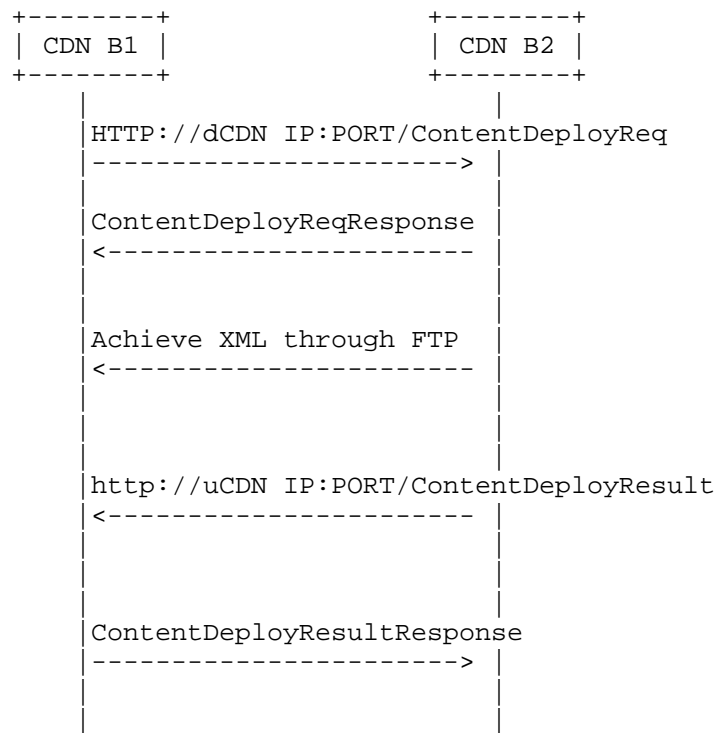


Figure 5 Message Exchange for Content Acquisition

The Message sequence of Figure 5 is shown below in details.

- (1) CDN B1 sends a content management requests to the CDN B2 including the content adding, content purging and content modifying. The content object can be live content, video on demand content or TV on demand. The object of ContentDeployReq includes the URL address of XML description of the content object.
- (2) CDN B2 checks if the URL FTP address from CDN B1 is OK. If the result is OK, CDN B2 responds positively (success) to the CDN B1.
- (3) CDN B2 login in the FTP of CDN B1 to achieve the XML data of content object and executes the operation according to the instruction of CDN B1, i.e., content adding, content purging or content modifying.
- (4) CDN B2 responds to the CDN B1 for the operation results, i.e., with either a success or a failure result.
- (5) CDN B1 confirms to the CDN B2 for the operation result and registers the operation result.

3. Lessons Learned

The CDNi functionality tested in this experiment is applicable to intra-operator case only. The inter-province service is ingested via CDN in one province and can be used throughout the entire trial network in two provinces. During the initial stage of CDNi standardization, the most practical scenario to be considered is the interconnection of CDNs distributed in different geographical regions within one operator so as to enhance the consistency and continuity of the services provided by operators themselves. Therefore, this experiment is intended to provide some experiences and references on CDNi networking to those operators who have initial requirements for internal CDN interconnection across different geographical regions.

3.1. Simplification of operation procedures

It is demonstrated that relatively simple methods can be used to simplify or optimize the Request Redirection, Content Acquisition, Content Pre-Positioning, Content Addition/Modification/Deletion procedures. This is conducive to operators when they also act as service providers to serve the end users by fulfilling the requirements of real-time service and demanding user experience.

3.2. Redirection

According to the HTTP/DNS-based Request Routing process defined in the current [RFC7336], multiple redirection processes are needed to determine the final CDN node that is suitable to serve the end user. This may be convenient in the case of two-level CDNs. But in case of large-scale CDN networking or complex CDN topology, this would cause serious delay. The method provided in this experiment, i.e., the one based on the local configuration of the relationship table of end users' IP addresses and load situation of dCDNs, the uCDN, as the national content storage center, can quickly acquire and locate the final dCDN that is suitable to serve the end user. By using this technique, the routing selection can be largely simplified especially when operators have large-scale internal networking.

3.3. UniContentID

In current IETF CDNi work [RFC7336], the content acquisition by dCDN from uCDN is achieved via embedding the URL of the original request as well as that of the CDN the request is redirected to during the redirection process. This would result in a very long URL. Limited by the length and format of URL, such approach would cause serious delay and waste of resources. In addition, it would be difficult to implement this, especially under the complex CDNi topology.

The unique content identification (named UniContentID in this document) can be used to uniquely identify the content item ingested by the content provider. The content source can also be resolved from UniContentID contained in the end user's content request for content acquisition. Due to the uniformity of UniContentID format and its unchangeable nature during the transmission among CDNs, it can all along identify the content requested by the end user even after multiple forwards under complex CDNi topology. Note that these introduce the need for defining a unique content identification for content item in CDNi framework.

3.4. Metadata and Logging

The metadata related to CDNi content delivery are as discussed in [I-D.ietf-cdni-metadata]. The policy information like content ingestion, content acquisition, etc. can be obtained from pre-configuration data. Also, since the scope of this experiment is one operator, there may not be any need for obtaining charging-related operations via logging interface. However, if needed the specifications that are being developed in [I-D.ietf-cdni-logging] would be useful.

3.5. Inter-Operator CDN Interconnection

For CDN interconnection across operators, if the operators are clearly aware of each other's CDN framework (according to their agreements), the method that is used in this experiment can also be utilized as a reference, i.e., for implementing end user's request redirection and content acquisition via maintaining the configuration table, which can also achieve relatively high content delivery efficiency. For those operators who have complicated internal CDN topology or proprietary APIs or CDN topology, it is our view that it requires to seek solutions for dynamic request redirection - as discussed in [I-D.ietf-cdni-redirection] - and content acquisition.

4. Security Considerations

This experiment is carried out within a single operator. No security issues are considered at this stage of the experiment.

5. IANA Considerations

There are no IANA considerations for this draft.

6. Acknowledgments

To be added later

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

7.2. Informative References

- [I-D.ietf-cdni-logging] Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-21 (work in progress), November 2015.
- [I-D.ietf-cdni-metadata] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-12 (work in progress), October 2015.
- [I-D.ietf-cdni-redirection] Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-16 (work in progress), January 2016.
- [I-D.murray-cdni-triggers] Murray, R. and B. Niven-Jenkins, "CDN Interconnect Triggers", February 2012.
- [I-D.narten-iana-considerations-rfc2434bis] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <<http://www.rfc-editor.org/info/rfc2629>>.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<http://www.rfc-editor.org/info/rfc3552>>.

- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <<http://www.rfc-editor.org/info/rfc6707>>.
- [RFC6770] Bertrand, G., Ed., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, DOI 10.17487/RFC6770, November 2012, <<http://www.rfc-editor.org/info/rfc6770>>.
- [RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <<http://www.rfc-editor.org/info/rfc7336>>.
- [RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <<http://www.rfc-editor.org/info/rfc7337>>.

Authors' Addresses

Ge Chen
China Telecom
109 West Zhongshan Ave
Guangzhou, Tianhe District
China

Email: cheng@gsta.com

Mian Li
ZTE Corporation
Nanjing 210012
China

Email: li.mian@zte.com.cn

Hongfei Xia
ZTE Corporation
Nanjing 210012
China

Email: xia.hongfei@zte.com.cn

Bhumip Khasnabish
ZTE (TX) Inc.
USA

Phone: +001-781-752-8003
Email: vumipl@gmail.com, bhumip.khasnabish@ztetx.com
URI: <http://tinyurl.com/bhumip/>

Jie Liang
China Telecom
109 West Zhongshan Ave
Guangzhou, Tianhe District
China

Email: liangj@gsta.com

INTERNET-DRAFT
Intended Status: Standards Track
Expires: April 24, 2013

TS Choi
ETRI
YI Seo
DJ Kim
KT
JM Lee
SKT
JR Koo
LGU+
JDH Shinn
Solbox Inc.
KS Park
KAIST
October 21, 2012

CDNi Request Routing Redirection with Loop Prevention
draft-choi-cdni-req-routing-redir-loop-prevention-01

Abstract

This document describes request routing redirection procedures, loop prevention mechanisms, and other operational considerations which are associated with redirection.

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 December 29, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1	Introduction	3
1.1	Terminology	3
2.	Redirection Procedures	3
2.1.	Preliminaries	3
2.2.	Iterative Redirection Procedures	4
2.2.1.	HTTP-based Redirection	4
2.2.2.	DNS-based Redirection	8
2.3.	Recursive Redirection Procedures	10
2.3.1.	HTTP-based Redirection	10
2.3.2.	DNS-based Redirection	14
3.	Redirection Loop Prevention	17
4.	Redirection Operational Considerations	18
5	Security Considerations	19
6	IANA Considerations	19
7	References	19
7.1	Normative References	19
7.2	Informative References	19
	Authors' Addresses	19

1 Introduction

According to the CDNi generic and request routing interface requirements[I-D.ietf-cdni-requirements], the CDNi solution shall support iterative and recursive CDNi request routing, efficient request routing for small and large objects, arbitrary number of levels of cascaded CDN redirection, looping prevention of any CDN request routing redirection, and subsequently allowing the request routing redirection. To meet such requirements, this document describes request routing redirection procedures, loop prevention mechanisms, and other operational considerations that are associated with redirection.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Redirection Procedures

2.1. Preliminaries

To meet the requirements of request routing redirection, we define the term "CDN-Provider-ID". It uniquely identifies each CDN provider during the course of request routing redirection. It consists of "CDN provider name" and "MaxNumRedHops". A pair of an AS number and an additional qualifier is used for CDN provider name. Since more than one CDN providers can belong to the same AS, an additional qualifier is used to guarantee the uniqueness. MaxNumRedHops represents a maximum allowed redirections. The value is decreased once every redirection occurs until it reaches 0. To avoid its usage abuse (e.g., end user or CDN operator can set huge number like 100 or above), a reasonable upper bound has to be agreed among CDN providers. Security aspect of it is for further study.

A few examples of the CDN provider names are 100:0 and 200:1. The former means that a CDN provider belong to AS 100 and it is the only CDN provider within that AS. The latter represents the first CDN provider in the AS 200. There are other CDN providers in the same AS.

One example of CDN-Provider-ID is "CDN-Provider-Name=100:0 & MaxNumRedHops=10", which means that a CDN provider that belong to AS number 100 and it is the only CDN provider and a maximum allowed redirection is 10. An example how a list of CDN-Provider-IDs can be carried in the URI query string when a certain cascaded request

routing redirection occurs is the following. We assume that redirection is cascaded three times: uCDN -> dCDN1 -> dCDN2. dCDN1, then, carries the following URL, "http://cdn.csp.com?uCDN-Provider-ID=100:0&dCDN1-Provider-ID=200:1&MaxNumRedHops=9". Note that MaxNumRedHops carries the latest number instead of adding in every CDN-Provider-ID to save the space in URI query string.

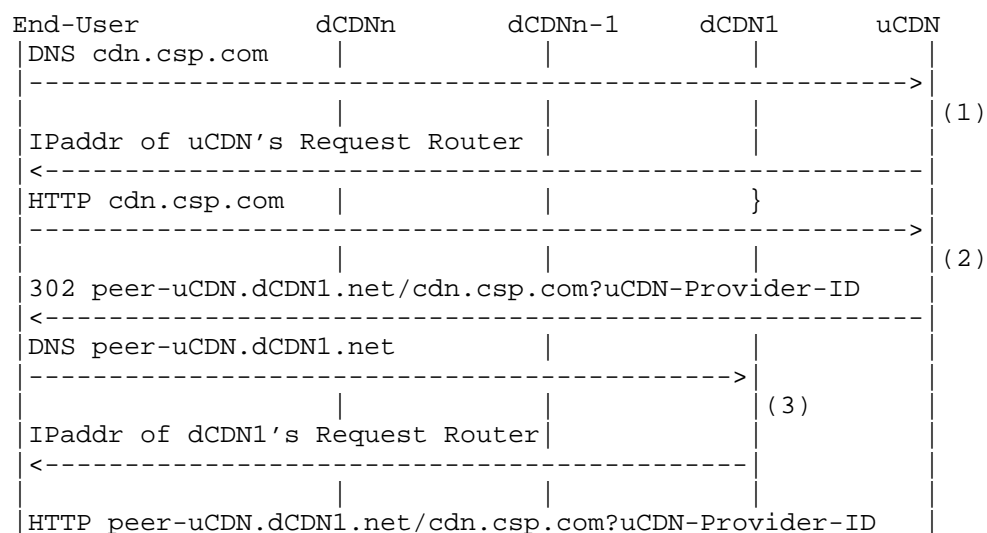
It is applicable for both HTTP-based and DNS redirections. For HTTP-based redirection, we define a HTTP request routing redirection header "CDN-Provider-ID". For each step of redirection, it is attached to the beginning of the provider domain URL. For example, uCDN initiates a redirection with its URL, http://100:0:10.cdn.csp.com. dCDN further attaches its own CDN-Provider-ID in the front when another level of redirection is required. For DNS-based redirection, the CDN-Provider-ID can be attached in the DNS CNAME.

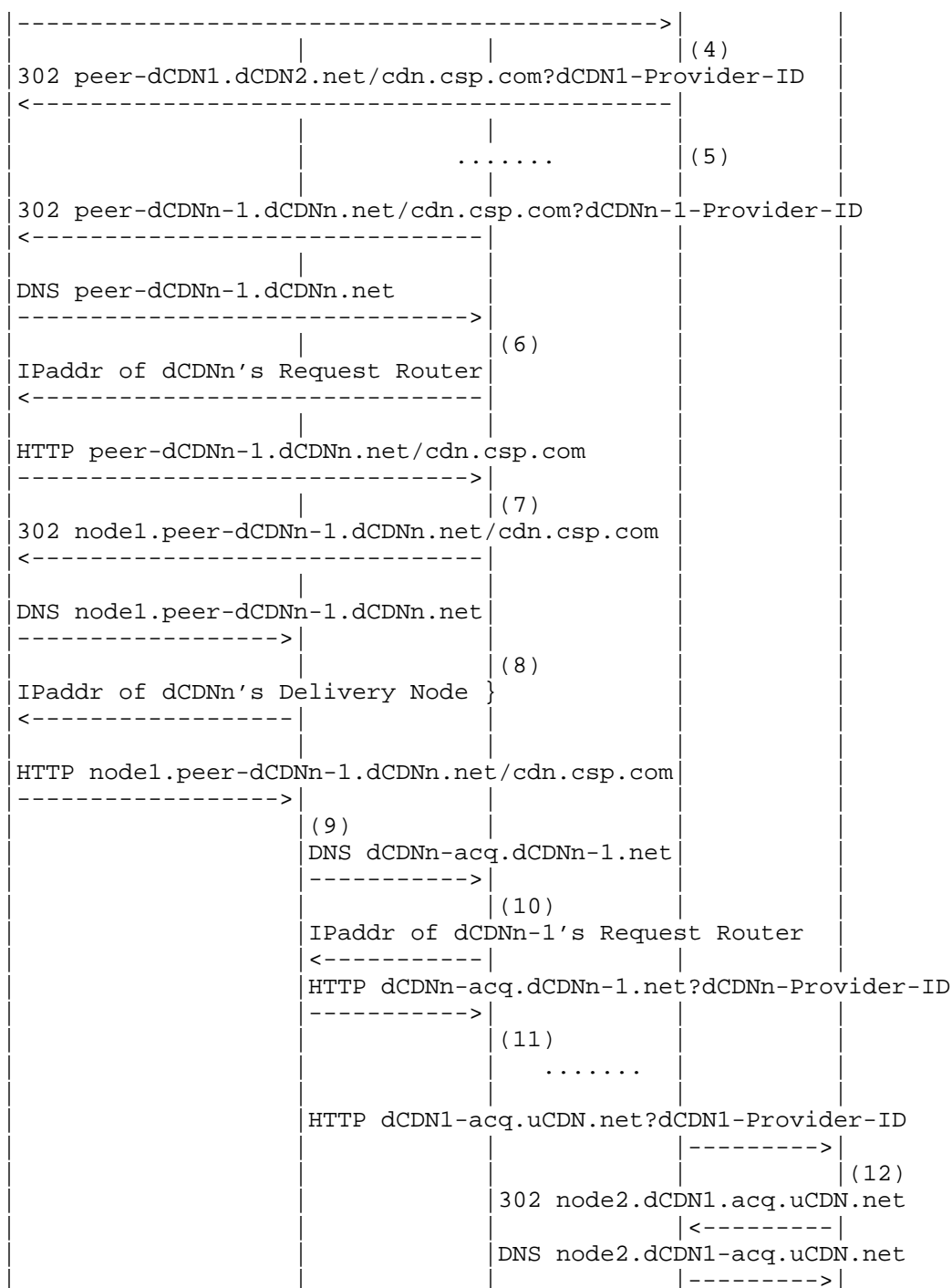
Since there is a CDNi requirement to support of arbitrary topology of interconnected CDNs, this document assumes that the redirection procedures and loop prevention mechanisms must also support arbitrary topology.

2.2. Iterative Redirection Procedures

2.2.1. HTTP-based Redirection

In this section, we describe an iterative procedure of HTTP-based request routing redirection with loop prevention.





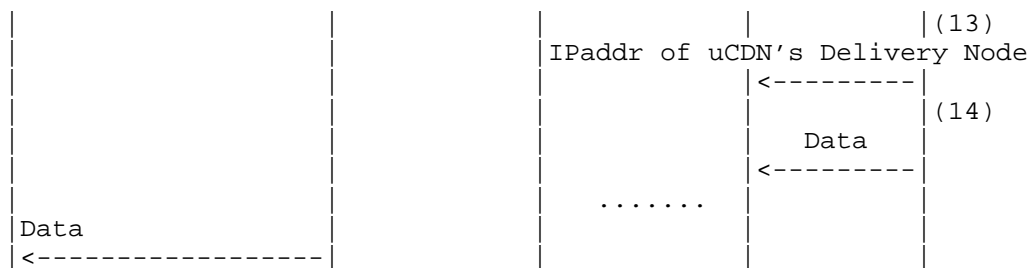


Figure 1: HTTP-based request routing redirection iterative procedure

The steps illustrated in the figure are as follows:

1. A DNS resolver for uCDN provider processes the DNS request for its customer based on CDN-domain `cdn.csp.com`. It returns the IP address of a request router in uCDN provider.
2. A Request Router for uCDN provider processes the HTTP request and recognizes that the end-user is best served by dCDN1. So it returns a 302 redirect message for a new URL constructed by "stacking" dCDN1's distinguished CDN-domain (`peer-uCDN.dCDN1.net`) on the front of the original URL. It also adds uCDN's CDN-Provider-ID in the URI query string of the HTTP request message. (e.g., `uCDN-Provider-ID=100:0 & MaxNumRedHops=10`). This information is not processed by the customer but conveyed in the HTTP message without any modification of the step 4. The details on how it is used for loop prevention is described in the step 4.
3. The end-user does a DNS lookup using dCDN1's distinguished CDN-domain (`peer-uCDN.dCDN1.net`). dCDN1's DNS resolver returns the IP address of a request router for dCDN1.
4. The request router for dCDN1 processes the HTTP request. There are two options: redirect further to another dCDN (i.e., cascading the request) or process it by itself. In either cases, it performs loop prevention step first. It checks a list of CDN-provider-IDs in the URI query string: it contains a list of CDN providers which requested redirections so far. If either it contains own CDN provider name or `MaxNumRedHops` becomes 0, it means that the redirection loop has occurred or the number of redirection hops has reached the maximum. Once loop is detected, details on the next steps is described in the section 3. If it is loop free, it either redirects further or processes based on the local policy. For the former, it selects another dCDN provider and sends an HTTP redirect message with its own CDN-Provider-ID included in its URI query string (e.g.,

uCDN-Provider-ID=100:0 & dCDN1-Provider-ID=200:1 & MaxNumRedHops=9) attached. For the latter, it selects a suitable delivery node to serve the end-user request, and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the dCDN1's distinguished CDN-domain that points to the selected delivery node. Then it goes to the step 6.

5. If further redirection is decided, it repeats steps 2 - 4 until it either selects dCDN provider to serve the request or MaxNumRedHops expires. If the former occurs, it resumes the step 6. If the latter occurs, it follows the processes described in the section 3.
6. Assuming that dCDNn is selected as a serving dCDN provider, the end-user does a DNS lookup using dCDNn's distinguished CDN-domain (peer-dCDNn-1.dCDNn.net). dCDNn-1's DNS resolver returns the IP address of a request router for dCDNn.
7. The request router for dCDN1 processes the HTTP request and selects a suitable delivery node to serve the end-user request, and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the dCDNn's distinguished CDN-domain that points to the selected delivery node.
8. The end-user does a DNS lookup using dCDNn's delivery node subdomain (node1.peer-dCDNn-1.dCDNn.net). dCDNn's DNS resolver returns the IP address of the delivery node.
9. The end-user requests the content from dCDNn's delivery node. In the case of a cache hit, steps 10 ~ 14 below do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDN from either parent dCDN or uCDN (not the CSP). The distinguished CDN-domain peer-dCDNn-1.dCDNn.net indicates that this content is to be acquired from dCDNn-1; stripping the CDN-domain reveals the original CDN-domain cdn.csp.com and dCDNn may verify that this CDN-domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an inter-CDN acquisition CDN-domain as agreed above (in this case, dCDNn-acq.dCDNn-1.net). This process repeats recursively until it finds a CDN provider that can serve the requested content.
10. dCDNn-1's DNS resolver processes the DNS request and returns the IP address of a request router in dCDNn-1.

11. The request router for dCDNn-1 processes the HTTP request from dCDNn's delivery node. dCDNn-1 request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (dCDNn-acq.dCDNn-1.net). It also performs loop prevention process as described in step 4 based on the provided CDN-Provider-ID (e.g., uCDN-Provider-ID=100:0 & dCDN1-Provider-ID=200:1 & ... & dCDNn-Provider-ID=1000:0 & MaxNumRedHops=1). Depending on the number of levels of redirection and availability of contents, the same process repeats until either content serving CDN provider is found or MaxNumRedHops expires.
12. Assuming that all intermediate dCDNs also have a cache miss, The request router for uCDN selects a suitable delivery node to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the uCDN's distinguished inter-CDN acquisition domain that points to the selected delivery node.
13. uCDN DNS resolver processes the DNS request and returns the IP address of the delivery node in uCDN.
14. uCDN serves content for the requested CDN-domain to dCDN and finally to end-user. Although not shown, it is at this point that uCDN processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

2.2.2. DNS-based Redirection

In this section, we describe an iterative procedure of DNS-based request routing redirection with loop prevention.

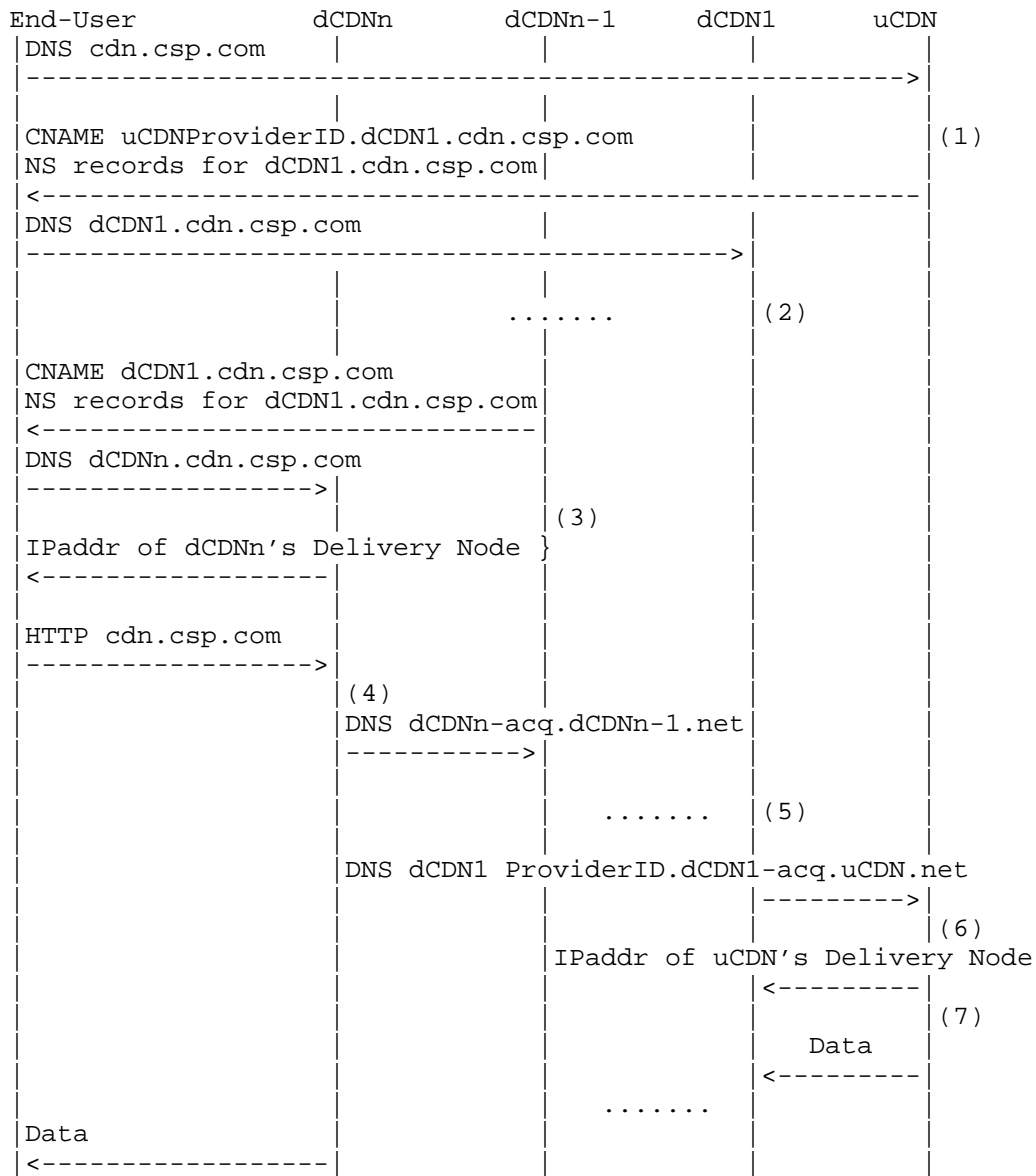


Figure 2: DNS-based request routing redirection iterative procedure

The steps illustrated in the figure are as follows:

1. Request Router for uCDN provider processes the DNS request for CDN- domain `cdn.csp.com` and recognizes that the end-user is best

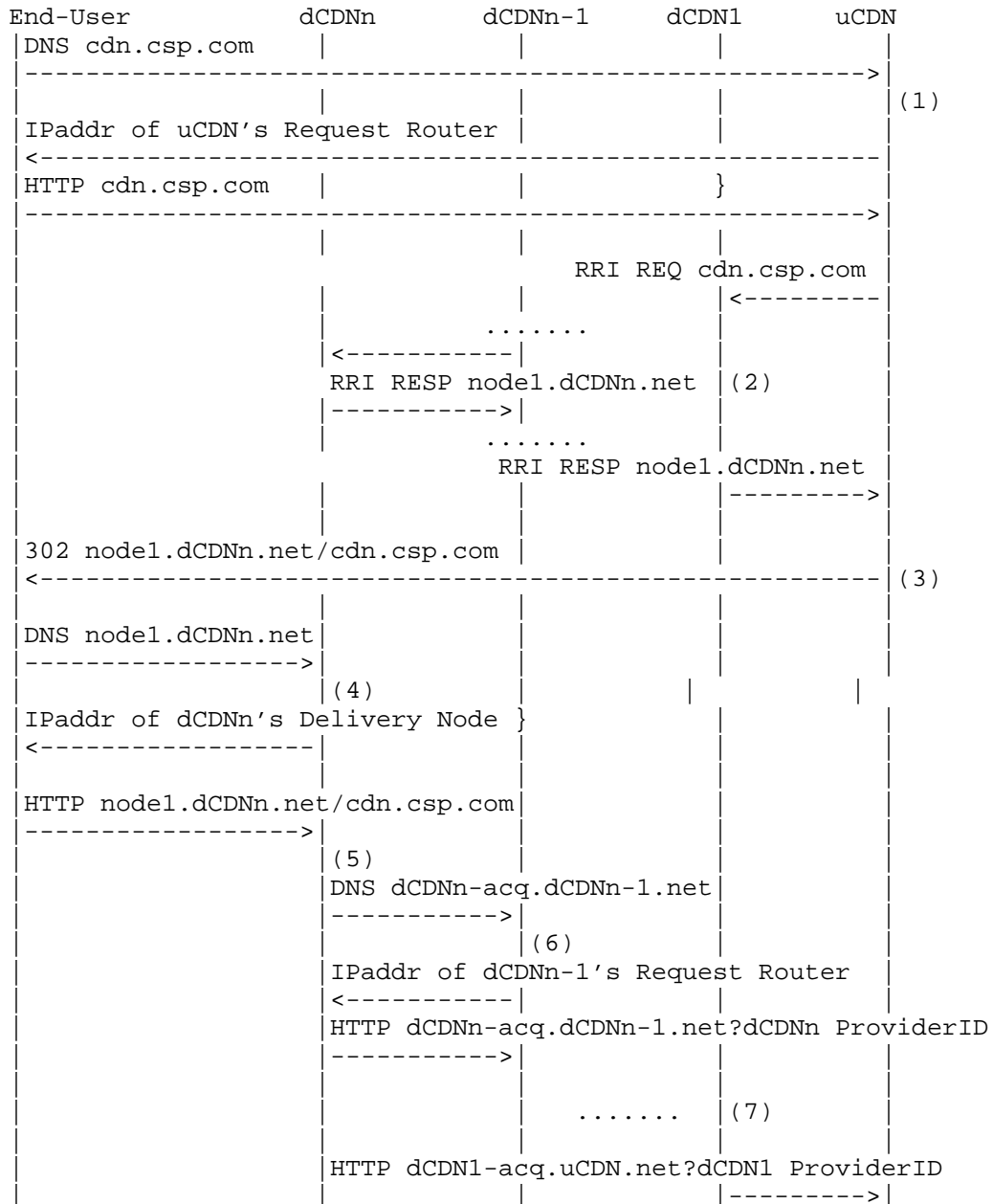
served by another CDN. (This may depend on the IP address of the user's local DNS resolver, or other information discussed below.) The Request Router returns a DNS CNAME response by "stacking" the distinguished identifier for dCDN1 and uCDN's CDN-Provider-ID (e.g., 100:0.10) onto the original CDN-domain (e.g., dCDN1.cdn.csp.com), plus an NS record that maps dCDN1.cdn.csp.com to dCDN1's Request Router.

2. The end-user does a DNS lookup using the modified CDN-domain (i.e., dCDN1.cdn.csp.com). dCDN1 Request Router processes the request and decides to serve the request or redirect further to another CDN provider. It also checks redirection loop. This process iterates until either serving dCDN is selected or MaxNumRedHops expires. In this case, dCDNn is selected as a serving dCDN. If the former occurs, it proceeds to step 3. If the latter occurs, it follows the processes described in the section 3.
3. The end-user does a DNS lookup using the modified CDN-domain (i.e., dCDN1.cdn.csp.com). This causes dCDNn's request router returns an IP address of a suitable delivery node.
4. The end-user requests the content from dCDNn's delivery node. In the case of a cache hit, steps 5 ~ 7 do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDNn from either parent dCDN or uCDN (not the CSP). It also performs loop prevention process as described in the step 2 based on the provided CDN-Provider-ID (e.g., 100:0.200:1.....900:0.1)
5. Depending on the number of levels of redirection and availability of contents, the same process repeats until either content serving CDN provider is found or MaxNumRedHops expires.
6. Assuming that all intermediate dCDNs also miss cache, uCDN is selected as a content delivery CDN provider. Thus, the request router for uCDN selects a suitable delivery node to serve the inter-CDN acquisition request and returns IP address of the suitable uCDN delivery node.
7. uCDN serves content to dCDN1 and further down to end-user. Although not shown, it is at this point that uCDN processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server.

2.3. Recursive Redirection Procedures

2.3.1. HTTP-based Redirection

In this section, we describe an recursive procedure of HTTP-based request routing redirection with loop prevention.



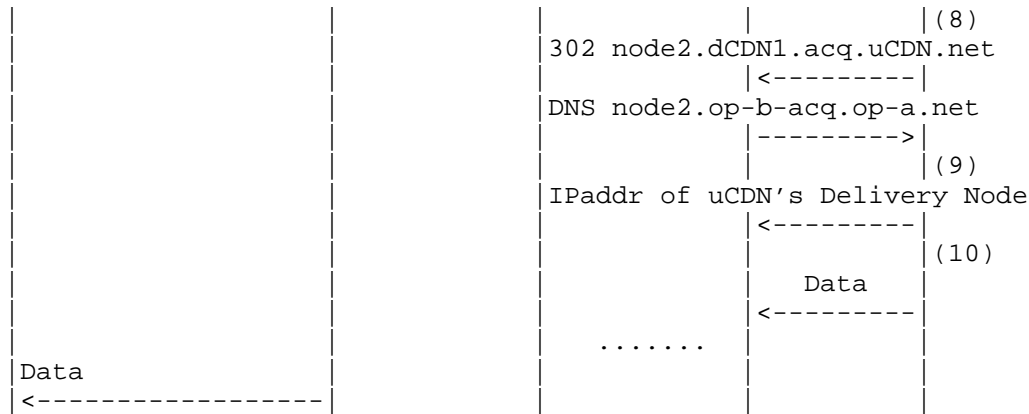


Figure 3: HTTP-based request routing redirection recursive procedure

The steps illustrated in the figure are as follows:

1. A DNS resolver for uCDN provider processes the DNS request for its customer based on CDN-domain `cdn.csp.com`. It returns the IP address of a request router in uCDN provider.
2. A Request Router for uCDN provider processes the HTTP request and recognizes that the end-user is best served by dCDN1. So it queries the CDNI Request Routing interface of dCDN1 providing a set of information about the request including the URL requested. It also provides uCDN's CDN-Provider-ID (e.g., uCDN-Provider-ID=100:0 & MaxNumRedHops=10) for loop prevention process. It contains a list of CDN providers that have requested redirections so far. If either it contains its own CDN provider name or MaxNumRedHops becomes 0, it means that the redirection loop has occurred or it has reached the maximum number of allowed number of redirection hops. Once loop is detected, details on the next steps are described in the section 3. If it is loop free, dCDN1 then either replies with the DNS name of a delivery node or redirect to another dCDN. Such cascading redirection can continue until a serving dCDN is decided. The RRI RESP can be sent in the reverse order of cascaded redirection or directly to the redirection origin CDN provider if contact information is known. The contact information can be embedded in the RRI REQ message or pre-configured during bootstrapping process. The default behavior is recursive RESP.
3. uCDN returns a 302 redirect message for a new URL obtained from

the Request Routing Interface.

4. The end-user does a DNS lookup using the host name of the URL just provided (node1.dCDNn.net). dCDNn's DNS resolver returns the IP address of the corresponding delivery node. Note that, since the name of the delivery node was already obtained from dCDNn using the CDNI Request Routing Interface, there should not be any further redirection here (in contrast to the iterative method described above.)
4. The request router for dCDN1 processes the HTTP request. There are two options: redirect further to another dCDN (i.e., cascading the request) or process it by itself. In either cases, it performs loop prevention step first. If it is loop free, it either redirect further or processes based on the local policy. For the former, it selects another dCDN provider and sends HTTP redirect message with its own CDN-Provider-ID (e.g., uCDN-Provider-ID=100:0 & dCDN1-Provider-ID=200:1 & MaxNumRedHops=9) attached. For the latter, it selects a suitable delivery node to serve the end-user request, and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the dCDN1's distinguished CDN-domain that points to the selected delivery node. Then it goes to step 6.
5. The end-user requests the content from dCDNn's delivery node. In the case of a cache hit, steps 6 ~ 10 below do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDN from either parent dCDN or uCDN (not the CSP). The distinguished CDN-domain dCDNn.net indicates that this content is to be acquired from dCDNn-1; stripping the CDN-domain reveals the original CDN-domain cdn.csp.com and dCDNn may verify that this CDN-domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an inter-CDN acquisition CDN-domain as agreed above (in this case, dCDNn-acq.dCDNn-1.net). This process repeats recursively until it finds CDN provider that can serve the requested content.
6. dCDNn-1's DNS resolver processes the DNS request and returns the IP address of a request router in dCDNn-1.
7. The request router for dCDNn-1 processes the HTTP request from dCDNn's delivery node. dCDNn-1 request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain

(dCDNn-acq.dCDNn-1.net). It also performs loop prevention process as described in the step 2 based on the provided CDN-Provider-ID (e.g., uCDN-Provider-ID=100:0 & dCDN1-Provider-ID=200:1 & ... & dCDNn-1-Provider-ID=900:0 & MaxNumRedHops=1). Depending on the number of levels of redirection and availability of contents, the same process repeats until either content serving CDN provider is found or MaxNumRedHops expires.

8. Assuming that all intermediate dCDNs also have a cache miss, The request router for uCDN selects a suitable delivery node to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the uCDN's distinguished inter-CDN acquisition domain that points to the selected delivery node.
9. uCDN DNS resolver processes the DNS request and returns the IP address of the delivery node in uCDN.
10. uCDN serves content for the requested CDN-domain to dCDN and finally to end-user. Although not shown, it is at this point that uCDN processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

2.3.2. DNS-based Redirection

In this section, we describe an recursive procedure of DNS-based request routing redirection with loop prevention.

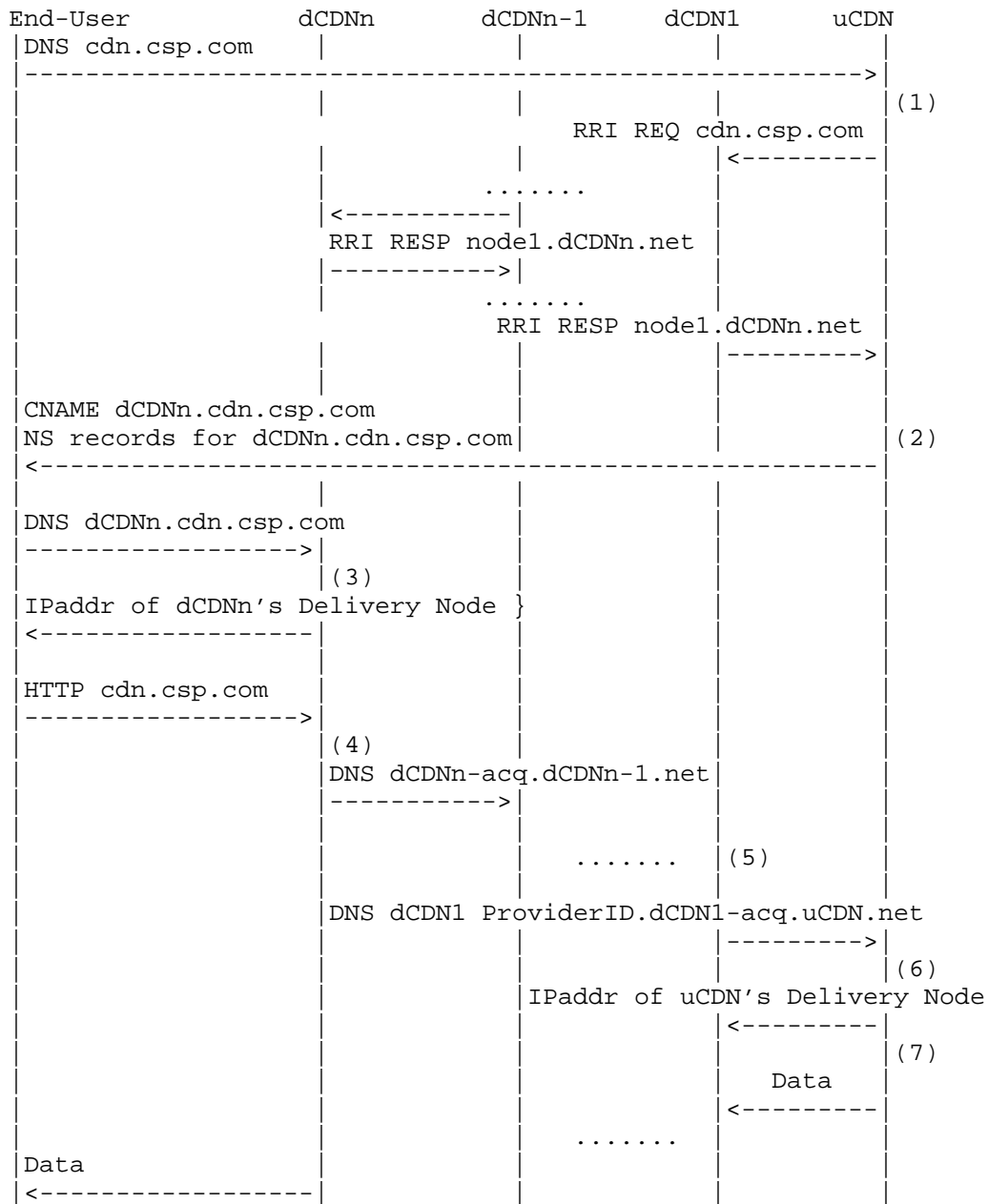


Figure 4: DNS-based request routing redirection recursive procedure

The steps illustrated in the figure are as follows:

1. Request Router for uCDN provider processes the DNS request for CDN-domain `cdn.csp.com` and recognizes that the end-user is best served by dCDN1. So it queries the CDNI Request Routing Interface of dCDN1 providing a set of information about the request including the URL requested. It also provides uCDN's CDN-Provider-ID (e.g., `uCDN-Provider-ID=100:0` & `MaxNumRedHops=10`) for loop prevention process. It checks CDN-Provider-ID. If either it contains own CDN provider name or `MaxNumRedHops` becomes 0, it means that the redirection loop is occurred. Once loop is detected, details on the next steps are described in the section 3. If it is loop-free, dCDN1 then either replies with the DNS name of a delivery node or redirect to another dCDN. Such cascaded redirection can continue until a serving dCDN is decided. The RRI RESP can be sent in the reverse order of cascaded redirection or directly to the redirection origin CDN provider if contact information is known. The contact information can be embedded in the RRI REQ message or pre-configured during bootstrapping process. The default behavior is recursive RESP. In this case, dCDNn is selected as a serving dCDN.
2. The Request Router returns a DNS CNAME response by "stacking" the distinguished identifier for dCDNn and uCDN's CDN-Provider-ID onto the original CDN-domain (e.g., `dCDN1.cdn.csp.com`), plus an NS record that maps `dCDN1.cdn.csp.com` to dCDN1's Request Router.
3. The end-user does a DNS lookup using the modified CDN-domain (i.e., `dCDN1.cdn.csp.com`). This causes dCDNn's request router returns an IP address of a suitable delivery node.
4. The end-user requests the content from dCDNn's delivery node. In the case of a cache hit, steps 5 ~ 7 do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDNn from either parent dCDN or uCDN (not the CSP). It also performs loop prevention process as described in the step 2 based on the provided CDN-Provider-ID (e.g., `uCDN-Provider-ID=100:0` & `dCDN1-Provider-ID=200:1` & ... & `dCDNn-1-Provider-ID=900:1` & `MaxNumRedHops=1`).
5. Depending on the number of levels of redirection and availability of contents, the same process repeats until either content serving CDN provider is found or `MaxNumRedHops` expires.
6. Assuming that all intermediate dCDNs also miss cache, uCDN is selected as a content delivery CDN provider. Thus, the request router for uCDN selects a suitable delivery node to serve the inter-CDN acquisition request and returns IP address of the suitable uCDN delivery node.

7. uCDN serves content to dCDN1 and further down to end-user. Although not shown, it is at this point that uCDN processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server.

3. Redirection Loop Prevention

According to the CDNi generic and request routing interface requirements, the CDNi solution shall support loop prevention of any CDN request routing redirection and subsequently allowing the request routing redirection. To meet such requirements, this section describes request routing redirection loop prevention mechanisms. Loop prevention mechanism should support both detection of the loop and post processing after loop detection. Also loop prevention should be applicable for the process of redirection CDN provider selection and inter CDN content acquisition.

This document specifies loop prevention mechanisms based on CDN-Provider-ID. Framework document [I-D.ietf-cdni-framework] recommends using distinguished acquisition domain for loop detection. It has some drawbacks such as overheads caused by length and processing time of domain name stacking in the case of cascading redirection. This document defines more general solution which can be applicable in both HTTP-based and DNS-based redirections. CDN-Provider-ID which is described in details in the section 2.1 is our proposed solution. Unlike distinguished domain name which is proposed in the framework, it is simple, unique, and efficient.

Post-processing after loop detection is also as important as its detection. The most strict option is to deny the service when the loop is detected. This option should be the last resort when other alternatives are not available. The simplest choice is the CDN provider which detected the loop provide the service by itself although the service quality may not be optimal. If it is not possible, it requests its parent. If the parent can provide the service, it does so. If not, it further checks with other dCDN providers at the peer level or downstream until it finds available one. If it all fails its attempts at the its parent, peer, and children, it checks with uCDN. If note is available, it finally rejects the service.

4. Redirection Operational Considerations

For efficient request routing redirection, various operational considerations need to be addressed. In the framework document, for the redirection selection criteria, CDNi uses end-user's IP address prefix. However, in real CDN service environments, there are various

other reasons such as service availability, QoS requirements, resource faults, etc. Both Routing Request Interface and redirection request should allow exchange of such information. The type and details of information that can be exchanged among CDN providers for the efficient redirection has to be considered together with footprint/capability advertisement. The details are for further study.

Performance feasibility of request routing redirection and loop prevention should be addressed. The requirements may vary depending on the CDN service types (e.g., CDN for small and/or large object). Also granularity of redirection within or between contents should be considered. It is for further study, too.

5 Security Considerations

6 IANA Considerations

7 References

7.1 Normative References

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[I-D.davie-cdni-framework] Peterson, L. and B. Davie, "Framework for CDN Interconnection", April 2012.

[I-D.ietf-cdni-requirements] Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", December 2011.

7.2 Informative References

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

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

Authors' Addresses

Taesang Choi
ETRI
161 Gajong-Dong
Yusong-Gu, Daejeon
Republic of Korea

Phone: +82-42-860-5628
Email: choits@etri.re.kr

Young-IL Seo
KT Network R&D Laboratory

463-1, Jeonmoin-dong,
Yuseong-gu, Daejeon
Republic of Korea

Phone: 82-10-3235-0135
Email: yohan.seo@kt.com

Dong-Ju Kim
KT Network R&D Laboratory
463-1, Jeonmoin-dong,
Yuseong-gu, Daejeon
Republic of Korea

Phone: 82-10-2686-9605
Email: dj.kim@kt.com

Jongmin Lee
SK Telecom
11, Euljiro-2ga
Jung-gu, Seoul
Republic of Korea

Phone: 82-10-9429-6260
Email: jminlee@sk.com

Ja-Ryeong Koo
LG U plus Corporation
Namdaemunro 5-ga
Jung-gu, Seoul
Republic of Korea

Phone: 82-10-8080-6115
Email: wjbkoo@lguplus.co.kr

John Dongho Shinn
Solbox Inc.
7F, Haesung Bldg. 747-2 Yeoksam-Dong
Kangnam-Gu, Seoul
Republic of Korea

Phone: +82-10-3005-4785
Email: eastsky@solbox.com

Content Delivery Networks
Interconnection Working Group
Internet-Draft
Intended status: Informational
Expires: March 31, 2013

J. Famaey
S. Latre
UGent - IBBT
September 27, 2012

Experiments on HTTP Adaptive Streaming over interconnected Content
Delivery Networks
draft-famaey-cdni-has-experiments-00

Abstract

This document reports experimental results on the delivery of HTTP Adaptive Streaming (HAS) content over interconnected Content Delivery Networks (CDNs). Specifically, the implications of CDN request routing between CDNs and HTTP redirection on the quality of delivered HAS content are investigated.

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 March 31, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Experimental Setup	3
3. Results	6
4. Conclusion	8
5. Security Considerations	9
6. References	9
6.1. Normative References	9
6.2. Informative References	9
Authors' Addresses	9

1. Introduction

HTTP Adaptive Streaming (HAS) refers to a set of novel streaming approaches, which deliver streaming media content over the HTTP protocol. The content is split into chunks, offered in several quality layers. This allows the client to dynamically adapt the requested quality, based on available network resources and device capabilities. Delivering HAS content across multiple interconnected CDNs introduces some new opportunities and challenges. Specifically, it becomes possible to distribute the chunks of a single HAS content stream across multiple CDNs, based on chunk popularity, Quality of Service requirements, resource availability, economic or other factors.

Every HAS content stream is accompanied by a Manifest File, which lists the chunks of each quality layer and specifies their location in the form of a URL. As stated in [I-D.brandenburg-cdni-has], several alternative methods exist for specifying chunk locations:

- o Relative URLs: The URLs specified in the Manifest File are relative to the Manifest File's location and thus all located on the same surrogate.
- o Absolute URLs with Redirection: The Manifest File specifies the fully qualified URL of each chunk. These URLs, however, direct the client towards the CDN's request routing node, which in turn uses HTTP redirection to send the client to the surrogate hosting the actual chunk.
- o Absolute URLs without Redirection: The URL points directly to the surrogate hosting the chunk, effectively allowing the client to circumvent the CDN request routing process.

This document aims to evaluate and compare different request routing policies for HAS content, derived from these addressing mechanisms, that can be used in CDN-interconnection scenarios.

2. Experimental Setup

The scenario used as a basis for the experiments consists of two interconnected CDNs. The downstream CDN is located close to the end-user (e.g., a telco CDN), while the upstream CDN is positioned further (e.g., in the core Internet). The upstream CDN is assumed to be the main storage facility of the original content. As such, it hosts the Manifest file but can offload content chunks to one or more downstream CDNs. Figure 1 graphically depicts the scenario and lists the parameters that were varied in the course of the experiments.

The upstream CDN request router, upstream CDN content server, downstream CDN request router and downstream CDN content server are depicted as uRR, uCS, dRR and dCS, respectively. During the experiments, three parameters were varied: the one-way Internet delay D , the per-client bandwidth B and the HAS client buffer size P . The bandwidth on all other network links was set to 100 Mbps, while the one-way network delay was set to 5 ms. The round trip time (RTT) between two nodes can be calculated as the sum of the one-way delays of the links on the path between them, multiplied by two. In the performed experiments, the RTT between the client and the dRR/dCS is 40 ms, as the path connecting them consists of 4 links. The RTT between the client and the uRR/uCS equals $(60+2D)$ ms, as the path between them contains 6 links and the Internet path. Note that the figure presents a high level, simplified view of the network topology and does not show all individual network links and routers. The processing delay on the CDN surrogates is not taken into account, as it is assumed to be negligible compared to the network delay.

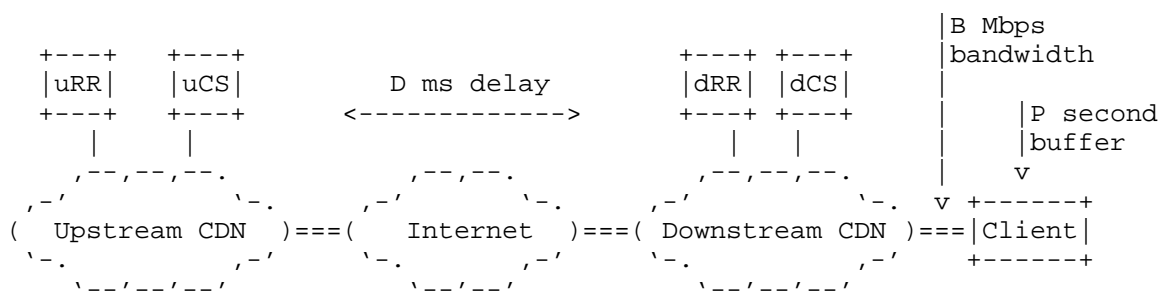


Figure 1: Evaluation scenario and parameters

Three alternative request routing policies are evaluated and compared:

- o UpstreamRR: The Manifest File points to the uRR for every chunk. If the chunk is located within the upstream CDN's network, the uRR sends the client a HTTP redirect request to point it to the correct uCS. Otherwise, the uRR redirects the client to dRR, which in turn redirects it to the correct dCS.
- o DirectRR: The Manifest File immediately points to the correct request router, which redirects the client to the correct content server. This policy thus allows the client to circumvent going via the upstream CDN's network if the chunk is located downstream.
- o DirectCS: The Manifest File immediately points to the correct content server, which allows the client to download segments without being redirected. Compared to the DirectRR policy, the

indirection of contacting the dRR is avoided.

The UpstreamRR policy can be seen as the traditional CDN-I approach, where clients always contact the original CDN and HTTP redirection is used to point them to interconnected CDNs when necessary. It does not require any Manifest File rewriting. Additionally, the upstream CDN does not need any detailed information about chunk locations, as it only needs to redirect clients to the downstream request router. The DirectRR and DirectCS policies are more complex, as they require the upstream CDN to rewrite the original Manifest File. Additionally, when using the DirectCS policy, the downstream CDN either needs to share detailed chunk location information with the upstream CDN or the interconnected CDNs need to collaborate in creating the Manifest File.

The presented policies differ mostly in addressing chunks located in the downstream CDN. As such, the experiments evaluate a scenario where a single client downloads 100 HAS video chunks (of 2 seconds each) from a content server located within the downstream CDN. The constant bitrate video is available in 3 qualities, with bitrates 500 kbps, 1 Mbps and 2 Mbps.

As the end-user Quality of Experience (QoE) depends on several factors, multiple evaluation metrics are used in the comparison:

- o Average played quality: The played quality layer, averaged over all chunks and specified in terms of bitrate. This is expressed in megabits per second (Mbps), representing the bandwidth required for downloading the played quality layers.
- o Total buffer starvation time: The accumulated time during which the client needs to rebuffer the chunks (excluding the original start-up). A rebuffering occurs when a chunk is not available at the client, while it is already required for decoding. This leads to frame freezes, as the client needs to wait for the next chunk to arrive, which significantly reduces QoE.
- o Start-up delay: The time between the initial HTTP request for the first chunk, performed by the client, and the time when the chunk actually starts playing.

All reported results were obtained using the NS-3 simulation environment [ns3] in combination with the Network Simulation Cradle (NSC) [nsc]. NS-3 is a discrete-event network simulator for Internet systems. NSC allows NS-3 to interface directly with the kernel's TCP implementation, generating more accurate and realistic results. The used HAS client rate adaptation algorithm is based on the first version of the client algorithm incorporated in Microsoft's

SmoothStreaming client. The source code of this algorithm can be retrieved from CodePlex [msscodel].

3. Results

This section lists and discusses experimental results on the average played video quality and the start-up delay. Results on buffer starvation are omitted, as they did not occur in the depicted scenarios.

Figure 2 depicts the average played quality (in Mbps) as a function of the client bandwidth B , client buffer size P and one-way Internet delay D . The depicted results show that, as expected, the delivered quality for the DirectRR and DirectCS policies is independent of the network delay D between the Interconnected CDNs. On the other hand, when using the standard UpstreamRR policy, quality of the delivered HAS content degenerates significantly as the delay increases. The exact breakpoint at which quality starts degrading does depend on other factors, of which the available bandwidth is the most important. Specifically, at a bandwidth of 5 Mbps, significant quality reductions are visible for upstream CDN network delays of more than 100 ms, while this breakpoint increases to over 200 ms for a bandwidth of 100 Mbps.

B (Mbps)	P (s)	D (ms)	Average played quality (Mbps)			
			UpstreamRR	DirectRR	DirectCS	
5	6	50	1.94	1.96	1.96	
		100	1.94	1.96	1.96	
		200	0.98	1.96	1.96	
		400	0.50	1.96	1.96	
	24	50	1.93	1.98	1.98	
		100	1.86	1.98	1.98	
		200	0.94	1.98	1.98	
		400	0.50	1.98	1.98	
	100	6	50	1.96	1.98	1.98
			100	1.94	1.98	1.98
200			1.94	1.98	1.98	
400			0.50	1.98	1.98	
24		50	1.96	1.98	1.98	
		100	1.96	1.98	1.98	
		200	1.88	1.98	1.98	
		400	0.50	1.98	1.98	

Figure 2: The average played quality as a function of client bandwidth B, client buffer size P and one-way delay D

Results on start-up delay are depicted in Figure 3. As the results are minimally influenced by the available bandwidth B and client buffer P, start-up delays are only shown for B equal to 5Mbps and P equal to 6s.

D (ms)	Start-up delay (s)		
	UpstreamRR	DirectRR	DirectCS
50	0.90	0.58	0.47
100	1.10	0.58	0.47
200	1.50	0.58	0.47
400	2.30	0.58	0.47

Figure 3: The start-up delay as a function of one-way delay D , for $B = 5\text{Mbps}$ and $P = 6\text{s}$

In line with the played quality, start-up delay is not negatively influenced by an increasing network delay between the interconnected CDNs when using the DirectRR and DirectCS policies. However, when using the UpstreamRR policy, the start-up delay increases linearly with the network delay D . Note that this start-up delay occurs whenever a new stream is requested. This occurs, for example, when switching channels in Internet TV scenarios as well as when users skips parts of a movie in a Video on Demand scenario.

4. Conclusion

In this document, we proposed, evaluated and compared several policies for routing requests and retrieving HAS content chunks distributed across multiple interconnected CDNs. Concretely, the traditional policy, herein called UpstreamRR, in which the original CDN's request router dynamically redirects the end-users towards the CDN currently hosting the requested content, is compared to two novel policies, called DirectRR and DirectCS. These novel policies employ HAS Manifest File rewriting to directly point end-users to the correct CDN (DirectRR) or even the correct content server (DirectCS).

A thorough evaluation, based on NS-3 simulation results, was conducted. It shows that the end-user QoE suffers greatly as a consequence of the HTTP redirects that occur when employing the standard UpstreamRR policy. Depending on the available bandwidth, QoE degradation can start occurring when the one-way network delay towards the upstream CDN is greater than 100 milliseconds. In contrast, the reported results also show that the novel DirectRR and DirectCS policies perform well under increasing network delays.

In summary, these results prove the need for advanced request routing mechanisms, as well as extensive cooperation between interconnected CDNs, to be able to satisfy end-user quality requirements of state-of-the-art HAS-based services.

5. Security Considerations

Not applicable.

6. References

6.1. Normative References

[I-D.brandenburg-cdni-has]
Brandenburg, R., Deventer, O., Faucheur, F., and K. Leung,
"Models for adaptive-streaming-aware CDN Interconnection",
draft-brandenburg-cdni-has-03 (work in progress),
July 2012.

6.2. Informative References

[msscode] "Microsoft's SmoothStreaming rate adaptation algorithm v1
source code", <[https://slextensions.svn.codeplex.com/svn/
trunk/SLExtensions/AdaptiveStreaming/](https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming/)>.

[ns3] "NS-3 discrete event network simulator",
<<http://www.nsnam.org/>>.

[nsc] "Network Simulation Cradle",
<<http://research.wand.net.nz/software/nsc.php>>.

Authors' Addresses

Jeroen Famaey
Ghent University - IBBT
Gaston Crommenlaan 8/201
Ghent 9050
Belgium

Phone: +32 9 331 49 38
Email: jeroen.famaey@intec.ugent.be

Steven Latre
Ghent University - IBBT
Gaston Crommenlaan 8/201
Ghent 9050
Belgium

Phone: +32 9 331 49 88
Email: steven.latre@intec.ugent.be

Content Delivery Networks
Interconnection Working Group
Internet-Draft
Intended status: Informational
Expires: July 28, 2013

J. Famaey
S. Latre
UGent - iMinds
January 24, 2013

Experiments on HTTP Adaptive Streaming over interconnected Content
Delivery Networks
draft-famaey-cdni-has-experiments-01

Abstract

This document reports experimental results on the delivery of HTTP Adaptive Streaming (HAS) content over interconnected Content Delivery Networks (CDNs). Specifically, the implications that CDN request routing between CDNs and HTTP redirection have on the quality of delivered HAS content are investigated.

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 July 28, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Experimental Setup	3
3. Results	6
3.1. Congested Scenario	6
3.2. Uncongested Scenario	9
3.3. Influence of Segment Duration	11
4. Conclusion	13
5. Security Considerations	14
6. References	14
6.1. Normative References	14
6.2. Informative References	14
Authors' Addresses	15

1. Introduction

HTTP Adaptive Streaming (HAS) refers to a set of novel streaming approaches, which deliver streaming media content over the HTTP protocol. The content is split into chunks, offered in several quality layers. This allows the client to dynamically adapt the requested quality, based on available network resources and device capabilities. Delivering HAS content across multiple interconnected CDNs introduces some new opportunities and challenges. Specifically, it becomes possible to distribute the chunks of a single HAS content stream across servers deployed on multiple CDNs, based on chunk popularity, Quality of Service requirements, resource availability, costs or other factors.

Every HAS content stream is accompanied by a Manifest File, which lists the chunks of each quality layer and specifies their location in the form of a URL. As stated in [I-D.brandenburg-cdni-has], several alternative methods exist for specifying chunk locations:

- o Relative URLs: The URLs specified in the Manifest File are relative to the Manifest File's location and thus all located on the same surrogate.
- o Absolute URLs with Redirection: The Manifest File specifies the fully qualified URL of each chunk. These URLs, however, direct the client towards the CDN's request routing node, which in turn uses HTTP redirection to send the client to the surrogate hosting the actual chunk.
- o Absolute URLs without Redirection: The URL points directly to the surrogate hosting the chunk, effectively allowing the client to circumvent the CDN request routing process.

This document aims to evaluate and compare different request routing policies for HAS content, derived from these addressing mechanisms, that can be used in federated CDN scenarios.

2. Experimental Setup

The scenario used as a basis for the experiments consists of two interconnected CDNs. The downstream CDN is located close to the end-user (e.g., a telco CDN), while the upstream CDN is positioned further (e.g., in the core Internet). The upstream CDN is assumed to be the main storage facility of the original content. As such, it hosts the Manifest file but can offload content chunks to one or more downstream CDNs. Figure 1 graphically depicts the scenario and lists the parameters that were varied in the course of the experiments.

The upstream CDN request router, upstream CDN content server, downstream CDN request router and downstream CDN content server are depicted as uRR, uCS, dRR and dCS, respectively. During the experiments, five parameters were varied: the one-way Internet delay ID, the one-way downstream CDN delay DD, the per-client bandwidth B, the HAS client buffer size P and the HAS segment duration S. The bandwidth on all other network links was set to 100 Mbps, while the one-way network delay was set to 5 ms. The round trip time (RTT) between two nodes can be calculated as the sum of the one-way delays of the links on the path between them, multiplied by two. In the performed experiments, the client and dRR/dCS are separated by three links, resulting in a RTT of $(2 * 5 + DD) * 2 = (20 + 2 DD)$ ms. On the other hand, the client and uRR/uCS are 5 links apart, resulting in a RTT of $(4 * 5 + ID) * 2 = (40 + 2 ID)$ ms. Note that the figure presents a high level, simplified view of the network topology and does not show all individual network links and routers. Additionally, the processing delay on the CDN surrogates is not taken into account, as it is assumed to be negligible compared to the network delay.

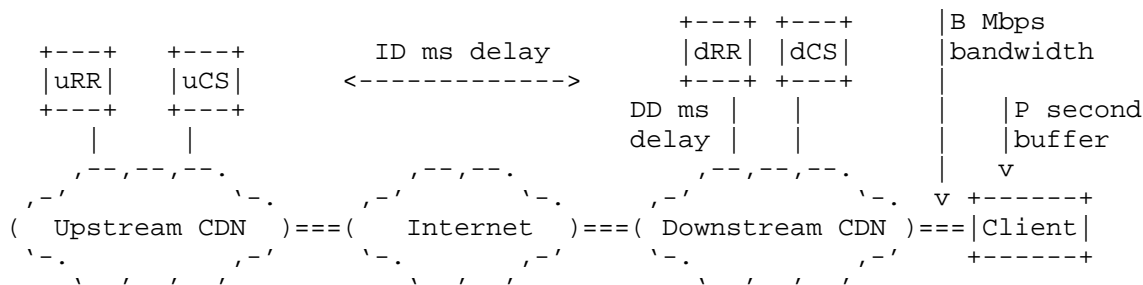


Figure 1: Evaluation scenario and parameters

Three alternative request routing policies are evaluated and compared:

- o UpstreamRR: The Manifest File points to the uRR for every chunk. If the chunk is located within the upstream CDN's network, the uRR sends the client a HTTP redirect request to point it to the correct uCS. Otherwise, the uRR redirects the client to dRR, which in turn redirects it to the correct dCS.
- o DirectRR: The Manifest File immediately points to the correct request router, which redirects the client to the correct content server. This policy thus allows the client to circumvent going via the upstream CDN's network if the chunk is located downstream.

- o DirectCS: The Manifest File immediately points to the correct content server, which allows the client to download segments without being redirected. Compared to the DirectRR policy, the indirection of contacting the dRR is avoided.

The UpstreamRR policy can be seen as the traditional CDN-I approach, where clients always contact the original CDN and HTTP redirection is used to point them to interconnected CDNs when necessary. It does not require any Manifest File rewriting. Additionally, the upstream CDN does not need any detailed information about chunk locations, as it only needs to redirect clients to the downstream request router. The DirectRR and DirectCS policies are more complex, as they require the upstream CDN to rewrite the original Manifest File. Additionally, when using the DirectCS policy, the downstream CDN either needs to share detailed chunk location information with the upstream CDN or the interconnected CDNs need to collaborate in creating the Manifest File.

The experiments evaluate a scenario where a single client downloads a 200 second video clip (split into 200/S segments). The first half is hosted by the downstream CDN, while the second half is hosted by the upstream CDN. The constant bitrate (CBR) video is available in 3 qualities, with bitrates 500 kbps, 1 Mbps and 2 Mbps respectively.

As the end-user Quality of Experience (QoE) depends on several factors, multiple evaluation metrics are used in the comparison:

- o Average played quality: The played quality layer, averaged over all chunks and specified in terms of bitrate. This is expressed in megabits per second (Mbps), representing the bandwidth required for downloading the played quality layers.
- o Total buffer starvation time: The accumulated time during which the client needs to rebuffer the chunks (excluding the original start-up). A rebuffering occurs when a chunk is not available at the client, while it is already required for decoding. This leads to frame freezes, as the client needs to wait for the next chunk to arrive, which significantly reduces QoE.
- o Start-up delay: The time between the initial HTTP request for the first chunk, performed by the client, and the time when the chunk actually starts playing.

All reported results were obtained using the NS-3 simulation environment [ns3] in combination with the Network Simulation Cradle (NSC) [nsc]. NS-3 is a discrete-event network simulator for Internet systems. NSC allows NS-3 to interface directly with the kernel's TCP implementation, generating more accurate and realistic results. The

used HAS client rate adaptation algorithm is based on the first version of the client algorithm incorporated in Microsoft's SmoothStreaming client. The source code of this algorithm can be retrieved from CodePlex [msscode].

3. Results

This section lists and discusses experimental results on the average played video quality, total buffer starvation time and the start-up delay. First, the effects of several parameters on the QoE metrics are studied, both in a congested and an uncongested network. Second, the influence of segment duration is evaluated.

3.1. Congested Scenario

The congested scenario considers a client-side bandwidth B of 1Mbps, which, due to protocol overhead allows only the lowest 500kbps quality to be streamed. As such, this section focuses on a comparison of the buffer starvation time and start-up delay. The segment duration S is fixed at 2s. The results on buffer starvation as a function of one-way Internet delay ID , one-way downstream CDN delay DD and client buffer size P are shown in Figure 2. The starvation time is shown separately for the first 50 segments downloaded from dCS (Dws) and the latter 50 downloaded from uCS (Ups). The results on start-up delay are depicted in Figure 3 as a function of delays ID and DD only, as they are unaffected by the buffer size P .

In general, the results depicted in Figure 2 clearly show that minimising the number of HTTP redirects benefits the QoE significantly, both for segments hosted at the downstream as well as the upstream CDN. Specifically, several observations can be made based on the depicted results. First, as expected, DirectCS and DirectRR are not influenced by an increasing Internet delay for downstream segments, as they completely circumvent the upstream CDN in this case. In contrast, when using the traditional UpstreamRR approach buffer starvations start occurring at a one-way Internet delay of as low as 100ms if the downstream CDN delay is 50ms or higher. If the downstream delay is low, then starvations start occurring at a 200ms Internet delay. Second, for upstream segments, DirectRR and DirectCS are also negatively impacted by an increasing Internet delay. However, DirectCS is significantly less influenced by it than DirectRR, as it circumvents DirectRR's redirect from uRR to uCS. Third, increasing the buffer size clearly helps reducing buffer starvations in the upstream scenario for DirectRR and DirectCS. This is due to the fact that the client can fill its buffer when downloading the first 50 segments from dCS. This set of

backup segments subsequently allows the client to temporarily maintain desirable QoE levels under high RTT. When using UpstreamRR, the client cannot fill its buffer during the initial phase, due to the larger number of redirects, and a larger buffer therefore does not improve results.

P (s)	DD (ms)	ID (ms)	Total buffer starvation time (s)					
			UpstreamRR		DirectRR		DirectCS	
			Dws	Ups	Dws	Ups	Dws	Ups
6	5	50	0.0	0.0	0.0	0.0	0.0	0.0
		100	0.0	0.1	0.0	0.0	0.0	0.0
		200	10.3	34.4	0.0	30.4	0.0	10.2
	50	50	0.0	0.0	0.0	0.0	0.0	0.0
		100	5.5	3.8	0.0	0.0	0.0	0.0
		200	18.6	34.5	0.0	30.4	0.0	10.2
24	5	50	0.0	0.0	0.0	0.0	0.0	0.0
		100	0.0	0.0	0.0	0.0	0.0	0.0
		200	10.4	34.4	0.0	12.4	0.0	0.0
	50	50	0.0	0.0	0.0	0.0	0.0	0.0
		100	5.5	3.8	0.0	0.0	0.0	0.0
		200	18.6	34.4	0.0	13.5	0.0	0.0

Figure 2: The total buffer starvation time as a function of client buffer size P and one-way Internet delay ID and one-way downstream CDN delay DD; for B = 1Mbps and S = 2s

ID (ms)	DD (ms)	Start-up delay (s)		
		UpstreamRR	DirectRR	DirectCS
50	5	2.11	1.81	1.72
	50	2.92	2.63	2.37
100	5	2.31	1.81	1.72
	50	3.12	2.63	2.37
200	5	2.71	1.81	1.72
	50	3.52	2.63	2.37

Figure 3: The start-up delay as a function of one-way Internet delay ID and one-way downstream CDN delay DD; for B = 1Mbps, S = 2s and P = 6s

The results in Figure 3 clearly show that the start-up delay is linearly proportional to the delay between the client and server. This explains the two evolutions visible in the table. First, for segments hosted at the downstream CDN, the start-up delay for UpstreamRR increases as a function of the one-way Internet delay ID, while DirectRR and DirectCS are unaffected. Second, the start-up delay for all routing policies increases as a function of the one-way downstream delay DD. Finally, due to the lower redirection delay of DirectCS compared to DirectRR, the DirectCS start-up delay is slightly lower. Note that this start-up delay occurs whenever the buffer needs to be flushed. As such, this not only happens when a client initiates a session, but also for example when switching channels in Internet TV scenarios or skipping to another part of a movie in a Video on Demand scenario.

In summary, it was shown that in congested scenarios, using the DirectRR or DirectCS routing policies can significantly reduce the amount of client-side buffer starvation compared to using the traditional UpstreamRR policy, when downloading HAS segments from the downstream CDN. Additionally, the use of DirectCS is beneficial in terms of buffer starvations compared to DirectRR and UpstreamRR when downloading segments from the upstream CDN. Although a larger buffer size was shown to help in temporarily overcoming the negative effects of high network latency, this only helps if a client can first fill its buffer by downloading segments with low latency. Finally, it was shown that the start-up delay is linearly proportional to the total

RTT, both caused by network latency to the content server, as well as redirection delay. As such, the DirectRR and DirectCS start-up delay is unaffected by the Internet delay when streaming from the downstream CDN, while that of UpstreamRR is not. Moreover, DirectCS has a lower start-up delay than DirectRR.

3.2. Uncongested Scenario

The uncongested scenario considers a client-side bandwidth B of 5Mbps, which is sufficient to download the highest 2Mbps quality layer stream. As such, this section does consider the delivered video quality. As buffer starvation and start-up delay results were already discussed in much detail in the previous section, and they show similar trends in the uncongested scenario, they are omitted here. The segment duration S is once again fixed at 2s. The results on average played video quality as a function of one-way Internet delay ID , one-way downstream CDN delay DD and client buffer size P are shown in Figure 4. The quality is shown separately for the first 50 segments downloaded from dCS (Dws) and the latter 50 downloaded from uCS (Ups).

The results in Figure 4 prove that an increased number of redirections significantly impacts video quality, even for a relatively low RTT. Specifically, the results show that UpstreamRR achieves a significantly lower quality than DirectRR and DirectCS for segments hosted at the downstream CDN for all depicted parameter combinations. Additionally, as expected, the delivered video quality when using UpstreamRR is inversely proportional to the Internet delay ID . In contrast, DirectRR and DirectCS are unaffected. In addition to the quality difference for segments hosted at the downstream CDN, there also are some remarkable differences for upstream CDN segments. Although UpstreamRR and DirectRR exhibit the same behaviour when downloading segments from the upstream CDN, they do show a difference in video quality. This is due to suboptimal decision making by the MSS client algorithm when switching servers. The UpstreamRR policy often results in a lower quality being requested for the downstream segments. However, when switching to the upstream CDN, the algorithm might not always decide to increase the quality, even if this would in theory be possible. This behaviour is caused by the way clients estimate the available throughput, which does not take into account multiple servers. In contrast, when using DirectRR the client is already downloading a higher quality from the dCDN, and will not change this when switching to the uCDN. Consequently, suboptimal decisions of the client algorithm, as a consequence of server switching, can lead to unexpected differences between UpstreamRR and DirectRR. Finally, the results show that increasing the buffer size leads to a higher delivered video quality in almost all cases.

P (s)	DD (ms)	ID (ms)	Average played quality (Mbps)					
			UpstreamRR		DirectRR		DirectCS	
			Dws	Ups	Dws	Ups	Dws	Ups
6	5	50	0.50	0.50	1.93	1.14	1.95	1.27
		100	0.50	0.50	1.93	1.02	1.95	1.03
		200	0.50	0.50	1.93	0.53	1.95	0.54
	50	50	0.50	0.50	0.97	1.00	0.98	1.00
		100	0.50	0.50	0.97	0.51	0.98	0.52
		200	0.50	0.50	0.97	0.51	0.98	0.52
24	5	50	1.64	2.00	1.93	2.00	1.95	2.00
		100	0.85	1.00	1.93	1.04	1.95	0.98
		200	0.50	0.50	1.93	0.69	1.95	0.66
	50	50	0.50	0.50	1.38	2.00	1.40	2.00
		100	0.50	0.50	1.38	1.01	1.40	0.98
		200	0.50	0.50	1.38	0.65	1.40	0.66

Figure 4: The average played quality as a function of client buffer size P, one-way Internet delay ID and one-way downstream CDN delay DD; for B = 5Mbps and S = 2s

In summary, the merits of DirectRR and DirectCS compared to UpstreamRR in uncongested networks were clearly shown. In addition to a reduction in buffer starvations and start-up delay, the DirectRR and DirectCS policies also result in an increased delivered video quality when enough bandwidth is available. Specifically, when using UpstreamRR and streaming content from the downstream CDN, the video quality is significantly impaired by an increase in Internet delay, while DirectRR and DirectCS are unaffected. Additionally, due to the fact that HAS client algorithms are unoptimised for delivery of a single stream from multiple servers, UpstreamRR additionally suffers a reduction in video quality compared to DirectRR for segments served from the upstream CDN in some scenarios.

3.3. Influence of Segment Duration

Previous sections only considered a short segment duration S of 2s. Although this value is widely used, by for example Microsoft SmoothStreaming-based services, others, such as Apple HTTP Live Streaming, generally recommend longer segment durations. This section evaluates the effect of segment duration S on the average played quality as well as the start-up delay in an uncongested scenario with a client-side bandwidth B of 5Mbps. As results showed that the used HAS algorithm performs poorly if the buffer fits only two segments or less and a segment duration of up to 12s is considered, a buffer size P of 36s is used. The results on average played quality as a function of segment duration S , one-way Internet delay ID and one-way downstream CDN delay DD are shown in Figure 5. The quality is shown separately for the first 50 segments downloaded from dCS (Dws) and the latter 50 downloaded from uCS (Ups). The results on start-up delay are depicted in Figure 6 as a function of S , ID and DD .

The results depicted in Figure 5 clearly prove that increasing the segment duration greatly improves performance of the three routing policies for large delays. If both ID and DD are large enough, it evens out performance of the three policies completely, removing the negative effects of redirects. This is obviously due to the fact that increasing the segment duration, decreases the number of requests and thus the relative delay introduced by redirects. On the other hand, longer segment durations result in lower average quality when the delay is very low (i.e., $ID = 50\text{ms}$, $DD = 5\text{ms}$). This is because increasing the segment duration results in a proportional increase in convergence time to the optimal video quality.

S (s)	DD (ms)	ID (ms)	Average played quality (Mbps)					
			UpstreamRR		DirectRR		DirectCS	
			Dws	Ups	Dws	Ups	Dws	Ups
2	5	50	1.95	2.00	1.93	2.00	1.95	2.00
		100	1.59	1.46	1.93	1.46	1.95	1.16
		200	1.15	0.75	1.93	0.74	1.95	0.72
	50	50	1.43	2.00	0.96	1.00	1.16	2.00
		100	0.81	1.00	0.96	1.00	1.16	1.16
		200	0.50	0.50	0.96	0.70	1.16	0.72
12	5	50	1.83	2.00	1.83	2.00	1.83	2.00
		100	1.72	2.00	1.83	2.00	1.83	2.00
		200	1.72	2.00	1.83	2.00	1.83	2.00
	50	50	1.61	2.00	1.61	2.00	1.61	2.00
		100	1.61	2.00	1.61	2.00	1.61	2.00
		200	1.61	2.00	1.61	2.00	1.61	2.00

Figure 5: The average played quality as a function of segment duration S, one-way Internet delay ID and one-way downstream CDN delay DD; for B = 5Mbps and P = 36s

Although using longer segment durations is an effective way to increase the video quality in face of redirects with high latency, it also has several disadvantages. As shown in Figure 6 the start-up delay increases significantly as a function of the segment duration. This is the case for all routing policies. Additionally, long segment durations are usually a poor choice in combination with live services, as they lead to significant lag of the streaming session compared to the live time.

S (s)	ID (ms)	DD (ms)	Start-up delay (s)		
			UpstreamRR	DirectRR	DirectCS
2	50	5	0.77	0.48	0.40
		50	1.56	1.26	1.00
	200	5	1.38	0.48	0.40
		50	2.16	1.26	1.00
12	50	5	1.81	1.51	1.43
		50	3.08	2.80	2.54
	200	5	2.41	1.51	1.43
		50	3.68	2.80	2.54

Figure 6: The start-up delay as a function of segment duration S, one-way Internet delay ID and one-way downstream CDN delay DD; for B = 5Mbps and P = 36s

In summary, results showed that increasing the HAS segment duration can help in overcoming the reduced video quality that occurs when using simple Inter-CDN routing policies, such as UpstreamRR. Nevertheless, long segment durations also have significant disadvantages, such as slower convergence to the optimal quality, an increased start-up delay and greater session lag in live scenarios. This makes them unsuitable in many use-cases, such as live or channel-switching intensive services.

4. Conclusion

In this document, we proposed, evaluated and compared several policies for routing requests and retrieving HAS content chunks distributed across multiple interconnected CDNs. Concretely, the traditional policy, herein called UpstreamRR, in which the original CDN's request router dynamically redirects the end-users towards the CDN currently hosting the requested content, is compared to two novel policies, called DirectRR and DirectCS. These novel policies employ HAS Manifest File rewriting to directly point end-users to the correct CDN (DirectRR) or even the correct content server (DirectCS).

A thorough evaluation, using an open source implementation of the Microsoft Smooth Streaming client algorithm and based on NS-3 simulation results, was conducted. It shows that the end-user QoE suffers greatly as a consequence of the HTTP redirects that occur when employing the standard UpstreamRR policy. Specifically, it was shown that when downloading segments from the downstream CDN, DirectRR and DirectCS result in a much lower buffer starvation rate and start-up delay, as well as an increased video quality compared to UpstreamRR. Additionally, DirectCS significantly outperforms the other two strategies in terms of buffer starvation rate and start-up delay for segments downloaded from the upstream CDN. Finally, the evaluation showed that increasing the segment duration can negate the negative effects of redirects on video quality when using the UpstreamRR policy. However, it also leads to increased start-up delay and slower convergence to the optimal quality.

In summary, these results prove the need for advanced request routing mechanisms, as well as extensive cooperation between interconnected CDNs, to be able to satisfy end-user quality requirements of state-of-the-art HAS-based services. Additionally, the results show the merits of the more complex DirectCS policy compared to the easier to implement DirectRR.

5. Security Considerations

Not applicable.

6. References

6.1. Normative References

- [I-D.brandenburg-cdni-has]
Brandenburg, R., Deventer, O., Faucheur, F., and K. Leung,
"Models for adaptive-streaming-aware CDN Interconnection",
draft-brandenburg-cdni-has-03 (work in progress),
July 2012.

6.2. Informative References

- [msscode] "Microsoft's SmoothStreaming rate adaptation algorithm v1
source code", <[https://slextensions.svn.codeplex.com/svn/
trunk/SLExtensions/AdaptiveStreaming/](https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming/)>.
- [ns3] "NS-3 discrete event network simulator",
<<http://www.nsnam.org/>>.

[nsc] "Network Simulation Cradle",
 <<http://research.wand.net.nz/software/nsc.php>>.

Authors' Addresses

Jeroen Famaey
Ghent University - iMinds
Gaston Crommenlaan 8/201
Ghent 9050
Belgium

Phone: +32 9 331 49 38
Email: jeroen.famaey@intec.ugent.be

Steven Latre
Ghent University - iMinds
Gaston Crommenlaan 8/201
Ghent 9050
Belgium

Phone: +32 9 331 49 88
Email: steven.latre@intec.ugent.be

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2013

Danhua. Wang, Ed.
Huawei Technologies
Xiaoyan. He
Spencer. Dawkins
Huawei
Chen. Ge
China Telecom
Wei. Ni
Yunfei. Zhang
China Mobile
B. Niven-Jenkins
Velocix (Alcatel-Lucent)
October 22, 2012

Routing Request Redirection for CDN Interconnection
draft-he-cdni-routing-request-redirection-03

Abstract

The Request Routing Interface comprises of (1) the asynchronous advertisement of footprint and capabilities by a downstream CDN that allows a upstream CDN to decide whether to redirect particular user requests to that downstream CDN; and (2) the synchronous operation of an upstream CDN requesting whether a downstream CDN is prepared to accept a user request and of a downstream CDN responding with how to actually redirect the user request. This document describes an interface for the latter part, i.e. the CDNI Request Routing Redirection interface.

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 April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Interface function and operation overview	4
3.1. Discussion on protocol type for CDNI RRRI	5
4. Data passed in CDNI request routing redirection requests & responses	6
4.1. Data passed in CDNI RRRI requests & response for DNS redirection	7
4.2. Data passed in CDNI RRRI request & responses for HTTP redirection	8
5. HTTP based RESTful interface for CDNI Request Routing Redirection	9
6. Protocol Specification	11
6.1. Recursive Request Routing	11
6.1.1. Downstream CDN Behavior for DNS based User Agent requests	11
6.1.2. Downstream CDN Behavior for HTTP based User Agent requests	12
6.1.3. Example CDNI RRRI Requests/Responses	12
6.2. Iterative Request Routing	16
7. Security Considerations	16
8. IANA Considerations	17
9. References	17
9.1. Normative References	17
9.2. Informative References	17
Authors' Addresses	17

1. Introduction

A Content Delivery Network (CDN) is a system built on an existing IP network which is used for large scale content delivery, via prefetching or dynamically caching content on its distributed surrogates (caching servers) that are typically deployed close to the end users so that a CDN can improve access to the content it caches, for example, by reducing access latency and improving the end user experience.

In recent years the volume of video and multimedia content delivered over the internet is rapidly increasing. To accommodate this increase, existing CDN providers are scaling up their infrastructure and many Network Service Providers (NSPs) are deploying their own CDNs. Another emerging requirement is CDN Interconnection (CDNI).

Several real world use cases are described in [I-D.ietf-cdni-use-cases] which prove the necessity for CDN interconnection. The most frequently mentioned use case is leveraging the collective CDN footprint of interconnected standalone CDNs to achieve the goal of delivering content to additional distributed end users regardless of their location.

[I-D.ietf-cdni-problem-statement] describes the problem area, where CDNs are interconnected as described in [I-D.ietf-cdni-use-cases] based on the requirements described in [I-D.ietf-cdni-requirements], and using the technology framework described in [I-D.ietf-cdni-framework].

The purpose of this document is to define the interface for synchronous redirection operation of the request routing interface, which is one of the main building blocks of the CDN interconnection architecture described in [I-D.ietf-cdni-requirements].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document reuses the terminology defined in [I-D.ietf-cdni-problem-statement]. The term "Distinguished CDN Domain" defined in [I-D.ietf-cdni-framework] is also reused in this document.

The following additional terms are introduced by this document:

DNS Redirection: The act of using DNS name resolution for the request routing process of a CDN. In DNS Redirection, the DNS resolver of the CDN makes the routing decision based on a local policy and returns the result as the response of a DNS query request to redirect a user agent to a new target. In CDNI, the result may point to a surrogate of the CDN, a request router in a downstream CDN or a surrogate in a downstream CDN, etc.

HTTP Redirection: The act of using an HTTP redirection response to redirect a user agent to a new target. The new target is the result of the routing decision of a CDN at the time it receives a content request via HTTP. In CDNI, the result may point to a surrogate of the CDN, a request router in a downstream CDN or a surrogate in a downstream CDN etc.

3. Interface function and operation overview

[[Editor's note: How an upstream CDN decides which downstream CDN(s) to query is a local policy within the upstream CDN and is outside the scope of this document.]]

The CDNI Request Routing Redirection interface (CDNI RRRI) is one of the main building blocks required in order to interconnect CDNs. The main function of the Request Routing interface is to allow the Request Routing systems in interconnected CDNs to communicate to facilitate the redirection of User Agent requests between interconnected CDNs.

The detailed requirements for the CDNI Request Routing Redirection interface and their relative priorities of those requirements are described in section 5 of [I-D.ietf-cdni-requirements].

The CDNI Request Routing Redirection interface operates between a pair of interconnected CDNs. To enable communication over the CDNI Request Routing Redirection interface, the two interconnected CDNs need to know the end point (URI) in the other CDN to query. For example, an Upstream CDN needs to know the URI (end point) in a Downstream CDN to send its CDNI Request Routing queries to.

The CDNI Request Routing Redirection interface URI may be statically pre-configured, dynamically discovered via the CDNI control interface, or discovered via other means. However, such discovery mechanisms are not specified in this document, as they are considered out of the scope of the CDNI Request Routing Redirection interface specification.

CDNI solutions must support both of the request routing mechanisms

illustrated in section 2.1 of [I-D.ietf-cdni-framework], namely Iterative Request Routing and Recursive Request Routing. However, the Iterative Request Routing method does not invoke any interaction over the request routing redirection interface across interconnected CDNs. Therefore, the Request Routing Redirection interface is only relevant in the case of Recursive Request Routing and so this document will not discuss Iterative Request Routing further.

In the case of Recursive Request Routing, an Upstream CDN forwards a CDNI request routing request from a user agent to a Downstream CDN for surrogate selection. The initial Upstream CDN->User Agent redirection protocols addressed in this draft are: DNS redirection and HTTP redirection. Moreover, the internal routing mechanisms used between the CDN and the user agent (DNS and HTTP Redirection) of the two interconnected CDNs should also be taken into account as they may affect the type of query request the Upstream CDN send to a Downstream CDN and the information the Downstream CDNs return in its query response.

3.1. Discussion on protocol type for CDNI RRRI

The request routing process between an Upstream CDN and a Downstream CDN has several variants depending on the following factors:

- o Which request routing mechanism is being used by an Upstream CDN to redirect the User Agent: DNS Redirection or HTTP Redirection.
- o Which request routing mechanism is being used by a Downstream CDN to redirect the User Agent: DNS Redirection or HTTP Redirection.

The possible combinations are shown in Table 1.

Case	uCDN Received Request	dCDN Response	Notes
1	DNS	DNS with IP address/hostname of Surrogate	dCDN uses DNS redirection (via the CDNI RRRI interface) to redirect the UA to one or more surrogates in the dCDN.

2	DNS	DNS with IP address/hostname of RR	dCDN uses DNS redirection (via the RRRI interface) to redirect the UA to a RR in the dCDN which then performs a HTTP redirection to redirect the UA to a surrogate in the dCDN.
3	HTTP	HTTP 302 redirection	dCDN uses HTTP redirection mode to redirect the UA to either (a) a RR in the dCDN, or (b) a surrogate in the dCDN.

Table 1. Recursive Routing Cases

4. Data passed in CDNI request routing redirection requests & responses

To assist the routing decision of a Downstream CDN, the Upstream CDN shall convey as much information as possible to the Downstream CDN, e.g. URI of the requested content, the client's location information.

In addition to details of the User Agent's request to the Upstream CDN, the Downstream CDN requires the appropriate CDNI metadata or policies related to the content the User Agent is requesting in order for the Downstream CDN to determine whether it is capable of delivering any requested content in the manner specified by the CDNI Metadata associated with that content. For example, the Downstream CDN needs to verify that the delivery protocol that will be used is a delivery protocol that the Downstream CDN supports.

Similarly, in addition to details of how to redirect the User Agent, the Downstream CDN may wish to return additional policy to the Upstream CDN to help the Upstream CDN with future CDNI RRRI requests. For example the Downstream CDN may wish to return a policy that expresses "this response can be reused without requiring a RRRI request for 60 seconds provided the User Agent's IP address is in the range 192.0.2.0 - 192.0.2.255".

The information passed in CDNI request routing redirection requests splits into two basic categories:

1. An encapsulation of the User Agent's request to the upstream CDN.
2. Properties/parameters that the uCDN can use to control the dCDN's response or that can help the dCDN make its decision.

The information passed in CDNI request routing redirection response splits into two basic categories:

1. An encapsulation of the DNS response or HTTP response to return to the End User.
2. parameters /policies that indicate the properties of the response, such as, whether it is cacheable, the scope of the response, etc.

The following sections provide more detailed descriptions of the information that should be passed in CDNI request routing redirection interface requests and responses for both DNS redirection and HTTP redirection mechanisms.

4.1. Data passed in CDNI RRRI requests & response for DNS redirection

[[Editor's Note: Align the terminology in this section with standards DNS terminology.]]

For DNS based redirection the uCDN needs to pass the following information to the dCDN in the CDNI RRRI request:

- o The IP address of the DNS resolver that made the DNS request to the Upstream CDN.
- o The type of DNS query made (A, AAAA, etc.).
- o The host/domain for which DNS redirection is being requested.
- o The IP address of the User Agent (if known to the Upstream CDN, e.g. through edns-client-subnet).

For DNS redirection, the uCDN also needs to provide in the CDNI RRRI request the a link to the associated CDNI Metadata for the host/domain being requested.

The information passed in CDNI request routing redirection requests are enumerated in the sections below, which covers both DNS redirection and HTTP redirection.

For DNS based redirection the dCDN needs to return one of the following to the uCDN in the CDNI RRRI response:

- o The IP address of the selected Surrogate in the dCDN (if the dCDN is performing DNS based redirection); or
- o The IP address of a Request Router in the dCDN (if the dCDN is performing HTTP based redirection).

In addition, the dCDN could also include the following information in its response:

- o An indication of the cacheability of the response (to reduce the number of subsequent CDNI RRI requests the uCDN needs to make).
- o The scope of the response (if it is cacheable). For example whether the response applies to a wider range of IP addresses than what was included in the CDNI RRRI request.

4.2. Data passed in CDNI RRRI request & responses for HTTP redirection

For HTTP based redirection the uCDN needs to pass the following information to the dCDN in the CDNI RRRI request:

- o The IP address of the User Agent.
- o The URL requested by the User Agent.

For HTTP redirection, the uCDN also needs to provide in the CDNI RRRI request the a link to the associated CDNI Metadata for the host/domain being requested and whether the uCDN would like the dCDN to return the IP address or the hostname of the surrogate in the returned redirection URL.

[[Editor's note: An alternative would be to have the RRRI response always contain both hostnames & IP addresses and just let the uCDN pick which one it want to return to the User Agent]]

For HTTP based redirection the dCDN needs to return one of the following to the uCDN in the CDNI RRRI response:

- o A URL pointing to the selected Surrogate in the dCDN (if the dCDN is redirecting the User Agent directly to a surrogate); or
- o A URL pointing to a Request Router in the dCDN (if the dCDN is not redirecting the User Agent directly to a surrogate).

In addition, the dCDN could also include the following information in its response:

- o An indication of the cacheability of the response (to reduce the number of subsequent CDNI RRI requests the uCDN needs to make).
- o The scope of the response (if it is cacheable). For example whether the response applies to a wider range of IP addresses than what was included in the CDNI RRRI request.

5. HTTP based RESTful interface for CDNI Request Routing Redirection

This document defines a simple HTTP based RESTful interface for the CDNI Request Routing Redirection interface, where End User's requests are encapsulated along with links to appropriate CDNI Metadata resources (records) and any other data that can aid the downstream CDN in processing the requests. The RRI response encapsulates the response that the upstream CDN should return to the End User (if it decides to utilize the Downstream CDN for delivery) along with the policy for how the response could be reused (through standard HTTP Cache-Control headers).

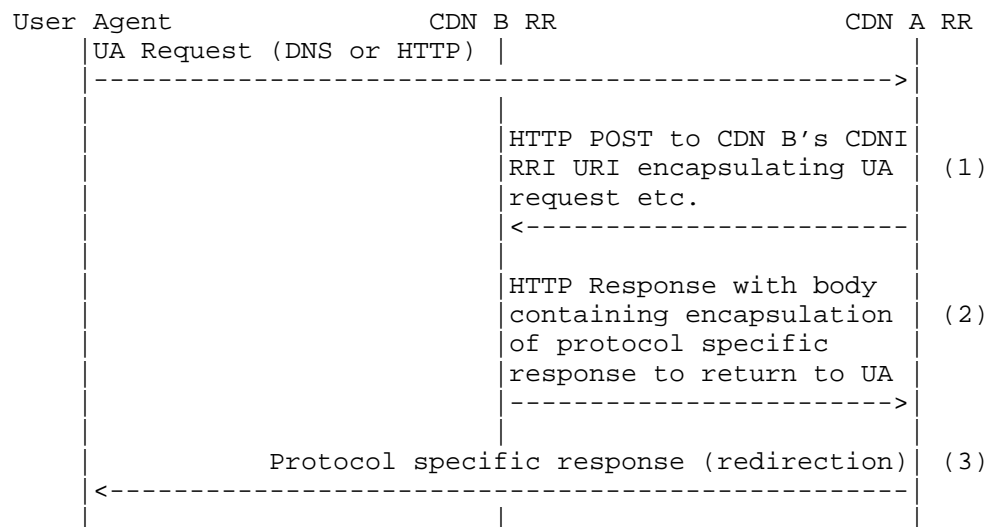
The same RESTful interface is used for both DNS and HTTP redirection of User Agent's requests, although the contents of the CDNI RRRI requests/responses contain data specific to either DNS or HTTP redirection.

This approach has been chosen because it enables CDN operators to only have to deploy a single (RESTful) interface for request routing between their CDNs, regardless of the User Agent redirection method. In this way, from an operational point of view there is only one interface to monitor, manage, develop troubleshooting tools for, etc.

In addition, having a single CDNI Request Routing interface where the User Agent's DNS or HTTP request are encapsulated along with the other data required for the downstream CDN to make a request routing decision, avoids having to try and encapsulate or proxy DNS/HTTP/RTMP/etc requests and find ways to somehow embed the additional CDNI Request Routing properties/data within those End User DNS/HTTP/RTMP/etc requests.

Finally, the interface is easily extendable to support other User Agent request redirection methods (e.g. RTMP 302 redirection).

The general call flow between Request Routers in a pair of interconnected CDNs is as follows:



1. The User Agent sends its request, either DNS request or HTTP request etc., to CDN A. The Request Routing System of CDN A processes the request, and it recognizes that the request is best served by another CDN, specifically CDN B (or that CDN B is one of a number of candidate dCDNs it could use).
2. The Request Routing System of CDN A sends an HTTP POST to CDN B's CDNI Request Routing Interface URI encapsulating the User Agent's request, as well as a link to the associated CDNI metadata, etc., which may help CDN B make its decision when processing the request.
3. The Request Routing System of CDN B processes the request and assuming the request is well formed, etc. responds with an HTTP "200" response with a message body containing an encapsulation of the protocol specific response to return to the User Agent (e.g., for HTTP redirection the URI that CDN A needs to return in a 302 response to the User Agent) as well as parameters that indicate the properties of the response (e.g., parameter that indicates if the response applies to a wider range of IP addresses than the single IP address being used by the User Agent).
4. The Request Routing System of CDN B sends the protocol specific response to the User Agent, and then the User Agent's request will be redirected to the CDN B.

6. Protocol Specification

This section only specifies a brief introduction of different fields to be exchanged in CDNI Request Routing Redirection requests/responses, and the specific encoding for them will be presented in a future revision.

[[Editor's note: This section needs some re-work, which we should do as part of documenting the specific protocol details.]]

6.1. Recursive Request Routing

Upon receiving a DNS query request from a User Agent, the Request Routing System of the Upstream CDN firsts determines how it should redirect the request using local policy. If it is aware that the End User is best served by another CDN, the Upstream CDN may make a CDNI RRRI request (that encapsulates the User Agent's request) to that Downstream CDN.

Upon receiving a response from the Downstream CDN, if the Request Routing System of the Upstream CDN decides to use that Downstream CDN, the Upstream CDN redirects the User Agent to the downstream CDN using the information in the CDNI RRRI response.

6.1.1. Downstream CDN Behavior for DNS based User Agent requests

Upon receiving a CDNI RRRI request, the Downstream CDN extracts End User's request information (e.g., the content provider's domain name) as well as associated CDNI Metadata to help it process the request. It then determines how it should redirect the request using local policy.

If the dCDN decides to use DNS Redirection, where the Downstream CDN is redirecting directly to a surrogate in the Downstream CDN: the Downstream CDN selects one or more surrogates and returns a CDNI RRRI response containing the IP addresses of the selected surrogate(s), as well as parameters/policies that indicate the properties of the response, such as, parameter that indicate whether the response is cacheable and for how long, parameter that indicate the scope of the response (if it is cacheable), etc.

If the dCDN decides to use DNS Redirection, where the Downstream CDN is redirecting to a Request Router in the dCDN: the Downstream CDN returns a CDNI RRRI response containing a CNAME of the Downstream CDN's Request Router(s), as well as parameters/policies that indicate the properties of the response, such as, parameter that indicate whether the response is cacheable and for how long, parameter that indicate the scope of the response (if it is cacheable), etc.

If the dCDN decides to use HTTP Redirection, where the original User Agent request was via DNS: the Downstream CDN returns a CDNI RRRI response containing the IP address of the Downstream CDN's Request Router(s), as well as parameters/policies that indicate the properties of the response, such as, parameter that indicate whether the response is cacheable and for how long, parameter that indicate the scope of the response (if it is cacheable), etc.

6.1.2. Downstream CDN Behavior for HTTP based User Agent requests

Upon receiving a CDNI RRRI request, the Downstream CDN extracts End User's request information (e.g., the content provider's domain name) as well as associated CDNI Metadata to help it process the request. It then determines how it should redirect the request using local policy.

The Downstream CDN selects one or more surrogates and returns a CDNI RRRI response containing a URL including the hostname and/or IP addresses of the selected surrogate(s), as well a path component that will return the resource originally request by the End User's User Agent. The CDNI RRRI response should also include parameters/policies that indicate the properties of the response, such as, parameter that indicate whether the response is cacheable and for how long, parameter that indicate the scope of the response (if it is cacheable), etc.

6.1.3. Example CDNI RRRI Requests/Responses

[[Editors' Note: Explanation of the format of the CDNI RRRI requests/responses is given as comments marked '#'. Need to extract & place in their own section which specifies the structure of CDNI RRRI requests/responses]]

CDN RRRI Request (uCDN->dCDN) for DNS based End User User Agent requests:

POST /dcdn/rrri HTTP/1.1

Host: rrl.dcdn.example.net

Accept: application/vnd.cdni.rrri.response+json

```
{
  "dns" : {
    "resolverip" : "192.0.2.1",
    "clientsubnet" : "198.51.100.1/24",
    "qtype" : "A",
    "qclass" : "IN",
    "qname" : "www.example.com"
  }
  "metadata" :
    "http://md.ucdn.example/www.example.com/"
}
```

Contains a representation of
the DNS query
IP address of DNS resolver
IP subnet of the UA/client
if known e.g. draft-
vandergaast-edns-client-subnet
DNS request type (one of A,
AAAA, CNAME)
class of query
domain name being queried
Link to associated CDNI
Metadata

CDNI RRRI successful Response (dCDN->uCDN) for DNS based End User
User Agent requests: [Currently shows both A/AAAA & CNAME in single
response, need to split to show the different use cases of
redirecting directly to a dCDN surrogate versus redirecting to a dCDN
Request Router]

HTTP/1.1 200 OK
 Date: Mon, 06 Aug 2012 18:41:38 GMT
 Content-Type: application/vnd.cdni.rrri.response+json
 Cache-Control: public, max-age=600

```
{
  "dns" : {
    # Contains a representation of
    # the DNS response to return to
    # the UA
    "rcode" : 0,
    # DNS response code
    "name" : "www.example.com",
    # Domain name response relates to
    # Response must contain at least
    # one of a, aaaa, cname
    "a" : ["192.0.2.200",
           "192.0.2.201"],
    # Set of IPv4 Addresses of dCDN
    # Surrogates or dCDN RRs
    "aaaa" : ["2001:DB8::C8",
              "2001:DB8::C9"],
    # Set of IPv6 Addresses of dCDN
    # or dCDN RRs
    "cname" : ["rr1.dcdn.example.net",
               "rr2.dcdn.example.net"],
    # Set of domain names of dCDN
    # Surrogates or dCDN RRs
    "ttl" : 60,
    # TTL to return in DNS response
  }
  "scope" : {
    # Scope of response
    "iprange" : "198.51.100.1/16"
    # Applies to a wider resolver/
    # client subnet
  }
}
```

CDN RRRI Request (uCDN->dCDN) for HTTP based End User User Agent requests:

POST/dcdn/rrri HTTP/1.1
 Host: rr1.dcdn.example.net
 Accept: application/vnd.cdni.rrri.response+json

```
{
  "http": {
    # Contains a representation of
    # the HTTP query
    "clientsubnet": "198.51.100.1/24",
    # IP subnet of the UA/client
    "url": "http://www.example.com",
    # URL requested by the UA
  }
  "metadata" :
    # Link to associated CDNI
    "http://md.ucdn.example/www.example.com/" # Metadata
}
```

CDNI RRRI successful Response (dCDN->uCDN) for HTTP based End User User Agent requests:

HTTP RRRI successful Response (dCDN->uCDN):

HTTP/1.1 200 OK

Date: Mon, 06 Aug 2012 18:41:38 GMT

Content-Type: application/vnd.cdni.rrri.response+json

Cache-Control: public, max-age=600

```
{
  "http": {
    # Contains a representation of
    # the HTTP response to return to
    # the UA
    "rcode": 200,
    # HTTP response code
    "url": "http://www.example.com",
    # URL response relates to
    "surrogates":
    # Set of URL pointing to the
    # selected dCDN Surrogates
    # or dCDN RR
    ["http://surl.dcdn.example.net/ucdn/example.com",
     "http://sur2.dcdn.example.net/ucdn/example.com"],
    "ttl" : 60,
    # TTL to return in DNS response,
  }
  "scope" : {
    # Scope of response
    "iprange" : "198.51.100.0/16"
    # Applies to a wider range of UA
    # IP addresses
  }
}
```

CDNI RRRI error response examples.

CDNI RRRI error response (dCDN->uCDN) for DNS based End User User Agent requests:

HTTP/1.1 500 Server Error

Date: Mon, 06 Aug 2012 18:41:38 GMT

Content-Type: application/vnd.cdni.rrri.error+json

Cache-Control: private, no-cache

```
{
  "dns" : {
    "rcode" : 4
    # DNS response code (e.g doesn't support AAAA)
    "name" : "www.example.com",
    # domain name response relates to
  },
  "error" : {
    "code" : TBD,
    # Give each error type its own
    # numeric code
    "description" :
    # Give more informative
    "IPv6/AAAA queries are not supported"
    # description than just protocol
    # specific error codes
  }
}
```

CDNI RRRI error response (dCDN->uCDN) for HTTP based End User User

Agent requests:

HTTP/1.1 500 Server Error

Date: Mon, 06 Aug 2012 18:41:38 GMT

Content-Type: application/vnd.cdni.rrri.error+json

Cache-Control: private, no-cache

```
{
  "http": {
    "rcode": 400,                # HTTP response code
    "url": http://www.example.com, # URL response relates to
  }
  "error" : {
    "code" : TBD,                # Give each error type its own
                                # numeric code
    "description" : TBD          # Give more informative
                                # description than just protocol
                                # specific error codes
  }
}
```

TBD: Work out more explanation & examples of errors that shouldn't be propagated to the EU User Agent.

6.2. Iterative Request Routing

[[Editor's note: Currently Iterative Request Routing is out of scope for this document as it does not require use of the CDNI RRRI. This section is just included to show that the authors' have not overlooked Iterative Request Routing.]]

7. Security Considerations

[[Editor's note: Not sure if this current text is really security considerations or whether it is better placed elsewhere in the document.]]

In HTTP based Recursive Request Routing, the end user's web browsers will not send cookies if the content request is redirected to a URL in a different domain rather than the original CP's domain, e.g. the Downstream CDN's domain. If the browser is expected to send any cookies associated with the original CP's domain, this will cause problem that the CP's policy is not enforced by the CDN.

The section 5.2 of draft [I-D.peterson-cdni-strawman] has discussed a similar question and given a solution.

8. IANA Considerations

This document makes no request of IANA.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

- [I-D.ietf-cdni-problem-statement]
Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-ietf-cdni-problem-statement-08 (work in progress), June 2012.
- [I-D.ietf-cdni-use-cases]
Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-09 (work in progress), July 2012.
- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-01 (work in progress), July 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.
- [I-D.peterson-cdni-strawman]
Peterson, L. and J. Hartman, "A Simple Approach to CDN Interconnection", draft-peterson-cdni-strawman-01 (work in progress), May 2011.

Authors' Addresses

Wang Danhua (editor)
Huawei Technologies
No. 101 Software Avenue
Nanjing, Jiangsu Province 210001
P.R.China

Phone: +86-25-56624734
Fax: +86-25-56624702
Email: wangdanhua@huawei.com

He Xiaoyan
Huawei
B2, Huawei Industrial Base
518129
P.R.China

Email: hexiaoyan@huawei.com

Spencer Dawkins
Huawei

Email: spencer.dawkins@wondermaster.com

Ge Chen
China Telecom
109 West Zhongshan Ave, Tianhe District
Guangzhou
P.R. China

Email: cheng@gsta.com

Ni Wei
China Mobile
No.32 Xuanwumen West Street Xicheng District
Beijing 100053
P.R. China

Email: niwei@chinamobile.com

Zhang Yunfei
China Mobile

Email: zhangyunfei@chinamobile.com

Ben Niven-Jenkins
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: ben@velocix.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 02, 2013

Danhua. Wang, Ed.
Huawei Technologies
B. Niven-Jenkins, Ed.
Velocix (Alcatel-Lucent)
Xiaoyan. He
Huawei
Chen. Ge
China Telecom
Wei. Ni
China Mobile
March 31, 2013

Request Routing Redirection Interface for CDN Interconnection
draft-he-cdni-routing-request-redirection-05

Abstract

The Request Routing Interface comprises of (1) the asynchronous advertisement of footprint and capabilities by a downstream CDN that allows a upstream CDN to decide whether to redirect particular user requests to that downstream CDN; and (2) the synchronous operation of an upstream CDN requesting whether a downstream CDN is prepared to accept a user request and of a downstream CDN responding with how to actually redirect the user request. This document describes an interface for the latter part, i.e. the CDNI request routing/Redirection Interface.

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 October 02, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Interface function and operation overview	4
4. HTTP based RESTful interface for the Redirection Interface .	5
4.1. Information passed in RI requests & responses	7
4.2. JSON encoding of RI requests & responses	9
4.3. DNS redirection	10
4.3.1. DNS Redirection requests	10
4.3.2. DNS Redirection responses	12
4.4. HTTP Redirection	13
4.4.1. HTTP Redirection requests	13
4.4.2. HTTP Redirection responses	14
4.5. Indicating the cacheability and scope of responses . . .	15
4.6. Error responses	17
4.7. Loop detection & prevention	18
5. Security Considerations	19
6. IANA Considerations	19
7. Acknowledgements	20
8. Outstanding considerations	20
9. Contributing Authors	20
10. References	20
10.1. Normative References	20
10.2. Informative References	21
Authors' Addresses	21

1. Introduction

A Content Delivery Network (CDN) is a system built on an existing IP network which is used for large scale content delivery, via prefetching or dynamically caching content on its distributed surrogates (caching servers). [RFC6707] describes the problem area of interconnecting CDNs.

The CDNI request routing interface outlined in [I-D.ietf-cdni-framework] comprises of:

1. The asynchronous advertisement of footprint and capabilities by a downstream CDN that allows a upstream CDN to decide whether to redirect particular user requests to that downstream CDN.
2. The synchronous operation of an upstream CDN requesting whether a downstream CDN is prepared to accept a user request and of a downstream CDN responding with how to actually redirect the user request.

This document describes an interface for the latter part, i.e. the CDNI request routing/Redirection Interface (RI).

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document reuses the terminology defined in [RFC6707]. The term "Distinguished CDN Domain" defined in [I-D.ietf-cdni-framework] is also reused in this document.

The following additional terms are introduced by this document:

Application Level Redirection: The act of using an application specific redirection mechanism for the request routing process of a CDN. The Redirection Target (RT) is the result of the routing decision of a CDN at the time it receives a content request via an application specific protocol response. Examples of an application level redirection are HTTP 302 Redirection and RTMP 302 Redirection.

DNS Redirection: The act of using DNS name resolution for the request routing process of a CDN. In DNS Redirection, the DNS name server of the CDN makes the routing decision based on a local policy and selects one or more Redirection Targets (RTs) and redirects the user agent to the RT(s) by returning the details of the RT(s) in response to the DNS query request from the user agent's DNS resolver.

HTTP Redirection: The act of using an HTTP redirection response for the request routing process of a CDN. The Redirection Target (RT) is the result of the routing decision of a CDN at the time it receives a content request via HTTP. HTTP Redirection is a particular case of Application Level Redirection.

Redirection Target (RT): A Redirection Target is the endpoint to which the user agent is redirected. In CDNI, a RT may point to a number of different components, some examples include a surrogate in the same CDN as the request router, a request router in a downstream CDN or a surrogate in a downstream CDN, etc.

3. Interface function and operation overview

[[Editor's note: Need to factor token authorisation into a future draft when that work is more stable/mature within the WG.]]

The CDNI request routing/Redirection Interface (RI) is one of the main building blocks required in order to interconnect CDNs. The main function of the Redirection Interface is to allow the Request Routing systems in interconnected CDNs to communicate to facilitate the redirection of User Agent requests between interconnected CDNs.

The detailed requirements for the Redirection Interface and their relative priorities are described in section 5 of [I-D.ietf-cdni-requirements].

The User Agent will make a request to a request router in the uCDN using one of either DNS or HTTP. If the RI is used between the uCDN and one or more dCDNs. The dCDN's RI response may contain a Redirection Target with a type that is compatible with the protocol used between User Agent and uCDN request router. The dCDN has control over the Redirection Target it provides and depending on the returned Redirection Target, the User Agent's request may be redirected to:

- o The final Surrogate, which may be in the dCDN or another dCDN (if dCDN delegates the delivery to another CDN).
- o A request router (in dCDN or another CDN) that will be using a redirection protocol (DNS or HTTP) which may or may not be the same as original redirection protocol.

The Redirection Interface operates between the Request Routing systems of a pair of interconnected CDNs. To enable communication over the Redirection Interface, the two interconnected CDNs need to know the end point (URI) in the other CDN to query. For example, an Upstream CDN needs to know the URI (end point) in a Downstream CDN to send its CDNI request routing queries to.

The Redirection Interface URI may be statically pre-configured, dynamically discovered via the CDNI control interface, or discovered via other means. However, such discovery mechanisms are not specified in this document, as they are considered out of the scope of the Redirection Interface specification.

CDNI solutions must support both of the request routing mechanisms illustrated in section 2.1 of [I-D.ietf-cdni-framework], namely Iterative Request Redirection and Recursive Request Redirection. However, the Iterative Request Redirection method does not invoke any interaction over the Redirection Interface between interconnected CDNs. Therefore, the Redirection Interface is only relevant in the case of Recursive Request Redirection and so this document will not discuss Iterative Request Redirection further.

In the case of Recursive Request Redirection, in order to perform redirection of a request received from a User Agent, the Upstream CDN queries the Downstream CDN so that the Downstream CDN can select and provide a Redirection Target. In cases where a uCDN has a choice of dCDNs it is down to the uCDN to decide (for example via configured policies) which dCDN(s) to query and in which order to query them. A number of strategies are possible including selecting a preferred dCDN based on local policy, possibly falling back to querying an alternative dCDN(s) if the first dCDN does not return a Redirection Target or otherwise reject the uCDN's RI request. A more complex strategy could be to query multiple dCDNs in parallel before selecting one and using the Redirection Target provided by that dCDN.

The Upstream CDN->User Agent redirection protocols addressed in this draft are: DNS redirection and HTTP redirection. Other types of application level redirection will not be discussed further in this draft. However the Redirection Interface is designed to be extensible and could be extended to support additional application level redirection protocols.

Also, according to the CDNI generic and request routing interface requirements, the CDNI solution shall support mechanisms to prevent and detect RI request loops. To meet such requirements, this document defines a loop prevention and detection mechanism as part of the Redirection Interface.

4. HTTP based RESTful interface for the Redirection Interface

This document defines a simple RESTful interface for the Redirection Interface based on HTTP [RFC2616], where the attributes of a User Agent's requests are encapsulated along with any other data that can aid the downstream CDN in processing the requests. The RI response encapsulates the attributes of the RT(s) that the upstream CDN should

return to the User Agent (if it decides to utilize the Downstream CDN for delivery) along with the policy for how the response can be reused.

The same RESTful interface is used for both DNS and HTTP redirection of User Agent's requests, although the contents of the RI requests/responses contain data specific to either DNS or HTTP redirection.

This approach has been chosen because it enables CDN operators to only have to deploy a single (RESTful) interface for the RI between their CDNs, regardless of the User Agent redirection method. In this way, from an operational point of view there is only one interface to monitor, manage, develop troubleshooting tools for, etc.

In addition, having a single RI where the attributes of the User Agent's DNS or HTTP request are encapsulated along with the other data required for the downstream CDN to make a request routing decision, avoids having to try and encapsulate or proxy DNS/HTTP/RTMP/etc requests and find ways to somehow embed the additional CDNI request routing/Redirection Interface properties/data within those End User DNS/HTTP/RTMP/etc requests.

Finally, the RI is easily extendable to support other User Agent request redirection methods (e.g. RTMP 302 redirection).

The generic Recursive Request Redirection message flow between Request Routing systems in a pair of interconnected CDNs is as follows:

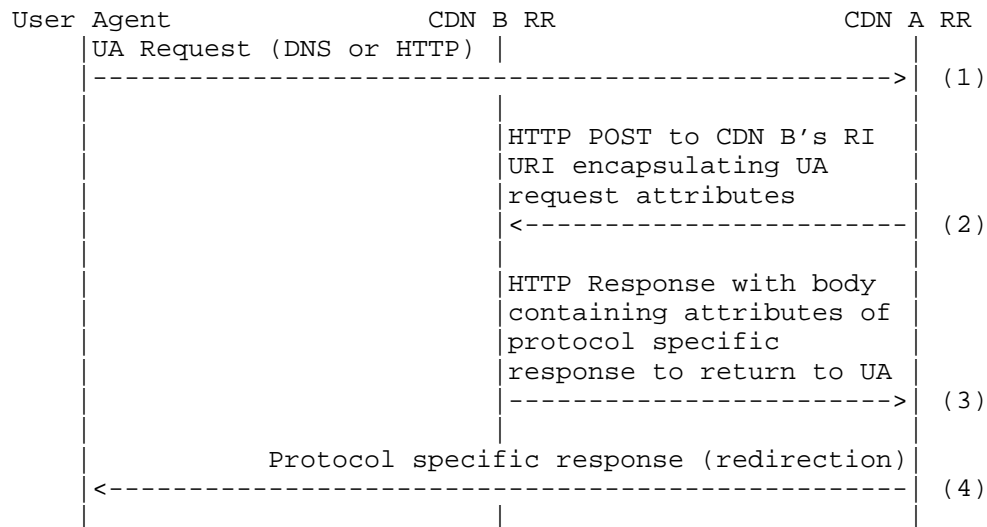


Figure 1: Generic Recursive Request Redirection message flow

1. The User Agent sends its request, either DNS request or HTTP request, to CDN A. The Request Routing System of CDN A processes the request and, through local policy, it recognizes that the request is best served by another CDN, specifically CDN B (or that CDN B is one of a number of candidate dCDNs it could use).
2. The Request Routing System of CDN A sends an HTTP POST to CDN B's RI URI containing the attributes of the User Agent's request.
3. The Request Routing System of CDN B processes the request and assuming the request is well formed, etc. responds with an HTTP "200" response with a message body containing the RT(s) to return to the User Agent as well as parameters that indicate the properties of the response (cacheability and scope).
4. The Request Routing System of CDN A sends a protocol specific response (containing the returned attributes) to the User Agent, so that the User Agent's request will be redirected to the RT(s) returned by CDN B.

4.1. Information passed in RI requests & responses

The information passed in RI requests splits into two basic categories:

1. The attributes of the User Agent's request to the upstream CDN.

2. Properties/parameters that the uCDN can use to control the dCDN's response or that can help the dCDN make its decision.

To assist the routing decision of a Downstream CDN, the Upstream CDN shall convey as much information as possible to the Downstream CDN, for example the URI of the requested content and the User Agent's location information, when those are known by the uCDN Request Routing system.

In order for the Downstream CDN to determine whether it is capable of delivering any requested content, it requires CDNI metadata related to the content the User Agent is requesting. That metadata will describe the content and any policies associated with it. It is expected that the RI request contains sufficient information for the Request Router in the Downstream CDN to be able to retrieve the require CDNI Metadata via the CDNI Metadata interface.

The information passed in RI responses splits into two basic categories:

1. The attributes of the RT to return to the User Agent in the DNS response or HTTP response.
2. Parameters/policies that indicate the properties of the response, such as, whether it is cacheable, the scope of the response, etc.

In addition to details of how to redirect the User Agent, the Downstream CDN may wish to return additional policy to the Upstream CDN to help the Upstream CDN with future RI requests. For example the Downstream CDN may wish to return a policy that expresses "this response can be reused without requiring a RI request for 60 seconds provided the User Agent's IP address is in the range 192.0.2.0 - 192.0.2.255".

These additional policies split into two basic categories:

- o An indication of the cacheability of the response carried in the HTTP response headers (to reduce the number of subsequent RI requests the uCDN needs to make).
- o The scope of the response (if it is cacheable) carried within the body of the HTTP response. For example whether the response applies to a wider range of IP addresses than what was included in the RI request.

The cacheability of the response is indicated using the standard HTTP Cache-Control mechanisms.

4.2. JSON encoding of RI requests & responses

The body of RI requests and responses is a JSON object containing a dictionary of keys. Keys MUST always be encoded in lowercase. Unknown keys MUST be ignored but the response MUST NOT be considered invalid unless the syntax of the request is invalid.

The following keys are defined:

Key	Request/Response	Description
dns	Both	The attributes of the UA's DNS request or the attributes of the RT(s) to return in a DNS response.
http	Both	The attributes of the UA's HTTP request or the attributes of the RT to return in a HTTP response.
scope	Response	The scope of the response (if it is cacheable). For example whether the response applies to a wider range of IP addresses than what was included in the RI request.
error	Response	Additional details if the response is an error response.
cdn-path	Both	A List of Strings. Contains the CDN Provider IDs of previous CDNs this RI request has passed through. When cascading a RI request the transit CDN appends its own CDN Provider ID to the list in cdn-path so that downstream CDNs can detect loops in the RI request chain. Transit CDNs should check the cdn-path and not cascade the RI request to downstream CDNs that are already listed in cdn-path. The cdn-path MUST be reflected back in RI responses.
max-hops	Request	Integer specifying the Maximum Number of hops (CDN Provider IDs) this request is allowed to be propagated along. This allows the uCDN to crudely constrain the latency of the request routing

```

|               |               | chain.               |
+-----+-----+-----+-----+

```

Top-Level keys in RI requests/responses

A single request or response MUST contain only one of the dns or http keys. Requests MUST contain a cdn-path key.

[[Editor's note: Need some text/section specifying the Media Types for RI requests/responses]]

[[Editor's note: Need some text on minimum attributes to be able to (at least parse) - e.g. A/AAAA/CNAME, etc]]

[[Editor's note: Need section detailing format/etc for scope and error keys]]

Note: All implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986] and MUST support all IPv6 address formats specified in [RFC4291]. Server implementations SHOULD use IPv6 address formats specified in [RFC5952].

4.3. DNS redirection

The following sections provide more detailed descriptions of the information that should be passed in RI requests and responses for DNS redirection.

4.3.1. DNS Redirection requests

For DNS based redirection the uCDN needs to pass the following information to the dCDN in the RI request:

- o The IP address of the DNS resolver that made the DNS request to the Upstream CDN.
- o The type of DNS query made (A, AAAA, RCODEs, etc.).
- o The class of DNS query made (usually IN). [[Editor's Note: Do we need to include class or can we always assume it is IN?]]
- o The fully qualified domain name for which DNS redirection is being requested.
- o The IP address or prefix of the User Agent (if known to the Upstream CDN, e.g. through draft-vandergaast-edns-client-subnet).

The information above is encoded as a set of key:value pairs within the dns dictionary as follows:

Key	Value	Mandatory	Description
resolver-ip	String	Yes	The IP address of the UA's DNS resolver.
qtype	String	Yes	The type of DNS query made by the UA's DNS resolvers in uppercase (A, AAAA, etc.).
qclass	String	Yes	The class of DNS query made in uppercase (IN, etc.).
qname	String	Yes	The fully qualified domain name being queried.
c-subnet	String	No	The IP address of the UA in CIDR format.
dns-only	Boolean	No	If True then dCDN MUST only use DNS redirection to a surrogate and MUST include the dns-only property set to True on any cascaded RI requests. Defaults to False.

An example RI request (uCDN->dCDN) for DNS based redirection:

```
POST /dcdn/ri HTTP/1.1
Host: rr1.dcdn.example.net
Accept: application/vnd.cdni.ri.response+json
```

```
{
  "dns" : {
    "resolver-ip" : "192.0.2.1",
    "c-subnet" : "198.51.100.0/24",
    "qtype" : "A",
    "qclass" : "IN",
    "qname" : "www.example.com"
  },
  "cdn-path": ["AS65551:0"],
  "max-hops": 3
}
```

4.3.2. DNS Redirection responses

For DNS based redirection the dCDN needs to return one of the following to the uCDN in the RI response:

- o The IP address of (or a CNAME to) the RT (if the dCDN is performing DNS based redirection); or
- o The IP address of (or a CNAME to) a RT which is a Request Router (if the dCDN is performing HTTP based redirection).

The information above is encoded as a set of key:value pairs within the dns dictionary as follows:

Key	Value	Mandatory	Description
rcode	Integer	Yes	DNS response code.
name	String	Yes	The fully qualified domain name the response relates to.
a	List of String	No	Set of IPv4 Addresses of RT(s).
aaaa	List of String	No	Set of IPv6 Addresses of RT(s).
cname	List of String	No	Set of fully qualified domain names of RT(s).
ttd	Integer	No	TTL of DNS response. Default is 0.

Response must contain at least one of a, aaaa, cname.

An example of a successful RI response (dCDN->uCDN) for DNS based redirection:

[[Editor's note: Currently shows both A/AAAA & CNAME in single response, need to split to show the different use cases]]

HTTP/1.1 200 OK

Date: Mon, 06 Aug 2012 18:41:38 GMT

Content-Type: application/vnd.cdni.ri.response+json

```
{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
```

```

    "a" : ["192.0.2.200", "192.0.2.201"],
    "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
    "cname" : ["rr1.dcdn.example",
               "rr2.dcdn.example"],
    "ttl" : 60
  }
}

```

4.4. HTTP Redirection

The following sections provide more detailed descriptions of the information that should be passed in RI requests and responses for HTTP redirection.

4.4.1. HTTP Redirection requests

For HTTP based redirection the uCDN MUST pass the following information to the dCDN in the RI request:

- o The IP address of the User Agent.
- o The URL requested by the User Agent.

The uCDN MAY also pass additional information to the dCDN in the RI request, such as:

- o The HTTP method or version number of the User Agent's request.
- o Additional HTTP header included in the User Agent request.

The information above is encoded as a set of key:value pairs within the http dictionary as follows:

Key	Value	Mandatory	Description
c-ip	String	Yes	The IP address of the UA/client
cs-uri	String	Yes	The URI requested by the UA/client.
cs(<HeaderName>)	String	No	The contents of the HTTP header named <HeaderName> as a string, for example cs(Cookie) would contain the content of the HTTP Cookie: header. Two

			special <HeaderName>s are defined: cs(Method) and cs(HTTP-Version) which contain the contents of the Method & HTTP-Version parts of the Request-Line as defined in Section 5.1 of [RFC2616].
--	--	--	--

An example RI request (uCDN->dCDN) for HTTP based redirection:

```
POST/dcdn/rrri HTTP/1.1
Host: rr1.dcdn.example.net
Accept: application/vnd.cdni.rrri.response+json
```

```
{
  "http": {
    "c-ip": "198.51.100.1",
    "cs-uri": "http://www.example.com"
  },
  "cdn-path": ["AS65551:0"],
  "max-hops": 3
}
```

4.4.2. HTTP Redirection responses

For HTTP based redirection the dCDN needs to return one of the following to the uCDN in the RI response:

- o A URL pointing to the selected RT (if the dCDN is redirecting the User Agent directly to a surrogate); or
- o A URL pointing to a RT which is a Request Router (if the dCDN is not redirecting the User Agent directly to a surrogate).

The information above is encoded as a set of key:value pairs within the http dictionary as follows:

Key	Value	Mandatory	Description
sc-status	Integer	Yes	The status code of the HTTP response to return to the UA

cs-uri	String	Yes	(usually 302). The URI requested by the UA/client.
sc-location	String	Yes	The contents of the Location header to return to the UA (i.e. a URI pointing to the RT(s)).
sc-cache-control	String	No	The contents of the Cache-Control header to return to the UA.

[[Editor's Note: Should we change the format above to align with the cs() format for headers on the RI request and allow the dCDN to signal back any headers it wants in the response as sc(<HeaderName>)? How to handle sc-status in that case - as a "special" header or separate key? Probably need to give some advice on HTTP headers the uCDN may want to override/not pass through, e.g. Server:?]]

An example of a successful RI response (dCDN->uCDN) for HTTP based redirection:

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/vnd.cdni.r1.response+json
```

```
{
  "http": {
    "sc-status": 302,
    "cs-uri": "http://www.example.com"
    "sc-location":
      "http://surl.dcdn.example/ucdn/example.com",
    "sc-cache-control" : "public, max-age=30"
  }
}
```

4.5. Indicating the cacheability and scope of responses

[[Editor's note: Need to expand text a little.]]

Cacheability is via the standard HTTP Cache-Control mechanisms.

Scope is encoded as a set of key:value pairs within the scope dictionary as follows:

Key	Value	Mandatory	Description
iprange	List of String	No	A List of IP subnets in CIDR notation that this RI response can be reused for, provided the RI response is still considered fresh.

If a uCDN has multiple cached responses with overlapping scopes, longest prefix matching of the User Agent's IP against the IP subnets in the scope of each response SHOULD be used to select the most appropriate RI response to use. [[Editor's note: is this always true? What about the most recent response, should that override older ones for the overlappign scope?]]

Example of DNS redirection response from Section 4.3.2 that is cacheable by the uCDN for 60 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/16.

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/vnd.cdni.ri.response+json
Cache-Control: public, max-age=60
```

```
{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "a" : ["192.0.2.200", "192.0.2.201"],
    "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
    "cname" : ["rr1.dcdn.example",
               "rr2.dcdn.example"],
    "ttl" : 60
  }
  "scope" : {
    "iprange" : ["198.51.100.0/16"]
  }
}
```

Example of HTTP redirection response from Section 4.4.2 that is cacheable by the uCDN for 60 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/16.


```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/vnd.cdni.r1.response+json
Cache-Control: public, max-age=60
```

```
{
  "http": {
    "sc-status": 302,
    "cs-uri": "http://www.example.com"
    "sc-location":
      "http://surl.dcdn.example/ucdn/example.com",
    "sc-cache-control" : "public, max-age=30"
  }
  "scope" : {
    "iprange" : ["198.51.100.0/16"]
  }
}
```

4.6. Error responses

[[Editor's note: Probably need more explanation & examples of errors that shouldn't be propagated to the User Agent?]]

RI error response examples.

RI error response (dCDN->uCDN) for DNS based User Agent requests:

HTTP/1.1 500 Server Error
 Date: Mon, 06 Aug 2012 18:41:38 GMT
 Content-Type: application/vnd.cdni.rrri.error+json
 Cache-Control: private, no-cache

```
{
  "dns" : {
    "rcode" : 4                                # DNS response code (e.g.
                                              # doesn't support AAAA)
    "name" : "www.example.com",               # domain name response
                                              # relates to
  },
  "error" : {
    "code" : TBD,                             # Give each error type its
                                              # own numeric code
    "description" :                          # Give more informative
    "IPv6/AAAA queries are not supported"    # description than just
  }                                           # protocol specific error
                                              # codes
}
```

RI error response (dCDN->uCDN) for HTTP based User Agent requests:

HTTP/1.1 500 Server Error
 Date: Mon, 06 Aug 2012 18:41:38 GMT
 Content-Type: application/vnd.cdni.rrri.error+json
 Cache-Control: private, no-cache

```
{
  "http": {
    "rcode": 400,                             # HTTP response code
    "url": "http://www.example.com",         # URL response
                                              # relates to
  }
  "error" : {
    "code" : TBD,                             # Give each error type its
                                              # own numeric code
    "description" : TBD                      # Give more informative
                                              # description than just
  }                                           # protocol specific error
                                              # codes
}
```

4.7. Loop detection & prevention

In order to prevent and detect RI request loops, each CDN MUST insert its CDN Provider ID into the cdn-path key of every RI request it originates or cascades. When receiving RI requests a dCDN should check the cdn-path and reject any RI requests which already contain the downstream CDN's Provider ID in the cdn-path. Transit CDNs should check the cdn-path and not cascade the RI request to downstream CDNs that are already listed in cdn-path. CDNs MUST NOT propagate to any downstream CDNs if the number of CDN Provider IDs in cdn-path (including the CDN's own Provider ID) is equal to or greater than max-hops.

The CDN Provider ID uniquely identifies each CDN provider during the course of request routing redirection. It consists of the the characters AS followed by the CDN Provider's AS number, then a colon (':') and an additional qualifier that is used to guarantee uniqueness in case a particular AS has multiple independent CDNs deployed. For example "AS65551:0".

If a downstream CDN receives a RI request whose cdn-path already contains that downstream CDN's Provider ID the downstream CDN MUST send a RI response with an error code of [[TBD]].

It should be noted that the loop detection & prevention mechanisms described above only cover preventing and detecting loops within the RI itself. In the cases where the IP address(es) or URI(s) returned in RI responses do not resolve directly to a surrogate in the final dCDN it is also possible to have redirection loops where Request Routers in different CDNs direct User Agents in a loop.

5. Security Considerations

[[Editor's note: Not sure if this current text is really security considerations or whether it is better placed elsewhere in the document.]]

In HTTP based Recursive Request Redirection, the end user's web browsers will not send cookies if the content request is redirected to a URL in a different domain rather than the original CP's domain, e.g. the Downstream CDN's domain. If the browser is expected to send any cookies associated with the original CP's domain, this will cause problem that the CP's policy is not enforced by the CDN.

The section 5.2 of draft [I-D.peterson-cdni-strawman] has discussed a similar question and given a solution.

6. IANA Considerations

This document makes no request of IANA.

7. Acknowledgements

The authors would like to thank Ray Brandenburg, Taesang Choi, Francois le Faucheur and Scott Wainner for their valuable comments and input to this document.

8. Outstanding considerations

Along with the various Editor's notes in the document, the following items still need to be addressed:

- o What extra properties/fields are required to cover all DNS/HTTP redirection cases?
- o Do we need Queries other than A/AAAA & response other than A/AAAA/CNAME?
- o Response scopes other than IP address? (AS? URL match?)
- o Better Security Considerations section.
- o Description/specification for how to extend the protocol with additional optional parameters/attributes.

9. Contributing Authors

[RFC Editor Note: Please move the contents of this section to the Authors' Addresses section prior to publication as an RFC.]

Spencer Dawkins
Huawei

Email: spencer@wonderhamster.org

Yunfei Zhang

Email: hishigh@gmail.com

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

10.2. Informative References

- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.
- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-03 (work in progress), February 2013.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-05 (work in progress), February 2013.
- [I-D.peterson-cdni-strawman]
Peterson, L. and J. Hartman, "A Simple Approach to CDN Interconnection", draft-peterson-cdni-strawman-01 (work in progress), May 2011.

Authors' Addresses

Wang Danhua (editor)
Huawei Technologies
No. 101 Software Avenue
Nanjing, Jiangsu Province 210001
P.R.China

Phone: +86-25-56624734
Fax: +86-25-56624702
Email: wangdanhua@huawei.com

Ben Niven-Jenkins (editor)
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: ben@velocix.com

He Xiaoyan
Huawei
B2, Huawei Industrial Base
518129
P.R.China

Email: hexiaoyan@huawei.com

Ge Chen
China Telecom
109 West Zhongshan Ave, Tianhe District
Guangzhou
P.R. China

Email: cheng@gsta.com

Ni Wei
China Mobile
No.32 Xuanwumen West Street Xicheng District
Beijing 100053
P.R. China

Email: niwei@chinamobile.com

Network Working Group
Internet-Draft
Obsoletes: 3466 (if approved)
Intended status: Informational
Expires: January 17, 2013

L. Peterson, Ed.
Verivue, Inc.
B. Davie
Nicira Networks, Inc.
July 16, 2012

Framework for CDN Interconnection
draft-ietf-cdni-framework-01

Abstract

This document presents a framework for Content Distribution Network Interconnection (CDNI). The purpose of the framework is to provide an overall picture of the problem space of CDNI and to describe the relationships among the various components necessary to interconnect CDNs. CDN Interconnection requires the specification of several interfaces and mechanisms to address issues such as request routing, metadata exchange, and the acquisition of content by one CDN from another. The intent of this document is to outline what each interface needs to accomplish, and to describe how these interfaces and mechanisms fit together, while leaving their detailed specification to other documents. It obsoletes RFC 3466.

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 January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Terminology	4
1.2. Reference Model	5
1.3. Structure Of This Document	8
2. Building Blocks	9
2.1. Request Redirection	9
2.1.1. DNS Redirection	9
2.1.2. HTTP Redirection	10
3. Overview of CDNI Operation	10
3.1. Preliminaries	13
3.2. HTTP Redirect Example	14
3.2.1. Comments on the example	18
3.3. Recursive Redirection Example	19
3.3.1. Comments on the example	23
3.4. DNS-based redirection example	23
3.4.1. Comments on the example	26
3.5. Dynamic Footprint Discovery	27
3.6. Content Removal	29
3.7. Pre-Positioned Content Acquisition Example	29
3.8. Asynchronous CDNI Metadata Example	31
3.9. Synchronous CDNI Metadata Acquisition Example	33
3.10. Content Acquisition with Multiple Upstream CDNs	36
4. Main Interfaces	37
4.1. In-Band versus Out-of-Band Interfaces	37
4.2. Cross Interface Concerns	38
4.3. Request Routing Interface	38
4.4. Logging Interface	40
4.5. Control Interface	42
4.6. Metadata Interface	42
5. Deployment Models	43
5.1. Meshed CDNs	44
5.2. CSP combined with CDN	44
5.3. CSP using CDNI Request Routing Interface	45
5.4. CDN Federations and CDN Exchanges	46
6. Trust Model	49
7. IANA Considerations	50
8. Security Considerations	50
8.1. Security of CDNI Interfaces	51
8.2. Digital Rights Management	52
9. Contributors	52
10. Acknowledgements	52
11. Informative References	52
Authors' Addresses	54

1. Introduction

The interconnection of Content Distribution Networks (CDNs) is motivated by several use cases, such as those described in [I-D.ietf-cdni-use-cases]. The overall problem space for CDN Interconnection is described in [I-D.ietf-cdni-problem-statement]. The purpose of this document is to provide an overview of the various components necessary to interconnect CDNs. CDN Interconnection requires the specification of several interfaces and mechanisms to address issues such as request routing, metadata exchange, and the acquisition of content by one CDN from another. The intent of this document is to describe how these interfaces and mechanisms fit together, leaving their detailed specification to other documents. We make extensive use of message flow examples to illustrate the operation of interconnected CDNs, but these examples should be considered illustrative rather than prescriptive.

RFC 3466 uses different terminology and models for "Content Internetworking (CDI)". It is also less prescriptive in terms of interfaces. To avoid confusion, this document obsoletes RFC 3466.

1.1. Terminology

This document draws freely on the core terminology defined in [I-D.ietf-cdni-problem-statement]. It also introduces the following terms:

CDN Domain: a host name (FQDN) at the beginning of a URL, representing a set of content that is served by a given CDN. For example, in the URL `http://cdn.csp.com/...rest of url...`, the CDN domain is `cdn.csp.com`. A major role of CDN Domain is to identify a region (subset) of the URI space relative to which various CDN Interconnection rules and policies are to apply. For example, a record of CDN Metadata might be defined for the set of resources corresponding to some CDN Domain.

Distinguished CDN Domain: a CDN domain that is allocated by a CDN for the purposes of communication with a peer CDN, but which is not found in client requests. Such CDN domains may be used for inter-CDN acquisition, or as redirection targets, and enable a CDN to distinguish a request from a peer CDN from an end-user request.

Delivering CDN: the CDN that ultimately delivers a piece of content to the end-user. The last in a potential sequence of downstream CDNs.

Recursive CDNI request routing: When an Upstream CDN elects to redirect a request towards a Downstream CDN, the Upstream CDN can

query the Downstream CDN Request Routing system via the CDNI Request Routing interface (or use information cached from earlier similar queries) to find out how the Downstream CDN wants the request to be redirected, which allows the Upstream CDN to factor in the Downstream CDN response when redirecting the user agent. This approach is referred to as "recursive" CDNI request routing. Note that the Downstream CDN may elect to have the request redirected directly to a Surrogate inside the Downstream CDN, to the Request-Routing System of the Downstream CDN, to another CDN, or to any other system that the Downstream CDN sees as fit for handling the redirected request.

Iterative CDNI Request Routing: When an Upstream CDN elects to redirect a request towards a Downstream CDN, the Upstream CDN can base its redirection purely on a local decision (and without attempting to take into account how the Downstream CDN may in turn redirect the user agent). In that case, the Upstream CDN redirects the request to the request routing system in the Downstream CDN, which in turn will decide how to redirect that request: this approach is referred to as "iterative" CDNI request routing.

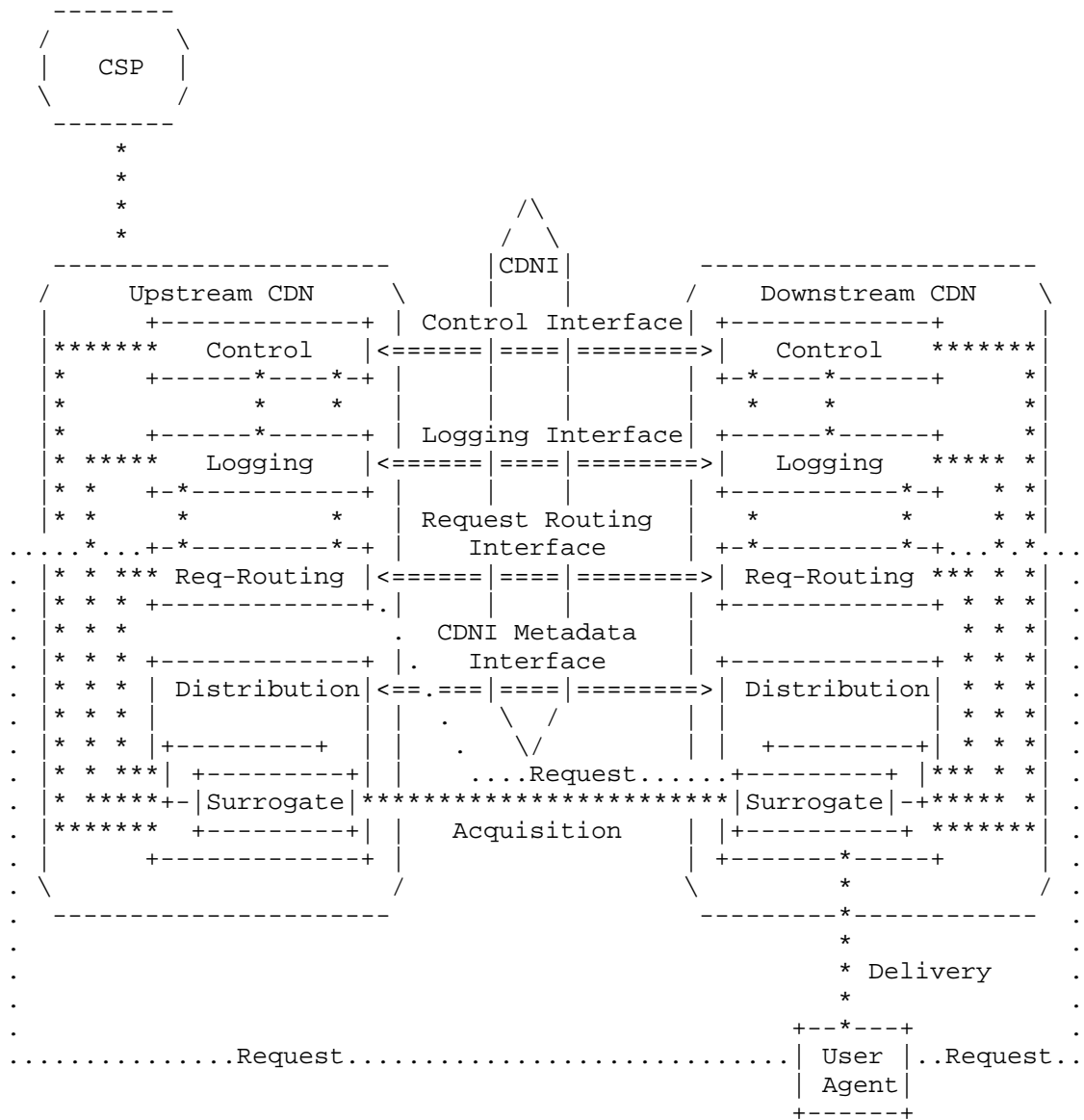
Synchronous CDNI operations: operations between CDNs that happen during the process of servicing a user request, i.e. between the time that the user agent begins its attempt to obtain content and the time at which that request is served.

Asynchronous CDNI operations: operations between CDNs that happen independently of any given user request, such as advertisement of footprint information or pre-positioning of content for later delivery.

Trigger Interface: a sub-set of the Control Interface that includes operations to pre-position, revalidate, and purge both metadata and content. These operations are typically called in response to some action (trigger) by the CSP on the upstream CDN.

1.2. Reference Model

This document uses the reference model in Figure 1 as originally created in [I-D.ietf-cdni-problem-statement].



`<==>` interfaces inside the scope of CDNI

```
**** interfaces outside the scope of CDNI
```

```
.... interfaces outside the scope of CDNI
```

Figure 1: CDNI Model and CDNI Interfaces

We note that while some interfaces in the reference model are "out of scope" for the CDNI WG (in the sense that there is no need to define new protocols for those interfaces) we still need to refer to them in this document to explain the overall operation of CDNI.

We also note that, while we generally show only one uCDN serving a given CSP, it is entirely possible that multiple uCDNs can serve a single CSP. In fact, this situation effectively exists today in the sense that a single CSP can currently connect to more than one CDN.

Definitions of the four CDNI interfaces follow. More discussion of these interfaces appears in Section 4.

- o Control Interface: Operations to discover, initialize, and parameterize the other CDNI interfaces, as well as operations to pre-position, revalidate, and purge both metadata and content. The latter sub-set of operations is sometimes collectively called the "trigger interface."
- o Request Routing Interface: Operations to determine what CDN (and optionally what surrogate within a CDN) is to serve end-user's requests. May include a combination of:
 - * Asynchronous operations to exchange routing information (e.g., the network footprint served by a given CDN) that enables CDN selection for subsequent user requests; and
 - * Synchronous operations to select a delivery CDN (surrogate) for a given user request.
- o Metadata Interface: Operations to communicate metadata that governs the how content is delivered by interconnected CDNs. Examples of CDNI metadata include geo-blocking directives, availability windows, access control mechanisms, and purge directives. May include a combination of:
 - * Asynchronous operations to exchange metadata that govern subsequent user requests for content; and
 - * Synchronous operations that govern behavior for a given user request for content.
- o Logging Interface: Operations that allow interconnected CDNs to exchange relevant activity logs. May include a combination of:
 - * Real-time exchanges, suitable for runtime traffic monitoring; and

- * Off-line exchanges, suitable for analytics and billing.

There is some ambiguity as to the line between the set of trigger-based operations in the Control interface and the Metadata interface. For both cases, the information passed from the upstream CDN to the downstream CDN can broadly be viewed as metadata that describes how content is to be managed by the downstream CDN. For example, the information conveyed by Control operations to pre-position, revalidate or purge metadata is similar to the information conveyed by posting updated metadata via the Metadata interface? Even the Control operation to purge content could be viewed as an metadata update for that content: purge simply says that the availability window for the named content ends now. The two interfaces share much in common, so minimally, there will need to be a consistent data model that spans both.

The distinction we draw has to do with what the caller knows the metadata being applied to content delivery by the callee. In the case of the Control interface, the downstream CDN returning a successful status message guarantees that the operation has been successfully completed; e.g., the content has been purged or pre-positioned. This implies that the downstream CDN accepts responsibility for having successfully completed the requested operation. In contrast, metadata passed between CDNs via the Metadata interface carries no such completion guarantee. Returning success implies successful receipt of the metadata, but nothing can be inferred about precisely when the metadata will take effect in the downstream CDN, only that it will take effect eventually. This is because of the challenge in globally synchronizing updates to metadata with end-user requests that are currently in progress (or indistinguishable from currently being in progress). Clearly, a CDN will not be viewed as a trusted peer if "eventually" often becomes an indefinite period of time, but the acceptance of responsibility cannot be as crisp for the Metadata interface.

1.3. Structure Of This Document

The remainder of this document is organized as follows:

- o Section 2 describes some essential building blocks for CDNI, notably the various options for redirecting user requests to a given CDN.
- o Section 3 provides a number of illustrative examples of various CDNI operations.
- o Section 4 describes the functionality of the four main CDNI interfaces.

- o Section 5 shows how various deployment models of CDNI may be achieved using the defined interfaces.
- o Section 6 describes the trust model of CDNI and the issues of transitive trust in particular that CDNI raises.

2. Building Blocks

2.1. Request Redirection

At its core, CDN Interconnection requires the redirection of requests from one CDN to another. For any given request that is received by an upstream CDN, it will either respond to the request directly, or somehow redirect the request to a downstream CDN. Two main mechanisms are available for redirecting a request to a downstream CDN. The first leverages the DNS name resolution process and the second uses in-protocol redirection mechanisms such as the HTTP 302 redirection response. We discuss these below as background before discussing some examples of their use in Section 3.

2.1.1. DNS Redirection

DNS redirection is based on returning different IP addresses for the same DNS name, for example, to balance server load or to account for the client's location in the network. A DNS server, sometimes called the Local DNS (LDNS), resolves DNS names on behalf of an end-user. The LDNS server in turn queries other DNS servers until it reaches the authoritative DNS server for the CDN-domain. The network operator typically provides the LDNS server, although the user is free to choose other DNS servers (e.g., OpenDNS, Google Public DNS).

The advantage of DNS redirection is that it is completely transparent to the end user--the user sends a DNS name to the LDNS server and gets back an IP address. On the other hand, DNS redirection is problematic because the DNS request comes from the LDNS server, not the end-user. This may affect the accuracy of server selection that is based on the user's location. The transparency of DNS redirection is also a problem in that there is no opportunity to modify the path component of the URL being accessed by the client. We consider two main forms of DNS redirection: simple and CNAME-based.

In simple DNS redirection, the authoritative DNS server for the name simply returns an IP address from a set of possible IP addresses. The answer is chosen from the set based on characteristics of the set (e.g., the relative loads on the servers) or characteristics of the client (e.g., the location of the client relative to the servers). Simple redirection is straightforward. The only caveats are (1)

there is a limit to the number of delivery nodes a single DNS server can manage; and (2) DNS responses are cached by downstream servers so the TTL on the response must be set to an appropriate value so as to preserve the timeliness of the redirection.

In CNAME-based DNS redirection, the authoritative server returns a CNAME response to the DNS request, telling the LDNS server to restart the name lookup using a new name. A CNAME is essentially a symbolic link in the DNS namespace, and like a symbolic link, redirection is transparent to the client--the LDNS server gets the CNAME response and re-executes the lookup. Only when the name has been resolved to an IP address does it return the result to the user. Note that DNAME would be preferable to CNAME if it becomes widely supported.

2.1.2. HTTP Redirection

HTTP redirection makes use of the "302" redirection response of the HTTP protocol. This response contains a new URL that the application should fetch instead of the original URL. By changing the URL appropriately, the server can cause the user to redirect to a different server. The advantages of 302 redirection are that (1) the server can change the URL fetched by the client to include, for example, both the DNS name of the particular server to use, as well as the original HTTP server that was being accessed; and (2) the client sends the HTTP request to the server, so that its IP address is known and can be used in selecting the server.

The disadvantages of HTTP redirection are (1) it is visible to the application, so it requires application support and may affect the application behavior (e.g., web browsers will not send cookies if the URL changes to a different domain); (2) HTTP is a heavy-weight protocol layered on TCP so it has relatively high overhead; and (3) the results of HTTP redirection are not cached so that all redirections must go through to the server.

3. Overview of CDNI Operation

To provide a big-picture overview of the various components of CDN Interconnection, we walk through a "day in the life" of a content item that is made available via a pair of interconnected CDNs. This will serve to illustrate many of the functions that need to be supported in a complete CDNI solution. We give examples using both DNS-based and HTTP-based redirection. We begin with very simple examples and then how additional capabilities, such as recursive request redirection and content removal, might be added.

Before walking through some specific examples, we present a high-

level view of the operations that may take place. This high-level overview is illustrated in Figure 2. Note that most operations will involve only a subset of all the messages shown below, and that the order and number of operations may vary considerably, as more detailed examples illustrate below.

The following shows Operator A as the upstream CDN (uCDN) and Operator B as the downstream CDN (dCDN), where the former has a relationship with a content provider and the latter being the best CDN to deliver content to the end-user. The interconnection relationship may be symmetric between these two CDN operators, but for simplicity we show the interaction in one direction only.

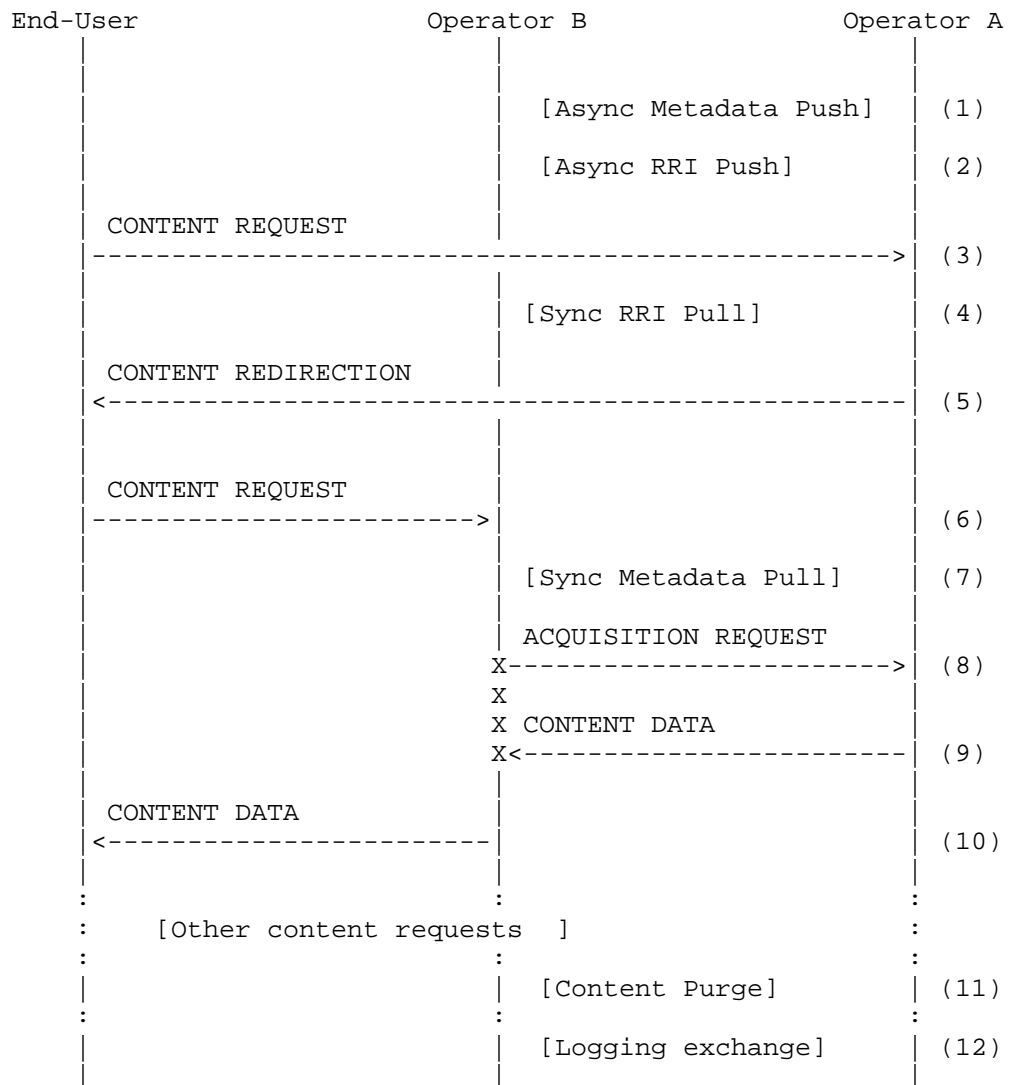


Figure 2: Overview of Operation

The operations shown in the Figure are as follows:

1. Prior to any content request, metadata may be asynchronously pushed from uCDN to dCDN so that it is available in readiness for later content requests.

2. dCDN may advertise information relevant to its delivery capabilities (e.g. geographic footprint, reachable address prefixes) prior to any content requests being redirected.
3. A content request from a user agent arrives at uCDN.
4. uCDN may synchronously request information from dCDN regarding its delivery capabilities to decide if dCDN is a suitable target for redirection of this request.
5. uCDN redirects the request to dCDN by sending some response (DNS, HTTP) to the user agent.
6. The user agent requests the content from dCDN.
7. dCDN may synchronously request metadata related to this content from uCDN, e.g. to decide whether to serve it.
8. If the content is not already in a suitable cache in dCDN, dCDN may acquire it from uCDN.
9. The content is delivered to dCDN from uCDN.
10. The content is delivered to the user agent by dCDN.
11. Some time later, perhaps at the request of the CSP (not shown) uCDN may instruct dCDN to purge the content to ensure it is not delivered again.
12. After one or more content delivery actions by dCDN, a log of delivery actions may be provided to uCDN.

The following sections show some more specific examples of how these operations may be combined to perform various delivery, control and logging operations across a pair of CDNs.

3.1. Preliminaries

Initially, we assume that there is at least one CSP that has contracted with an upstream CDN (uCDN) to deliver content on its behalf. We are not particularly concerned with the interface between the CSP and uCDN, other than to note that it is expected to be the same as in the "traditional" (non-interconnected) CDN case. Existing mechanisms such as DNS CNAMEs or HTTP redirects (Section 2) can be used to direct a user request for a piece of content from the CSP towards the CSP's chosen upstream CDN.

We use the term "CDN-domain" to refer to the host name (a FQDN) at

the beginning of each URL. We assume Operator A provides an upstream CDN that serves content on behalf of a CSP with CDN-domain `cdn.csp.com`. We assume that Operator B provides a downstream CDN. An end user at some point makes a request for URL

`http://cdn.csp.com/...rest of url...`

It may well be the case that `cdn.csp.com` is just a CNAME for some other CDN-domain (such as `csp.op-a.net`). Nevertheless, the HTTP request in the examples that follow is assumed to be for the example URL above.

Our goal is to enable content identified by the above URL to be served by the CDN of operator B. In the following sections we will walk through some scenarios in which content is served, as well as other CDNI operations such as the removal of content from a downstream CDN.

3.2. HTTP Redirect Example

In this section we walk through a simple, illustrative example using HTTP redirection from uCDN to dCDN. The example also assumes the use of HTTP redirection inside uCDN and dCDN; however, this is independent of the choice of redirection approach across CDNs, so an alternative example could be constructed still showing HTTP redirection from uCDN to dCDN but using DNS for handling of request inside each CDN.

We assume for this example that Operators A and B have established an agreement to interconnect their CDNs, with A being upstream and B being downstream. (It is likely that the agreement would be made in both directions, but we focus on just one here for clarity.)

The operators agree that a CDN-domain `peer-a.op-b.net` will be used as the target of redirections from uCDN to dCDN. The name of this domain must be communicated by some means to each CDN. (This could be established out-of-band or via a CDNI interface.) We refer to this domain as a "distinguished" CDN domain to convey the fact that its use is limited to the interconnection mechanism; such a domain is never embedded in URLs that end-users request.

The operators must also agree on some distinguished CDN-domain that will be used for inter-CDN acquisition of CSP's content from uCDN by dCDN. In this example, we'll use `op-b-acq.op-a.net`.

The operators must also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical region or a set

of IP address prefixes. This information may again be provided out of band or via a defined interface.

DNS must be configured in the following way:

- o The content provider must be configured to make operator A the authoritative DNS server for `cdn.csp.com` (or to return a CNAME for `cdn.csp.com` for which operator A is the authoritative DNS server).
- o Operator A must be configured so that a DNS request for `op-b-acq.op-a.net` returns a request router in Operator A.
- o Operator B must be configured so that a DNS request for `peer-a.op-b.net/cdn.csp.com` returns a request router in Operator B.

Figure 3 illustrates how a client request for

`http://cdn.csp.com/...rest of url...`

is handled.



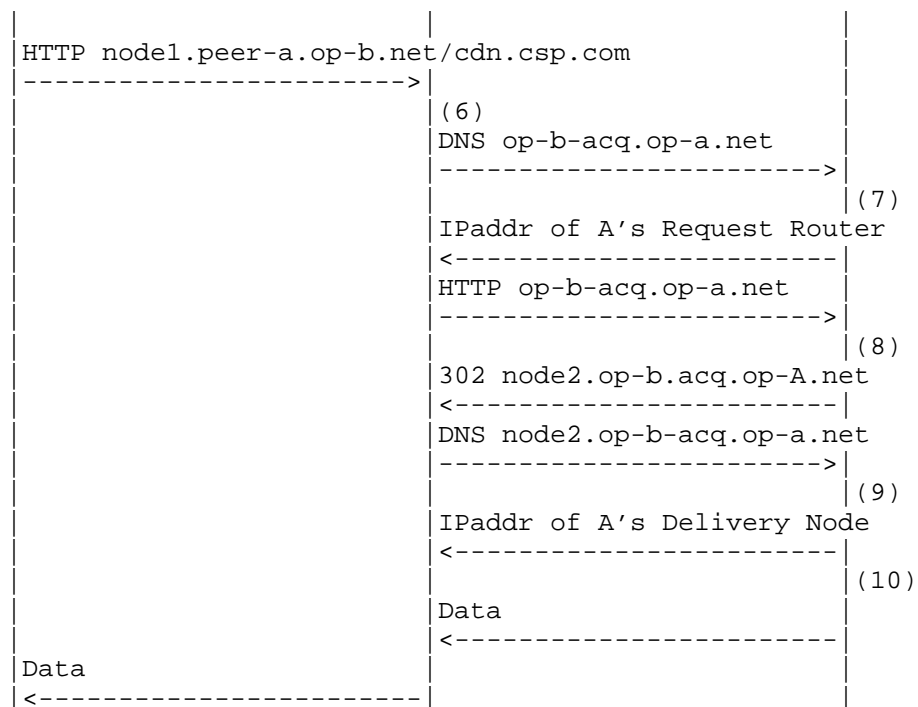


Figure 3: Request Trace for HTTP redirection method

The steps illustrated in the figure are as follows:

1. A DNS resolver for Operator A processes the DNS request for its customer based on CDN-domain `cdn.csp.com`. It returns the IP address of a request router in Operator A.
2. A Request Router for Operator A processes the HTTP request and recognizes that the end-user is best served by another CDN--specifically one provided by Operator B--and so it returns a 302 redirect message for a new URL constructed by "stacking" Operator B's distinguished CDN-domain (`peer-a.op-b.net`) on the front of the original URL. (Note that more complex URL manipulations are possible, such as replacing the initial CDN-domain by some opaque handle.)
3. The end-user does a DNS lookup using Operator B's distinguished CDN-domain (`peer-a.op-b.net`). B's DNS resolver returns the IP address of a request router for Operator B. Note that if request routing within dCDN was performed using DNS instead of HTTP redirection, B's DNS resolver would also behave as the request router and directly return the IP address of a delivery node.

4. The request router for Operator B processes the HTTP request and selects a suitable delivery node to serve the end-user request, and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the Operator B's distinguished CDN-domain that points to the selected delivery node.
5. The end-user does a DNS lookup using Operator B's delivery node subdomain (nodel.peer-a.op-b.net). B's DNS resolver returns the IP address of the delivery node.
6. The end-user requests the content from B's delivery node. In the case of a cache hit, steps 6, 7, 8, 9 and 10 below do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDN from uCDN (not the CSP). The distinguished CDN-domain peer-a.op-b.net indicates to dCDN that this content is to be acquired from uCDN; stripping the CDN-domain reveals the original CDN-domain cdn.csp.com and dCDN may verify that this CDN-domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an inter-CDN acquisition CDN-domain as agreed above (in this case, op-b-acq.op-a.net).
7. Operator A's DNS resolver processes the DNS request and returns the IP address of a request router in operator A.
8. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (op-b-acq.op-a.net). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.
9. Operator A DNS resolver processes the DNS request and returns the IP address of the delivery node in operator A.
10. Operator A serves content for the requested CDN-domain to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has

been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

3.2.1. Comments on the example

The main advantage of this design is that it is simple: each CDN need only know the distinguished CDN-domain for each peer, with the upstream CDN "pushing" the downstream CDN-domain onto the URL as part of its redirect (step 2) and the downstream CDN "popping" its CDN-domain off the URL to expose a CDN-domain that the upstream CDN can correctly process. Neither CDN needs to be aware of the internal structure of the other's URLs. Moreover, the inter-CDN redirection is entirely supported by a single HTTP redirect; neither CDN needs to be aware of the other's internal redirection mechanism (i.e., whether it is DNS or HTTP based).

One disadvantage is that the end-user's browser is redirected to a new URL that is not in the same domain of the original URL. This has implications on a number of security or validation mechanisms sometimes used on endpoints. For example, it is important that any redirected URL be in the same domain (e.g., csp.com) if the browser is expected to send any cookies associated with that domain. As another example, some video players enforce validation of a cross domain policy that needs to allow for the domains involved in the CDN redirection. These problems are generally soluble, but the solutions complicate the example, so we do not discuss them further in this version of the draft.

We note that this example begins to illustrate some of the interfaces that may be required for CDNI, but does not require all of them. For example, obtaining information from dCDN regarding the set of client IP addresses or geographic regions it might be able to serve is an aspect of the request routing interface. Important configuration information such as the distinguished names used for redirection and inter-CDN acquisition could also be conveyed via a CDNI interface (e.g., perhaps the control interface). The example also shows how existing HTTP-based methods suffice for the acquisition interface. Arguably, the absolute minimum metadata required for CDNI is the information required to acquire the content, and this information was provided "in-band" in this example by means of the URI handed to the client in the HTTP 302 response. Hence, there is no explicit metadata interface invoked in this example. There is also no explicit logging interface discussed in this example.

We also note that the step of deciding when a request should be redirected to dCDN rather than served by uCDN has been somewhat

glossed over. It may be as simple as checking the client IP address against a list of prefixes, or it may be considerably more complex, involving a wide range of factors, such as the geographic location of the client (perhaps determined from a third party service), CDN load, or specific business rules.

This example uses the "iterative" CDNI request routing approach. That is, uCDN performs part of the request routing function to determine that dCDN should serve the request, and then redirects the client to a request router in dCDN to perform the rest of the request routing function. If request routing is performed in the dCDN using HTTP redirection, this translates in the end-user experiencing two successive HTTP redirections. By contrast, the alternative approach of "recursive" CDNI request routing effectively coalesces these two successive HTTP redirections into a single one, sending the end-user directly to the right delivery node in the dCDN. This "recursive" CDNI request routing approach is discussed in the next section.

3.3. Recursive Redirection Example

The following example builds on the previous one to illustrate the use of the Request Routing interface to enable "recursive" CDNI request routing. We build on the HTTP-based redirection approach because it illustrates the principles and benefits clearly, but it is equally possible to perform recursive redirection when DNS-based redirection is employed.

In contrast to the prior example, the operators need not agree in advance on a CDN-domain to serve as the target of redirections from uCDN to dCDN. The operators still must agree on some distinguished CDN-domain that will be used for inter-CDN acquisition of CSP's content by dCDN. In this example, we'll use op-b-acq.op-a.net.

The operators must also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical region or a set of IP address prefixes. This information may again be provided out of band or via a defined protocol.

DNS must be configured in the following way:

- o The content provider must be configured to make operator A the authoritative DNS server for cdn.csp.com (or to return a CNAME for cdn.csp.com for which operator A is the authoritative DNS server).
- o Operator A must be configured so that a DNS request for op-b-acq.op-a.net returns a request router in Operator A.

- o Operator B must be configured so that a request for `node1.opb.net/cdn.csp.com` returns the IP address of a delivery node. Note that there might be a number of such delivery nodes.

Figure 3 illustrates how a client request for

`http://cdn.csp.com/...rest of url...`

is handled.

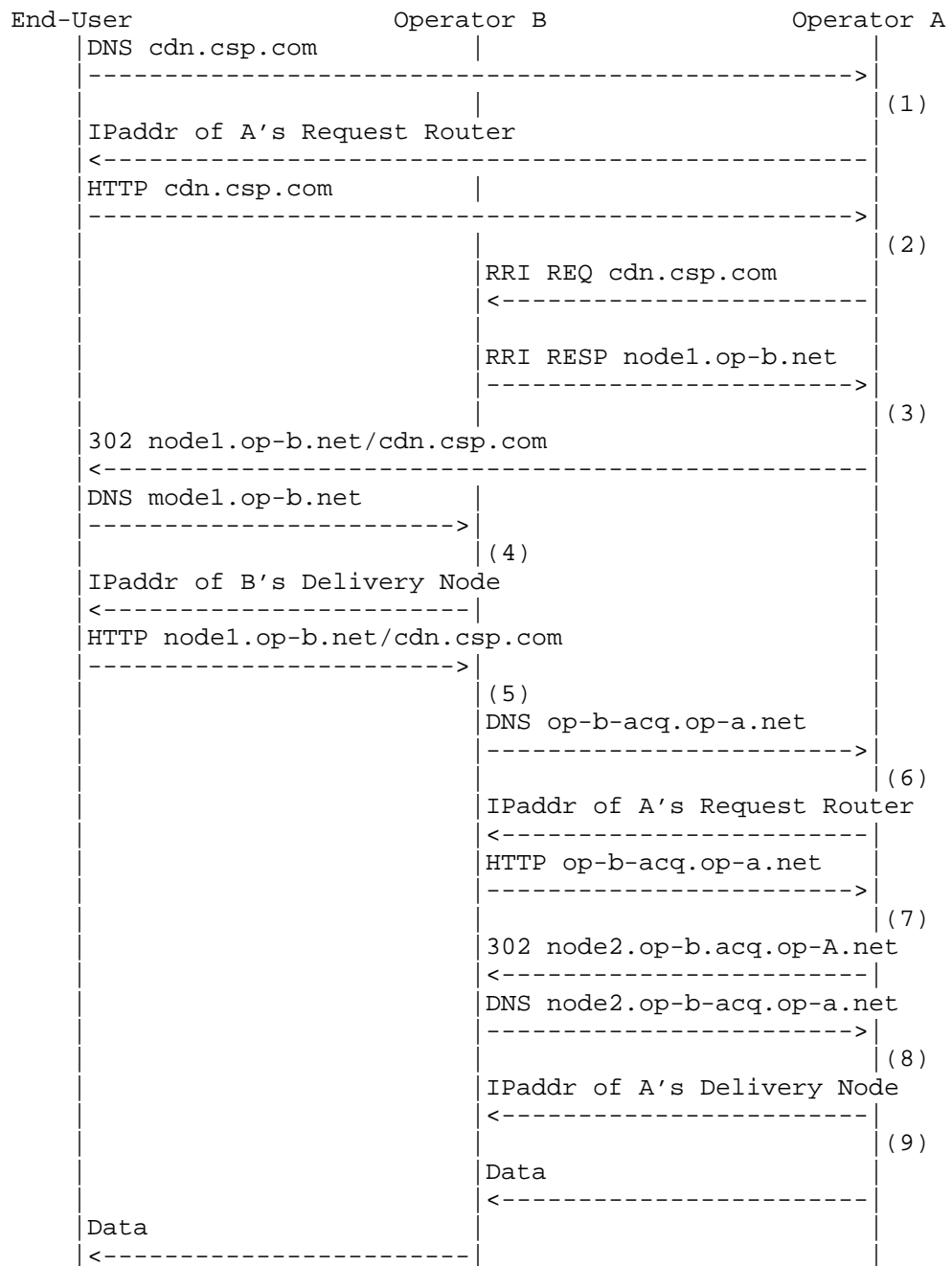


Figure 4: Request Trace for Recursive HTTP redirection method

The steps illustrated in the figure are as follows:

1. A DNS resolver for Operator A processes the DNS request for its customer based on CDN-domain `cdn.csp.com`. It returns the IP address of a Request Router in Operator A.
2. A Request Router for Operator A processes the HTTP request and recognizes that the end-user is best served by another CDN--specifically one provided by Operator B--and so it queries the CDNI Request Routing interface of Operator B, providing a set of information about the request including the URL requested. Operator B replies with the DNS name of a delivery node.
3. Operator A returns a 302 redirect message for a new URL obtained from the Request Routing Interface.
4. The end-user does a DNS lookup using the host name of the URL just provided (`node1.op-b.net`). B's DNS resolver returns the IP address of the corresponding delivery node. Note that, since the name of the delivery node was already obtained from B using the CDNI Request Routing Interface, there should not be any further redirection here (in contrast to the iterative method described above.)
5. The end-user requests the content from B's delivery node, potentially resulting in a cache miss. In the case of a cache miss, the content needs to be acquired from uCDN (not the CSP.) The distinguished CDN-domain `op-b.net` indicates to dCDN that this content is to be acquired from another CDN; stripping the CDN-domain reveals the original CDN-domain `cdn.csp.com`, dCDN may verify that this CDN-domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for the inter-CDN Acquisition "distinguished" CDN-domain as agreed above (in this case, `op-b-acq.op-a.net`).
6. Operator A DNS resolver processes the DNS request and returns the IP address of a request router in operator A.
7. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (`op-b-acq.op-a.net`). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL

constructed by replacing the hostname by a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.

8. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-domain) and so returns the address of a delivery node. (Note that without this specially defined internal domain, Operator A would be at risk of redirecting the request back to Operator B, resulting in an infinite loop.)
9. Operator A serves content for the requested CDN-domain to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

3.3.1. Comments on the example

Recursive redirection has the advantage over iterative of being more transparent from the end-user's perspective, but the disadvantage of each CDN exposing more of its internal structure (in particular, the addresses of edge caches) to peer CDNs. By contrast, iterative redirection does not require dCDN to expose the addresses of its edge caches to uCDN.

This example happens to use HTTP-based redirection in both CDN A and CDN B, but a similar example could be constructed using DNS-based redirection in either CDN. Hence, the key point to take away here is simply that the end user only sees a single redirection of some type, as opposed to the pair of redirections in the prior (iterative) example.

The use of the Request Routing Interface requires that interface to be appropriately configured and bootstrapped, which is not shown here. More discussion on the bootstrapping of interfaces is provided in Section 4

3.4. DNS-based redirection example

In this section we walk through a simple example using DNS-based redirection for request redirection from uCDN to dCDN (as well as for request routing inside dCDN and uCDN). As noted in Section 2.1, DNS-based redirection has certain advantages over HTTP-based redirection (notably, it is transparent to the end-user) as well as some drawbacks (notably the client IP address is not visible to the

request router).

As before, Operator A must learn the set of requests that dCDN is willing or able to serve (e.g. which client IP address prefixes or geographic regions are part of the dCDN footprint). Operator B must have and make known to operator A some unique identifier that can be used for the construction of a distinguished CDN domain, as shown in more detail below. (This identifier strictly needs only to be unique within the scope of Operator A, but a globally unique identifier, such as an AS number assigned to B, is one easy way to achieve that.) Also, Operator A must obtain the NS records for Operator B's externally visible redirection servers. Also, as before, a distinguished CDN-domain, such as op-b-acq.op-a.net, must be assigned for inter-CDN acquisition.

DNS must be configured in the following way:

- o The CSP must be configured to make Operator A the authoritative DNS server for cdn.csp.com (or to return a CNAME for cdn.csp.com for which operator A is the authoritative DNS server).
- o When uCDN sees a request best served by dCDN, it returns CNAME and NS records for "b.cdn.csp.com", where "b" is the unique identifier assigned to Operator B. (It may, for example, be an AS number assigned to Operator B.)
- o dCDN must be configured so that a request for "b.cdn.csp.com" returns a delivery node in dCDN.
- o uCDN must be configured so that a request for "op-b-acq.op-a.net" returns a delivery node in uCDN.

Figure 5 depicts the exchange of DNS and HTTP requests. The main differences from Figure 3 are the lack of HTTP redirection and transparency to the end-user.

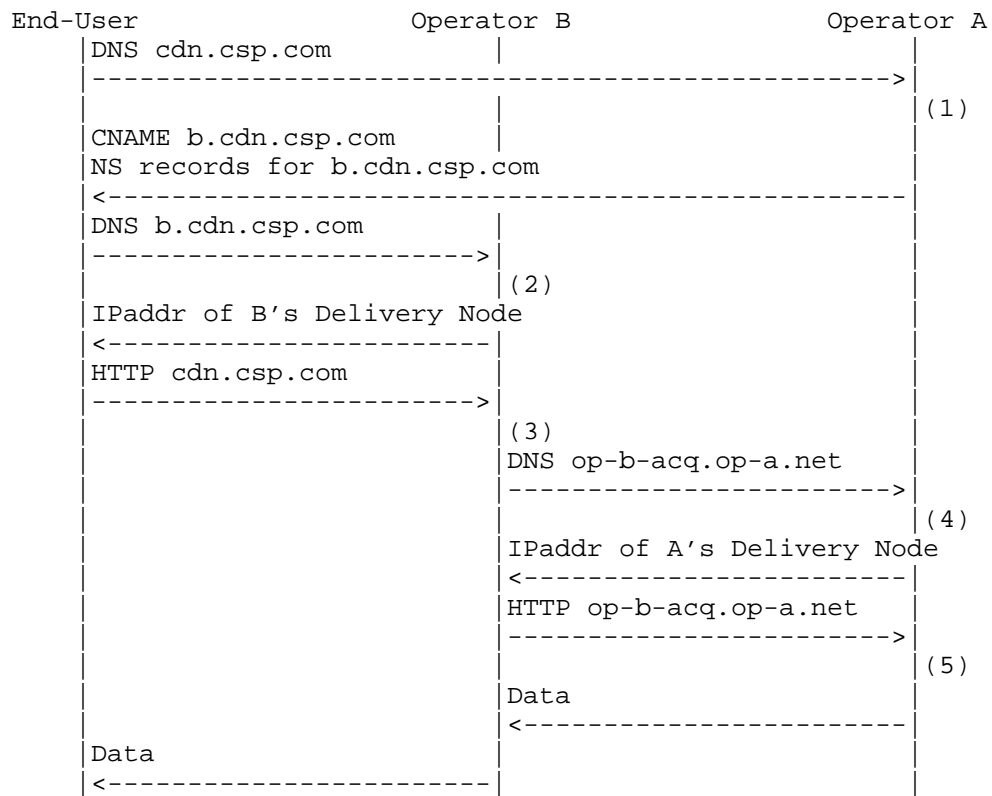


Figure 5: Request Trace for DNS-based Redirection Example

The steps illustrated in the figure are as follows:

1. Request Router for Operator A processes the DNS request for CDN-domain `cdn.csp.com` and recognizes that the end-user is best served by another CDN. (This may depend on the IP address of the user's local DNS resolver, or other information discussed below.) The Request Router returns a DNS CNAME response by "stacking" the distinguished identifier for Operator B onto the original CDN-domain (e.g., `b.cdn.csp.com`), plus an NS record that maps `b.cdn.csp.com` to B's Request Router.
2. The end-user does a DNS lookup using the modified CDN-domain (i.e., `b.cdn.csp.com`). This causes B's Request Router to respond with a suitable delivery node.
3. The end-user requests the content from B's delivery node. The requested URL contains the name `cdn.csp.com`. (Note that the returned CNAME does not affect the URL.) At this point the

delivery node has the correct IP address of the end-user and can do an HTTP 302 redirect if the redirections in steps 2 and 3 were incorrect. Otherwise B verifies that this CDN-domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an "internal" CDN-domain as agreed above (op-b-acq.op-a.net).

4. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-domain) and so returns the address of a delivery node in uCDN.
5. Operator A serves content to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server.

3.4.1. Comments on the example

The advantages of this approach are that it is more transparent to the end-user and requires fewer round trips than HTTP-based redirection. A potential problem is that the upstream CDN depends on being able to learn the correct downstream CDN that serves the end-user from the client address in the DNS request. In standard DNS operation, uCDN will only obtain the address of the client's local DNS resolver (LDNS), which is not guaranteed to be in the same network (or geographic region) as the client. If not--e.g., the end-user uses a global DNS service--then the upstream CDN cannot determine the appropriate downstream CDN to serve the end-user. In this case, one option is for the upstream CDN to treat the end-user as it would any user not connected to a peer CDN. Another option is for the upstream CDN to "fall back" to a pure HTTP-based redirection strategy in this case (i.e., use the first method). Note that this problem affects existing CDNs that rely on DNS to determine where to redirect client requests, but the consequences are arguably less serious since the LDNS is likely in the same network as the dCDN serves. One approach to ensuring that the client's IP address prefix is correctly determined in such situations is described in [I-D.vandergaast-edns-client-subnet].

As with the prior example, this example partially illustrates the various interfaces involved in CDNI. Operator A could learn dynamically from Operator B the set of prefixes or regions that B is willing and able to serve via the request routing interface. The distinguished name used for acquisition and the identifier for Operator B that is prepended to the CDN domain on redirection are examples of information elements that might also be conveyed by CDNI interfaces (or, alternatively, statically configured). As before,

minimal metadata sufficient to obtain the content is carried "in-band" as part of the redirection process, and standard HTTP is used for inter-CDN acquisition. There is no explicit logging interface discussed in this example.

3.5. Dynamic Footprint Discovery

There could be situations where being able to dynamically discover the set of requests that a given dCDN is willing and able to serve is beneficial. For example, a CDN might at one time be able to serve a certain set of client IP prefixes, but that set might change over time due to changes in the topology and routing policies of the IP network. The following example illustrates this capability. We have chosen the example of DNS-based redirection, but HTTP-based redirection could equally well use this approach.

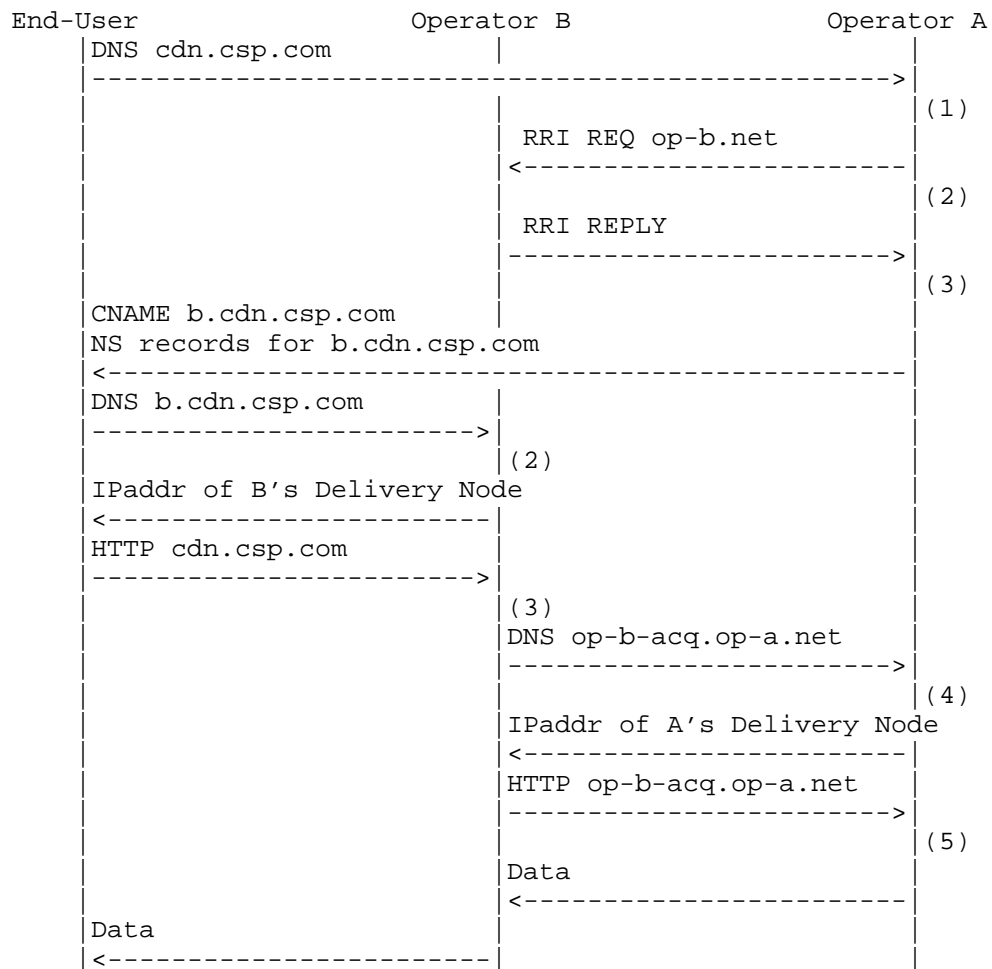


Figure 6: Request Trace for Dynamic Footprint Discovery Example

This example differs from the one in Figure 5 only in the addition of a CDNI Request Routing Interface request (step 2) and corresponding response (step 3). The RRI Req could be a message such as "Can you serve clients from this IP Prefix?" or it could be "Provide the list of client IP prefixes you can currently serve". In either case the response might be cached by operator A to avoid repeatedly asking the same question. Alternatively, or in addition, Operator B may spontaneously advertise to Operator A information (or changes) on the set of requests it is willing and able to serve on behalf of operator A; in that case, Operator B may spontaneously issue RRI REPLY messages that are not in direct response to a corresponding RRI REQ message. (Note that the issues of determining the client's subnet

from DNS requests, as described above, are exactly the same here as in Section 3.4.)

Once Operator A obtains the RRI response, it is now able to determine that Operator B's CDN is an appropriate dCDN for this request and therefore a valid candidate dCDN to consider in its Redirection decision. If that dCDN is selected, the redirection and serving of the request proceeds as before (i.e. in the absence of dynamic footprint discovery).

3.6. Content Removal

The following example illustrates how the Control interface may be used to remove an item of content. In this example, user requests for a particular content, and corresponding redirection of such requests from Operator A to Operator B CDN, may (or may not) have taken place earlier. Then, at some point in time, the uCDN (for example, in response to a corresponding trigger from the Content Provider) uses the Control Interface to request that content identified by a particular URL be removed from dCDN. The following diagram illustrates the operation.

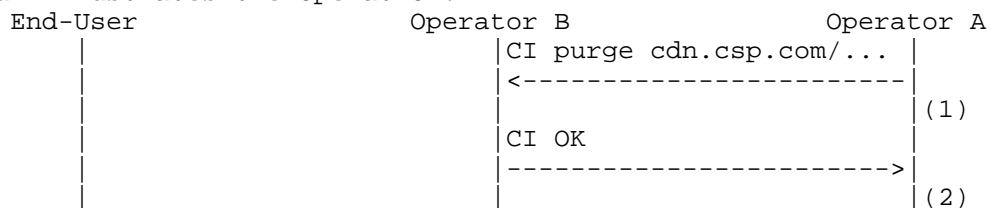


Figure 7: Request Trace for Content Removal

The Control interface is used to convey the request from uCDN to dCDN that some previously acquired content should be deleted. The URL in the request specifies which content to remove. This example corresponds to a DNS-based redirection scenario such as Section 3.4. If HTTP-based redirection had been used, the URL for removal would be of the form peer-a.op-b.net/cdn.csp.com/...

The dCDN is expected to confirm to the uCDN, as illustrated by the CI OK message, the completion of the removal of the targeted content from all the caches in dCDN.

3.7. Pre-Positioned Content Acquisition Example

The following example illustrates how the Control interface may be used to pre-position an item of content in the dCDN. In this example, Operator A uses the Metadata interface to request that

content identified by a particular URL be pre-positioned into Operator B CDN.

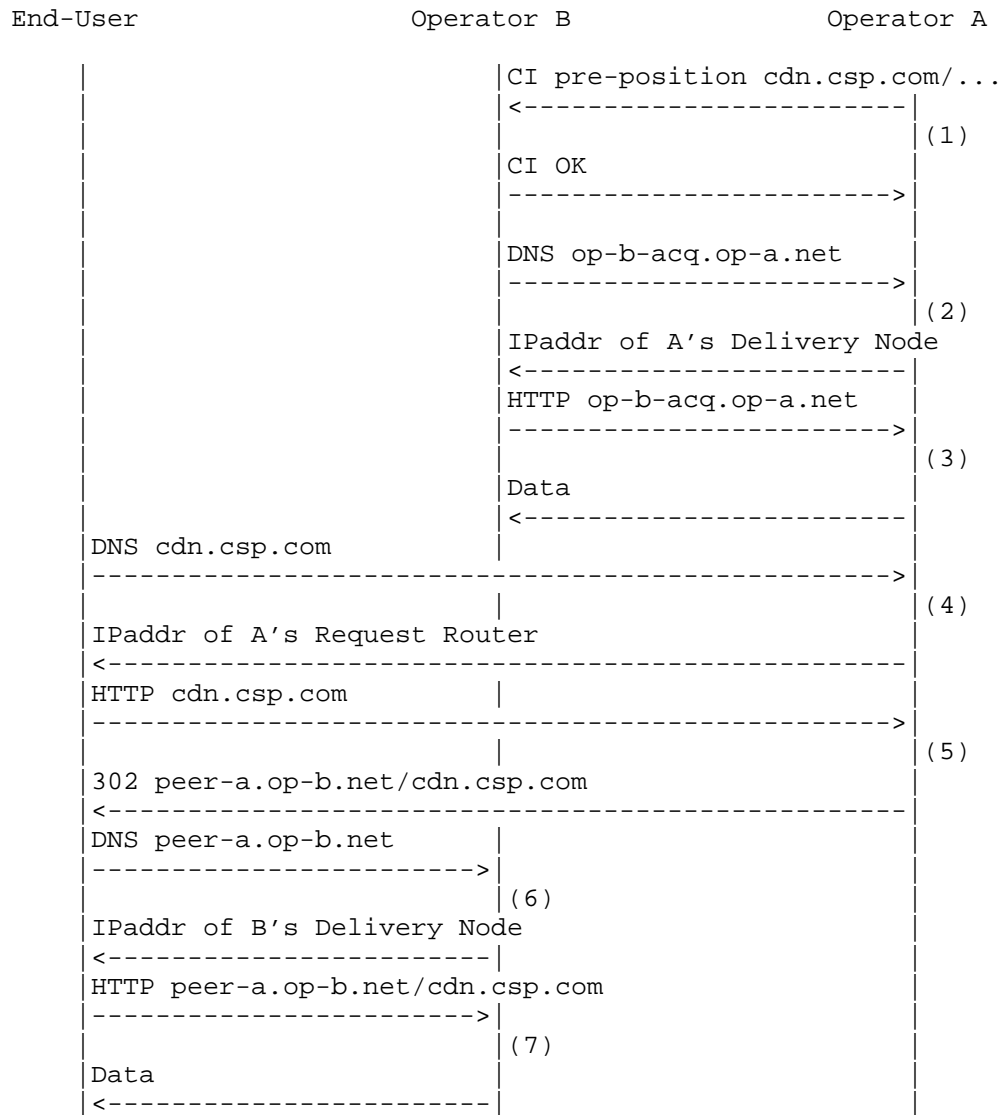


Figure 8: Request Trace for Content Pre-Positioning

The steps illustrated in the figure are as follows:

1. Operator A uses the Control Interface to request that Operator B pre-positions a particular content item identified by its URL.

Operator B responds by confirming that it is willing to perform this operation.

Steps 2 and 3 are exactly the same as steps 5 and 6 of Figure 3, only this time those steps happen as the result of the Pre-positioning request instead of as the result of a cache miss.

Steps 4, 5, 6, 7 are exactly the same as steps 1, 2, 3, 4 of Figure 3, only this time Operator B CDN can serve the end-user request without triggering dynamic content acquisition, since the content has been pre-positioned in dCDN. Note that, depending on dCDN operations and policies, the content pre-positioned in the dCDN may be pre-positioned to all, or a subset of, dCDN caches. In the latter case, intra-CDN dynamic content acquisition may take place inside the dCDN serving requests from caches on which the content has not been pre-positioning; however, such intra-CDN dynamic acquisition would not involve the uCDN.

3.8. Asynchronous CDNI Metadata Example

In this section we walk through a simple example illustrating a scenario of asynchronously exchanging CDNI metadata, where the downstream CDN obtains CDNI metadata for content ahead of a corresponding content request. The example that follows assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used, as in Section 3.3. However, asynchronous exchange of CDNI Metadata is similarly applicable to DNS-based inter-CDN redirection and iterative request routing (in which cases the CDNI metadata may be used at slightly different processing stages of the message flows).

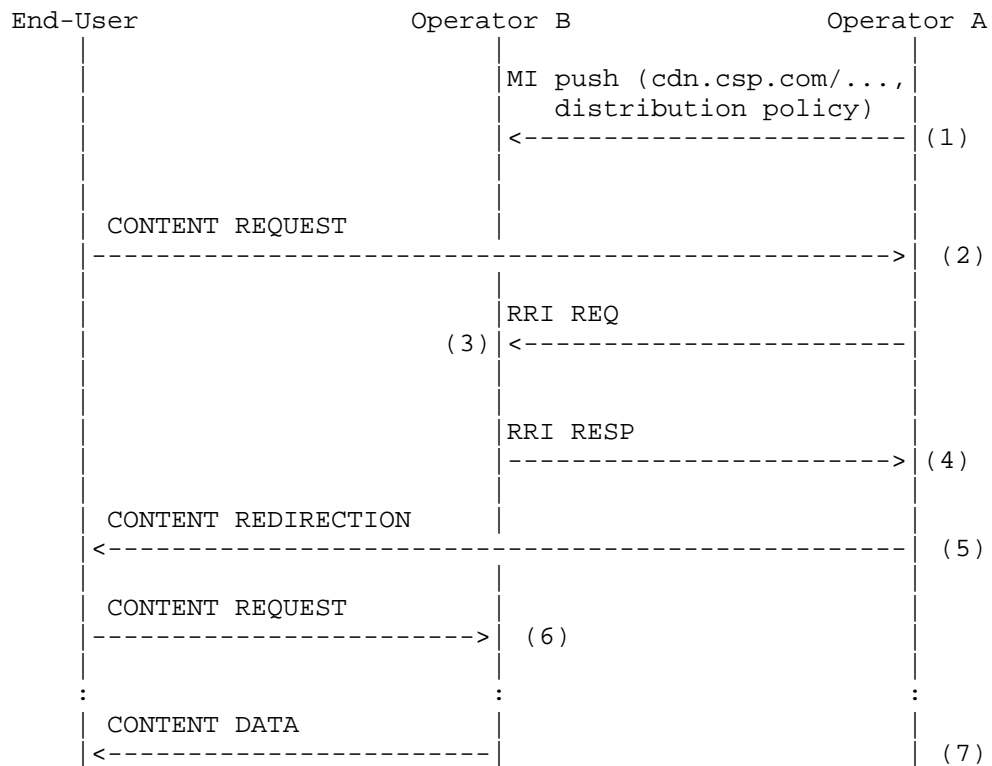


Figure 9: Request Trace for Asynchronous CDNI Metadata

The steps illustrated in the figure are as follows:

- Operator A uses the Metadata Interface to asynchronously push CDNI metadata to Operator B. The present document does not constrain how the CDNI metadata information is actually represented. For the purposes of this example, we assume that Operator A provides CDNI metadata to Operator B indicating that:
 - * this CDNI Metadata is applicable to any content referenced by "cdn.csp.com/op-b.net/..." (assuming HTTP redirection is used - it would be applicable to "cdn.csp.com/..." if DNS redirection were used as in Section 3.4).
 - * this CDNI metadata consists of a distribution policy requiring enforcement by the delivery node of a specific per-request authorization mechanism (e.g. URI signature or token validation).

2. A Content Request occurs as usual.
3. A CDNI Request Routing Request (RRI REQ) is issued by operator A CDN, as discussed in Section 3.3. Operator B's request router can access the CDNI Metadata that are relevant to the requested content and that have been pre-positioned as per Step 1, which may or may not affect the response.
4. Operator B's request router issues a CDNI Request Routing Response (RRI RESP) as in Section 3.3.
5. Operator B performs content redirection as discussed in Section 3.3.
6. On receipt of the Content Request by the end user, the delivery node detects that previously acquired CDNI metadata is applicable to the requested content. In accordance with the specific CDNI metadata of this example, the delivery node will invoke the appropriate per-request authorization mechanism, before serving the content. (Details of this authorization are not shown.)
7. Assuming successful per-request authorization, serving of Content Data (possibly preceded by inter-CDN acquisition) proceeds as in Section 3.3.

3.9. Synchronous CDNI Metadata Acquisition Example

In this section we walk through a simple example illustrating a scenario of synchronous CDNI metadata acquisition, in which the downstream CDN obtains CDNI metadata for content at the time of handling a first request for the corresponding content. As in the preceding section, this example assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used (as in Section 3.3), but dynamic CDNI metadata acquisition is applicable to other variations of request routing.

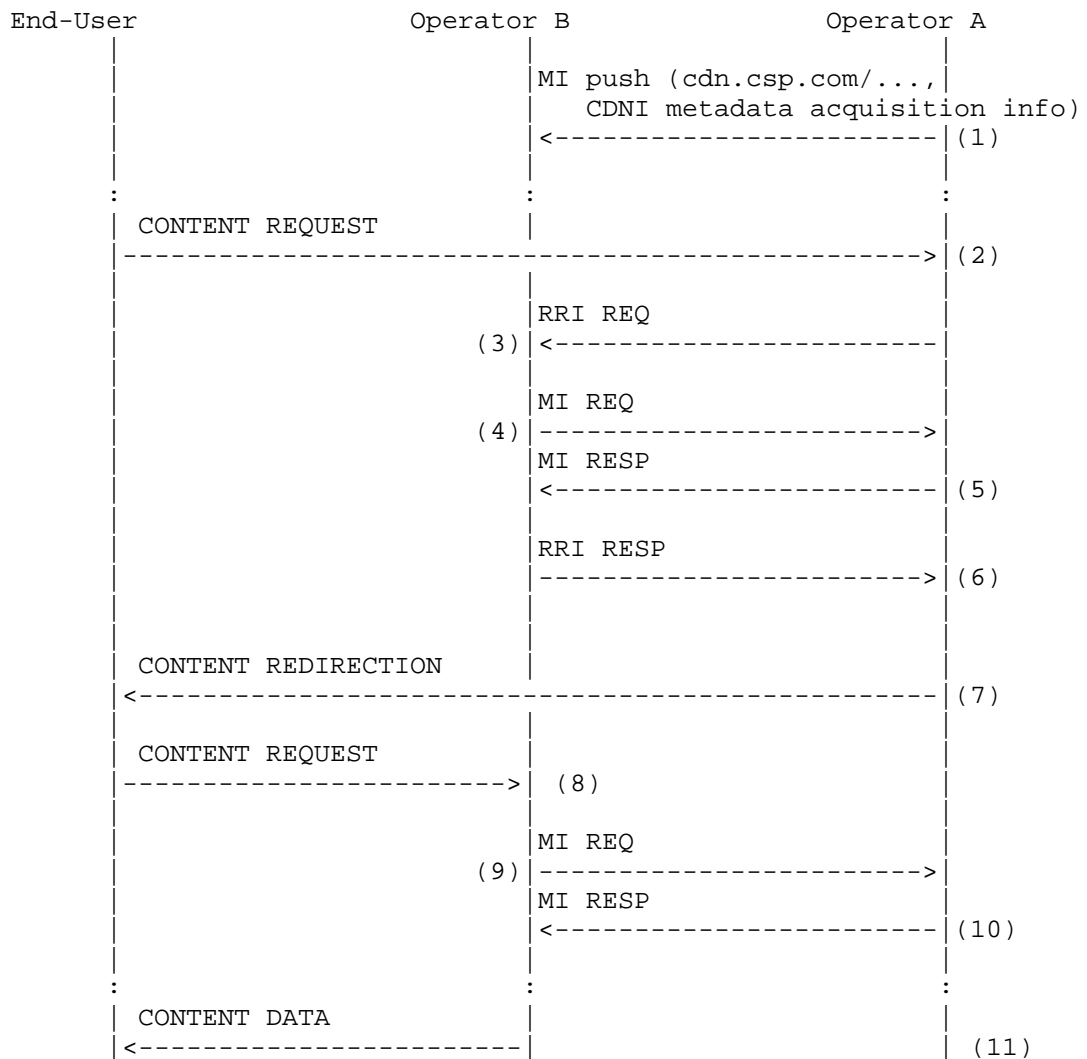


Figure 10: Request Trace for Synchronous CDNI Metadata Acquisition

The steps illustrated in the figure are as follows:

1. Operator A initially uses the Metadata Interface to asynchronously push seed metadata to Operator B. For example, this seed information may include a URI indicating where CDNI Metadata can later be pulled from for some content set. (There are alternative ways that this seeding information may be provided, such as piggybacking on the CDNI RRI REQ message of

Step 3.)

2. A Content Request arrives as normal.
3. A Request Routing Interface request occurs as in the prior example.
4. On receipt of the CDNI Request Routing Request, Operator B's CDN initiates synchronous acquisition of CDNI Metadata that are needed for routing of the end-user request. The seeding information provided in Step 1 is used to determine how to obtain the metadata. Note that there may exist cases in which this step does not occur (e.g., because the CDNI metadata seeding information indicates CDNI metadata are not needed at that stage).
5. On receipt of a CDNI Metadata MI Request, Operator A's CDN responds, making the corresponding CDNI metadata information available to Operator B's CDN. This metadata is considered by operator B's CDN before responding to the Request Routing request. (In a simple case, the metadata could simply be an allow or deny response for this particular request.)
6. Response to the RRI request as normal.
7. Redirection message is sent to the end user.
8. A delivery node of Operator B receives the end user request.
9. The delivery node triggers dynamic acquisition of additional CDNI metadata that are needed to process the end-user content request. Again the seeding information provided in Step 1 is used to determine how to acquire the needed CDNI metadata. Note that there may exist cases where this step need not happen, either because the metadata were already acquired previously, or because the seeding information indicates no metadata are required.
10. Operator A's CDN responds to the CDNI Metadata Request and makes the corresponding CDNI metadata available to Operator B. This metadata influence how Operator B's CDN processes the end-user request.
11. Content is served (possibly preceded by inter-CDN acquisition) as in Section 3.3.

3.10. Content Acquisition with Multiple Upstream CDNs

A single dCDN may receive end-user requests from multiple uCDNs. When a dCDN receives an end-user request, it must determine the identity of the uCDN from which it should acquire the requested content.

Ideally, the acquisition path of an end-user request will follow the redirection path of the request. The dCDN should acquire the content from the same uCDN which redirected the request.

Determining the acquisition path requires the dCDN to reconstruct the redirection path based on information in the end-user request. The method for reconstructing the redirection path differs based on the redirection approach: HTTP or DNS.

With HTTP-redirection, the rewritten URI must include sufficient information for the dCDN to directly or indirectly determine the uCDN when the end-user request is received. The HTTP-redirection approach can be further broken-down based on how the URL is rewritten during redirection: HTTP-redirection with or without Site Aggregation. HTTP-redirection with Site Aggregation hides the identity of the original CSP. HTTP-redirection without Site Aggregation does not attempt to hide the identity of the original CSP. With both approaches, the rewritten URI must include enough information to identify the immediate neighbor uCDN.

With DNS-redirection, the dCDN receives the published URI (instead of a rewritten URI) and does not have sufficient information for the dCDN to identify the appropriate uCDN. Given that the dCDN has established a business relationship with each of its uCDNs, assume that the dCDN can trust any uCDN to acquire the content. The dCDN may narrow the set of viable uCDNs by examining the CDNI metadata from each to determine which uCDNs are hosting metadata for the requested content.

Content acquisition may be preceded by content metadata acquisition. If possible, the acquisition path for metadata should also follow the redirection path. Additionally, metadata must be indexed based on rewritten URIs in the case of HTTP-redirection and must be indexed based on published URIs in the case of DNS-redirection. Thus, the request routing interface and the metadata interface are tightly coupled in that the result of request routing (a rewritten URI pointing to the dCDN) must serve as an input to metadata lookup. If the content metadata includes information for acquiring the content, then the metadata interface is also tightly coupled with the acquisition interface in that the result of the metadata lookup (an acquisition URL likely hosted by the uCDN) should serve as input to

the content acquisition.

4. Main Interfaces

Figure 1 illustrates the four main interfaces that are in scope for the CDNI WG, along with several others. The detailed specifications of these interfaces are left to other documents (mostly still to be written, but see [I-D.ietf-cdni-problem-statement] and [I-D.ietf-cdni-requirements] for some discussion of the interfaces).

One interface that is not shown in Figure 1 is the interface between the user and the CSP. While for the purposes of CDNI that interface is out of scope, it is worth noting that it does exist and can provide useful functions, such as end-to-end performance monitoring and some forms of authentication and authorization.

There is also an important interface between the user and the Request Routing function of both uCDN and dCDN. As we saw in some of the preceding examples, that interface can be used as a way of passing information such as the metadata that is required to obtain the content in dCDN from uCDN.

In this section we will provide an overview of the functions performed by each of the CDNI interfaces and discuss how they fit into the overall solution. We also examine some of the design tradeoffs. We begin with an examination of one such tradeoff that affects all the interfaces - the use of in-band or out-of-band communication.

4.1. In-Band versus Out-of-Band Interfaces

Before getting to the individual interfaces, we observe that there is a high-level design choice for each, involving the use of existing in-band communication channels versus defining new out-of-band interfaces.

It is possible that the information needed to carry out various interconnection functions can be communicated between peer CDNs using existing in-band protocols. The use of HTTP 302 redirect is an example of how certain aspects of request routing can be implemented in-band (embedded in URIs). Note that using existing in-band protocols does not imply that the CDNI interfaces are null; it is still necessary to establish the rules (conventions) by which such protocols are used to implement the various interface functions.

There are other opportunities for in-band communication beyond HTTP redirects. For example, many of the HTTP directives used by proxy

servers can also be used by peer CDNs to inform each other of caching activity. Of these, one that is particularly relevant is the If-Modified-Since directive, which is used with the GET method to make it conditional: if the requested object has not been modified since the time specified in this field, a copy of the object will not be returned, and instead, a 304 (not modified) response will be returned.

4.2. Cross Interface Concerns

Although the CDNI interfaces are largely independent, there are a set of conventions practiced consistently across all interfaces. Most important among these is how resources are named, for example, how the Metadata and Control interfaces identify the set of resources to which a given directive applies, or the Logging interface identifies the set of resources for which a summary record applies.

While in the limit the CDNI interfaces could explicitly identify every individual resource, in practice, they name resource aggregates (sets of URIs) that are to be treated in a similar way. For example, URI aggregates can be identified by a CDN-Domain (i.e., the FQDN at the beginning of a URI) or by a URI-Filter (i.e., a regular expression that matches a subset of URIs contained in some CDN-Doman). In other words, CDN-Domains and URI-Filters provide a uniform means to aggregate sets (and subsets) of URIs for the purpose of defining the scope for some operation in one of the CDNI interfaces.

4.3. Request Routing Interface

We may think of the request routing interface as comprising two parts: the asynchronous advertisement of footprint and capabilities by a dCDN that allows a uCDN to decide whether to redirect particular user requests to that dCDN; and the synchronous operation of actually redirecting a user request. (These are somewhat analogous to the operations of routing and forwarding in IP.)

As illustrated in Section 3, the synchronous part of the request routing interface may be implemented in part by DNS and HTTP. Naming conventions may be established by which CDN peers communicate whether a request should be routed or content served.

In support of these exchanges, it is necessary for CDN peers to exchange additional information with each other. Depending on the method(s) supported, this includes

- o The operator's unique id (operator-id) or distinguished CDN-domain (operator-domain);

- o NS records for the operator's set of externally visible request routers;
- o The set of requests the dCDN operator is prepared to serve (e.g. a set of client IP prefixes or geographic regions that may be served by dCDN).

Of these, the two operator identifiers are fixed, and can be exchanged off-line as part of a peering agreement. The NS records potentially change with some frequency, but an existing protocol--DNS--can be used to dynamically track this information. That is, a peer can do a DNS lookup on operator-domain to retrieve the set of NS records corresponding to the peer's redirection service.

The set of requests that dCDN is willing to serve could in some cases be relatively static (e.g., a set of IP prefixes) which could be exchanged off-line, or might even be negotiated as part of a peering agreement. However, it may also be more dynamic, in which case an explicit protocol for its exchange would be helpful.

A variety of options exist for the dCDN operator to advertise its footprint to uCDN. As discussed in [I-D.previdi-cdni-footprint-advertisement], footprint is comprised of two components:

- o a class of end user requests (represented, for example, by a set of IP prefixes, or a geographic region) that the dCDN is willing and able to serve directly, without use of another dCDN;
- o the connectivity of the dCDN to other CDNs that may be able to serve content to users on behalf of dCDN.

[I-D.previdi-cdni-footprint-advertisement] describes an approach to advertising such footprint information asynchronously using BGP. In addition to this sort of information, a dCDN might also advertise "capabilities" such as the ability to handle certain types of content (e.g. specific streaming formats) or quality of service (QoS) capabilities. [I-D.xiaoyan-cdni-request-routing-protocol] describes an approach that exchanges CDN "capabilities" over HTTP, while [I-D.seedorf-alto-for-cdni] describes how ALTO [RFC5693] may be used to obtain request routing information.

We also note that the Request Routing interface plays a key role in enabling recursive redirection, as illustrated in Section 3.3. It enables the user to be redirected to the correct delivery node in dCDN with only a single redirection step (as seen by the user). This may be particularly valuable as the chain of interconnected CDNs increases beyond two CDNs.

One final issue is how HTTP adaptive streaming impacts the Request Routing interface, and in particular, the interplay between the request router and manifest files, which may contain either absolute or relative URLs. These issues are discussed in more detail in [I-D.brandenburg-cdni-has].

4.4. Logging Interface

It is necessary for the upstream CDN to have visibility into the delivery of content it originates to end-users connected to the downstream CDN. This allows the upstream CDN to properly bill its customers for multiple deliveries of content cached by the downstream CDN, as well as to report accurate traffic statistics to those content providers. This is one role of the Logging interface.

Other operational data that may be relevant to CDNI can also be exchanged by the Logging interface. For example, dCDN may report the amount of content it has acquired from uCDN, and how much cache storage has been consumed by content cached on behalf of uCDN.

Traffic logs are easily exchanged off-line. For example, the following traffic log is a small deviation from the Apache log file format, where entries include the following fields:

- o Domain - the full domain name of the origin server
- o IP address - the IP address of the client making the request
- o End time - the ending time of the transfer
- o Time zone - any time zone modifier for the end time
- o Method - the transfer command itself (e.g., GET, POST, HEAD)
- o URL - the requested URL
- o Version - the protocol version, such as HTTP/1.0
- o Response - a numeric response code indicating transfer result
- o Bytes Sent - the number of bytes in the body sent to the client
- o Request ID - a unique identifier for this transfer
- o User agent - the user agent, if supplied
- o Duration - the duration of the transfer in milliseconds

- o Cached Bytes - the number of body bytes served from the cache
- o Referrer - the referrer string from the client, if supplied

Of these, only the Domain field is indirect in the downstream CDN--it is set to the CDN-domain used by the upstream CDN rather than the actual origin server. This field could then be used to filter traffic log entries so only those entries matching the upstream CDN are reported to the corresponding operator. Further discussion of the Logging interface can be found in [I-D.bertrand-cdni-logging] and [I-D.lefaucheur-cdni-logging-delivery].

One open question is who does the filtering. One option is that the downstream CDN filters its own logs, and passes the relevant records directly to each upstream peer. This requires that the downstream CDN knows the set of CDN-domains that belong to each upstream peer. If this information is already exchanged between peers as part of the request routing interface, then direct peer-to-peer reporting is straightforward. If it is not available, and operators do not wish to advertise the set of CDN-domains they serve to their peers, then the second option is for each CDN to send both its non-local traffic records and the set of CDN-domains it serves to an independent third-party (i.e., a CDN Exchange), which subsequently filters, merges, and distributes traffic records on behalf of each participating CDN operator.

A second open question is how timely traffic information should be. For example, in addition to off-line traffic logs, accurate real-time traffic monitoring might also be useful, but such information requires that the downstream CDN inform the upstream CDN each time it serves upstream content from its cache. The downstream CDN can do this, for example, by sending a conditional HTTP GET request (If-Modified-Since) to the upstream CDN each time it receives an HTTP GET request from one of its end-users. This allows the upstream CDN to record that a request has been issued for the purpose of real-time traffic monitoring. The upstream CDN can also use this information to validate the traffic logs received later from the downstream CDN.

There is obviously a tradeoff between accuracy of such monitoring and the overhead of the downstream CDN having to go back to the upstream CDN for every request.

Another design tradeoff in the Logging interface is the degree of aggregation or summarization of data. One situation that lends itself to summarization is the delivery of HTTP-based adaptive bit-rate video. Most schemes to deliver such video use a large number of relatively small HTTP requests (e.g. one request per two-second chunk of video.) It may be desirable to aggregate logging information so

that a single log entry is provided for the entire video rather than for each chunk. Note however that such aggregation requires a degree of application awareness in dCDN to recognize that the many HTTP requests correspond to a single video. The implications of HTTP adaptive streaming are discussed further in [I-D.brandenburg-cdni-has].

Other forms of aggregation may also be useful. For example, there may be situations where bulk metrics such as bytes delivered per hour may suffice rather than the detailed per-request logs outlined above. It seems likely that a range of granularities of logging will be needed along with ways to specify the type and degree of aggregation required.

4.5. Control Interface

The control interface is initially used to bootstrap the other interfaces. As a simple example, it could be used to provide the address of the logging server in dCDN to uCDN in order to bootstrap the logging interface. It may also be used, for example, to establish security associations for the other interfaces.

The other role the Control interface plays is to allow the uCDN to pre-position, revalidate, or purge metadata and content on a dCDN. These operations, sometimes collectively called the trigger interface, are discussed further in [I-D.murray-cdni-triggers].

4.6. Metadata Interface

The role of the metadata interface is to enable CDNI distribution metadata to be conveyed to the downstream CDN by the upstream CDN. For example, see [I-D.ma-cdni-metadata] and [I-D.cjlmw-cdni-metadata]. Such metadata includes geo-blocking restrictions, availability windows, access control policies, and so on. It may also include policy information such as the desire to pre-position content rather than fetch it on demand.

Some metadata may be able to be conveyed using in-band mechanisms. For example, to inform the downstream CDN of any geo-blocking restrictions or availability windows, the upstream can elect to redirect a request to the downstream CDN only if that CDN's advertised delivery footprint is acceptable for the requested URL. Similarly, the request could be forwarded only if the current time is within the availability window.

Similarly, some forms of access control may also be performed on a per-request basis using HTTP directives. For example, being able to respond to a conditional GET request gives the upstream CDN an

opportunity to influence how the downstream CDN delivers its content. Minimally, the upstream CDN can invalidate (purge) content previously cached by the downstream CDN.

Fine-grain control over how the downstream CDN delivers content on behalf of the upstream CDN is also possible. For example, by including the X-Forwarded-For HTTP header with the conditional GET request, the downstream CDN can report the end-user's IP address to the upstream CDN, giving it an opportunity to control whether the downstream CDN should serve the content to this particular end-user. The upstream CDN would communicate its directive through its response to the conditional GET. The downstream CDN can cache information for a period of time specified by the upstream CDN, thereby reducing control overhead.

All of these in-band techniques serve to illustrate that uCDNs have the option of enforcing their access control policies themselves, rather than delegating enforcement to dCDNs using the Metadata interface. As a consequence, the Metadata interface must provide a means for the uCDN to express this its desire to retain enforcement for itself. For example, this might be done by including a "check with me" flag in the metadata associated with certain content.

5. Deployment Models

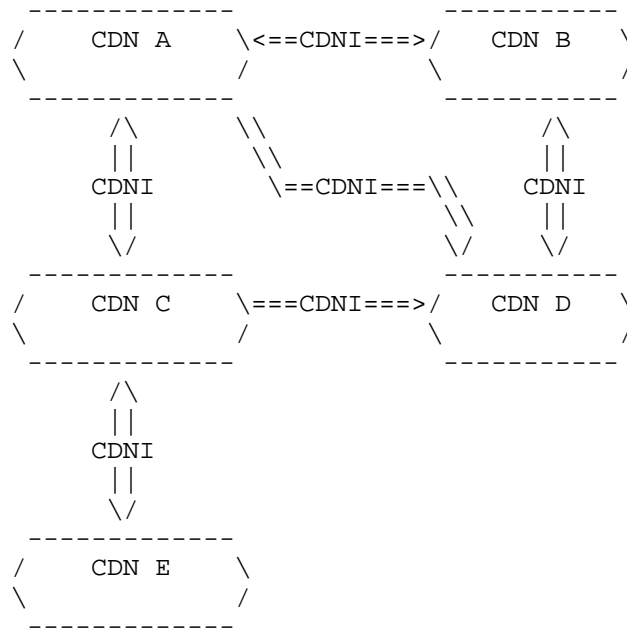
In this section we describe a number of possible deployment models that may be achieved using the CDNI interfaces described above. We note that these models are by no means exhaustive, and that many other models may be possible.

Although the reference model of Figure 1 shows all CDN functions on each side of the CDNI interface, deployments can rely on entities that are involved in any subset of these functions, and therefore only support the relevant subset of CDNI interfaces. As already noted in Section 3, effective CDNI deployments can be built without necessarily implementing all four interfaces. Some examples of such deployments are shown below.

Note that, while we refer to upstream and downstream CDNs, this distinction applies to specific content items and transactions. That is, a given CDN may be upstream for some transactions and downstream for others, depending on many factors such as location of the requesting client and the particular piece of content requested.

5.1. Meshed CDNs

Although the reference model illustrated in Figure 1 shows a unidirectional CDN interconnection with a single uCDN and a single dCDN, any arbitrary CDNI meshing can be built from this, such as the example meshing illustrated in Figure 11. (Support for arbitrary meshing may or may not be in the initial scope for the working group, but the model allows for it.)



```

====>  CDNI interfaces, with right-hand side CDN acting as dCDN
        to left-hand side CDN
<====>  CDNI interfaces, with right-hand side CDN acting as dCDN
        to left-hand side CDN and with left-hand side CDN acting
        as dCDN to right-hand side CDN

```

Figure 11: CDNI Deployment Model: CDN Meshing Example

5.2. CSP combined with CDN

Note that our terminology refers to functional roles and not economic or business roles. That is, a given organization may be operating as both a CSP and a fully-fledged uCDN when we consider the functions performed, as illustrated in Figure 12.

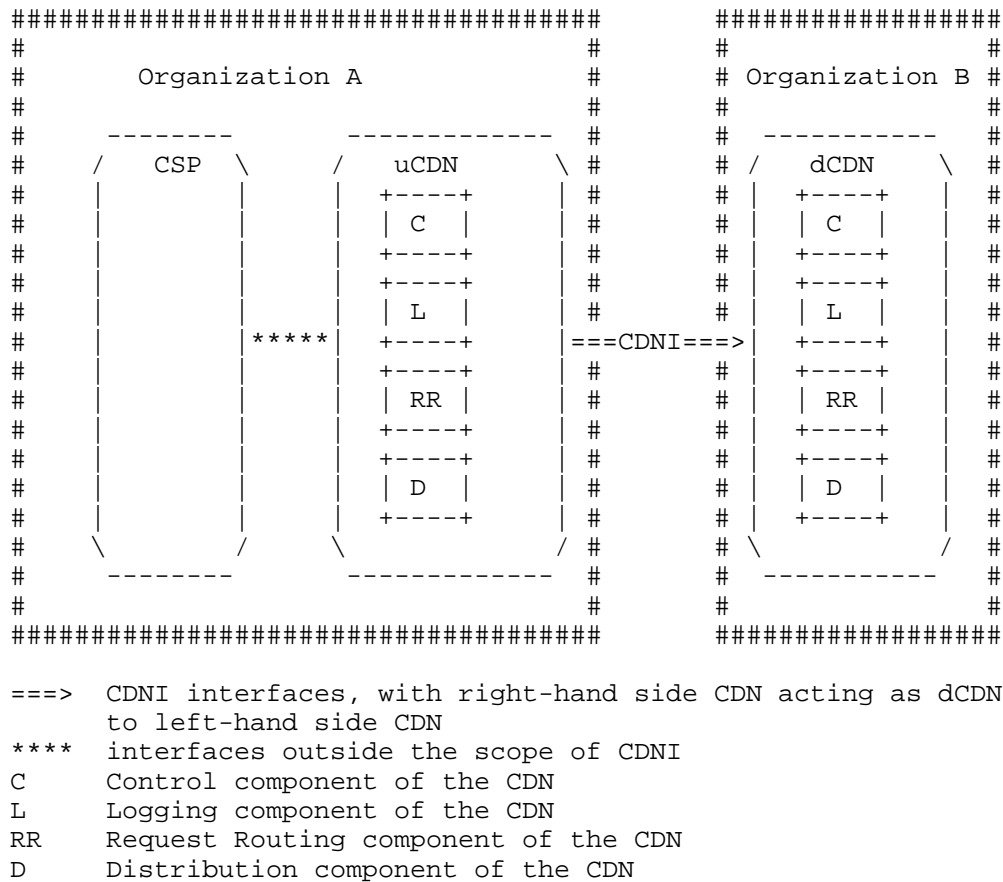


Figure 12: CDNI Deployment Model: Organization combining CSP & uCDN

5.3. CSP using CDNI Request Routing Interface

As another example, a content provider organization may choose to run its own request routing function as a way to select among multiple candidate CDN providers; In this case the content provider may be modeled as the combination of a CSP and of a special, restricted case of a CDN. In that case, as illustrated in Figure 13, the CDNI Request Routing interface can be used between the restricted CDN operated by the content provider Organization and the CDN operated by the full-CDN organization acting as a dCDN in the request routing control plane. Interfaces outside the scope of the CDNI work can be used between the CSP functional entities of the content provider organization and the CDN operated by the full-CDN organization acting as a uCDN) in the CDNI control planes other than the request routing plane (i.e. Control, Distribution, Logging).

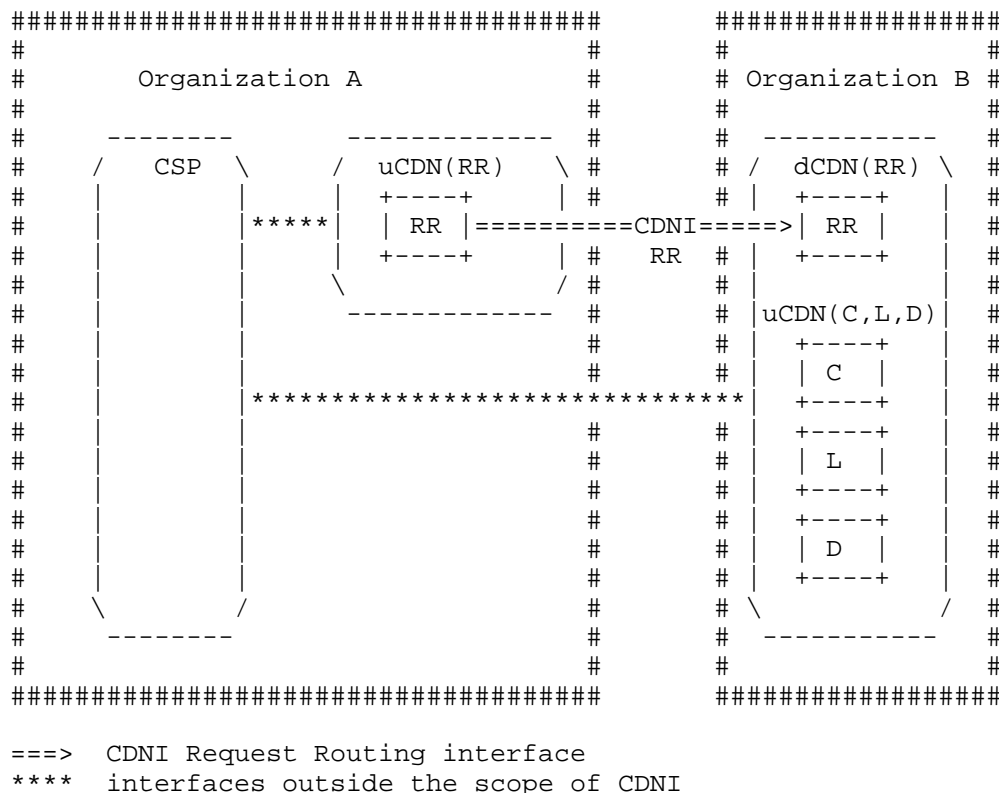


Figure 13: CDNI Deployment Model: Organization combining CSP and partial CDN

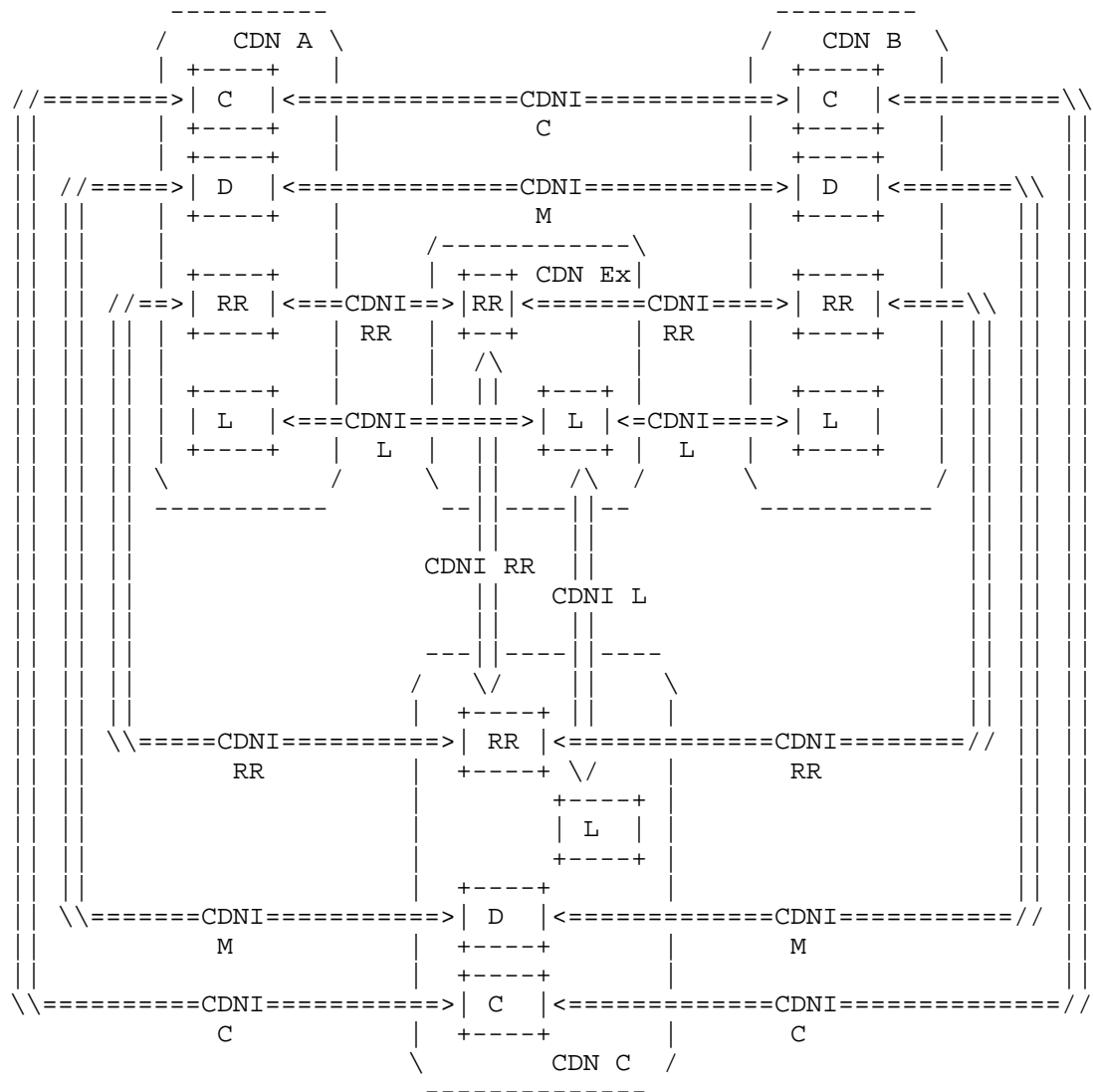
5.4. CDN Federations and CDN Exchanges

There are two additional concepts related to, but distinct from CDN Interconnection. The first is CDN Federation. Our view is that CDNI is the more general concept, involving two or more CDNs serving content to each other's users, while federation implies a multi-lateral interconnection arrangement, but other CDN interconnection agreements are also possible (e.g., symmetric bilateral, asymmetric bilateral). An important conclusion is that CDNI technology should not presume (or bake in) a particular interconnection agreement, but should instead be general enough to permit alternative interconnection arrangements to evolve.

The second concept often used in the context of CDN Federation is CDN Exchange--a third party broker or exchange that is used to facilitate a CDN federation. Our view is that a CDN exchange offers valuable machinery to scale the number of CDN operators involved in a multi-

lateral (federated) agreement, but that this machinery is built on top of the core CDNI interconnection mechanisms. For example, as illustrated in Figure 14, the exchange might aggregate and redistribute information about each CDN footprint and capacity, as well as collect, filter, and re-distribute traffic logs that each participant needs for interconnection settlement, but inter-CDN request routing, inter-CDN content distribution (including inter-CDN acquisition) and inter-CDN control which fundamentally involve a direct interaction between an upstream CDN and a downstream CDN-- operate exactly as in a pair-wise peering arrangement. Turning to Figure 14, we observe that in this example:

- o each CDN supports a direct CDNI Control interface to every other CDN
- o each CDN supports a direct CDNI Metadata interface to every other CDN
- o each CDN supports a CDNI Logging interface with the CDN Exchange
- o each CDN supports both a CDNI request Routing interface with the CDN Exchange (for aggregation and redistribution of dynamic CDN footprint discovery information) and a direct CDNI Request Routing interface to every other CDN (for actual request redirection).



<=CDNI RR=> CDNI Request Routing interface
 <=CDNI M==> CDNI Metadata interface
 <=CDNI C==> CDNI Control interface
 <=CDNI L==> CDNI Logging interface

Figure 14: CDNI Deployment Model: CDN Exchange

Note that a CDN exchange may alternatively support a different set of functionality (e.g. Logging only, or Logging and full request

routing, or all the functionality of a CDN including content distribution). All these options are expected to be allowed by the IETF CDNI specifications.

6. Trust Model

There are a number of trust issues that need to be addressed by a CDNI solution. Many of them are in fact similar or identical to those in a simple CDN without interconnection. In a standard CDN environment (without CDNI), the CSP places a degree of trust in a single CDN operator to perform many functions. The CDN is trusted to deliver content with appropriate quality of experience for the end user. The CSP trusts the CDN operator not to corrupt or modify the content. The CSP often relies on the CDN operator to provide reliable accounting information regarding the volume of delivered content. The CSP may also trust the CDN operator to perform actions such as timely invalidation of content and restriction of access to content based on certain criteria such as location of the user and time of day, and to enforce per-request authorization performed by the CSP using techniques such as URI signing.

A CSP also places trust in the CDN not to distribute any information that is confidential to the CSP (e.g., how popular a given piece of content is) or confidential to the end user (e.g., which content has been watched by which user).

A CSP does not necessarily have to place complete trust in a CDN. A CSP will in some cases take steps to protect its content from improper distribution by a CDN, e.g. by encrypting it and distributing keys in some out of band way. A CSP also depends on monitoring (possibly by third parties) and reporting to verify that the CDN has performed adequately. A CSP may use techniques such as client-based metering to verify that accounting information provided by the CDN is reliable. HTTP conditional requests may be used to provide the CSP with some checks on CDN operation. In other words, while a CSP may trust a CDN to perform some functions in the short term, the CSP is able in most cases to verify whether these actions have been performed correctly and to take action (such as moving the content to a different CDN) if the CDN does not live up to expectations.

The main trust issue raised by CDNI is that it introduces transitive trust. A CDN that has a direct relationship with a CSP can now "outsource" the delivery of content to another (downstream) CDN. That CDN may in turn outsource delivery to yet another downstream CDN, and so on.

The top level CDN in such a chain of delegation is responsible for ensuring that the requirements of the CSP are met. Failure to do so is presumably just as serious as in the traditional single CDN case. Hence, an upstream CDN is essentially trusting a downstream CDN to perform functions on its behalf in just the same way as a CSP trusts a single CDN. Monitoring and reporting can similarly be used to verify that the downstream CDN has performed appropriately. However, the introduction of multiple CDNs in the path between CSP and end user complicates the picture. For example, third party monitoring of CDN performance (or other aspects of operation, such as timely invalidation) might be able to identify the fact that a problem occurred somewhere in the chain but not point to the particular CDN at fault.

In summary, we assume that an upstream CDN will invest a certain amount of trust in a downstream CDN, but that it will verify that the downstream CDN is performing correctly, and take corrective action (including potentially breaking off its relationship with that CDN) if behavior is not correct. We do not expect that the trust relationship between a CSP and its "top level" CDN will differ significantly from that found today in single CDN situations. However, it does appear that more sophisticated tools and techniques for monitoring CDN performance and behavior will be required to enable the identification of the CDN at fault in a particular delivery chain.

We expect that the detailed designs for the specific interfaces for CDNI will need to take the transitive trust issues into account. For example, explicit confirmation that some action (such as content removal) has taken place in a downstream CDN may help to mitigate some issues of transitive trust.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

While there is a variety of security issues introduced by a single CDN, we are concerned here specifically with the additional issues that arise when CDNs are interconnected. For example, when a single CDN has the ability to distribute content on behalf of a CSP, there may be concerns that such content could be distributed to parties who are not authorized to receive it, and there are mechanisms to deal with such concerns. Our focus in this section is on how CDN interconnection introduces new security issues not found in the

single CDN case.

Many of the security issues that arise in CDNI are related to the transitivity of trust (or lack thereof) described in Section 6. As noted above, the design of the various interfaces for CDNI must take account of the additional risks posed by the fact that a CDN with whom a CSP has no direct relationship is now potentially distributing content for that CSP. The mechanisms used to mitigate these risks may be similar to those used in the single CDN case, but their suitability in this more complex environment must be validated.

Another concern that arises in any CDN is that information about the behavior of users (what content they access, how much content they consume, etc.) may be gathered by the CDN. This risk certainly exists in inter-connected CDNs, but it should be possible to apply the same techniques to mitigate it as in the single CDN case.

CDNs today offer a variety of means to control access to content, such as time-of-day restrictions, geo-blocking, and URI signing. These mechanisms must continue to function in CDNI environments, and this consideration is likely to affect the design of certain CDNI interfaces (e.g. metadata, request routing.)

Just as with a single CDN, each peer CDN must ensure that it is not used as an "open proxy" to deliver content on behalf of a malicious CSP. Whereas a single CDN typically addresses this problem by having CSPs explicitly register content (or origin servers) that is to be served, simply propagating this information to peer downstream CDNs may be problematic because it reveals more information than the upstream CDN is willing to specify. (To this end, the content acquisition step in the earlier examples force the dCDN to retrieve content from the uCDN rather than go directly to the origin server.)

There are several approaches to this problem. One is for the uCDN to encode a signed token generated from a shared secret in each URL routed to a dCDN, and for the dCDN to validate the request based on this token. Another one is to have each upstream CDN advertise the set of CDN-domains they serve, where the downstream CDN checks each request against this set before caching and delivering the associated object. Although straightforward, this approach requires operators to reveal additional information, which may or may not be an issue.

8.1. Security of CDNI Interfaces

It is noted in [I-D.ietf-cdni-requirements] that all CDNI interfaces must be able to operate securely over insecure IP networks. Since it is expected that the CDNI interfaces will be implemented using existing application protocols such as HTTP or XMPP, we also expect

that the security mechanisms available to those protocols may be used by the CDNI interfaces. Details of how these interfaces are secured will be specified in the relevant interface documents.

8.2. Digital Rights Management

Issues of digital rights management (DRM, also sometimes called digital restrictions management) is often employed for content distributed via CDNs. In general, DRM relies on the CDN to distribute encrypted content, with decryption keys distributed to users by some other means (e.g. directly from the CSP to the end user.) For this reason, DRM is considered out of scope for the CDNI WG [I-D.ietf-cdni-problem-statement] and does not introduce additional security issues for CDNI.

9. Contributors

The following individuals contributed to this document:

- o Matt Caulfield
- o Francois le Faucheur
- o Aaron Falk
- o David Ferguson
- o John Hartman
- o Ben Niven-Jenkins
- o Kent Leung

10. Acknowledgements

We thank Huw Jones for helpful input to the draft.

11. Informative References

[I-D.bertrand-cdni-logging]
Bertrand, G. and S. Emile, "CDNI Logging Interface",
draft-bertrand-cdni-logging-00 (work in progress),
February 2012.

[I-D.brandenburg-cdni-has]

Brandenburg, R., Deventer, O., Faucheur, F., and K. Leung, "Models for adaptive-streaming-aware CDN Interconnection", draft-brandenburg-cdni-has-03 (work in progress), July 2012.

[I-D.cjlmw-cdni-metadata]

Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., and K. Leung, "CDN Interconnect Metadata", draft-cjlmw-cdni-metadata-00 (work in progress), July 2012.

[I-D.ietf-cdni-problem-statement]

Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-ietf-cdni-problem-statement-08 (work in progress), June 2012.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.

[I-D.ietf-cdni-use-cases]

Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-09 (work in progress), July 2012.

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

Faucheur, F., Viveganandhan, M., and K. Leung, "CDNI Logging Formats for HTTP and HTTP Adaptive Streaming Deliveries", draft-lefaucheur-cdni-logging-delivery-01 (work in progress), July 2012.

[I-D.ma-cdni-metadata]

Ma, K., "Content Distribution Network Interconnection (CDNI) Metadata Interface", draft-ma-cdni-metadata-03 (work in progress), July 2012.

[I-D.murray-cdni-triggers]

Murray, R. and B. Niven-Jenkins, "CDN Interconnect Triggers", draft-murray-cdni-triggers-00 (work in progress), February 2012.

[I-D.previdi-cdni-footprint-advertisement]

Previdi, S., Faucheur, F., Faucheur, F., Medved, J., and L. Faucheur, "CDNI Footprint Advertisement",

draft-previdi-cdni-footprint-advertisement-01 (work in progress), March 2012.

[I-D.seedorf-alto-for-cdni]

Seedorf, J., "ALTO for CDNi Request Routing",
draft-seedorf-alto-for-cdni-00 (work in progress),
October 2011.

[I-D.vandergaast-edns-client-subnet]

Contavalli, C., Gaast, W., Leach, S., and E. Lewis,
"Client subnet in DNS requests",
draft-vandergaast-edns-client-subnet-01 (work in
progress), April 2012.

[I-D.xiaoyan-cdni-request-routing-protocol]

He, X., Dawkins, S., Li, J., and G. Chen, "Request Routing
Protocol for CDN Interconnection",
draft-xiaoyan-cdni-request-routing-protocol-00 (work in
progress), October 2011.

[RFC3466] Day, M., Cain, B., Tomlinson, G., and P. Rzewski, "A Model
for Content Internetworking (CDI)", RFC 3466,
February 2003.

[RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic
Optimization (ALTO) Problem Statement", RFC 5693,
October 2009.

Authors' Addresses

Larry Peterson (editor)
Verivue, Inc.
2 Technology Park Drive
Westford, MA
USA

Phone: +1 978 303 8032
Email: lpeterson@verivue.com

Bruce Davie
Nicira Networks, Inc.
3460 W. Bayshore Rd.
Palo Alto, CA 94303
USA

Email: bsd@nicira.com

Network Working Group
Internet-Draft
Obsoletes: 3466 (if approved)
Intended status: Informational
Expires: December 8, 2014

L. Peterson
Akamai Technologies, Inc.
B. Davie
VMware, Inc.
R. van Brandenburg, Ed.
TNO
June 6, 2014

Framework for CDN Interconnection
draft-ietf-cdni-framework-14

Abstract

This document presents a framework for Content Distribution Network Interconnection (CDNI). The purpose of the framework is to provide an overall picture of the problem space of CDNI and to describe the relationships among the various components necessary to interconnect CDNs. CDN Interconnection requires the specification of interfaces and mechanisms to address issues such as request routing, distribution metadata exchange, and logging information exchange across CDNs. The intent of this document is to outline what each interface needs to accomplish, and to describe how these interfaces and mechanisms fit together, while leaving their detailed specification to other documents. This document, in combination with RFC 6707, obsoletes RFC 3466.

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 December 8, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Reference Model	5
1.3. Structure Of This Document	9
2. Building Blocks	9
2.1. Request Redirection	9
2.1.1. DNS Redirection	9
2.1.2. HTTP Redirection	11
3. Overview of CDNI Operation	11
3.1. Preliminaries	13
3.2. Iterative HTTP Redirect Example	14
3.3. Recursive HTTP Redirection Example	19
3.4. Iterative DNS-based Redirection Example	23
3.4.1. Notes on using DNSSEC	27
3.5. Dynamic Footprint Discovery Example	28
3.6. Content Removal Example	30
3.7. Pre-Positioned Content Acquisition Example	31
3.8. Asynchronous CDNI Metadata Example	32
3.9. Synchronous CDNI Metadata Acquisition Example	34
3.10. Content and Metadata Acquisition with Multiple Upstream CDNs	36
4. Main Interfaces	37
4.1. In-Band versus Out-of-Band Interfaces	38
4.2. Cross Interface Concerns	38
4.3. Request Routing Interfaces	39
4.4. CDNI Logging Interface	40
4.5. CDNI Control Interface	42
4.6. CDNI Metadata Interface	42
4.7. HTTP Adaptive Streaming Concerns	43
4.8. URI Rewriting	44
5. Deployment Models	45

5.1. Meshed CDNs	46
5.2. CSP combined with CDN	47
5.3. CSP using CDNI Request Routing Interface	47
5.4. CDN Federations and CDN Exchanges	48
6. Trust Model	51
7. IANA Considerations	52
8. Privacy Considerations	52
9. Security Considerations	53
9.1. Security of CDNI Interfaces	54
9.2. Digital Rights Management	54
10. Contributors	54
11. Acknowledgements	54
12. Informative References	55
Authors' Addresses	56

1. Introduction

This document provides an overview of the various components necessary to interconnect CDNs, expanding on the problem statement and use cases introduced in [RFC6770] and [RFC6707]. It describes the necessary interfaces and mechanisms in general terms and outlines how they fit together to form a complete system for CDN Interconnection. Detailed specifications are left to other documents. This document makes extensive use of message flow examples to illustrate the operation of interconnected CDNs, but these examples should be considered illustrative rather than prescriptive.

[RFC3466] uses different terminology and models for "Content Internetworking (CDI)". It is also less prescriptive in terms of interfaces. To avoid confusion, this document obsoletes [RFC3466].

1.1. Terminology

This document uses the core terminology defined in [RFC6707]. It also introduces the following terms:

CDN-Domain: a host name (FQDN) at the beginning of a URL (excluding port and scheme), representing a set of content that is served by a given CDN. For example, in the URL `http://cdn.csp.example/...rest of url...`, the CDN domain is `cdn.csp.example`. A major role of CDN-Domain is to identify a region (subset) of the URI space relative to which various CDN Interconnection rules and policies are to apply. For example, a record of CDN Metadata might be defined for the set of resources corresponding to some CDN-Domain.

Distinguished CDN-Domain: a CDN-Domain that is allocated by a CDN for the purposes of communication with a peer CDN, but which is not found

in client requests. Such CDN-Domains may be used for inter-CDN acquisition, or as redirection targets, and enable a CDN to distinguish a request from a peer CDN from an end-user request.

Delivering CDN: the CDN that ultimately delivers a piece of content to the end-user. The last in a potential sequence of downstream CDNs.

Iterative CDNI Request Redirection: When an upstream CDN elects to redirect a request towards a downstream CDN, the upstream CDN can base its redirection purely on a local decision (and without attempting to take into account how the downstream CDN may in turn redirect the user agent). In that case, the upstream CDN redirects the request to the request routing system in the downstream CDN, which in turn will decide how to redirect that request: this approach is referred to as "Iterative" CDNI Request Redirection.

Recursive CDNI Request Redirection: When an upstream CDN elects to redirect a request towards a downstream CDN, the upstream CDN can query the downstream CDN Request Routing system via the CDNI Request Routing Redirection Interface (or use information cached from earlier similar queries) to find out how the downstream CDN wants the request to be redirected. This allows the upstream CDN to factor in the downstream CDN response when redirecting the user agent. This approach is referred to as "Recursive" CDNI Request Redirection. Note that the downstream CDN may elect to have the request redirected directly to a Surrogate inside the downstream CDN, or to any other element in the downstream CDN (or in another CDN) to handle the redirected request appropriately.

Synchronous CDNI operations: operations between CDNs that happen during the process of servicing a user request, i.e. between the time that the user agent begins its attempt to obtain content and the time at which that request is served.

Asynchronous CDNI operations: operations between CDNs that happen independently of any given user request, such as advertisement of footprint information or pre-positioning of content for later delivery.

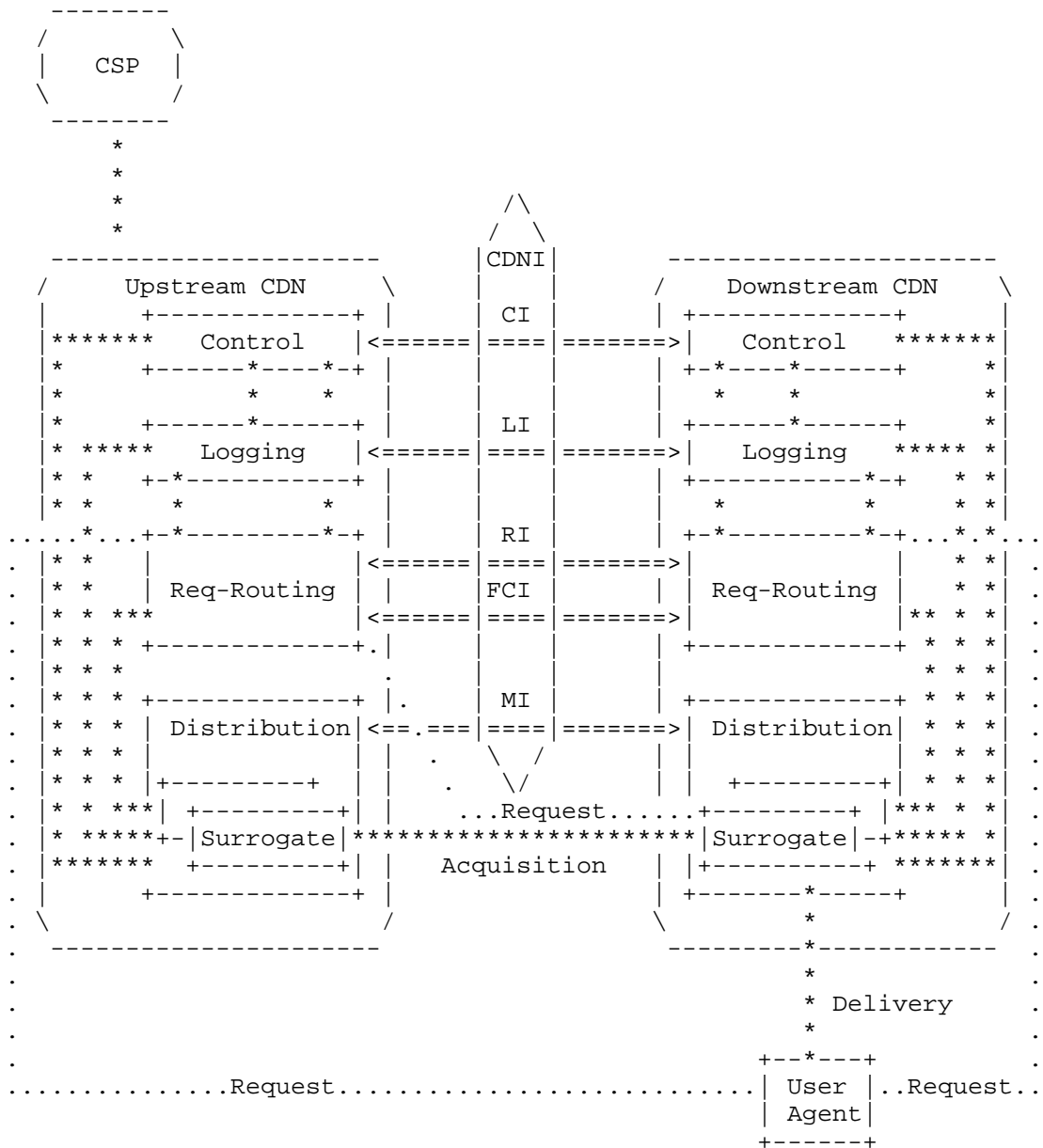
Trigger Interface: a subset of the CDNI Control interface that includes operations to pre-position, revalidate, and purge both metadata and content. These operations are typically called in response to some action (Trigger) by the Content Service Provider (CSP) on the upstream CDN.

We also sometimes use uCDN and dCDN as shorthand for upstream CDN and downstream CDN (see [RFC6707]), respectively.

At various points in this document, the concept of a CDN footprint is used. For a discussion on what constitutes a CDN footprint, the reader is referred to [I-D.ietf-cdni-footprint-capabilities-semantics].

1.2. Reference Model

This document uses the reference model in Figure 1, which expands the reference model originally defined in [RFC6707]. (The difference is that the expanded model splits the Request Routing Interface into its two distinct parts: the Request Routing Redirection interface and the Footprint and Capabilities Advertisement interface, as described below.)



\Leftrightarrow interfaces inside the scope of CDNI

**** and interfaces outside the scope of CDNI

Figure 1: CDNI Expanded Model and CDNI Interfaces

We note that while some interfaces in the reference model are "out of scope" for the CDNI WG (in the sense that there is no need to define new protocols for those interfaces) we still need to refer to them in this document to explain the overall operation of CDNI.

We also note that, while we generally show only one upstream CDN serving a given CSP, it is entirely possible that multiple uCDNs can serve a single CSP. In fact, this situation effectively exists today in the sense that a single CSP can currently delegate its content delivery to more than one CDN.

The following briefly describes the five CDNI interfaces, paraphrasing the definitions given in [RFC6707]. We discuss these interfaces in more detail in Section 4.

- o CDNI Control interface (CI): Operations to bootstrap and parameterize the other CDNI interfaces, as well as operations to pre-position, revalidate, and purge both metadata and content. The latter subset of operations is sometimes collectively called the "Trigger interface."
- o CDNI Request Routing interface: Operations to determine what CDN (and optionally what surrogate within a CDN) is to serve end-user's requests. This interface is actually a logical bundling of two separate but related interfaces:
 - * CDNI Footprint & Capabilities Advertisement interface (FCI): Asynchronous operations to exchange routing information (e.g., the network footprint and capabilities served by a given CDN) that enables CDN selection for subsequent user requests; and
 - * CDNI Request Routing Redirection interface (RI): Synchronous operations to select a delivery CDN (surrogate) for a given user request.
- o CDNI Metadata interface (MI): Operations to communicate metadata that governs how the content is delivered by interconnected CDNs. Examples of CDNI metadata include geo-blocking directives, availability windows, access control mechanisms, and purge directives. It may include a combination of:
 - * Asynchronous operations to exchange metadata that govern subsequent user requests for content; and
 - * Synchronous operations that govern behavior for a given user request for content.

- o CDNI Logging interface (LI): Operations that allow interconnected CDNs to exchange relevant activity logs. It may include a combination of:
 - * Real-time exchanges, suitable for runtime traffic monitoring; and
 - * Offline exchanges, suitable for analytics and billing.

The division between the sets of Trigger-based operations in the CDNI Control interface and the CDNI Metadata interface is somewhat arbitrary. For both cases, the information passed from the upstream CDN to the downstream CDN can broadly be viewed as metadata that describes how content is to be managed by the downstream CDN. For example, the information conveyed by CI to pre-position, revalidate or purge metadata is similar to the information conveyed by posting updated metadata via the MI. Even the CI operation to purge content could be viewed as a metadata update for that content: purge simply says that the availability window for the named content ends now. The two interfaces share much in common, so minimally, there will need to be a consistent data model that spans both.

The distinction we draw has to do with what the uCDN knows about the successful application of the metadata by the dCDN. In the case of the CI, the downstream CDN returning a successful status message guarantees that the operation has been successfully completed; e.g., the content has been purged or pre-positioned. This implies that the downstream CDN accepts responsibility for having successfully completed the requested operation. In contrast, metadata passed between CDNs via the MI carries no such completion guarantee. Returning success implies successful receipt of the metadata, but nothing can be inferred about precisely when the metadata will take effect in the downstream CDN, only that it will take effect eventually. This is because of the challenge in globally synchronizing updates to metadata with end-user requests that are currently in progress (or indistinguishable from currently being in progress). Clearly, a CDN will not be viewed as a trusted peer if "eventually" often becomes an indefinite period of time, but the acceptance of responsibility cannot be as crisply defined for the MI.

Finally, there is a practical issue that impacts all of the CDNI interfaces, and that is whether or not to optimize CDNI for HTTP Adaptive Streaming (HAS). We highlight specific issues related to delivering HAS content throughout this document, but for a more thorough treatment of the topic, see [RFC6983].

1.3. Structure Of This Document

The remainder of this document is organized as follows:

- o Section 2 describes some essential building blocks for CDNI, notably the various options for redirecting user requests to a given CDN.
- o Section 3 provides a number of illustrative examples of various CDNI operations.
- o Section 4 describes the functionality of the main CDNI interfaces.
- o Section 5 shows how various deployment models of CDNI may be achieved using the defined interfaces.
- o Section 6 describes the trust model of CDNI and the issues of transitive trust in particular that CDNI raises.

2. Building Blocks

2.1. Request Redirection

At its core, CDN Interconnection requires the redirection of requests from one CDN to another. For any given request that is received by an upstream CDN, it will either respond to the request directly, or somehow redirect the request to a downstream CDN. Two main mechanisms are available for redirecting a request to a downstream CDN. The first leverages the DNS name resolution process and the second uses application-layer redirection mechanisms such as the HTTP 302 or RTSP 302 redirection responses. While there exists a large variety of application-layer protocols that include some form of redirection mechanism, this document will use HTTP (and HTTPS) in its examples. Similar mechanisms can be applied to other application-layer protocols. What follows is a short discussion of both DNS- and HTTP-based redirection, before presenting some examples of their use in Section 3.

2.1.1. DNS Redirection

DNS redirection is based on returning different IP addresses for the same DNS name, for example, to balance server load or to account for the client's location in the network. A DNS server, sometimes called the Local DNS (LDNS), resolves DNS names on behalf of an end-user. The LDNS server in turn queries other DNS servers until it reaches the authoritative DNS server for the CDN-Domain. The network operator typically provides the LDNS server, although the user is free to choose other DNS servers (e.g., OpenDNS, Google Public DNS).

This latter possibility is important because the authoritative DNS server sees only the IP address of the DNS server that queries it, not the IP address of the original end-user.

The advantage of DNS redirection is that it is completely transparent to the end user; the user sends a DNS name to the LDNS server and gets back an IP address. On the other hand, DNS redirection is problematic because the DNS request comes from the LDNS server, not the end-user. This may affect the accuracy of server selection that is based on the user's location. The transparency of DNS redirection is also a problem in that there is no opportunity to take the attributes of the user agent or the URI path component into account. We consider two main forms of DNS redirection: simple and CNAME-based.

In simple DNS redirection, the authoritative DNS server for the name simply returns an IP address from a set of possible IP addresses. The answer is chosen from the set based on characteristics of the set (e.g., the relative loads on the servers) or characteristics of the client (e.g., the location of the client relative to the servers). Simple redirection is straightforward. The only caveats are (1) there is a limit to the number of alternate IP addresses a single DNS server can manage; and (2) DNS responses are cached by downstream servers so the TTL on the response must be set to an appropriate value so as to preserve the freshness of the redirection.

In CNAME-based DNS redirection, the authoritative server returns a CNAME response to the DNS request, telling the LDNS server to restart the name lookup using a new name. A CNAME is essentially a symbolic link in the DNS namespace, and like a symbolic link, redirection is transparent to the client; the LDNS server gets the CNAME response and re-executes the lookup. Only when the name has been resolved to an IP address does it return the result to the user. Note that DNAME would be preferable to CNAME if it becomes widely supported.

One of the advantages of DNS redirection compared to HTTP redirection is that it can be cached, reducing load on the redirecting CDN's DNS server. However, this advantage can also be a drawback, especially when a given DNS resolver doesn't strictly adhere to the TTL, which is a known problem in some real world environments. In such cases, an end-user might end up at a dCDN without first having passed through the uCDN, which might be an undesirable scenario from a uCDN point of view.

2.1.2. HTTP Redirection

HTTP redirection makes use of the redirection response of the HTTP protocol (e.g., "302" or "307"). This response contains a new URL that the application should fetch instead of the original URL. By changing the URL appropriately, the server can cause the user to redirect to a different server. The advantages of HTTP redirection are that (1) the server can change the URL fetched by the client to include, for example, both the DNS name of the particular server to use, as well as the original HTTP server that was being accessed; (2) the client sends the HTTP request to the server, so that its IP address is known and can be used in selecting the server; and (3) other attributes (e.g., content type, user agent type) are visible to the redirection mechanism.

Just as is the case for DNS redirection, there are some potential disadvantages of using HTTP redirection. For example, it may affect application behavior, e.g. web browsers will not send cookies if the URL changes to a different domain. In addition, although this might also be an advantage, results of HTTP redirection are not cached so that all redirections must go through to the uCDN.

3. Overview of CDNI Operation

To provide a big picture overview of the various components of CDN Interconnection, we walk through a "day in the life" of a content item that is made available via a pair of interconnected CDNs. This will serve to illustrate many of the functions that need to be supported in a complete CDNI solution. We give examples using both DNS-based and HTTP-based redirection. We begin with very simple examples and then show how additional capabilities, such as recursive request redirection and content removal, might be added.

Before walking through the specific examples, we present a high-level view of the operations that may take place. This high-level overview is illustrated in Figure 2. Note that most operations will involve only a subset of all the messages shown below, and that the order and number of operations may vary considerably, as the more detailed examples illustrate.

The following shows Operator A as the upstream CDN (uCDN) and Operator B as the downstream CDN (dCDN), where the former has a relationship with a content provider and the latter being the CDN selected by Operator A to deliver content to the end-user. The interconnection relationship may be symmetric between these two CDN operators, but each direction can be considered as operating independently of the other so for simplicity we show the interaction in one direction only.

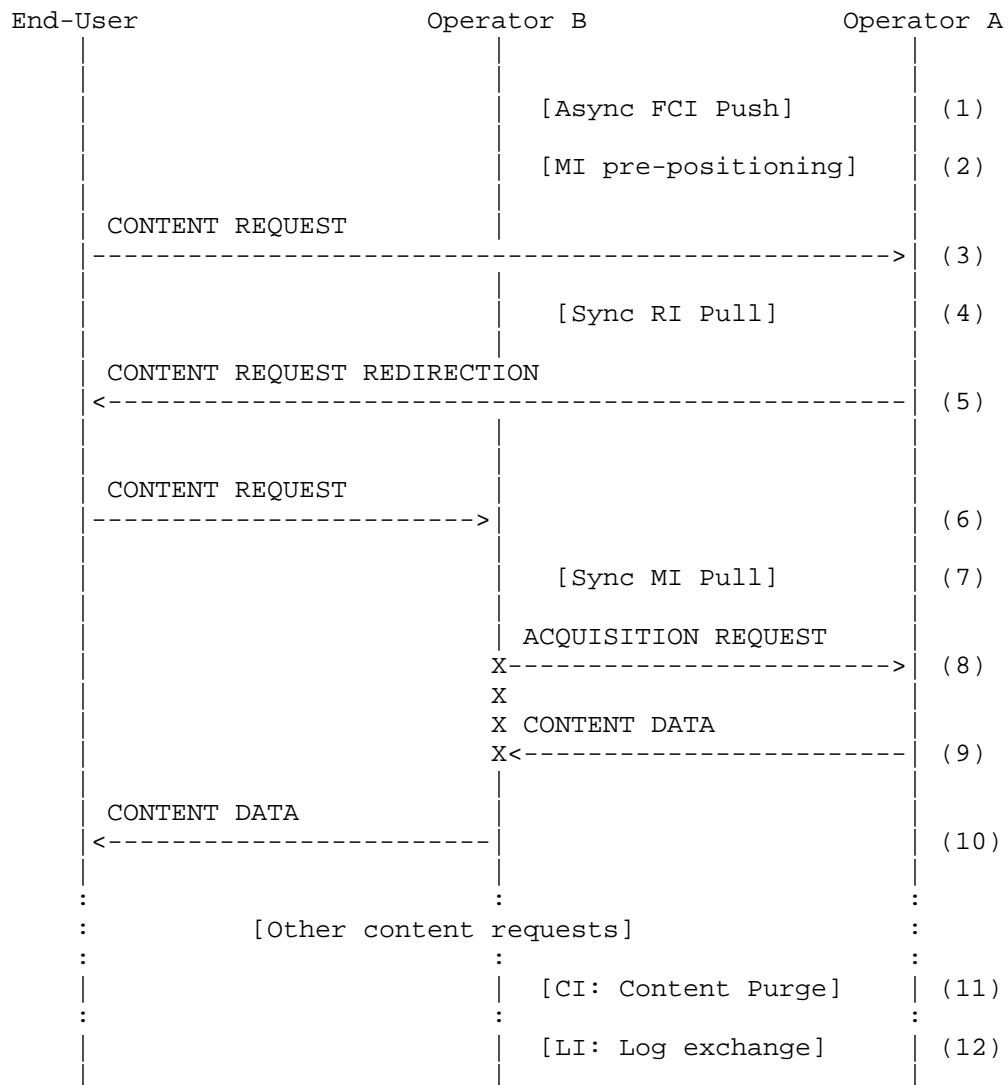


Figure 2: Overview of Operation

The operations shown in the Figure are as follows:

1. dCDN uses the FCI to advertise information relevant to its delivery footprint and capabilities prior to any content requests being redirected.

2. Prior to any content request, the uCDN uses the MI to pre-position CDNI metadata to the dCDN, thereby making that metadata available in readiness for later content requests.
3. A content request from a user agent arrives at uCDN.
4. uCDN may use the RI to synchronously request information from dCDN regarding its delivery capabilities to decide if dCDN is a suitable target for redirection of this request.
5. uCDN redirects the request to dCDN by sending some response (DNS, HTTP) to the user agent.
6. The user agent requests the content from dCDN.
7. dCDN may use the MI to synchronously request metadata related to this content from uCDN, e.g. to decide whether to serve it.
8. If the content is not already in a suitable cache in dCDN, dCDN may acquire it from uCDN.
9. The content is delivered to dCDN from uCDN.
10. The content is delivered to the user agent by dCDN.
11. Some time later, perhaps at the request of the CSP (not shown) uCDN may use the CI to instruct dCDN to purge the content, thereby ensuring it is not delivered again.
12. After one or more content delivery actions by dCDN, a log of delivery actions may be provided to uCDN using the LI.

The following sections show some more specific examples of how these operations may be combined to perform various delivery, control and logging operations across a pair of CDNs.

3.1. Preliminaries

Initially, we assume that there is at least one CSP that has contracted with an upstream CDN (uCDN) to deliver content on its behalf. We are not particularly concerned with the interface between the CSP and uCDN, other than to note that it is expected to be the same as in the "traditional" (non-interconnected) CDN case. Existing mechanisms such as DNS CNAMEs or HTTP redirects (Section 2) can be used to direct a user request for a piece of content from the CSP towards the CSP's chosen upstream CDN.

We assume Operator A provides an upstream CDN that serves content on behalf of a CSP with CDN-Domain `cdn.csp.example`. We assume that Operator B provides a downstream CDN. An end user at some point makes a request for URL

`http://cdn.csp.example/...rest of url...`

It may well be the case that `cdn.csp.example` is just a CNAME for some other CDN-Domain (such as `csp.op-a.example`). Nevertheless, the HTTP request in the examples that follow is assumed to be for the example URL above.

Our goal is to enable content identified by the above URL to be served by the CDN of operator B. In the following sections we will walk through some scenarios in which content is served, as well as other CDNI operations such as the removal of content from a downstream CDN.

3.2. Iterative HTTP Redirect Example

In this section we walk through a simple, illustrative example using HTTP redirection from uCDN to dCDN. The example also assumes the use of HTTP redirection inside uCDN and dCDN; however, this is independent of the choice of redirection approach across CDNs, so an alternative example could be constructed still showing HTTP redirection from uCDN to dCDN but using DNS for handling of request inside each CDN.

We assume for this example that Operators A and B have established an agreement to interconnect their CDNs, with A being upstream and B being downstream.

The operators agree that a CDN-Domain `peer-a.op-b.example` will be used as the target of redirections from uCDN to dCDN. We assume the name of this domain is communicated by some means to each CDN. (This could be established out-of-band or via a CDNI interface.) We refer to this domain as a "distinguished" CDN-Domain to convey the fact that its use is limited to the interconnection mechanism; such a domain is never used directly by a CSP.

We assume the operators also agree on some distinguished CDN-Domain that will be used for inter-CDN acquisition of CSP's content from uCDN by dCDN. In this example, we'll use `op-b-acq.op-a.example`.

We assume the operators also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical

region or a set of IP address prefixes. This information may again be provided out of band or via a defined CDNI interface.

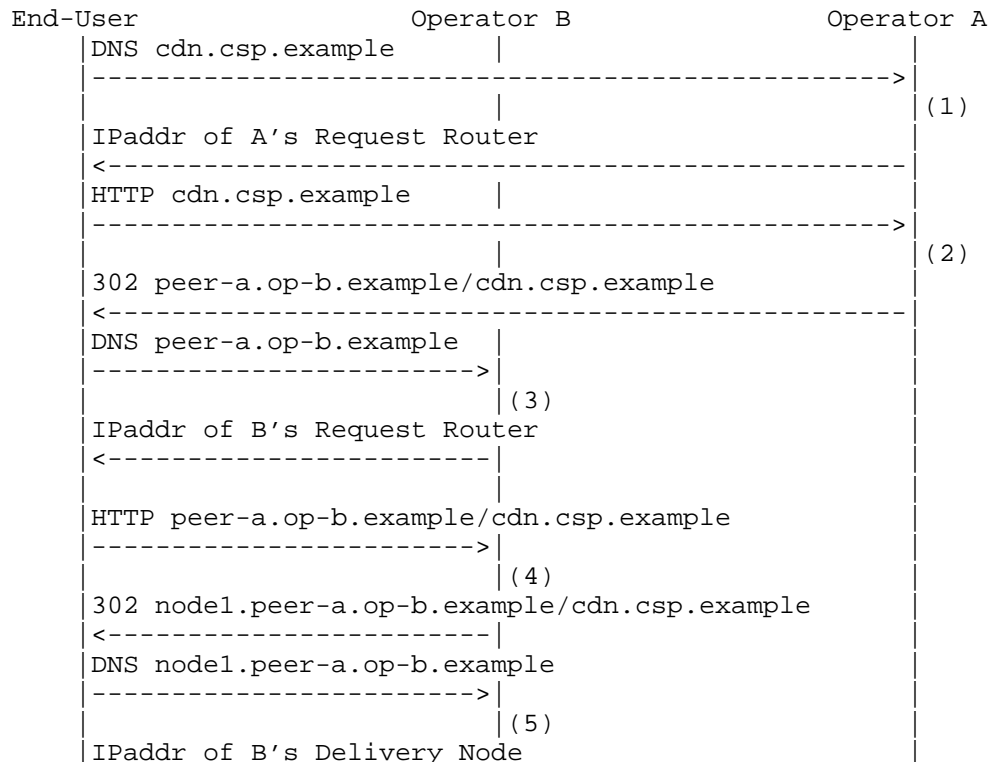
We assume DNS is configured in the following way:

- o The content provider is configured to make operator A the authoritative DNS server for `cdn.csp.example` (or to return a CNAME for `cdn.csp.example` for which operator A is the authoritative DNS server).
- o Operator A is configured so that a DNS request for `op-b-acq.op-a.example` returns a request router in Operator A.
- o Operator B is configured so that a DNS request for `peer-a.op-b.example/cdn.csp.example` returns a request router in Operator B.

Figure 3 illustrates how a client request for

`http://cdn.csp.example/...rest of url...`

is handled.



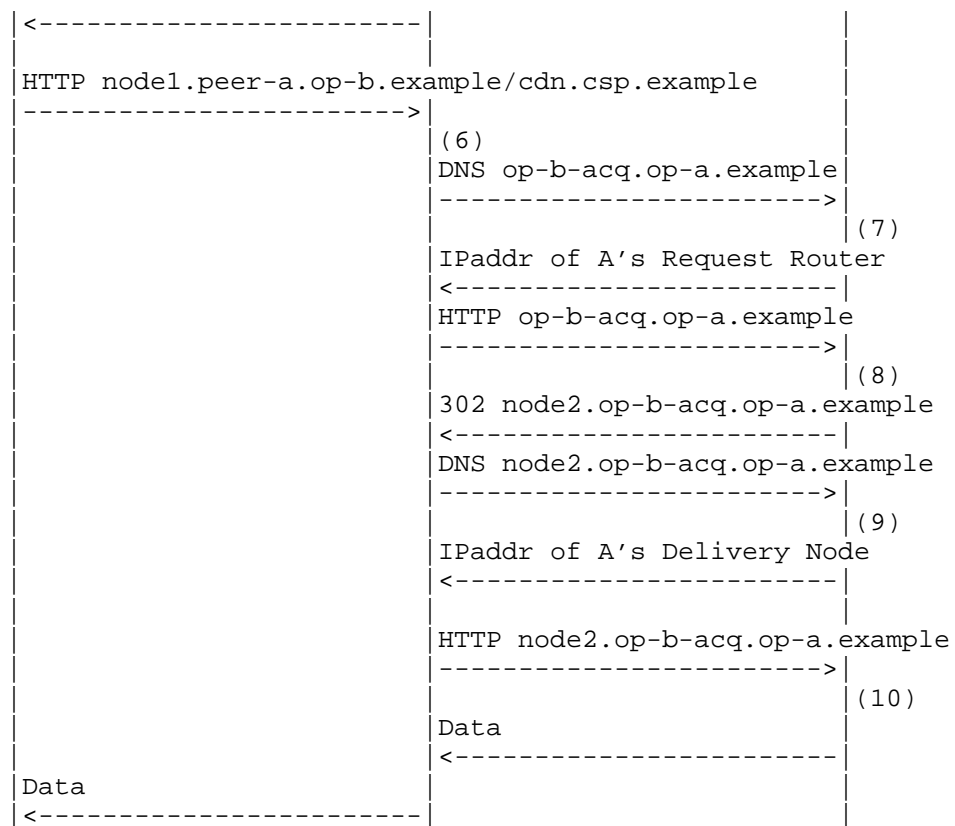


Figure 3: Message Flow for Iterative HTTP Redirection

The steps illustrated in the figure are as follows:

1. A DNS resolver for Operator A processes the DNS request for its customer based on CDN-Domain `cdn.csp.example`. It returns the IP address of a request router in Operator A.
2. A Request Router for Operator A processes the HTTP request and recognizes that the end-user is best served by another CDN, specifically one provided by Operator B, and so it returns a 302 redirect message for a new URL constructed by "stacking" Operator B's distinguished CDN-Domain (`peer-a.op-b.example`) on the front of the original URL. (Note that more complex URL manipulations are possible, such as replacing the initial CDN-Domain by some opaque handle.)
3. The end-user does a DNS lookup using Operator B's distinguished CDN-Domain (`peer-a.op-b.example`). B's DNS resolver returns the

IP address of a request router for Operator B. Note that if request routing within dCDN was performed using DNS instead of HTTP redirection, B's DNS resolver would also behave as the request router and directly return the IP address of a delivery node.

4. The request router for Operator B processes the HTTP request and selects a suitable delivery node to serve the end-user request, and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator B's distinguished CDN-Domain that points to the selected delivery node.
5. The end-user does a DNS lookup using Operator B's delivery node subdomain (node1.peer-a.op-b.example). B's DNS resolver returns the IP address of the delivery node.
6. The end-user requests the content from B's delivery node. In the case of a cache hit, steps 6, 7, 8, 9 and 10 below do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDN from uCDN (not the CSP). The distinguished CDN-Domain peer-a.op-b.example indicates to dCDN that this content is to be acquired from uCDN; stripping the CDN-Domain reveals the original CDN-Domain cdn.csp.example and dCDN may verify that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an inter-CDN acquisition CDN-Domain as agreed above (in this case, op-b-acq.op-a.example).
7. Operator A's DNS resolver processes the DNS request and returns the IP address of a request router in operator A.
8. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (op-b-acq.op-a.example). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.

9. Operator A DNS resolver processes the DNS request and returns the IP address of the delivery node in operator A.
10. Operator B requests (acquires) the content from Operator A. Although not shown, Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

The main advantage of this design is that it is simple: each CDN need only know the distinguished CDN-Domain for each peer, with the upstream CDN "pushing" the downstream CDN-Domain onto the URL as part of its redirect (step 2) and the downstream CDN "popping" its CDN-Domain off the URL to expose a CDN-Domain that the upstream CDN can correctly process. Neither CDN needs to be aware of the internal structure of the other's URLs. Moreover, the inter-CDN redirection is entirely supported by a single HTTP redirect; neither CDN needs to be aware of the other's internal redirection mechanism (i.e., whether it is DNS or HTTP based).

One disadvantage is that the end-user's browser is redirected to a new URL that is not in the same domain of the original URL. This has implications on a number of security or validation mechanisms sometimes used on endpoints. For example, it is important that any redirected URL be in the same domain (e.g., csp.example) if the browser is expected to send any cookies associated with that domain. As another example, some video players enforce validation of a cross domain policy that needs to accommodate the domains involved in the CDN redirection. These problems are generally solvable, but the solutions complicate the example, so we do not discuss them further in this document.

We note that this example begins to illustrate some of the interfaces that may be required for CDNI, but does not require all of them. For example, obtaining information from dCDN regarding the set of client IP addresses or geographic regions it might be able to serve is an aspect of request routing (specifically of the CDNI Footprint & Capabilities Advertisement interface). Important configuration information such as the distinguished names used for redirection and inter-CDN acquisition could also be conveyed via a CDNI interface (e.g., perhaps the CDNI Control interface). The example also shows how existing HTTP-based methods suffice for the acquisition interface. Arguably, the absolute minimum metadata required for CDNI is the information required to acquire the content, and this information was provided "in-band" in this example by means of the URI handed to the client in the HTTP 302 response. The example also

assumes that the CSP does not require any distribution policy (e.g. time window, geo-blocking) or delivery processing to be applied by the interconnected CDNs. Hence, there is no explicit CDNI Metadata interface invoked in this example. There is also no explicit CDNI Logging interface discussed in this example.

We also note that the step of deciding when a request should be redirected to dCDN rather than served by uCDN has been somewhat glossed over. It may be as simple as checking the client IP address against a list of prefixes, or it may be considerably more complex, involving a wide range of factors, such as the geographic location of the client (perhaps determined from a third party service), CDN load, or specific business rules.

This example uses the "iterative" CDNI request redirection approach. That is, uCDN performs part of the request redirection function by redirecting the client to a request router in the dCDN, which then performs the rest of the redirection function by redirecting to a suitable surrogate. If request routing is performed in the dCDN using HTTP redirection, this translates in the end-user experiencing two successive HTTP redirections. By contrast, the alternative approach of "recursive" CDNI request redirection effectively coalesces these two successive HTTP redirections into a single one, sending the end-user directly to the right delivery node in the dCDN. This "recursive" CDNI request routing approach is discussed in the next section.

While the example above uses HTTP, the iterative HTTP redirection mechanism would work over HTTPS in a similar fashion. In order to make sure an end-user's HTTPS request is not downgraded to HTTP along the redirection path, it is necessary for every request router along the path from the initial uCDN Request Router to the final surrogate in the dCDN to respond to an incoming HTTPS request with an HTTP Redirect containing an HTTPS URL. It should be noted that using HTTPS will have the effect of increasing the total redirection process time and increasing the load on the request routers, especially when the redirection path includes many redirects and thus many TLS/SSL sessions. In such cases, a recursive HTTP redirection mechanism, as described in an example in the next section, might help to reduce some of these issues.

3.3. Recursive HTTP Redirection Example

The following example builds on the previous one to illustrate the use of the request routing interface (specifically the CDNI Request Routing Redirection interface) to enable "recursive" CDNI request routing. We build on the HTTP-based redirection approach because it illustrates the principles and benefits clearly, but it is equally

possible to perform recursive redirection when DNS-based redirection is employed.

In contrast to the prior example, the operators need not agree in advance on a CDN-Domain to serve as the target of redirections from uCDN to dCDN. We assume that the operators agree on some distinguished CDN-Domain that will be used for inter-CDN acquisition of CSP's content by dCDN. In this example, we'll use op-b-acq.op-a.example.

We assume the operators also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical region or a set of IP address prefixes. This information may again be provided out of band or via a defined protocol.

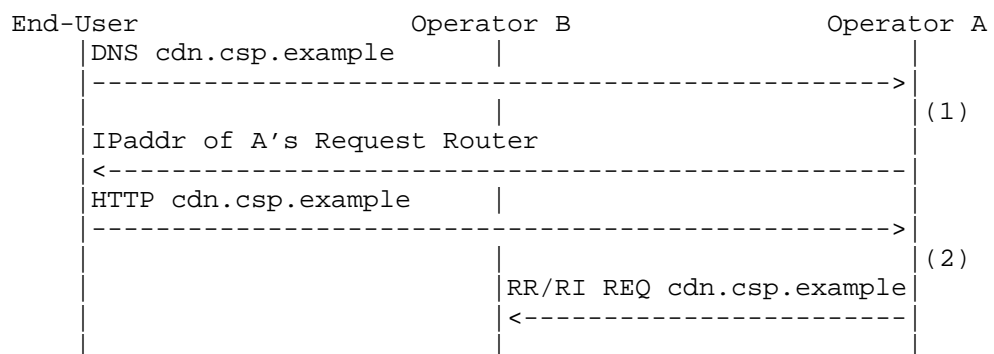
We assume DNS is configured in the following way:

- o The content provider is configured to make operator A the authoritative DNS server for cdn.csp.example (or to return a CNAME for cdn.csp.example for which operator A is the authoritative DNS server).
- o Operator A is configured so that a DNS request for op-b-acq.op-a.example returns a request router in Operator A.
- o Operator B is configured so that a request for node1.op-b.example/cdn.csp.example returns the IP address of a delivery node. Note that there might be a number of such delivery nodes.

Figure 3 illustrates how a client request for

http://cdn.csp.example/...rest of url...

is handled.



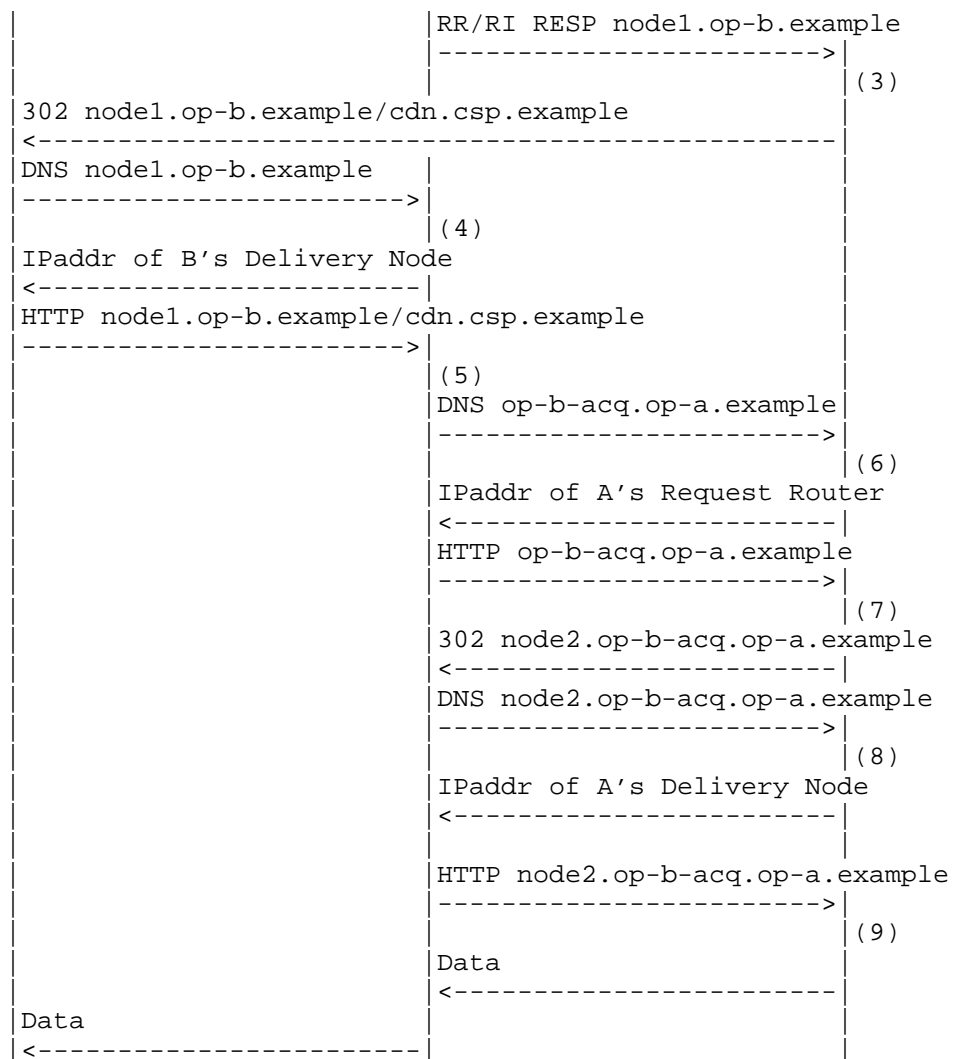


Figure 4: Message Flow for Recursive HTTP Redirection

The steps illustrated in the figure are as follows:

1. A DNS resolver for Operator A processes the DNS request for its customer based on CDN-Domain `cdn.csp.example`. It returns the IP address of a Request Router in Operator A.
2. A Request Router for Operator A processes the HTTP request and recognizes that the end-user is best served by another CDN--specifically one provided by Operator B--and so it queries the

CDNI Request Routing Redirection interface of Operator B, providing a set of information about the request including the URL requested. Operator B replies with the DNS name of a delivery node.

3. Operator A returns a 302 redirect message for a new URL obtained from the RI.
4. The end-user does a DNS lookup using the host name of the URL just provided (node1.op-b.example). B's DNS resolver returns the IP address of the corresponding delivery node. Note that, since the name of the delivery node was already obtained from B using the RI, there should not be any further redirection here (in contrast to the iterative method described above.)
5. The end-user requests the content from B's delivery node, potentially resulting in a cache miss. In the case of a cache miss, the content needs to be acquired from uCDN (not the CSP.) The distinguished CDN-Domain op-b.example indicates to dCDN that this content is to be acquired from another CDN; stripping the CDN-Domain reveals the original CDN-Domain cdn.csp.example, dCDN may verify that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for the inter-CDN Acquisition "distinguished" CDN-Domain as agreed above (in this case, op-b-acq.op-a.example).
6. Operator A DNS resolver processes the DNS request and returns the IP address of a request router in operator A.
7. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (op-b-acq.op-a.example). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.
8. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-Domain) and so returns the address of a delivery node. (Note that without this specially defined internal domain, Operator A would be at risk of

redirecting the request back to Operator B, resulting in an infinite loop.)

9. Operator B requests (acquires) the content from Operator A. Operator A serves content for the requested CDN-Domain to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

Recursive redirection has the advantage over iterative of being more transparent from the end-user's perspective, but the disadvantage of each CDN exposing more of its internal structure (in particular, the addresses of edge caches) to peer CDNs. By contrast, iterative redirection does not require dCDN to expose the addresses of its edge caches to uCDN.

This example happens to use HTTP-based redirection in both CDN A and CDN B, but a similar example could be constructed using DNS-based redirection in either CDN. Hence, the key point to take away here is simply that the end user only sees a single redirection of some type, as opposed to the pair of redirections in the prior (iterative) example.

The use of the RI requires that the request routing mechanism be appropriately configured and bootstrapped, which is not shown here. More discussion on the bootstrapping of interfaces is provided in Section 4

3.4. Iterative DNS-based Redirection Example

In this section we walk through a simple example using DNS-based redirection for request redirection from uCDN to dCDN (as well as for request routing inside dCDN and uCDN). As noted in Section 2.1, DNS-based redirection has certain advantages over HTTP-based redirection (notably, it is transparent to the end-user) as well as some drawbacks (notably the client IP address is not visible to the request router).

As before, Operator A has to learn the set of requests that dCDN is willing or able to serve (e.g. which client IP address prefixes or geographic regions are part of the dCDN footprint). We assume Operator B has and makes known to Operator A some unique identifier that can be used for the construction of a distinguished CDN-Domain, as shown in more detail below. (This identifier strictly needs only to be unique within the scope of Operator A, but a globally unique

identifier, such as an AS number assigned to B, is one easy way to achieve that.) Also, Operator A obtains the NS records for Operator B's externally visible redirection servers. Also, as before, a distinguished CDN-Domain, such as `op-b-acq.op-a.example`, must be assigned for inter-CDN acquisition.

We assume DNS is configured in the following way:

- o The CSP is configured to make Operator A the authoritative DNS server for `cdn.csp.example` (or to return a CNAME for `cdn.csp.example` for which operator A is the authoritative DNS server).
- o When uCDN sees a request best served by dCDN, it returns CNAME and NS records for `"b.cdn.csp.example"`, where "b" is the unique identifier assigned to Operator B. (It may, for example, be an AS number assigned to Operator B.)
- o dCDN is configured so that a request for `"b.cdn.csp.example"` returns a delivery node in dCDN.
- o uCDN is configured so that a request for `"op-b-acq.op-a.example"` returns a delivery node in uCDN.

Figure 5 depicts the exchange of DNS and HTTP requests. The main differences from Figure 3 are the lack of HTTP redirection and transparency to the end-user.

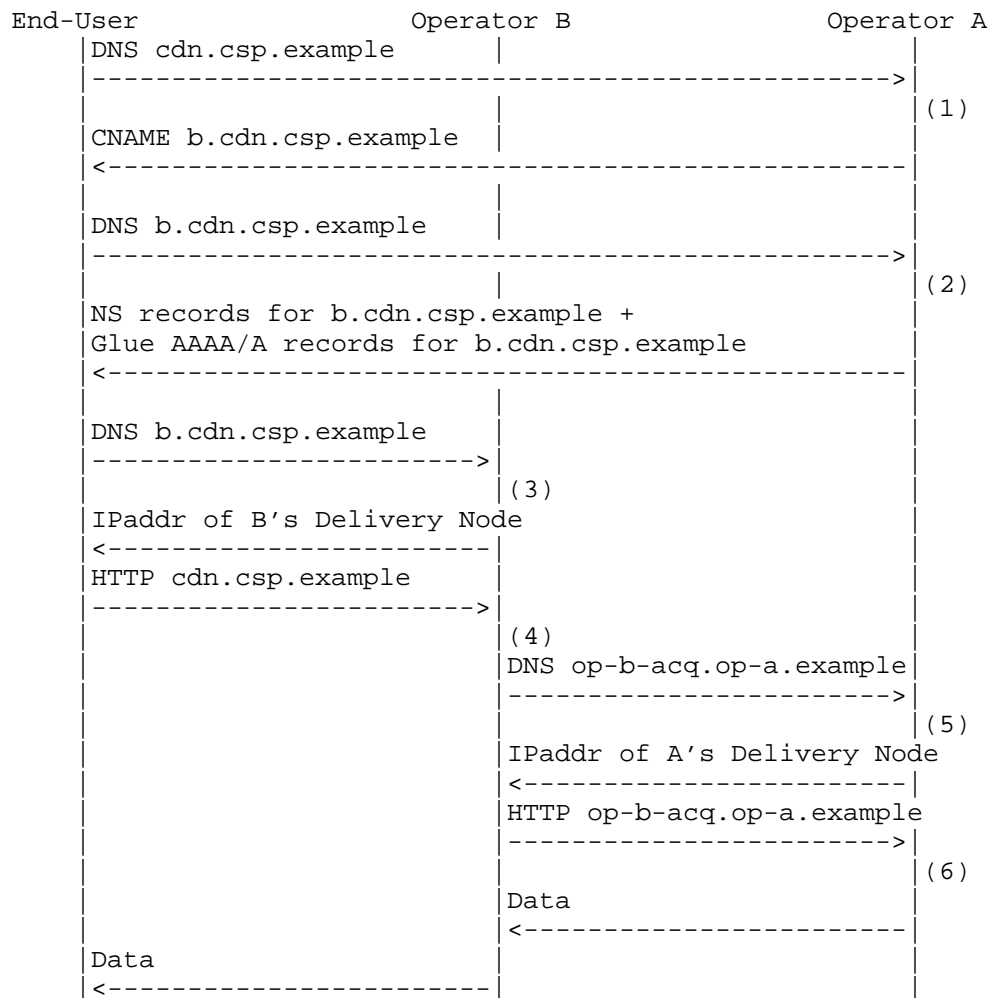


Figure 5: Message Flow for DNS-based Redirection

The steps illustrated in the figure are as follows:

1. Request Router for Operator A processes the DNS request for CDN-Domain `cdn.csp.example` and recognizes that the end-user is best served by another CDN. (This may depend on the IP address of the user's local DNS resolver, or other information discussed below.) The Request Router returns a DNS CNAME response by "stacking" the distinguished identifier for Operator B onto the original CDN-Domain (e.g., `b.cdn.csp.example`).

2. The end-user sends a DNS query for the modified CDN-Domain (i.e. b.cdn.csp.example) to Operator A's DNS server. The Request Router for Operator A processes the DNS request and return a delegation to b.cdn.csp.example by sending an NS record plus glue AAAA/A records pointing to Operator B's DNS server. (This extra step is necessary since typical DNS implementation won't follow an NS record when it is sent together with a CNAME record, thereby necessitating a two-step approach).
3. The end-user sends a DNS query for the modified CDN-Domain (i.e., b.cdn.csp.example) to Operator B's DNS server, using the NS and AAAA/A records received in step 2. This causes B's Request Router to respond with a suitable delivery node.
4. The end-user requests the content from B's delivery node. The requested URL contains the name cdn.csp.example. (Note that the returned CNAME does not affect the URL.) At this point the delivery node has the correct IP address of the end-user and can do an HTTP 302 redirect if the redirections in steps 2 and 3 were incorrect. Otherwise B verifies that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an "internal" CDN-Domain as agreed above (op-b-acq.op-a.example).
5. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-Domain) and so returns the address of a delivery node in uCDN.
6. Operator A serves content to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server.

The advantages of this approach are that it is more transparent to the end-user and requires fewer round trips than HTTP-based redirection (in its worst case, i.e., when none of the needed DNS information is cached). A potential problem is that the upstream CDN depends on being able to learn the correct downstream CDN that serves the end-user from the client address in the DNS request. In standard DNS operation, uCDN will only obtain the address of the client's local DNS resolver (LDNS), which is not guaranteed to be in the same network (or geographic region) as the client. If not--e.g., the end-user uses a global DNS service--then the upstream CDN cannot determine the appropriate downstream CDN to serve the end-user. In this case, and assuming the uCDN is capable of detecting that situation, one option is for the upstream CDN to treat the end-user as it would any user not connected to a peer CDN. Another option is

for the upstream CDN to "fall back" to a pure HTTP-based redirection strategy in this case (i.e., use the first method). Note that this problem affects existing CDNs that rely on DNS to determine where to redirect client requests, but the consequences are arguably less serious for CDNI since the LDNS is likely in the same network as the dCDN serves.

As with the prior example, this example partially illustrates the various interfaces involved in CDNI. Operator A could learn dynamically from Operator B the set of prefixes or regions that B is willing and able to serve via the CDNI Footprint & Capabilities Advertisement interface. The distinguished name used for acquisition and the identifier for Operator B that is prepended to the CDN-Domain on redirection are examples of information elements that might also be conveyed by CDNI interfaces (or, alternatively, statically configured). As before, minimal metadata sufficient to obtain the content is carried "in-band" as part of the redirection process, and standard HTTP is used for inter-CDN acquisition. There is no explicit CDNI Logging interface discussed in this example.

3.4.1. Notes on using DNSSEC

Although it is possible to use DNSSEC in combination with the Iterative DNS-based Redirection mechanism explained above, it is important to note that the uCDN might have to sign records on the fly, since the CNAME returned, and thus the signature provided, can potentially be different for each incoming query. Although there is nothing preventing a uCDN from performing such on-the-fly signing, this might be computationally expensive. In the case where the number of dCDNs, and thus the number of different CNAMEs to return, is relatively stable, an alternative solution would be for the uCDN to pre-generate signatures for all possible CNAMEs. For each incoming query the uCDN would then determine the appropriate CNAME and return it together with the associated pre-generated signature. Note: In the latter case maintaining the serial and signature of SOA might be an issue since technically it should change every time a different CNAME is used. However, since in practice direct SOA queries are relatively rare, a uCDN could defer incrementing the serial and resigning the SOA until it is queried and then do it on-the-fly.

Note also that the NS record and the glue AAAA/A records used in step 2 in the previous section should generally be identical to those of their authoritative zone managed by Operator B. Even if they differ, this will not make the DNS resolution process fail, but the client DNS server will prefer the authoritative data in its cache and use it for subsequent queries. Such inconsistency is a general operational issue of DNS, but it may be more important for this architecture

because the uCDN (operator A) would rely on the consistency to make the resulting redirection work as intended. In general, it is the administrator's responsibility to make them consistent.

3.5. Dynamic Footprint Discovery Example

There could be situations where being able to dynamically discover the set of requests that a given dCDN is willing and able to serve is beneficial. For example, a CDN might at one time be able to serve a certain set of client IP prefixes, but that set might change over time due to changes in the topology and routing policies of the IP network. The following example illustrates this capability. We have chosen the example of DNS-based redirection, but HTTP-based redirection could equally well use this approach.

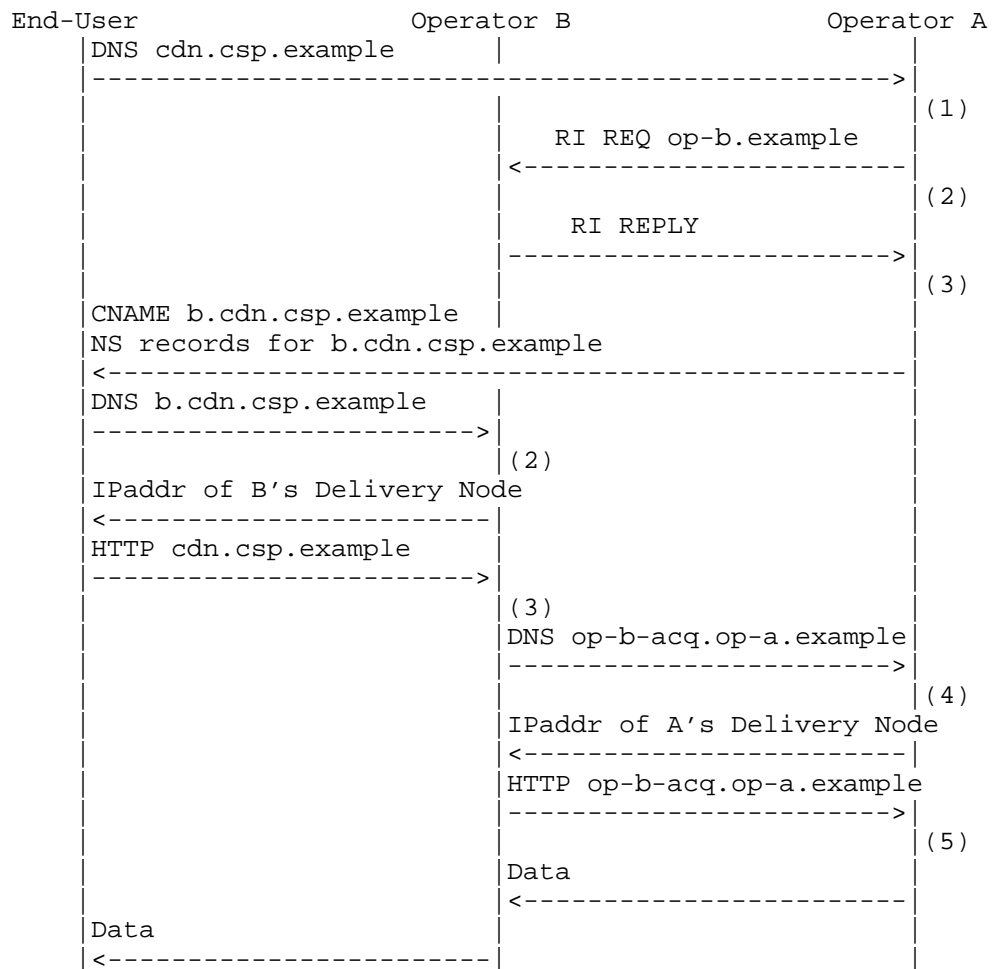


Figure 6: Message Flow for Dynamic Footprint Discovery

This example differs from the one in Figure 5 only in the addition of a RI request (step 2) and corresponding response (step 3). The RI REQ could be a message such as "Can you serve clients from this IP Prefix?" or it could be "Provide the list of client IP prefixes you can currently serve". In either case the response might be cached by operator A to avoid repeatedly asking the same question. Alternatively, or in addition, Operator B may spontaneously advertise to Operator A information (or changes) on the set of requests it is willing and able to serve on behalf of operator A; in that case, Operator B may spontaneously issue RR/RI REPLY messages that are not in direct response to a corresponding RR/RI REQ message. (Note that

the issues of determining the client's subnet from DNS requests, as described above, are exactly the same here as in Section 3.4.)

Once Operator A obtains the RI response, it is now able to determine that Operator B's CDN is an appropriate dCDN for this request and therefore a valid candidate dCDN to consider in its Redirection decision. If that dCDN is selected, the redirection and serving of the request proceeds as before (i.e. in the absence of dynamic footprint discovery).

3.6. Content Removal Example

The following example illustrates how the CDNI Control interface may be used to achieve pre-positioning of an item of content in the dCDN. In this example, user requests for a particular content, and corresponding redirection of such requests from Operator A to Operator B CDN, may (or may not) have taken place earlier. Then, at some point in time, the uCDN (for example, in response to a corresponding Trigger from the Content Provider) uses the CI to request that content identified by a particular URL be removed from dCDN. The following diagram illustrates the operation. It should be noted that a uCDN will typically not know whether a dCDN has cached a given content item, however, it may send the content removal request to make sure no cached versions remain to satisfy any contractual obligations it may have.

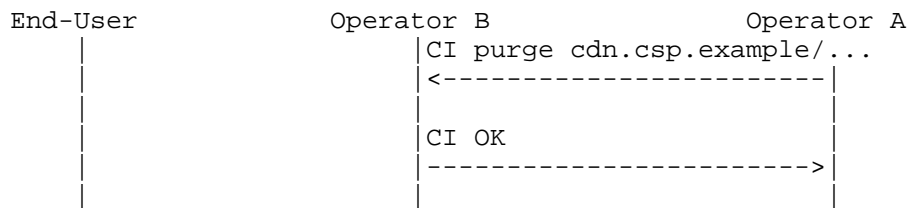


Figure 7: Message Flow for Content Removal

The CI is used to convey the request from uCDN to dCDN that some previously acquired content should be deleted. The URL in the request specifies which content to remove. This example corresponds to a DNS-based redirection scenario such as Section 3.4. If HTTP-based redirection had been used, the URL for removal would be of the form peer-a.op-b.example/cdn.csp.example/...

The dCDN is expected to confirm to the uCDN, as illustrated by the CI OK message, the completion of the removal of the targeted content from all the caches in dCDN.

3.7. Pre-Positioned Content Acquisition Example

The following example illustrates how the CI may be used to pre-position an item of content in the dCDN. In this example, Operator A uses the CDNI Metadata interface to request that content identified by a particular URL be pre-positioned into Operator B CDN.

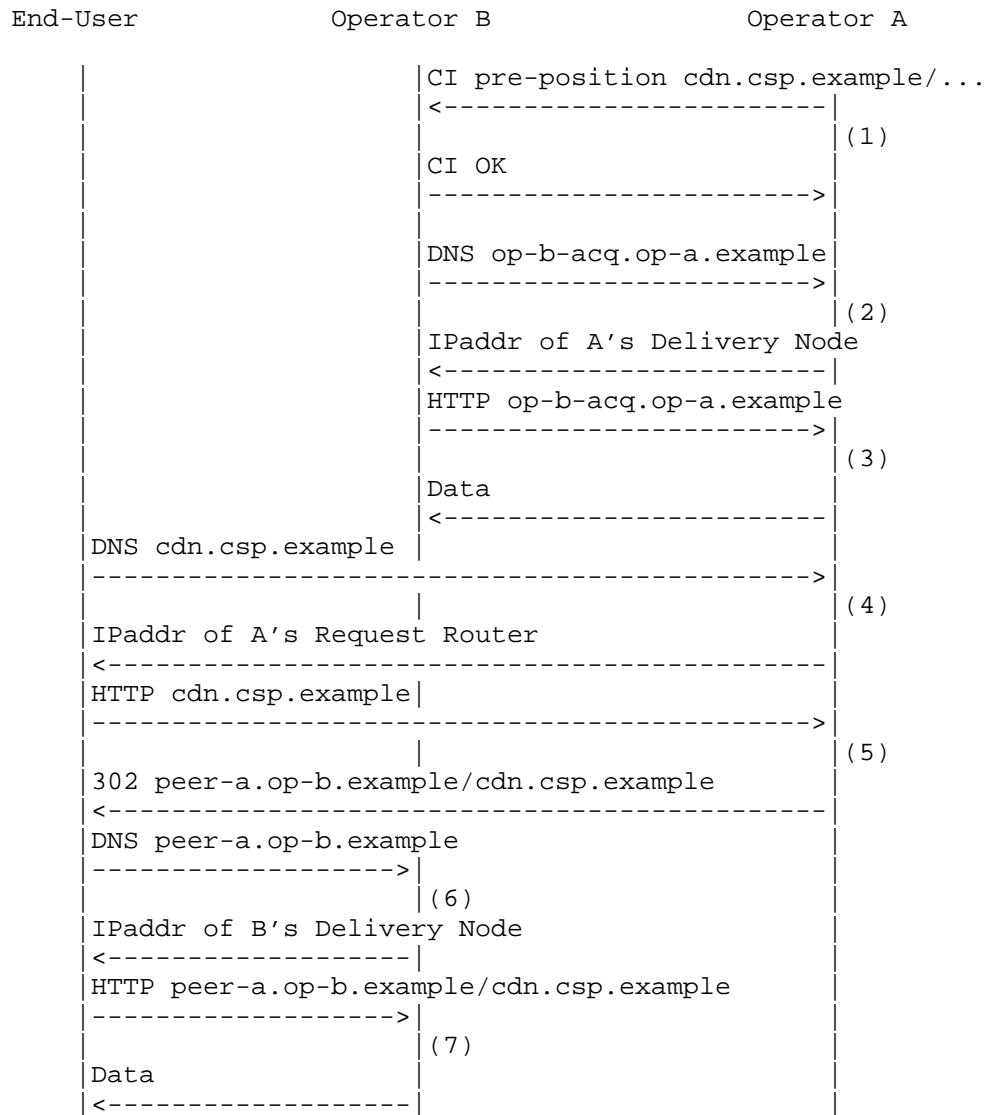


Figure 8: Message Flow for Content Pre-Positioning

The steps illustrated in the figure are as follows:

1. Operator A uses the CI to request that Operator B pre-positions a particular content item identified by its URL. Operator B responds by confirming that it is willing to perform this operation.

Steps 2 and 3 are exactly the same as steps 5 and 6 of Figure 3, only this time those steps happen as the result of the Pre-positioning request instead of as the result of a cache miss.

Steps 4, 5, 6, 7 are exactly the same as steps 1, 2, 3, 4 of Figure 3, only this time Operator B CDN can serve the end-user request without triggering dynamic content acquisition, since the content has been pre-positioned in dCDN. Note that, depending on dCDN operations and policies, the content pre-positioned in the dCDN may be pre-positioned to all, or a subset of, dCDN caches. In the latter case, intra-CDN dynamic content acquisition may take place inside the dCDN serving requests from caches on which the content has not been pre-positioning; however, such intra-CDN dynamic acquisition would not involve the uCDN.

3.8. Asynchronous CDNI Metadata Example

In this section we walk through a simple example illustrating a scenario of asynchronously exchanging CDNI metadata, where the downstream CDN obtains CDNI metadata for content ahead of a corresponding content request. The example that follows assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used, as in Section 3.3. However, Asynchronous exchange of CDNI Metadata is similarly applicable to DNS-based inter-CDN redirection and iterative request routing (in which cases the CDNI metadata may be used at slightly different processing stages of the message flows).

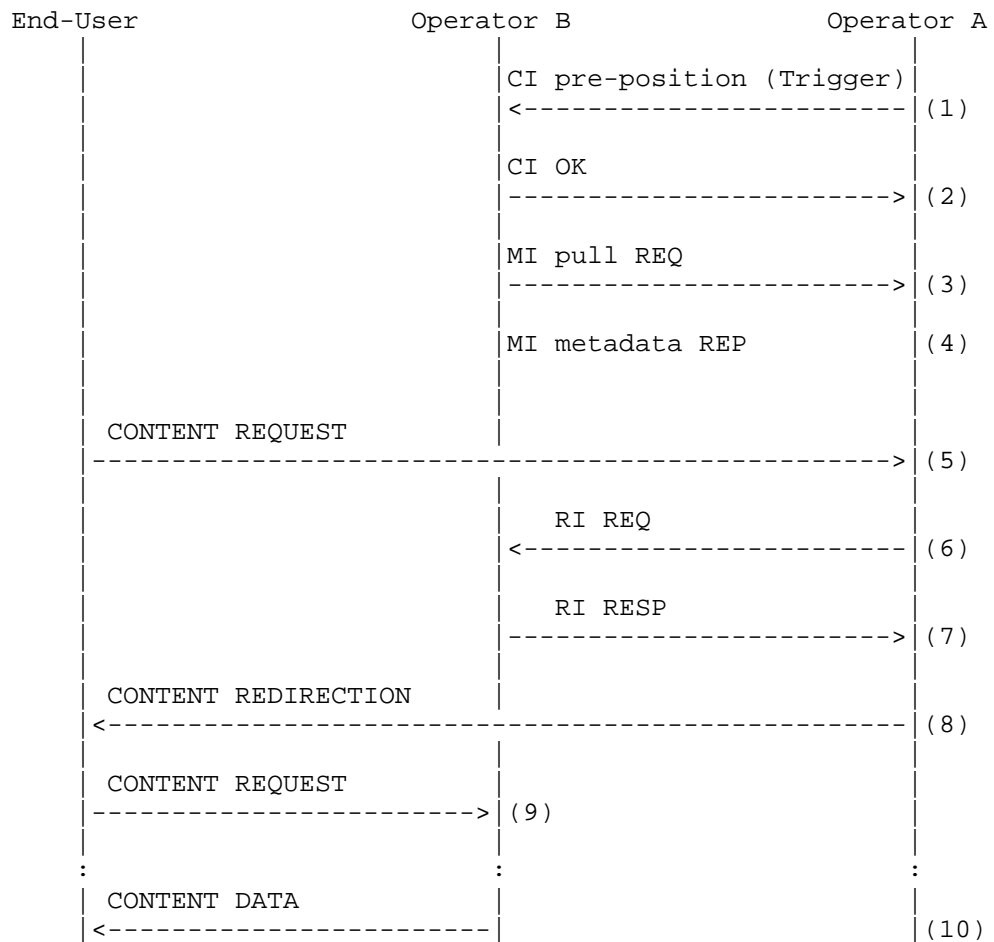


Figure 9: Message Flow for Asynchronous CDNI Metadata

The steps illustrated in the figure are as follows:

1. Operator A uses the CI to Trigger to signal the availability of CDNI metadata to Operator B.
2. Operator B acknowledges the receipt of this Trigger.
3. Operator B requests the latest metadata from Operator A using the MI.
4. Operator A replies with the requested metadata. This document does not constrain how the CDNI metadata information is actually

represented. For the purposes of this example, we assume that Operator A provides CDNI metadata to Operator B indicating that:

- * this CDNI Metadata is applicable to any content referenced by some CDN-Domain.
- * this CDNI metadata consists of a distribution policy requiring enforcement by the delivery node of a specific per-request authorization mechanism (e.g. URI signature or token validation).

5. A Content Request occurs as usual.
6. A CDNI Request Routing Redirection request (RI REQ) is issued by operator A CDN, as discussed in Section 3.3. Operator B's request router can access the CDNI Metadata that are relevant to the requested content and that have been pre-positioned as per Steps 1-4, which may or may not affect the response.
7. Operator B's request router issues a CDNI Request Routing Redirection response (RI RESP) as in Section 3.3.
8. Operator B performs content redirection as discussed in Section 3.3.
9. On receipt of the Content Request by the end user, the delivery node detects that previously acquired CDNI metadata is applicable to the requested content. In accordance with the specific CDNI metadata of this example, the delivery node will invoke the appropriate per-request authorization mechanism, before serving the content. (Details of this authorization are not shown.)
10. Assuming successful per-request authorization, serving of Content Data (possibly preceded by inter-CDN acquisition) proceeds as in Section 3.3.

3.9. Synchronous CDNI Metadata Acquisition Example

In this section we walk through a simple example illustrating a scenario of Synchronous CDNI metadata acquisition, in which the downstream CDN obtains CDNI metadata for content at the time of handling a first request for the corresponding content. As in the preceding section, this example assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used (as in Section 3.3), but dynamic CDNI metadata acquisition is applicable to other variations of request routing.

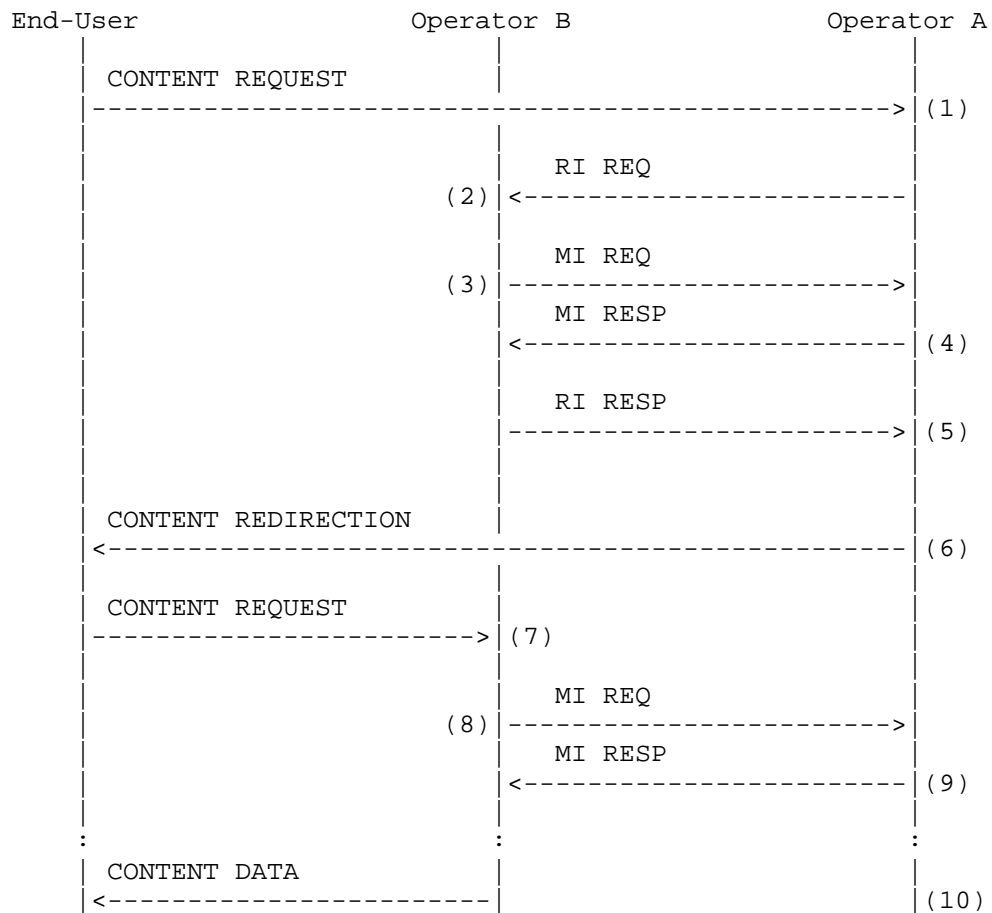


Figure 10: Message Flow for Synchronous CDNI Metadata Acquisition

The steps illustrated in the figure are as follows:

1. A Content Request arrives as normal.
2. An RI request occurs as in the prior example.
3. On receipt of the CDNI Request Routing Request, Operator B's CDN initiates Synchronous acquisition of CDNI Metadata that are needed for routing of the end-user request. We assume the URI for the a Metadata server is known ahead of time through some out-of-band means.

4. On receipt of a CDNI Metadata Request, Operator A's CDN responds, making the corresponding CDNI metadata information available to Operator B's CDN. This metadata is considered by operator B's CDN before responding to the Request Routing request. (In a simple case, the metadata could simply be an allow or deny response for this particular request.)
5. Response to the RI request as normal.
6. Redirection message is sent to the end user.
7. A delivery node of Operator B receives the end user request.
8. The delivery node Triggers dynamic acquisition of additional CDNI metadata that are needed to process the end-user content request. Note that there may exist cases where this step need not happen, for example because the metadata were already acquired previously.
9. Operator A's CDN responds to the CDNI Metadata Request and makes the corresponding CDNI metadata available to Operator B. This metadata influence how Operator B's CDN processes the end-user request.
10. Content is served (possibly preceded by inter-CDN acquisition) as in Section 3.3.

3.10. Content and Metadata Acquisition with Multiple Upstream CDNs

A single dCDN may receive end-user requests from multiple uCDNs. When a dCDN receives an end-user request, it must determine the identity of the uCDN from which it should acquire the requested content.

Ideally, the acquisition path of an end-user request will follow the redirection path of the request. The dCDN should acquire the content from the same uCDN which redirected the request.

Determining the acquisition path requires the dCDN to reconstruct the redirection path based on information in the end-user request. The method for reconstructing the redirection path differs based on the redirection approach: HTTP or DNS.

With HTTP-redirection, the rewritten URI should include sufficient information for the dCDN to directly or indirectly determine the uCDN when the end-user request is received. The HTTP-redirection approach can be further broken-down based on the how the URL is rewritten during redirection: HTTP-redirection with or without Site

Aggregation. HTTP-redirection with Site Aggregation hides the identity of the original CSP. HTTP-redirection without Site Aggregation does not attempt to hide the identity of the original CSP. With both approaches, the rewritten URI includes enough information to identify the immediate neighbor uCDN.

With DNS-redirection, the dCDN receives the published URI (instead of a rewritten URI) and does not have sufficient information for the dCDN to identify the appropriate uCDN. The dCDN may narrow the set of viable uCDNs by examining the CDNI metadata from each to determine which uCDNs are hosting metadata for the requested content. If there is a single uCDN hosting metadata for the requested content, the dCDN can assume that the request redirection is coming from this uCDN and can acquire content from that uCDN. If there are multiple uCDNs hosting metadata for the requested content, the dCDN may be ready to trust any of these uCDNs to acquire the content (provided the uCDN is in a position to serve it). If the dCDN is not ready to trust any of these uCDNs, it needs to ensure via out of band arrangements that, for a given content, only a single uCDN will ever redirect requests to the dCDN.

Content acquisition may be preceded by content metadata acquisition. If possible, the acquisition path for metadata should also follow the redirection path. Additionally, we assume metadata is indexed based on rewritten URIs in the case of HTTP-redirection and is indexed based on published URIs in the case of DNS-redirection. Thus, the RI and the MI are tightly coupled in that the result of request routing (a rewritten URI pointing to the dCDN) serves as an input to metadata lookup. If the content metadata includes information for acquiring the content, then the MI is also tightly coupled with the acquisition interface in that the result of the metadata lookup (an acquisition URL likely hosted by the uCDN) should serve as input to the content acquisition.

4. Main Interfaces

Figure 1 illustrates the main interfaces that are in scope for the CDNI WG, along with several others. The detailed specifications of these interfaces are left to other documents, but see [RFC6707] and [I-D.ietf-cdni-requirements] for some discussion of the interfaces.

One interface that is not shown in Figure 1 is the interface between the user and the CSP. While for the purposes of CDNI that interface is out of scope, it is worth noting that it does exist and can provide useful functions, such as end-to-end performance monitoring and some forms of authentication and authorization.

There is also an important interface between the user and the Request Routing function of both uCDN and dCDN (shown as the "Request" Interface in Figure 1). As we saw in some of the preceding examples, that interface can be used as a way of passing metadata, such as the minimum information that is required for dCDN to obtain the content from uCDN.

In this section we will provide an overview of the functions performed by each of the CDNI interfaces and discuss how they fit into the overall solution. We also examine some of the design tradeoffs, and explore several cross-interface concerns. We begin with an examination of one such tradeoff that affects all the interfaces - the use of in-band or out-of-band communication.

4.1. In-Band versus Out-of-Band Interfaces

Before getting to the individual interfaces, we observe that there is a high-level design choice for each, involving the use of existing in-band communication channels versus defining new out-of-band interfaces.

It is possible that the information needed to carry out various interconnection functions can be communicated between peer CDNs using existing in-band protocols. The use of HTTP 302 redirect is an example of how certain aspects of request routing can be implemented in-band (embedded in URIs). Note that using existing in-band protocols does not imply that the CDNI interfaces are null; it is still necessary to establish the rules (conventions) by which such protocols are used to implement the various interface functions.

There are other opportunities for in-band communication beyond HTTP redirects. For example, many of the HTTP directives used by proxy servers can also be used by peer CDNs to inform each other of caching activity. Of these, one that is particularly relevant is the If-Modified-Since directive, which is used with the GET method to make it conditional: if the requested object has not been modified since the time specified in this field, a copy of the object will not be returned, and instead, a 304 (not modified) response will be returned.

4.2. Cross Interface Concerns

Although the CDNI interfaces are largely independent, there are a set of conventions practiced consistently across all interfaces. Most important among these is how resources are named, for example, how the CDNI Metadata and Control interfaces identify the set of resources to which a given directive applies, or the CDNI Logging

interface identifies the set of resources for which a summary record applies.

While in the limit the CDNI interfaces could explicitly identify every individual resource, in practice, they name resource aggregates (sets of URIs) that are to be treated in a similar way. For example, URI aggregates can be identified by a CDN-Domain (i.e., the FQDN at the beginning of a URI) or by a URI-Filter (i.e., a regular expression that matches a subset of URIs contained in some CDN-Doman). In other words, CDN-Domains and URI-Filters provide a uniform means to aggregate sets (and subsets) of URIs for the purpose of defining the scope for some operation in one of the CDNI interfaces.

4.3. Request Routing Interfaces

The Request Routing interface comprises two parts: the Asynchronous interface used by a dCDN to advertize footprint and capabilities (denoted FCI) to a uCDN, allowing the uCDN to decide whether to redirect particular user requests to that dCDN; and the Synchronous interface used by the uCDN to redirect a user request to the dCDN (denoted RI). (These are somewhat analogous to the operations of routing and forwarding in IP.)

As illustrated in Section 3, the RI part of request routing may be implemented in part by DNS and HTTP. Naming conventions may be established by which CDN peers communicate whether a request should be routed or content served.

We also note that RI plays a key role in enabling recursive redirection, as illustrated in Section 3.3. It enables the user to be redirected to the correct delivery node in dCDN with only a single redirection step (as seen by the user). This may be particularly valuable as the chain of interconnected CDNs increases beyond two CDNs. For further discussion on the RI, see [I-D.ietf-cdni-redirection].

In support of these redirection requests, it is necessary for CDN peers to exchange additional information with each other, and this is the role of the FCI part of request routing. Depending on the method(s) supported, this might include:

- o The operator's unique id (operator-id) or distinguished CDN-Domain (operator-domain);
- o NS records for the operator's set of externally visible request routers;

- o The set of requests the dCDN operator is prepared to serve (e.g. a set of client IP prefixes or geographic regions that may be served by dCDN).
- o Additional capabilities of the dCDN, such as its ability to support different CDNI Metadata requests.

Note that the set of requests that dCDN is willing to serve could in some cases be relatively static (e.g., a set of IP prefixes) which could be exchanged off-line, or might even be negotiated as part of a peering agreement. However, it may also be more dynamic, in which case the exchange supported by FCI would be helpful. A further discussion of the Footprint & Capability Advertisement interface can be found in [I-D.ietf-cdni-footprint-capabilities-semantics].

4.4. CDNI Logging Interface

It is necessary for the upstream CDN to have visibility into the delivery of content that it redirected to a downstream CDN. This allows the upstream CDN to properly bill its customers for multiple deliveries of content cached by the downstream CDN, as well as to report accurate traffic statistics to those content providers. This is one role of the LI.

Other operational data that may be relevant to CDNI can also be exchanged by the LI. For example, dCDN may report the amount of content it has acquired from uCDN, and how much cache storage has been consumed by content cached on behalf of uCDN.

Traffic logs are easily exchanged off-line. For example, the following traffic log is a small deviation from the Apache log file format, where entries include the following fields:

- o Domain - the full domain name of the origin server
- o IP address - the IP address of the client making the request
- o End time - the ending time of the transfer
- o Time zone - any time zone modifier for the end time
- o Method - the transfer command itself (e.g., GET, POST, HEAD)
- o URL - the requested URL
- o Version - the protocol version, such as HTTP/1.0
- o Response - a numeric response code indicating transfer result

- o Bytes Sent - the number of bytes in the body sent to the client
- o Request ID - a unique identifier for this transfer
- o User agent - the user agent, if supplied
- o Duration - the duration of the transfer in milliseconds
- o Cached Bytes - the number of body bytes served from the cache
- o Referer - the referrer string from the client, if supplied

Of these, only the Domain field is indirect in the downstream CDN--it is set to the CDN-Domain used by the upstream CDN rather than the actual origin server. This field could then be used to filter traffic log entries so only those entries matching the upstream CDN are reported to the corresponding operator. Further discussion of the LI can be found in [I-D.ietf-cdni-logging].

One open question is who does the filtering. One option is that the downstream CDN filters its own logs, and passes the relevant records directly to each upstream peer. This requires that the downstream CDN knows the set of CDN-Domains that belong to each upstream peer. If this information is already exchanged between peers as part of another interface, then direct peer-to-peer reporting is straightforward. If it is not available, and operators do not wish to advertise the set of CDN-Domains they serve to their peers, then the second option is for each CDN to send both its non-local traffic records and the set of CDN-Domains it serves to an independent third-party (i.e., a CDN Exchange), which subsequently filters, merges, and distributes traffic records on behalf of each participating CDN operator.

A second open question is how timely traffic information should be. For example, in addition to offline traffic logs, accurate real-time traffic monitoring might also be useful, but such information requires that the downstream CDN inform the upstream CDN each time it serves upstream content from its cache. The downstream CDN can do this, for example, by sending a conditional HTTP GET request (If-Modified-Since) to the upstream CDN each time it receives an HTTP GET request from one of its end-users. This allows the upstream CDN to record that a request has been issued for the purpose of real-time traffic monitoring. The upstream CDN can also use this information to validate the traffic logs received later from the downstream CDN.

There is obviously a tradeoff between accuracy of such monitoring and the overhead of the downstream CDN having to go back to the upstream CDN for every request.

Another design tradeoff in the LI is the degree of aggregation or summarization of data. One situation that lends itself to summarization is the delivery of HTTP adaptive streaming (HAS), since the large number of individual chunk requests potentially results in large volumes of logging information. This case is discussed below, but other forms of aggregation may also be useful. For example, there may be situations where bulk metrics such as bytes delivered per hour may suffice rather than the detailed per-request logs outlined above. It seems likely that a range of granularities of logging will be needed along with ways to specify the type and degree of aggregation required.

4.5. CDNI Control Interface

The CDNI Control interface is initially used to bootstrap the other interfaces. As a simple example, it could be used to provide the address of the logging server in dCDN to uCDN in order to bootstrap the CDNI Logging interface. It may also be used, for example, to establish security associations for the other interfaces.

The other role the CI plays is to allow the uCDN to pre-position, revalidate, or purge metadata and content on a dCDN. These operations, sometimes collectively called the Trigger interface, are discussed further in [I-D.ietf-cdni-control-triggers].

4.6. CDNI Metadata Interface

The role of the CDNI Metadata interface is to enable CDNI distribution metadata to be conveyed to the downstream CDN by the upstream CDN. Such metadata includes geo-blocking restrictions, availability windows, access control policies, and so on. It may also include information to facilitate acquisition of content by dCDN (e.g., alternate sources for the content, authorization information needed to acquire the content from the source). For a full discussion of the CDNI Metadata Interface, see [I-D.ietf-cdni-metadata]

Some distribution metadata may be partially emulated using in-band mechanisms. For example, in case of any geo-blocking restrictions or availability windows, the upstream CDN can elect to redirect a request to the downstream CDN only if that CDN's advertised delivery footprint is acceptable for the requested URL. Similarly, the request could be forwarded only if the current time is within the availability window. However, such approaches typically come with shortcomings such as inability to prevent from replay outside the time window or inability to make use of a downstream CDN that covers a broader footprint than the geo-blocking restrictions.

Similarly, some forms of access control may also be performed on a per-request basis using HTTP directives. For example, being able to respond to a conditional GET request gives the upstream CDN an opportunity to influence how the downstream CDN delivers its content. Minimally, the upstream CDN can invalidate (purge) content previously cached by the downstream CDN.

All of these in-band techniques serve to illustrate that uCDNs have the option of enforcing some of their access control policies themselves (at the expense of increased inter-CDN signaling load), rather than delegating enforcement to dCDNs using the MI. As a consequence, the MI could provide a means for the uCDN to express its desire to retain enforcement for itself. For example, this might be done by including a "check with me" flag in the metadata associated with certain content. The realization of such in-band techniques over the various inter-CDN acquisition protocols (e.g., HTTP) requires further investigation and may require small extensions or semantic changes to the acquisition protocol.

4.7. HTTP Adaptive Streaming Concerns

We consider HTTP Adaptive Streaming (HAS) and the impact it has on the CDNI interfaces because large objects (e.g., videos) are broken into a sequence of small, independent chunks. For each of the following, a more thorough discussion, including an overview of the tradeoffs involved in alternative designs, can be found in RFC 6983.

First, with respect to Content Acquisition and File Management, which are out-of-scope for the CDNI interfaces but nonetheless relevant to the overall operation, we assume no additional measures are required to deal with large numbers of chunks. This means that the dCDN is not explicitly made aware of any relationship between different chunks and the dCDN handles each chunk as if it were an individual and independent content item. The result is that content acquisition between uCDN and dCDN also happens on a per-chunk basis. This approach is in line with the recommendations made in RFC 6983, which also identifies potential improvements in this area that might be considered in the future.

Second, with respect to Request Routing, we note that HAS manifest files have the potential to interfere with request routing since manifest files contain URLs pointing to the location of content chunks. To make sure that a manifest file does not hinder CDNI request routing and does not place excessive load on CDNI resources, the use of manifest files could either be limited to those containing relative URLs or the uCDN could modify the URLs in the manifest. Our approach for dealing with these issues is twofold. As a mandatory requirement, CDNs should be able to handle unmodified manifest files

containing either relative or absolute URLs. To limit the number of redirects, and thus the load placed on the CDNI interfaces, as an optional feature uCDNs can use the information obtained through the CDNI Request Routing Redirection interface to modify the URLs in the manifest file. Since the modification of the manifest file is an optional uCDN-internal process, this does not require any standardization effort beyond being able to communicate chunk locations in the CDNI Request Routing Redirection interface.

Third, with respect to the CDNI Logging interface, there are several potential issues, including the large number of individual chunk requests potentially resulting in large volumes of logging information, and the desire to correlate logging information for chunk requests that correspond to the same HAS session. For the initial CDNI specification, our approach is to expect participating CDNs to support per-chunk logging (e.g. logging each chunk request as if it were an independent content request) over the CDNI Logging interface. Optionally, the LI may include a Content Collection Identifier (CCID) and/or a Session Identifier (SID) as part of the logging fields, thereby facilitating correlation of per-chunk logs into per-session logs for applications benefiting from such session level information (e.g. session-based analytics). This approach is in line with the recommendations made in RFC 6983, which also identifies potential improvements in this area that might be considered in the future.

Fourth, with respect to the CDNI Control interface, and in particular purging HAS chunks from a given CDN, our approach is to expect each CDN supports per-chunk content purge (e.g. purging of chunks as if they were individual content items). Optionally, a CDN may support content purge on the basis of a "Purge Identifier (Purge-ID)" allowing the removal of all chunks related to a given Content Collection with a single reference. It is possible that this Purge-ID could be merged with the CCID discussed above for HAS Logging, or alternatively, they may remain distinct.

4.8. URI Rewriting

When using HTTP redirection, content URIs may be rewritten when redirection takes place within an uCDN, from an uCDN to a dCDN, and within the dCDN. In the case of cascaded CDNs, content URIs may be rewritten at every CDN hop (e.g., between the uCDN and the dCDN acting as the transit CDN, and between the transit CDN and the dCDN serving the request. The content URI used between any uCDN/dCDN pair becomes a common handle that can be referred to without ambiguity by both CDNs in all their inter-CDN communications. This handle allows the uCDN and dCDN to correlate information exchanged using other CDNI

interfaces in both the downstream direction (e.g., when using the MI) and the upstream direction (e.g., when using the LI).

Consider the simple case of a single uCDN/dCDN pair using HTTP redirection. We introduce the following terminology for content URIs to simplify the discussion:

"u-URI" represents a content URI in a request presented to the uCDN;

"ud-URI" is a content URI acting as the common handle across uCDN and dCDN for requests redirected by the uCDN to a specific dCDN;

"d-URI" represents a content URI in a request made within the delegate dCDN.

In our simple pair-wise example, the "ud-URI" effectively becomes the handle that the uCDN/dCDN pair use to correlate all CDNI information. In particular, for a given pair of CDNs executing the HTTP redirection, the uCDN needs to map the u-URI to the ud-URI handle for all MI message exchanges, while the dCDN needs to map the d-URI to the ud-URI handle for all LI message exchanges.

In the case of cascaded CDNs, the transit CDN will rewrite the content URI when redirecting to the dCDN, thereby establishing a new handle between the transit CDN and the dCDN, that is different from the handle between the uCDN and transit CDN. It is the responsibility of the transit CDN to manage its mapping across handles so the right handle for all pairs of CDNs is always used in its CDNI communication.

In summary, all CDNI interfaces between a given pair of CDNs need to always use the "ud-URI" handle for that specific CDN pair as their content URI reference.

5. Deployment Models

In this section we describe a number of possible deployment models that may be achieved using the CDNI interfaces described above. We note that these models are by no means exhaustive, and that many other models may be possible.

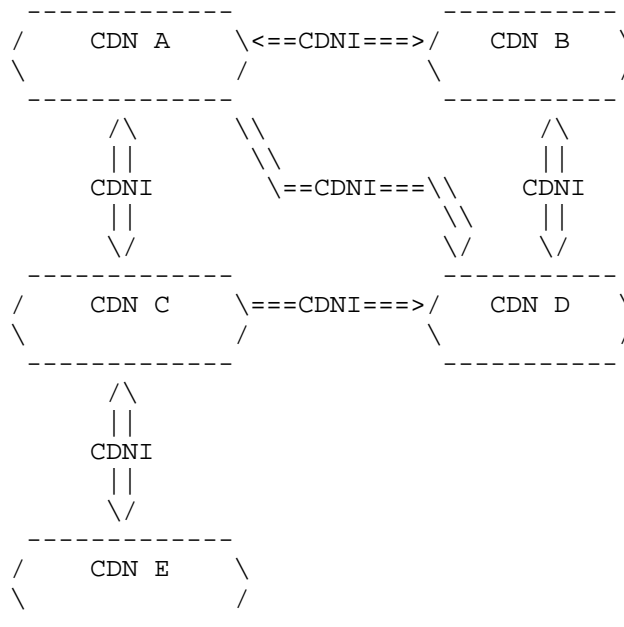
Although the reference model of Figure 1 shows all CDN functions on each side of the CDNI interface, deployments can rely on entities that are involved in any subset of these functions, and therefore only support the relevant subset of CDNI interfaces. As already noted in Section 3, effective CDNI deployments can be built without

necessarily implementing all the interfaces. Some examples of such deployments are shown below.

Note that, while we refer to upstream and downstream CDNs, this distinction applies to specific content items and transactions. That is, a given CDN may be upstream for some transactions and downstream for others, depending on many factors such as location of the requesting client and the particular piece of content requested.

5.1. Meshed CDNs

Although the reference model illustrated in Figure 1 shows a unidirectional CDN interconnection with a single uCDN and a single dCDN, any arbitrary CDNI meshing can be built from this, such as the example meshing illustrated in Figure 11. (Support for arbitrary meshing may or may not be in the initial scope for the working group, but the model allows for it.)



- ==> CDNI interfaces, with right-hand side CDN acting as dCDN to left-hand side CDN
- <==> CDNI interfaces, with right-hand side CDN acting as dCDN to left-hand side CDN and with left-hand side CDN acting as dCDN to right-hand side CDN

Figure 11: CDNI Deployment Model: CDN Meshing Example

5.2. CSP combined with CDN

Note that our terminology refers to functional roles and not economic or business roles. That is, a given organization may be operating as both a CSP and a fully fledged uCDN when we consider the functions performed, as illustrated in Figure 12.

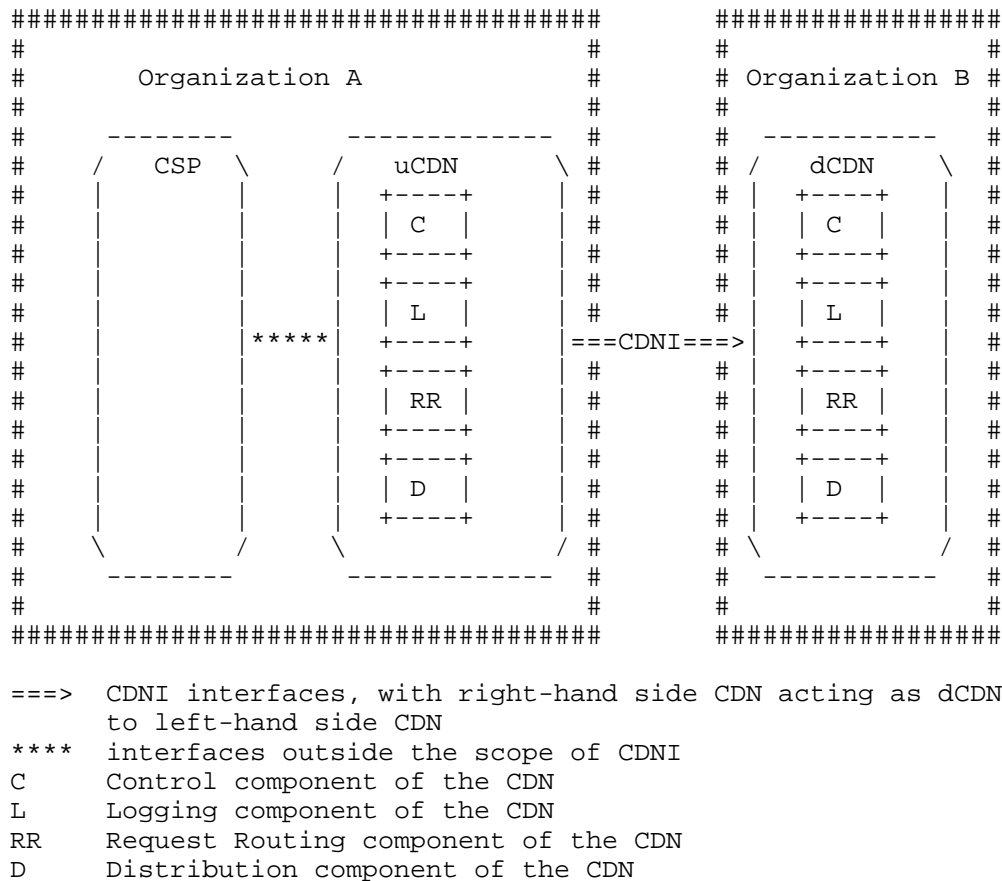


Figure 12: CDNI Deployment Model: Organization combining CSP & uCDN

5.3. CSP using CDNI Request Routing Interface

As another example, a content provider organization may choose to run its own request routing function as a way to select among multiple candidate CDN providers; In this case the content provider may be modeled as the combination of a CSP and of a special, restricted case of a CDN. In that case, as illustrated in Figure 13, the CDNI Request Routing interfaces can be used between the restricted CDN

operated by the content provider Organization and the CDN operated by the full CDN organization acting as a dCDN in the request routing control plane. Interfaces outside the scope of the CDNI work can be used between the CSP functional entities of the content provider organization and the CDN operated by the full CDN organization acting as a uCDN) in the CDNI control planes other than the request routing plane (i.e. Control, Distribution, Logging).

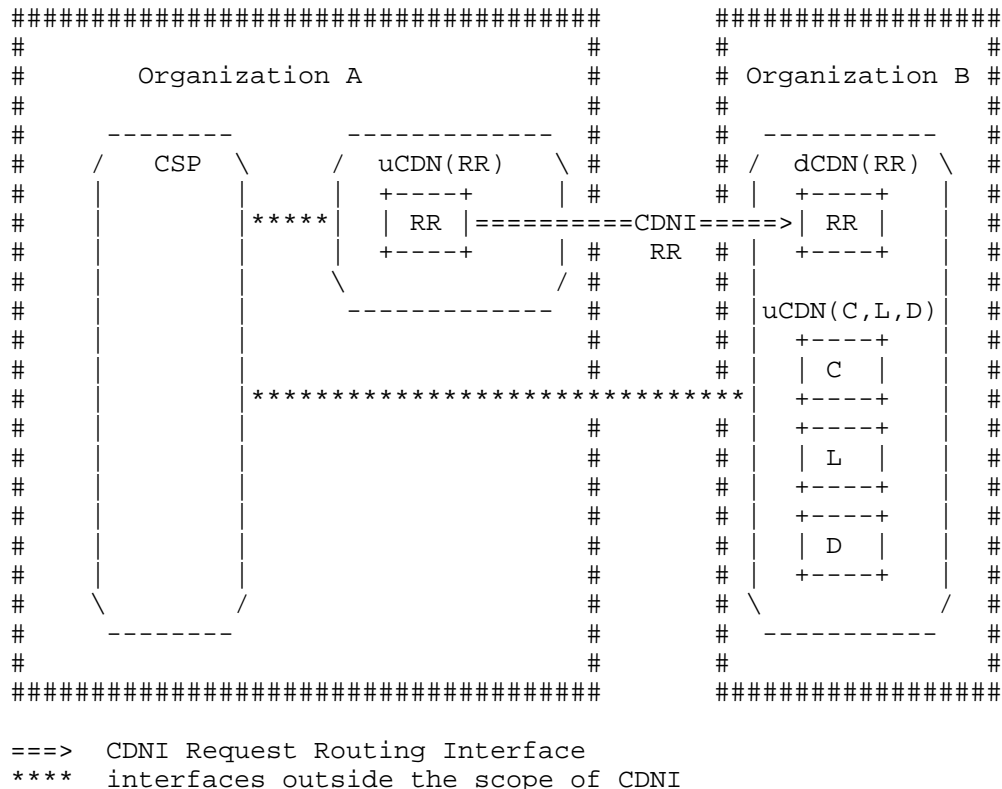


Figure 13: CDNI Deployment Model: Organization combining CSP and partial CDN

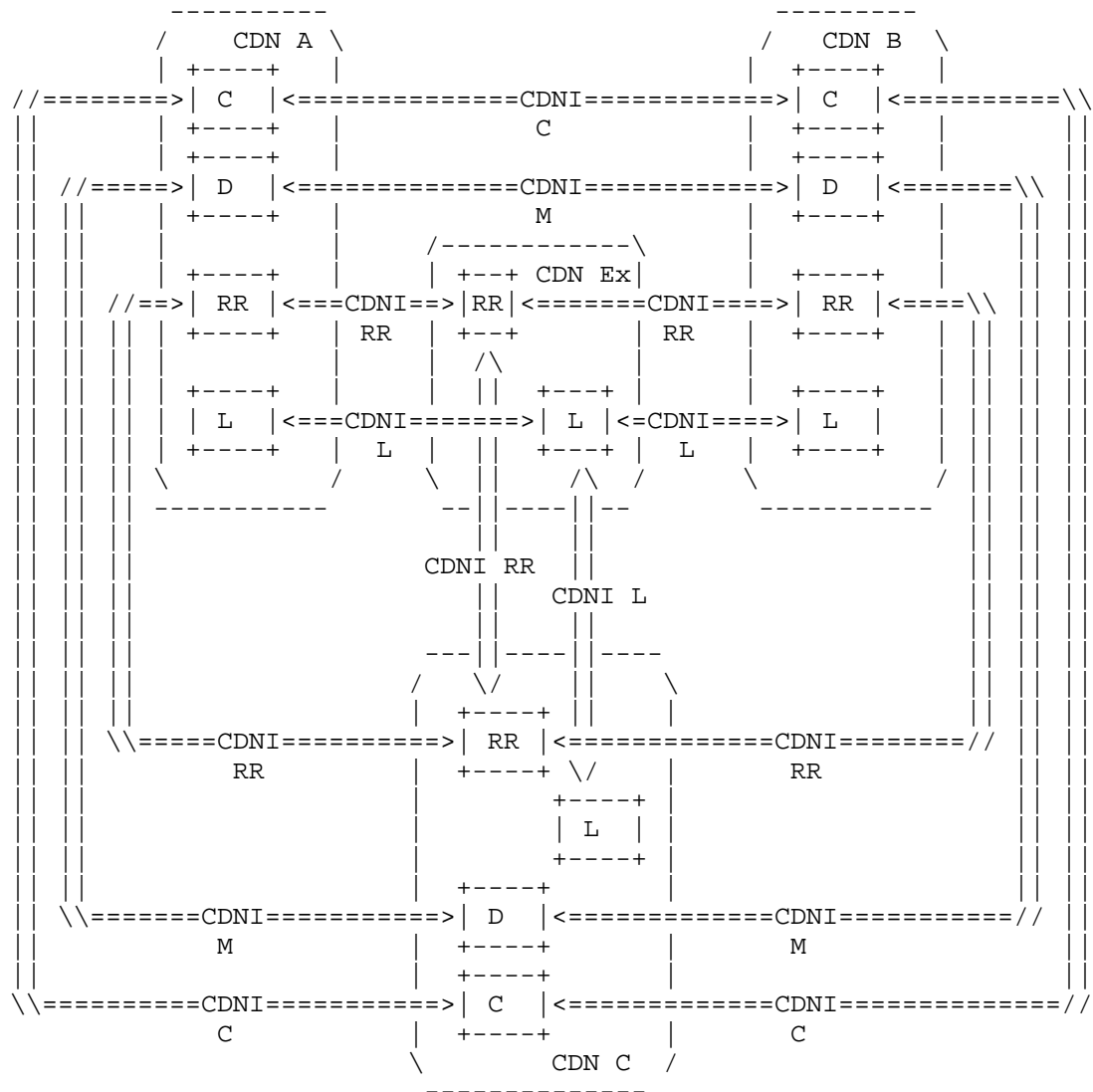
5.4. CDN Federations and CDN Exchanges

There are two additional concepts related to, but distinct from CDN Interconnection. The first is CDN Federation. Our view is that CDNI is the more general concept, involving two or more CDNs serving content to each other's users, while federation implies a multi-lateral interconnection arrangement, but other CDN interconnection agreements are also possible (e.g., symmetric bilateral, asymmetric bilateral). An important conclusion is that CDNI technology should

not presume (or bake in) a particular interconnection agreement, but should instead be general enough to permit alternative interconnection arrangements to evolve.

The second concept often used in the context of CDN Federation is CDN Exchange--a third party broker or exchange that is used to facilitate a CDN federation. Our view is that a CDN exchange offers valuable machinery to scale the number of CDN operators involved in a multi-lateral (federated) agreement, but that this machinery is built on top of the core CDNI interconnection mechanisms. For example, as illustrated in Figure 14, the exchange might aggregate and redistribute information about each CDN footprint and capacity, as well as collect, filter, and redistribute traffic logs that each participant needs for interconnection settlement, but inter-CDN request routing, inter-CDN content distribution (including inter-CDN acquisition) and inter-CDN control which fundamentally involve a direct interaction between an upstream CDN and a downstream CDN--operate exactly as in a pair-wise peering arrangement. Turning to Figure 14, we observe that in this example:

- o each CDN supports a direct CDNI Control interface to every other CDN
- o each CDN supports a direct CDNI Metadata interface to every other CDN
- o each CDN supports a CDNI Logging interface with the CDN Exchange
- o each CDN supports both a CDNI Request Routing interface with the CDN Exchange (for aggregation and redistribution of dynamic CDN footprint discovery information) and a direct RI to every other CDN (for actual request redirection).



<=CDNI RR=> CDNI Request Routing Interface
 <=CDNI M==> CDNI Metadata Interface
 <=CDNI C==> CDNI Control Interface
 <=CDNI L==> CDNI Logging Interface

Figure 14: CDNI Deployment Model: CDN Exchange

Note that a CDN exchange may alternatively support a different set of functionality (e.g. Logging only, or Logging and full request

routing, or all the functionality of a CDN including content distribution). All these options are expected to be allowed by the IETF CDNI specifications.

6. Trust Model

There are a number of trust issues that need to be addressed by a CDNI solution. Many of them are in fact similar or identical to those in a simple CDN without interconnection. In a standard CDN environment (without CDNI), the CSP places a degree of trust in a single CDN operator to perform many functions. The CDN is trusted to deliver content with appropriate quality of experience for the end user. The CSP trusts the CDN operator not to corrupt or modify the content. The CSP often relies on the CDN operator to provide reliable accounting information regarding the volume of delivered content. The CSP may also trust the CDN operator to perform actions such as timely invalidation of content and restriction of access to content based on certain criteria such as location of the user and time of day, and to enforce per-request authorization performed by the CSP using techniques such as URI signing.

A CSP also places trust in the CDN not to distribute any information that is confidential to the CSP (e.g., how popular a given piece of content is) or confidential to the end user (e.g., which content has been watched by which user).

A CSP does not necessarily have to place complete trust in a CDN. A CSP will in some cases take steps to protect its content from improper distribution by a CDN, e.g. by encrypting it and distributing keys in some out of band way. A CSP also depends on monitoring (possibly by third parties) and reporting to verify that the CDN has performed adequately. A CSP may use techniques such as client-based metering to verify that accounting information provided by the CDN is reliable. HTTP conditional requests may be used to provide the CSP with some checks on CDN operation. In other words, while a CSP may trust a CDN to perform some functions in the short term, the CSP is able in most cases to verify whether these actions have been performed correctly and to take action (such as moving the content to a different CDN) if the CDN does not live up to expectations.

One of the trust issues raised by CDNI is transitive trust. A CDN that has a direct relationship with a CSP can now "outsource" the delivery of content to another (downstream) CDN. That CDN may in turn outsource delivery to yet another downstream CDN, and so on.

The top level CDN in such a chain of delegation is responsible for ensuring that the requirements of the CSP are met. Failure to do so

is presumably just as serious as in the traditional single CDN case. Hence, an upstream CDN is essentially trusting a downstream CDN to perform functions on its behalf in just the same way as a CSP trusts a single CDN. Monitoring and reporting can similarly be used to verify that the downstream CDN has performed appropriately. However, the introduction of multiple CDNs in the path between CSP and end user complicates the picture. For example, third party monitoring of CDN performance (or other aspects of operation, such as timely invalidation) might be able to identify the fact that a problem occurred somewhere in the chain but not point to the particular CDN at fault.

In summary, we assume that an upstream CDN will invest a certain amount of trust in a downstream CDN, but that it will verify that the downstream CDN is performing correctly, and take corrective action (including potentially breaking off its relationship with that CDN) if behavior is not correct. We do not expect that the trust relationship between a CSP and its "top level" CDN will differ significantly from that found today in single CDN situations. However, it does appear that more sophisticated tools and techniques for monitoring CDN performance and behavior will be required to enable the identification of the CDN at fault in a particular delivery chain.

We expect that the detailed designs for the specific interfaces for CDNI will need to take the transitive trust issues into account. For example, explicit confirmation that some action (such as content removal) has taken place in a downstream CDN may help to mitigate some issues of transitive trust.

7. IANA Considerations

This memo includes no request to IANA.

8. Privacy Considerations

In general, a CDN has the opportunity to collect detailed information about the behavior of end-users e.g. by logging which files are being downloaded. While the concept of interconnected CDNs as described in this document doesn't necessarily allow any given CDN to gather more information on any specific user, it potentially facilitates sharing of this data by a CDN with more parties. As an example, the purpose of the CDNI Logging Interface is to allow a dCDN to share some of its log records with a uCDN, both for billing purposes as well as for sharing traffic statistics with the Content Provider on which behalf the content was delivered. The fact that the CDNI Interfaces provide mechanisms for sharing such potentially sensitive user data, shows that it is necessary to include in these interface appropriate

privacy and confidentiality mechanisms. The definition of such mechanisms is dealt with in the respective CDN interface documents.

9. Security Considerations

While there are a variety of security issues introduced by a single CDN, we are concerned here specifically with the additional issues that arise when CDNs are interconnected. For example, when a single CDN has the ability to distribute content on behalf of a CSP, there may be concerns that such content could be distributed to parties who are not authorized to receive it, and there are mechanisms to deal with such concerns. Our focus in this section is on how CDN interconnection introduces new security issues not found in the single CDN case. For a more detailed analysis of the security requirements of CDNI, see section 9 of [I-D.ietf-cdni-requirements].

Many of the security issues that arise in CDNI are related to the transitivity of trust (or lack thereof) described in Section 6. As noted above, the design of the various interfaces for CDNI must take account of the additional risks posed by the fact that a CDN with whom a CSP has no direct relationship is now potentially distributing content for that CSP. The mechanisms used to mitigate these risks may be similar to those used in the single CDN case, but their suitability in this more complex environment must be validated.

CDNs today offer a variety of means to control access to content, such as time-of-day restrictions, geo-blocking, and URI signing. These mechanisms must continue to function in CDNI environments, and this consideration is likely to affect the design of certain CDNI interfaces (e.g. metadata, request routing). For more information on URI signing in CDNI, see [I-D.leung-cdni-uri-signing].

Just as with a single CDN, each peer CDN must ensure that it is not used as an "open proxy" to deliver content on behalf of a malicious CSP. Whereas a single CDN typically addresses this problem by having CSPs explicitly register content (or origin servers) that are to be served, simply propagating this information to peer downstream CDNs may be problematic because it reveals more information than the upstream CDN is willing to specify. (To this end, the content acquisition step in the earlier examples force the dCDN to retrieve content from the uCDN rather than go directly to the origin server.)

There are several approaches to this problem. One is for the uCDN to encode a signed token generated from a shared secret in each URL routed to a dCDN, and for the dCDN to validate the request based on this token. Another one is to have each upstream CDN advertise the set of CDN-Domains they serve, where the downstream CDN checks each request against this set before caching and delivering the associated

object. Although straightforward, this approach requires operators to reveal additional information, which may or may not be an issue.

9.1. Security of CDNI Interfaces

It is noted in [I-D.ietf-cdni-requirements] that all CDNI interfaces must be able to operate securely over insecure IP networks. Since it is expected that the CDNI interfaces will be implemented using existing application protocols such as HTTP or XMPP, we also expect that the security mechanisms available to those protocols may be used by the CDNI interfaces. Details of how these interfaces are secured will be specified in the relevant interface documents.

9.2. Digital Rights Management

Issues of digital rights management (DRM, also sometimes called digital restrictions management) is often employed for content distributed via CDNs. In general, DRM relies on the CDN to distribute encrypted content, with decryption keys distributed to users by some other means (e.g. directly from the CSP to the end user.) For this reason, DRM is considered out of scope [RFC6707] and does not introduce additional security issues for CDNI.

10. Contributors

The following individuals contributed to this document:

- o Matt Caulfield
- o Francois le Faucheur
- o Aaron Falk
- o David Ferguson
- o John Hartman
- o Ben Niven-Jenkins
- o Kent Leung

11. Acknowledgements

The authors would like to thank Huw Jones and Jinmei Tatuya for their helpful input to this document. In addition, the authors would like to thank Stephen Farrell, Ted Lemon and Alissa Cooper for their reviews, which have helped to improve this document.

12. Informative References

- [I-D.ietf-cdni-control-triggers]
Murray, R. and B. Niven-Jenkins, "CDNI Control Interface / Triggers", draft-ietf-cdni-control-triggers-02 (work in progress), December 2013.
- [I-D.ietf-cdni-footprint-capabilities-semantics]
Seedorf, J., Peterson, J., Previdi, S., Brandenburg, R., and K. Ma, "CDNI Request Routing: Footprint and Capabilities Semantics", draft-ietf-cdni-footprint-capabilities-semantics-02 (work in progress), February 2014.
- [I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-11 (work in progress), March 2014.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata", draft-ietf-cdni-metadata-06 (work in progress), February 2014.
- [I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection Interface for CDN Interconnection", draft-ietf-cdni-redirection-02 (work in progress), April 2014.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-17 (work in progress), January 2014.
- [I-D.leung-cdni-uri-signing]
Leung, K., Faucheur, F., Downey, B., Brandenburg, R., and S. Leibrand, "URI Signing for CDN Interconnection (CDNI)", draft-leung-cdni-uri-signing-05 (work in progress), March 2014.
- [RFC3466] Day, M., Cain, B., Tomlinson, G., and P. Rzewski, "A Model for Content Internetworking (CDI)", RFC 3466, February 2003.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

- [RFC6770] Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, November 2012.
- [RFC6983] van Brandenburg, R., van Deventer, O., Le Faucheur, F., and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware Content Distribution Network Interconnection (CDNI)", RFC 6983, July 2013.

Authors' Addresses

Larry Peterson
Akamai Technologies, Inc.
8 Cambridge Center
Cambridge, MA 02142
USA

Email: lapeters@akamai.com

Bruce Davie
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
USA

Email: bdavie@vmware.com

Ray van Brandenburg (editor)
TNO
Brassersplein 2
Delft 2612CT
the Netherlands

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 5, 2013

B. Niven-Jenkins
R. Murray
G. Watson
Velocix (Alcatel-Lucent)
M. Caulfield
K. Leung
Cisco Systems
K. Ma
Azuki Systems, Inc.
October 2, 2012

CDN Interconnect Metadata
draft-ietf-cdni-metadata-00

Abstract

The CDNI Metadata Interface enables interconnected CDNs to exchange content distribution metadata in order to enable content acquisition and delivery. The CDNI metadata associated with a piece of content provides a downstream CDN with sufficient information for the downstream CDN to service content requests on behalf of an upstream CDN. This document describes both the core set of CDNI metadata and the protocol for exchanging that metadata.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 April 5, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Terminology	4
2. Design Principles	5
3. CDNI Metadata Data Model	5
3.1. HostIndex, HostMetadata & PathMetadata objects	6
3.2. Generic CDNI Metadata Object Properties	9
3.3. Metadata Inheritance	10
3.4. Metadata Naming	10
4. Encoding-Independent CDNI Metadata Object Descriptions	11
4.1. CDNI Metadata Structural Object Descriptions	11
4.1.1. HostIndex	11
4.1.2. HostMatch	12
4.1.3. HostMetadata	12
4.1.4. PathMatch	12
4.1.5. PathMetadata	13
4.1.6. PatternMatch	13
4.1.7. GenericMetadata	14
4.2. CDNI Metadata Property Object Descriptions	14
4.2.1. Source Metadata	14
4.2.1.1. Source	14
4.2.2. LocationACL Metadata	15
4.2.2.1. LocationRule	15
4.2.2.2. Location	15
4.2.3. TimeWindowACL Metadata	16
4.2.3.1. TimeWindowRule	16
4.2.3.2. TimeWindow	16
4.2.4. ProtocolACL Metadata	17
4.2.4.1. ProtocolRule	17
4.2.5. Authorization Metadata	17
4.2.6. Auth	17

4.3.	CDNI Metadata Simple Data Type Descriptions	18
4.3.1.	Link	18
4.3.2.	Protocol	18
4.3.3.	Endpoint	19
4.3.4.	IPRange	19
4.3.5.	URI	19
4.3.6.	Time	19
5.	CDNI Metadata interface	20
5.1.	Transport	20
5.2.	Retrieval of CDNI Metadata resources	21
5.3.	Bootstrapping	22
5.4.	Encoding	22
5.4.1.	MIME Media Types	22
5.4.2.	JSON Encoding of Objects	23
5.4.2.1.	JSON Example	24
5.4.3.	XML Encoding of Objects	27
5.4.3.1.	XML Example	27
5.5.	Extensibility	30
5.5.1.	Metadata Enforcement	30
5.5.2.	Metadata Override	31
6.	IANA Considerations	31
7.	Security Considerations	31
8.	Acknowledgements	32
9.	References	32
9.1.	Normative References	32
9.2.	Informative References	32
	Appendix A. Relationship to the CDNI Requirements	33
	Authors' Addresses	34

1. Introduction

CDNI enables a downstream CDN to service content requests on behalf of an upstream CDN. The CDNI metadata associated with a piece of content (or with a set of contents) provides a downstream CDN with sufficient information for servicing content requests on behalf of an upstream CDN in accordance with the policies defined by the upstream CDN.

The CDNI Metadata Interface is introduced by [I-D.ietf-cdni-problem-statement] along with three other interfaces that may be used to compose a CDNI solution (Control, Request Routing and Logging). [I-D.davie-cdni-framework] expands on the information provided in [I-D.ietf-cdni-problem-statement] and describes each interface, and the relationships between them, in more detail. The requirements for the CDNI metadata interface are specified in [I-D.ietf-cdni-requirements]

This document focuses on the CDNI Metadata interface which enables a downstream CDN to obtain CDNI Metadata from an upstream CDN so that the downstream CDN can properly process and respond to:

- o Redirection Requests received over the CDNI Request Routing protocol.
- o Content Requests received directly from User Agents.

Specifically this document proposes:

- o A data structure for mapping content requests to CDNI Metadata properties (Section 3).
- o An initial set of CDNI Metadata properties (Section 4.2).
- o A RESTful web service for the transfer of CDNI Metadata (Section 5).

1.1. Terminology

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

Additionally, the following terms are used throughout this document and are defined as follows:

- o Object - a collection of properties
- o Property - a key and value pair where the key is a property name and the value is the property value or an object.

2. Design Principles

The proposed CDNI Metadata Interface aims to achieve the following design principles:

1. Cacheability of CDNI metadata objects
2. Deterministic mapping from content requests to CDNI metadata properties
3. Support for DNS redirection as well as application-specific redirection (for example HTTP redirection)
4. Minimal duplication of CDNI metadata
5. Leverage existing protocols

Cacheability improves the latency of acquiring metadata and therefore improves the latency of serving content requests. The CDNI Metadata Interface uses HTTP to achieve cacheability.

Deterministic mappings from content requests to metadata properties eliminates ambiguity and ensures that the same policies are applied consistently by all downstream CDNs.

Support for both HTTP and DNS redirection ensures that the CDNI Metadata Interface can be used for HTTP and DNS redirection and also meets the same design principles for both HTTP and DNS based redirection schemes.

Minimal duplication of CDNI metadata provides space efficiency on storage in the CDNs, on caches in the network, and across the network between CDNs.

Leveraging existing protocols avoids reinventing common mechanisms such as data structure encoding (e.g. XML, JSON) and data transport (e.g. HTTP).

3. CDNI Metadata Data Model

The CDNI Metadata Model describes a data structure for mapping content requests to metadata properties. Metadata properties describe how to acquire, authorize, and deliver content from a downstream CDN. The data model relies on the assumption that these metadata properties may be aggregated based on the hostname of the content and subsequently on the resource path of the content. The data model associates a set of CDNI Metadata properties with a Hostname to form a default set of metadata properties for content delivered for that Hostname. That default set of metadata properties can be overridden by properties that apply to specific paths within a URI.

Different Hostnames and URI paths will contain different sets of CDNI Metadata properties in order to describe the required behaviour when a dCDN surrogate is processing User Agent requests for content at that Hostname or URI path. As a result of this structure, significant commonality may exist between the CDNI Metadata properties specified for different Hostnames, different URI paths within a Hostname and different URI paths on different Hostnames. For example the definition of which User Agent IP addresses should be treated as being grouped together into a single network or geographic location is likely to be common for a number of different Hostnames. Another example is that although a uCDN is likely to have several different policies configured to express geo-blocking rules, it is likely that a single geo-blocking policy would be applied to multiple Hostnames delivered through the CDN.

In order to enable the CDNI Metadata for a given Hostname or URI Path to be decomposed into sets of CDNI Metadata properties that can be reused by multiple Hostnames and URI Paths, the CDNI Metadata interface specified in this document splits the CDNI Metadata into a number of objects. Efficiency is improved by enabling a single CDNI Metadata object (that is shared across Hostname and/or URI paths) to be retrieved by a dCDN once, even if it is referenced by the CDNI Metadata of multiple Hostnames.

Section 3.1 introduces a high level description of the HostIndex, HostMetadata and PathMetadata objects and describes the relationships between those objects.

Section 3.2 introduces a high level description of the CDNI GenericMetadata object which represents the level at which CDNI Metadata override occurs between HostMetadata and PathMetadata objects.

Section 4 describes in detail the specific CDNI Metadata objects and properties which may be contained within a CDNI GenericMetadata object.

3.1. HostIndex, HostMetadata & PathMetadata objects

A HostIndex object contains a list of Hostnames (and/or IP addresses) that may be delegated to the downstream CDN. The HostIndex is the starting point for accessing the uCDN's CDNI Metadata data store. It enables surrogates in the dCDN to deterministically discover, on receipt of a User Agent request for content, which other CDNI Metadata objects it requires in order to deliver the requested content.

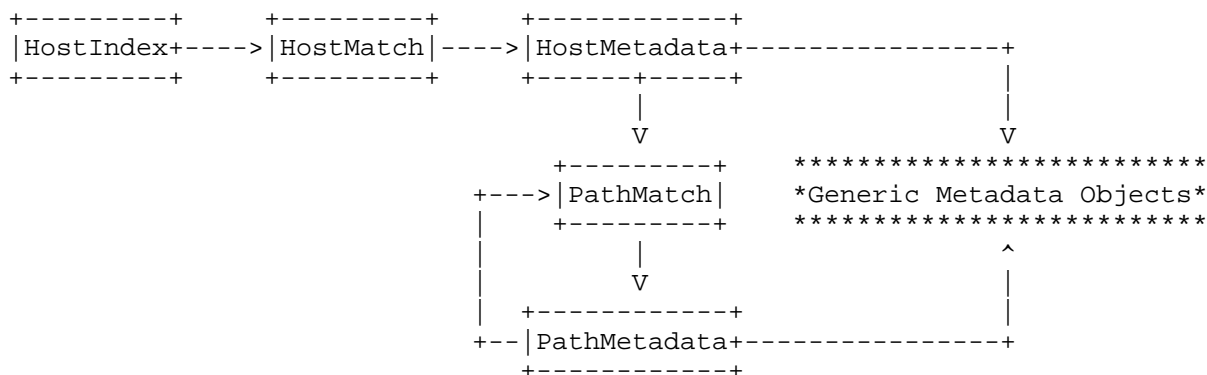
The HostIndex links Hostnames (and/or IP addresses) to HostMetadata

objects via HostMatch objects. HostMetadata objects contain (or reference) the default CDNI Metadata required to serve content for that host. When looking up CDNI Metadata, the downstream CDN looks up the requested Hostname (or IP address) in the HostIndex, from there it can find HostMetadata which describes delivery rules for a host and PathMetadata which may override those rules for given URI paths within the host.

As well as containing the default CDNI Metadata for the specified Hostname, HostMetadata and PathMetadata objects may also contain PathMatch objects which in turn contain PathMetadata objects. PathMatch objects override the CDNI Metadata in the HostMetadata object or one or more preceding PathMetadata objects with more specific CDNI Metadata that applies to content requests matching the pattern defined in that PathMatch object.

For the purposes of retrieving CDNI Metadata all other required CDNI Metadata objects and their properties are discoverable from the appropriate HostMetadata, PathMatch and PathMetadata objects for the requested content.

The relationships between the HostIndex, HostMatch, HostMetadata, PathMatch and PathMetadata objects are described in Figure 1.



Key: ----> = References

Figure 1: Relationships between the HostIndex, HostMetadata & PathMetadata CDNI Metadata Objects

The relationships in Figure 1 are summarised in Table 1 below.

Data Object	Objects it References
HostIndex	0 or more HostMatch objects.
HostMatch	1 HostMetadata object.
HostMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.
PathMatch	1 PathMetadata object.
PathMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.

Table 1: Relationships between CDNI Metadata Objects

The table below describes the HostIndex, HostMetadata and PathMetadata objects in more detail.

Data Object	Description
HostIndex	A HostIndex object lists the Hostnames (and/or IP addresses) that an upstream CDN can provide CDNI Metadata for and the URIs to use for retrieving that CDNI Metadata. For example, if "example.com" is a content provider, the HostIndex object may include an entry for "example.com" with the URI of the associated HostMetadata object. These hostnames are contained inside a list of HostMatch objects.
HostMatch	A HostMatch object defines a hostname to match against a requested host, and contains or references a HostMetadata object which contains CDNI Metadata objects to be applied when a content request matches against the hostname.
HostMetadata	A HostMetadata object contains (or references) the default CDNI Metadata objects for content served from that host, i.e. the CDNI Metadata objects for content requests that do not match any of the PathMatch objects contained or referenced by that HostMetadata object. For example, a HostMetadata object may describe the metadata properties which apply to "example.com" and may contain PathMatches for "example.com/movies/*" and "example.com/music/*" which reference corresponding PathMetadata objects that contain the CDNI Metadata objects for those more specific URI paths.

PathMatch	A PathMatch object defines a pattern to match against the requested path, and contains or references a PathMetadata object which contains (or references) the CDNI Metadata objects to be applied when a content request matches against the defined URI path pattern.
PathMetadata	A PathMetadata object contains the CDNI GenericMetadata objects for content served with the associated URI path (defined in a PathMatch object). A PathMetadata object may also contain PathMatch objects in order to recursively define more specific URI paths that require different (e.g. more specific) CDNI Metadata to this one. For example, the PathMetadata object which applies to "example.com/movies/*" may describe CDNI Metadata which apply to that resource path and may contain a PathMatch object for "example.com/movies/hd/*" which would reference the corresponding PathMetadata object for the "example.com/movies/hd/" path prefix.
GenericMetadata	A GenericMetadata object contains individual CDNI Metadata property objects which define the specific policies and attributes needed to properly deliver the associated content.

Table 2: HostIndex, HostMetadata and PathMetadata CDNI Metadata Objects

3.2. Generic CDNI Metadata Object Properties

The HostMetadata and PathMetadata objects contain or can reference other CDNI Metadata objects that contain properties which describe how User Agent requests for content should be processed, for example where to acquire the content, authorization rules that should be applied, delivery location restrictions and so on. Each such CDNI Metadata object is a specialization of a CDNI GenericMetadata object. The GenericMetadata object abstracts the basic information required for Metadata override and opaque Metadata distribution, from the specifics of any given property (e.g., property semantics, enforcement options, serialization rules, etc.).

The GenericMetadata object defines the type of properties contained within it as well as whether or not the properties are mandatory to enforce. If the dCDN does not understand or support the property type and the property type is mandatory to enforce, the dCDN MUST NOT serve the content to the User Agent. If the dCDN does not understand

or support the property type it is also not going to be able to properly deserialize and reserialize the Metadata for cascaded distribution.

For Metadata which does not require customization, the data representation received off the wire MAY be stored and redistributed without being natively understood or supported by the transit CDN. However, for Metadata which require for translations, transparent redistribution of the uCDN Metadata values may not be appropriate. Certain Metadata may be safely, though possibly not optimally, redistributed unmodified, e.g., source acquisition address may not be optimal if transparently redistributed, but may still work. Redistribution safety MUST be specified for each GenericMetadata.

3.3. Metadata Inheritance

In the data model, a HostMetadata object may contain (or reference) multiple PathMetadata objects (via PathMatch objects). Each PathMetadata object may in turn contain (or reference) other PathMetadata objects. HostMetadata and PathMetadata objects form an inheritance tree where each node in the tree inherits or overrides the property values set by its parent.

GenericMetadata objects of a given type override all GenericMetadata objects of the same type previously defined by any parent object in the tree. For example, if HostMetadata for the host "example.com" contains GenericMetadata objects of type LocationACL and TimeWindowACL, while a PathMetadata object which applies to "example.com/movies/*" defines an alternate GenericMetadata object of type TimeWindowACL, The PathMetadata defined TimeWindowACL would override the TimeWindowACL defined in the HostMetadata for all User Agent requests for movies.

3.4. Metadata Naming

GenericMetadata objects are identified by their type. The type SHOULD be descriptive, and MAY be hierarchical to support aggregating groups of properties for the purpose of readability and for avoiding name conflicts between vendor extensions. A dotted alpha-numeric notation is suggested for human readability. For example:

```
ext.vendor1.featurex
ext.vendor1.featurey
ext.vendor2.featurex
```

Metadata types defined by this document are not hierarchical.

[Ed. It is intended that Metadata capability advertisements will

allow either individual Metadata names or Metadata bundle identifiers to be used. Need to have a procedure for defining and distributing bundle information to be used in Metadata capability advertisement.]

4. Encoding-Independent CDNI Metadata Object Descriptions

Section 4.1 provides the definitions of each object type declared in Section 3. These objects are described as structural objects as they provide the structure for the inheritance tree and identifying which specific properties apply to a given User Agent content request.

Section 4.2 provides the definitions for the set of core metadata objects which may be contained within a GenericMetadata object. These objects are described as property objects as they define the semantics, enforcement options, and serialization rules for specific properties. These properties govern how User Agent requests for content are handled. Property objects may be composed of or contain references to other objects. In those cases the value of the property can be either an object of that type (the object is embedded) or a Link object that contains a URI and relationship that can be dereferenced to retrieve the CDNI Metadata object that represents the value of that property.

Note: In the following sections, the term "mandatory-to-specify" is used to convey which objects or properties must be specified for a given parent object or property. When mandatory-to-specify is set to true, it implies that if the parent object is specified, then the defined object or property MUST also be specified, e.g., a HostMatch object without a host to match against does not make sense, therefore, the host is mandatory-to-specify inside a parent HostMatch object.

4.1. CDNI Metadata Structural Object Descriptions

Each of the sub-sections below describe the structural objects defined in Table 2.

4.1.1. HostIndex

The HostIndex object is the entry point into the CDNI Metadata hierarchy. An incoming content request is matched against the list of hosts to find the HostMatch object which applies to the request.

Property: hosts

Description: List of HostMatch objects.

Type: List of HostMatch
Mandatory-to-Specify: Yes.

4.1.2. HostMatch

The HostMatch object contains a hostname or IP address to match against content requests. The HostMatch object references Metadata objects to apply if a match is found.

Property: host
Description: String (hostname or IP address) to match against the requested host.
Type: String
Mandatory-to-Specify: Yes.
Property: host-metadata
Description: CDNI Metadata to apply when delivering content that matches this host.
Type: HostMetadata
Mandatory-to-Specify: Yes.

4.1.3. HostMetadata

The HostMetadata object contains both Metadata that applies to content requests for a particular host and a list of pattern matches for finding more specific Metadata based on the resource path in a content request.

Property: metadata
Description: List of host related metadata.
Type: List of GenericMetadata
Mandatory-to-Specify: Yes.
Property: paths
Description: Path specific rules. First match applies.
Type: List of PathMatch
Mandatory-to-Specify: No.

4.1.4. PathMatch

The PathMatch object contains an expression given as a PatternMatch object to match against a resource URI path and Metadata objects to apply if a match is found.

Property: path-pattern
Description: Pattern to match against the requested path, i.e. against the [RFC3986] path-absolute.
Type: PatternMatch

Mandatory-to-Specify: Yes.
Property: path-metadata
Description: CDNI Metadata to apply when delivering content that matches this pattern.
Type: PathMetadata
Mandatory-to-Specify: Yes.

4.1.5. PathMetadata

A PathMetadata object contains the CDNI Metadata properties for content served with the associated URI path (defined in a PathMatch object). Note that if CDNI metadata is used as an input to CDNI request routing and DNS-based redirection is employed, then any metadata at the PathMetadata level or below will be inaccessible at request routing time.

Property: metadata
Description: List of path related metadata.
Type: List of GenericMetadata
Mandatory-to-Specify: Yes.
Property: paths
Description: Path specific rules. First match applies.
Type: List of PathMatch
Mandatory-to-Specify: No.

4.1.6. PatternMatch

A PatternMatch object contains the pattern string and flags that describe the PathMatch expression.

Property: pattern
Description: >A pattern for string matching. The pattern may contain the wildcards * and ?, where * matches any sequence of characters (including the empty string) and ? matches exactly one character. The three literals \ , * and ? should be escaped as \\, * and \?
Type: String
Mandatory-to-Specify: Yes.
Property: case-sensitive
Description: Flag indicating whether or not case-sensitive matching should be used.
Type: Boolean
Mandatory-to-Specify: No. Default is case-insensitive match.
Property: match-query-string
Description: Flag indicating whether or not the query string should be included in the pattern match.

Type: Boolean
Mandatory-to-Specify: No. Default is not to include query strings when matching.

4.1.7. GenericMetadata

A GenericMetadata object is an abstraction for managing individual CDNI Metadata properties in an opaque manner.

Property: type
Description: CDNI Metadata property object type.
Type: String
Mandatory-to-Specify: Yes.
Property: value
Description: CDNI Metadata property object.
Type: matches the type property above
Mandatory-to-Specify: Yes.
Property: mandatory-to-enforce
Description: Flag identifying whether or not the enforcement of the property Metadata is required.
Type: Boolean
Mandatory-to-Specify: Yes.
Property: safe-to-redistribute
Description: Flag identifying whether or not the the property Metadata may be safely redistributed without modification.
Type: Boolean
Mandatory-to-Specify: No. Default is allow transparent redistribution.

4.2. CDNI Metadata Property Object Descriptions

4.2.1. Source Metadata

Source Metadata provides the dCDN information about content acquisition e.g. how to contact an uCDN Surrogate or an Origin Server. The sources are not necessarily the actual Origin Servers operated by the CSP but might be a set of Surrogates in the uCDN.

Property: sources
Description: Sources from which the dCDN can acquire content.
Type: List of Source
Mandatory-to-Specify: No.

4.2.1.1. Source

A Source object describes the Source which should be used by the dCDN for content acquisition, e.g. a Surrogate within the uCDN or an alternate Origin Server, the protocol to be used and any

authentication method.

Property: auth
Description: Authentication method to use when requesting content from this source.
Type: Auth
Mandatory-to-Specify: No. Default is no authentication is required.
Property: endpoints
Description: Origins from which the dCDN can acquire content.
Type: List of EndPoint
Mandatory-to-Specify: Yes.
Property: protocol
Description: Protocol to use for content acquisition.
Type: Protocol
Mandatory-to-Specify: Yes.

4.2.2. LocationACL Metadata

LocationACL Metadata defines location-based restrictions.

Property: locations
Description: Access control list which applies restrictions to delivery based on client location.
Type: List of LocationRule
Mandatory-to-Specify: No. Default is allow all locations.

4.2.2.1. LocationRule

A LocationRule contains or references a list of Location objects. LocationRule objects are used to construct a LocationACL to apply restrictions to content delivery.

Property: locations
Description: List of locations to which the rule applies.
Type: List of Location
Mandatory-to-Specify: Yes.
Property: action
Description: Defines whether the rule specifies locations to allow or deny.
Type: Enumeration [allow|deny]
Mandatory-to-Specify: Yes.

4.2.2.2. Location

A Location object describes a Location which may be applied by an ACLRule, e.g. a Location may be an IPv4 address range or a geographic location.

Property: iprange
Description: A set of IP Addresses.
Type: List of IPRange.
Mandatory-to-Specify: Yes.

[Ed: Location as specified above only supports the Class 1a names described in [I-D.jenkins-cdni-names]. Need to add support for Class 1b names to a later version.]

4.2.3. TimeWindowACL Metadata

TimeWindowACL Metadata defines time-based restrictions.

Property: times
Description: Access control list which applies restrictions to delivery based on request time.
Type: List of TimeWindowRule
Mandatory-to-Specify: No. Default is allow all time windows.

4.2.3.1. TimeWindowRule

A TimeWindowRule contains or references a list of TimeWindow objects. TimeWindowRule objects are used to construct a TimeWindowACL to apply restrictions to content delivery.

Property: times
Description: List of time windows to which the rule applies.
Type: List of TimeWindow
Mandatory-to-Specify: Yes.
Property: action
Description: Defines whether the rule specifies time windows to allow or deny.
Type: Enumeration [allow|deny]
Mandatory-to-Specify: Yes.

4.2.3.2. TimeWindow

A TimeWindow object describes a time range which may be applied by an ACLRule, e.g. Start 09:00AM 01/01/2000 UTC End 17:00PM 01/01/2000 UTC.

Property: start
Description: The start time of the window.
Type: Time
Mandatory-to-Specify: Yes.
Property: end

Description: The end time of the window.
Type: Time
Mandatory-to-Specify: Yes.

4.2.4. ProtocolACL Metadata

ProtocolACL Metadata defines delivery protocol restrictions.

Property: protocols
Description: Access control list which applies restrictions to delivery based on delivery protocol.
Type: List of ProtocolRule
Mandatory-to-Specify: No. Default is allow all protocols.

4.2.4.1. ProtocolRule

A ProtocolRule contains or references a list of Protocol objects. ProtocolRule objects are used to construct a ProtocolACL to apply restrictions to content delivery.

Property: protocols
Description: List of protocols to which the rule applies.
Type: List of protocol
Mandatory-to-Specify: Yes.
Property: action
Description: Defines whether the rule specifies protocols to allow or deny.
Type: Enumeration [allow|deny]
Mandatory-to-Specify: Yes.

4.2.5. Authorization Metadata

Authorization Metadata define content authorization methods.

Property: methods
Description: Options for authenticating content requests. All options in the list are equally valid.
Type: List of Auth
Mandatory-to-Specify: No. Default is no authorization required.

4.2.6. Auth

An Auth object defines authentication and authorization methods to be used during content delivery and content acquisition, e.g. methods such as tokenization and URL Signing.

[Ed. Need to synchronize authentication configuration with CDNI URL

signing draft definitions.]

[Ed. Need to consider how to separate protocol specific method configuration (e.g., HTTP basic/digest authentication), which must match the HostMatch protocol, from protocol agnostic method configurations (e.g., URL signing/tokenization).]

4.3. CDNI Metadata Simple Data Type Descriptions

This section describes the simpler data types that are used for properties of CDNI Metadata objects.

4.3.1. Link

A link object may be used in place of any of the objects described above. Links can be used to avoid duplication if the same metadata information is repeated within the metadata tree. When a link replaces an object, its href property is set to the URI of the resource, its rel property is set to the name of the property it is replacing, and its type property is set to the type of the object it is replacing.

Property: href

Description: The URI of the of the addressable object being referenced.

Type: URI

Mandatory: Yes

Property: rel

Description: The Relationship between the referring object and the object it is referencing.

Type: String

Mandatory: Yes

Property: type

Description: The type of the object being referenced.

Type: String

Mandatory: Yes

4.3.2. Protocol

This type only appears in Links. Links with this type are not machine readable but rather represent particular feature sets of a protocol defined in a specification and implemented in code. The URI contained in the link needs to be defined for each delivery protocol with an associated interoperable feature set.

The following examples are illustrative:

- o `http://url.cdni.ietf.example/protocol/delivery/http/rfcABCD`
- o `http://url.cdni.ietf.example/protocol/delivery/rtmp/rfcEFGH`
- o `http://url.vendorY.ietf.example/protocol/delivery/rtmp/releaseP.Q`

[Editor's Note: It may be more appropriate to use the 'tag' URI scheme [RFC4151] for these URIs.]

4.3.3. Endpoint

A hostname (with optional port) or an IP address (with optional port).

Note: All implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986] and MUST support all IPv6 address formats specified in [RFC4291]. Server implementations SHOULD use IPv6 address formats specified in [RFC5952].

4.3.4. IPRange

One of:

- o A range of consecutive IP addresses (IPv4 or IPv6) expressed as Address1-Address2 which does not have to be to power of two aligned, for example the range 192.0.2.1-192.0.2.10 is valid. The first Address in the range MUST be 'lower' than the final address in the range.
- o A valid IP subnet (IPv4 or IPv6) expressed using CIDR notation.
- o A single IP address (IPv4 or IPv6).

Note: Client implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986] and MUST support all IPv6 address formats specified in [RFC4291]. Server implementations SHOULD use IPv6 address formats specified in [RFC5952].

4.3.5. URI

A URI as specified in [RFC3986].

4.3.6. Time

A time value expressed in seconds since Unix epoch in the UTC timezone.

5. CDNI Metadata interface

This section specifies an interface to enable a Downstream CDN to retrieve CDNI Metadata objects from an Upstream CDN.

The interface can be used by a Downstream CDN to retrieve CDNI Metadata objects either dynamically as required by the Downstream CDN to process received requests (for example in response to receiving a CDNI Request Routing request from an Upstream CDN or in response to receiving a request for content from a User Agent) or in advance of being required.

The CDNI Metadata interface is built on the principles of RESTful web services. This means that requests and responses over the interface are built around the transfer of representations of hyperlinked resources. A resource in the context of the CDNI Metadata interface is any object in the Data Model (as described in Section 3 through Section 4).

In the general case a CDNI Metadata server makes each instance of an addressable CDNI Metadata object available via a unique URI that returns a representation of that instance of that CDNI Metadata object. When an object needs to reference another addressable CDNI Metadata object (for example a HostIndex object referencing a HostMetadata object) it does so by including a link to the referenced object.

CDNI Metadata servers are free to assign whatever structure they desire to the URIs for CDNI Metadata objects and CDNI Metadata clients MUST NOT make any assumptions regarding the structure of CDNI Metadata URIs or the mapping between CDNI Metadata objects and their associated URIs. Therefore any URIs present in the examples below are purely illustrative and are not intended impose a definitive structure on CDNI Metadata interface implementations.

5.1. Transport

The CDNI Metadata interface uses HTTP as the underlying protocol transport.

The HTTP Method in the request defines the operation the request would like to perform. Servers implementing the CDNI Metadata interface MUST support the HTTP GET and HEAD methods.

The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses from servers implementing the CDNI Metadata interface that contain a

response body SHOULD include an ETag to enable validation of cached versions of returned resources.

The CDNI Metadata interface specified in this document is a read-only interface. Therefore support for other HTTP methods such as PUT, POST and DELETE etc. is not specified. Server implementations of this interface SHOULD reject all methods other than GET and HEAD.

As the CDNI Metadata interface builds on top of HTTP, CDNI Metadata servers may make use of any HTTP feature when implementing the CDNI Metadata interface, for example a CDNI Metadata server may make use of HTTP's caching mechanisms to indicate that the returned response/representation can be reused without re-contacting the CDNI Metadata server.

5.2. Retrieval of CDNI Metadata resources

In the general case a CDNI Metadata server makes each instance of an addressable CDNI Metadata object available via a unique URI and therefore in order to retrieve CDNI Metadata, a CDNI Metadata client first makes a HTTP GET request for the URI of the HostIndex which provides the CDNI Metadata client with a list of Hostnames that the upstream CDN may delegate to the downstream CDN.

In order to retrieve the CDNI Metadata for a particular request the CDNI Metadata client processes the received HostIndex object and finds the corresponding HostMetadata entry (by matching the hostname in the request against the hostnames in the HostIndex). The CDNI metadata client then makes a GET request for the URI specified in the href key of that Host's entry in the HostIndex.

In order to retrieve the most specific metadata for a particular request, the CDNI metadata client inspects the HostMetadata for references to more specific PathMetadata objects. If any PathMetadata match the request, the CDNI metadata client makes another GET request for the PathMetadata. Each PathMetadata object may also include references to yet more specific metadata. If this is the case, the CDNI metadata client continues requesting PathMetadata recursively.

Where a downstream CDN is interconnected with multiple upstream CDNs, the downstream CDN must decide which upstream CDN's CDNI metadata should be used to handle a particular User Agent request.

When application level redirection (e.g. HTTP 302 redirects) is being used between CDNs, it is expected that the downstream CDN will be able to determine the upstream CDN that redirected a particular request from information contained in the received request (e.g. via

the URI in case of HTTP redirection across CDNs). With knowledge of which upstream CDN routed the request, the downstream CDN can choose the correct metadata server from which to obtain the HostIndex. Note that the HostIndex served by each uCDN may be unique.

In the case of DNS redirection there is not sufficient information carried in the DNS request from User Agents to determine the upstream CDN that redirected a particular request and therefore downstream CDNs may have to apply local policy when deciding which upstream CDN's metadata to apply.

5.3. Bootstrapping

The URI for the HostIndex object of a given upstream CDN needs to be either discovered by or configured in the downstream CDN. All other objects/resources are then discoverable from the HostIndex object by following the links in the HostIndex object and the referenced HostMetadata and PathMetadata objects.

If the URI for the HostIndex object is not manually configured in the downstream CDN then the HostIndex URI could be discovered via the CDNI Control interface. An upstream CDN would provide the URI of the HostIndex object to the downstream CDN via the CDNI Control Interface.

5.4. Encoding

Object are resources that may be:

- o Addressable, where the object is a resource that may be retrieved or referenced via its own URI.
- o Embedded, where the object is contained (or inlined) within a property of an addressable object.

In the descriptions of objects we use the term "X contains Y" to mean either Y is directly embedded in X or that Y is linked to by X. It is generally a deployment choice for the uCDN implementation to decide when and which CDNI Metadata objects to embed and which are separately addressable.

5.4.1. MIME Media Types

All MIME types are prefixed with "application/cdni." The MIME type for each object matches the type name of that object as defined by this document. Table 3 lists a few examples of the MIME Media Type for each object (resource) that is retrievable through the CDNI Metadata interface. The MIME type suffix depends on the metadata encoding, either "+xml" or "+json".

Data Object	MIME Media Type
HostIndex	application/cdni.HostIndex
HostMatch	application/cdni.HostMatch
HostMetadata	application/cdni.HostMetadata
PathMatch	application/cdni.PathMatch
PathMetadata	application/cdni.PathMetadata

Table 3: MIME Media Types for CDNI Metadata resources

See <http://www.iana.org/assignments/media-types/index.html> for reference.

5.4.2. JSON Encoding of Objects

One possible encoding for a CDNI Metadata object is a JSON object containing a dictionary of (key,value) pairs where the keys are the property names and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource). Likewise, the values associated with each key are dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource).

Dictionary keys in JSON are case sensitive and therefore any dictionary key defined by this document (for example the names of CDNI Metadata object properties) MUST always be represented in lowercase.

In addition to the properties specific to each object type, the keys defined below may be present in any object.

Key: base

Description: Provides a prefix for any relative URLs in the object. This is similar to the XML base tag [XML-BASE]. If absent, all URLs in the remainder of the document must be absolute URLs.

Type: URI

Mandatory: No

Key: links

Description: The links of this object to other addressable objects. Any property may be replaced by a link to an object with the same type as the property it replaces.

Type: List of Link
Mandatory: Yes

5.4.2.1. JSON Example

A downstream CDN may request the HostIndex and receive the following object of type "application/cdni.HostIndex+json":

```
{
  "hosts": [
    {
      "host": "video.example.com",
      "links": [
        {
          "rel": "host-metadata",
          "type": "application/cdni.HostMetadata",
          "href": "http://metadata.example.ucdn.com/video"
        }
      ]
    },
    {
      "host": "images.example.com",
      "links": [
        {
          "rel": "host-metadata",
          "type": "application/cdni.HostMetadata",
          "href": "http://metadata.ucdn.example.com/images"
        }
      ]
    }
  ]
}
```

If the incoming request has a Host header with "video.example.com" then the downstream CDN would fetch from the next metadata object from "http://metadata.ucdn.example.com/video" expecting a MIME type of "application/cdni.HostMetadata+json":

```
{
  "metadata": [
    {
      "type": "application/cdni.SourceMetadata",
      "value": {
        "sources": [
          {
            "links": [{
              "rel": "auth",
              "type": "application/cdni.Auth",

```

```
        "href": "http://metadata.ucdn.example.com/auth1234"
      }],
      "endpoint": "acq1.ucdn.example.com",
      "protocol": "ftp"
    },
    {
      "links": [{
        "rel": "auth",
        "type": "application/cdni.Auth",
        "href": "http://metadata.ucdn.example.com/auth1234"
      }],
      "endpoint": "acq2.ucdn.example.com",
      "protocol": "http"
    }
  ]
}
},
{
  "type": "application/cdni.LocationACL",
  "value": {
    "locations": [
      {
        "locations": [
          { "iprange": "192.168.0.0/16" }
        ],
        "action": "deny"
      }
    ]
  }
},
{
  "type": "application/cdni.ProtocolACL",
  "value": {
    "protocols": [
      {
        "protocols": [
          "ftp"
        ],
        "action": "deny"
      }
    ]
  }
},
{
  "paths": [
    {
      "path-pattern": {
        "pattern": "/videos/trailers/*"
      }
    }
  ]
}
```

```

    },
    "links": [{
      "rel": "path-metadata",
      "type": "application/cdni.PathMetadata",
      "href": "http://metadata.ucdn.example.com/videos/trailers"
    }]
  },
  {
    "path-pattern": {
      "pattern": "/videos/movies/*"
    },
    "links": [{
      "rel": "pathmetadata",
      "type": "application/cdni.PathMetadata",
      "href": "http://metadata.ucdn.example.com/videos/movies"
    }]
  }
]
}

```

Suppose the path of the requested resource matches the `"/video/movies/*"` pattern, the next metadata requested would be for `"http://metadata.ucdn.example.com/video/movies"` with an expected type of `"application/cdni.PathMetadata"`:

```

{
  "metadata": [],
  "paths": [
    {
      "path-pattern": {
        "pattern": "/videos/movies/hd/*"
      },
      "links": [{
        "rel": "pathmetadata",
        "type": "application/cdni.PathMetadata",
        "href": "http://metadata.ucdn.example.com/videos/movies/hd"
      }]
    }
  ]
}

```

Finally, if the path of the requested resource also matches the `"/videos/movies/hd/*"` pattern, the downstream CDN would also fetch the following object from `"http://metadata.ucdn.example.com/videos/movies/hd"` with MIME type `"application/cdni.PathMetadata"`:


```
{
  "metadata": [
    {
      "type": "application/cdni.TimeWindowACL",
      "value": {
        "times": [
          {
            "start": "1213948800",
            "end": "1327393200"
          }
        ],
        "type": "allow"
      }
    }
  ]
}
```

5.4.3. XML Encoding of Objects

Another possible encoding for a CDNI Metadata object is an XML document containing elements with tag names which match property names and values which match the associated property values.

Tag names of elements are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource). Likewise, the values associated with each element are dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource).

Lists are encoded by repeating the singular form of a property name. For example the "hosts" property is a list of "HostMatch" objects. This list would be encoded as multiple "host" elements.

Link objects are a special case. If a Link object replaces a property then a "link" element replaces the expected element. The properties of the Link object are encoded as XML attributes. The type attribute is set to the MIME type of the target object. The href attribute is set to the URI of the target object. The rel attribute is set to the name of the element being replaced.

5.4.3.1. XML Example

A downstream CDN may request the HostIndex and receive the following object of type "application/cdni.HostIndex+xml":

```
<HostIndex>
  <host>
    <host>video.example.com</host>
    <link rel="host-metadata" type="application/cdni.HostMetadata"
      href="http://metadata.ucdn.example.com/video"/>
  </host>
  <host>
    <host>images.example.com</host>
    <link rel="host-metadata" type="application/cdni.HostMetadata"
      href="http://metadata.ucdn.example.com/images"/>
  </host>
</HostIndex>
```

If the incoming request has a Host header with "video.example.com" then the downstream CDN would fetch from the next metadata object from "http://metadata.ucdn.example.com/video" expecting a MIME type of "application/cdni.HostMetadata+xml":

```
<HostMetadata>
  <metadata>
    <type>application/cdni.SourceMetadata</type>
    <value>
      <sources>
        <link rel="auth" type="application/cdni.Auth"
          href="http://metadata.ucdn.example.com/auth1234"/>
        <endpoint>acq1.ucdn.example.com</endpoint>
        <protocol>ftp</protocol>
      </source>
      <source>
        <link rel="auth" type="application/cdni.Auth"
          href="http://metadata.ucdn.example.com/auth1234"/>
        <endpoint>acq2.ucdn.example.com</endpoint>
        <protocol>http</protocol>
      </source>
    </value>
  </metadata>
  <metadata>
    <type>application/cdni.LocationACL</type>
    <value>
      <location>
        <location>
          <iprange>192.168.0.0/16</iprange>
        </location>
        <action>deny</type>
      </location>
    </value>
  </metadata>
</metadata>
```

```

    <type>application/cdni.ProtocolACL</type>
    <value>
      <protocol>
        <protocol>ftp</protocol>
        <action>deny</action>
      </protocol>
    </value>
  </metadata>
  <path>
    <path-pattern>
      <pattern>/videos/trailers/*</pattern>
    </path-pattern>
    <link rel="path-metadata" type="application/cdni.PathMetadata"
      href="http://metadata.ucdn.example.com/videos/trailers"/>
  </path>
  <path>
    <path-pattern>
      <pattern>/videos/movies/*</pattern>
    </path-pattern>
    <link rel="path-metadata" type="application/cdni.PathMetadata"
      href="http://metadata.ucdn.example.com/videos/movies"/>
  </path>
</HostMetadata>

```

Suppose the path of the requested resource matches the `"/video/movies/*"` pattern, the next metadata requested would be for `"http://metadata.ucdn.example.com/video/movies"` with an expected type of `"application/cdni.PathMetadata"`:

```

<PathMetadata>
  <path>
    <path-pattern>
      <pattern>/videos/movies/hd/*</pattern>
    </path-pattern>
    <link rel="path-metadata" type="application/cdni.PathMetadata"
      href="http://metadata.ucdn.example.com/videos/movies/hd"/>
  </path>
</PathMetadata>

```

Finally, if the path of the requested resource also matches the `"/videos/movies/hd/*"` pattern, the downstream CDN would also fetch the following object from `"http://metadata.ucdn.example.com/videos/movies/hd"` with MIME type `"application/cdni.PathMetadata"`:

```
<PathMetadata>
  <metadata>
    <type>application/cdni.TimeWindowACL</type>
    <value>
      <time>
        <time>
          <start>1213948800</start>
          <end>1327393200</end>
        </time>
      <type>allow</type>
    </time>
  </metadata>
</PathMetadata>
```

5.5. Extensibility

The set of property Metadata may be extended with proprietary and/or custom property Metadata. The GenericMetadata object defined in Section 4.1.7 allows any Metadata property to be included in either the HostMetadata or PathMetadata lists. As described in Section 3.4, it is suggested that proprietary and/or custom property Metadata be identified by the "ext." prefix in an appropriately descriptive type which conveys the organization defining the property Metadata and the function of the property Metadata.

Note: Identification of the property Metadata defining organization in the property Metadata type decreases the possibility of property Metadata type collision.

5.5.1. Metadata Enforcement

At any given time, the set of property Metadata supported by the uCDN may not match the set of property Metadata supported by the dCDN. The uCDN may or may not know which property Metadata the dCDN supports. In cases where the uCDN supports Metadata that the dCDN does not, the dCDN MUST be aware of any Metadata marked as "mandatory-to-enforce". If a CDN does not understand or is unable to perform the functions associated with any "mandatory-to-enforce" Metadata, the CDN MUST NOT service any requests for the corresponding content.

Note: Ideally, uCDNs would not delegate content requests to a dCDN which does not support the Metadata associated with the content being requested. However, even if the uCDN has a priori knowledge of the Metadata supported by the dCDN (e.g., via the CDNI capabilities interface or through out-of-band negotiation between CDN operators) Metadata support may fluctuate or be inconsistent (e.g., due to miscommunication, mis-configuration, or temporary outage). The dCDN

MUST evaluate all Metadata associated with content requests and reject any requests where "mandatory-to-enforce" Metadata associated with the content cannot be enforced.

5.5.2. Metadata Override

It is possible that new Metadata definitions may obsolete or override existing property Metadata (e.g., a future revision of the CDNI Metadata interface may redefine the Auth Metadata or a custom vendor extension may implement an alternate Auth Metadata option). If multiple Metadata (e.g., cdni.v2.Auth, ext.vendor1.Auth, and ext.vendor2.Auth) all override an existing Metadata (e.g., cdni.Auth) and all are marked as "mandatory-to-enforce", it may be ambiguous which Metadata should be applied, especially if the functionality of the Metadata conflict.

As described in Section 3.3, Metadata override only applies to Metadata objects of the same exact type, found in HostMetadata and nested PathMetadata structures. The CDNI Metadata interface does not support enforcement of dependencies between different Metadata types. It is the responsibility of the CSP and the CDN operators to ensure that Metadata assigned to a given content asset do not conflict.

Note: Because Metadata is inherently ordered in GenericMetadata lists, as well as in the PathMetadata hierarchy and PathMatch lists, multiple conflicting Metadata types MAY be used, however, Metadata hierarchies MUST ensure that independent PathMatch root objects are used to prevent ambiguous or conflicting Metadata definitions.

6. IANA Considerations

This document requests the registration of the "application/cdni" MIME type.

[Ed. Need to consider a registry for Metadata type identifiers.]

7. Security Considerations

The CDNI Metadata Interface is expected to be secured as a function of the transport protocol (e.g. HTTP authentication, HTTPS, or inter-domain IPSec).

If a malicious metadata server is contacted by a downstream CDN, the malicious server may provide metadata to the downstream CDN which denies service for any piece of content to any user agent. The malicious server may also provide metadata which directs a downstream

CDN to a malicious origin server instead of the actual origin server.

A malicious metadata client could request metadata for a piece of content from an upstream CDN. The metadata information may then be used to glean information regarding the uCDN or to contact an upstream origin server. The uCDN is expected to authenticate client requests to prevent this situation.

8. Acknowledgements

The authors would like to thank David Ferguson and Francois le Faucheur for their valuable comments and input to this document.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

9.2. Informative References

- [I-D.davie-cdni-framework]
Davie, B. and L. Peterson, "Framework for CDN Interconnection", draft-davie-cdni-framework-00 (work in progress), July 2011.
- [I-D.ietf-cdni-problem-statement]
Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-ietf-cdni-problem-statement-03 (work in progress), January 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-02 (work in progress), December 2011.
- [I-D.zyp-json-schema]

Zyp, K. and G. Court, "A JSON Media Type for Describing the Structure and Meaning of JSON Documents", draft-zyp-json-schema-03 (work in progress), November 2010.

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

[RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", RFC 4151, October 2005.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[XML-BASE] Marsh, J., Ed. and R. Tobin, Ed., "XML Base (Second Edition) - <http://www.w3.org/TR/xmlbase/>", January 2009.

Appendix A. Relationship to the CDNI Requirements

Section 6 of [I-D.ietf-cdni-requirements] lists the requirements for the CDNI Metadata Distribution interface. This section outlines which of those requirements are met by the CDNI Metadata interface specified in this document.

All metadata requirements are met either directly or indirectly by the CDNI Metadata Interface described in this document. The following paragraphs describe notable exceptions.

Requirements related to pre-positioning of metadata are not met directly by this document. Triggering metadata pre-positioning is beyond the scope of the CDNI Metadata interface. However, the interface as described by this document supports pulling metadata on-demand for the purpose of pre-positioning.

Requirement META-13 relating to feedback from the downstream CDN to the upstream CDN with respect to metadata is not directly supported by the pull-based interface described in this document. As an alternative, the downstream CDN may use the CDNI Logging interface to convey error conditions related to metadata.

Requirement META-18 relating to surrogate cache behavior parameters is supported via extensibility. However, the example parameters in META-18 are not described in this document.

Authors' Addresses

Ben Niven-Jenkins
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: ben@velocix.com

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: rmurray@velocix.com

Grant Watson
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: gwatson@velocix.com

Matt Caulfield
Cisco Systems
1414 Massachusetts Avenue
Boxborough, MA 01719
USA

Phone: +1 978 936 9307
Email: mcaulfie@cisco.com

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose 95134
USA

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

Kevin J. Ma
Azuki Systems, Inc.
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.ma@azukisystems.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 1, 2017

B. Niven-Jenkins
R. Murray
Velocix (Alcatel-Lucent)
M. Caulfield
Cisco Systems
K. Ma
Ericsson
August 28, 2016

CDN Interconnection Metadata
draft-ietf-cdni-metadata-21

Abstract

The Content Delivery Networks Interconnection (CDNI) metadata interface enables interconnected Content Delivery Networks (CDNs) to exchange content distribution metadata in order to enable content acquisition and delivery. The CDNI metadata associated with a piece of content provides a downstream CDN with sufficient information for the downstream CDN to service content requests on behalf of an upstream CDN. This document describes both a base set of CDNI metadata and the protocol for exchanging that metadata.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 March 1, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Terminology	5
1.2. Supported Metadata Capabilities	5
2. Design Principles	6
3. CDNI Metadata object model	7
3.1. HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects	8
3.2. Generic CDNI Metadata Objects	10
3.3. Metadata Inheritance and Override	13
4. CDNI Metadata objects	14
4.1. Definitions of the CDNI structural metadata objects . . .	15
4.1.1. HostIndex	15
4.1.2. HostMatch	15
4.1.3. HostMetadata	17
4.1.4. PathMatch	18
4.1.5. PatternMatch	19
4.1.6. PathMetadata	20
4.1.7. GenericMetadata	21
4.2. Definitions of the initial set of CDNI Generic Metadata objects	23
4.2.1. SourceMetadata	23
4.2.1.1. Source	24
4.2.2. LocationACL Metadata	25
4.2.2.1. LocationRule	27
4.2.2.2. Footprint	27
4.2.3. TimeWindowACL	29
4.2.3.1. TimeWindowRule	30
4.2.3.2. TimeWindow	31
4.2.4. ProtocolACL Metadata	31
4.2.4.1. ProtocolRule	32
4.2.5. DeliveryAuthorization Metadata	33

4.2.6.	Cache	34
4.2.7.	Auth	36
4.2.8.	Grouping	37
4.3.	CDNI Metadata Simple Data Type Descriptions	37
4.3.1.	Link	37
4.3.1.1.	Link Loop Prevention	39
4.3.2.	Protocol	39
4.3.3.	Endpoint	39
4.3.4.	Time	40
4.3.5.	IPv4CIDR	40
4.3.6.	IPv6CIDR	40
4.3.7.	ASN	41
4.3.8.	CountryCode	41
5.	CDNI Metadata Capabilities	41
6.	CDNI Metadata interface	42
6.1.	Transport	42
6.2.	Retrieval of CDNI Metadata resources	43
6.3.	Bootstrapping	44
6.4.	Encoding	44
6.5.	Extensibility	45
6.6.	Metadata Enforcement	46
6.7.	Metadata Conflicts	46
6.8.	Versioning	47
6.9.	Media Types	48
6.10.	Complete CDNI Metadata Example	48
7.	IANA Considerations	52
7.1.	CDNI Payload Types	52
7.1.1.	CDNI MI HostIndex Payload Type	53
7.1.2.	CDNI MI HostMatch Payload Type	53
7.1.3.	CDNI MI HostMetadata Payload Type	54
7.1.4.	CDNI MI PathMatch Payload Type	54
7.1.5.	CDNI MI PatternMatch Payload Type	54
7.1.6.	CDNI MI PathMetadata Payload Type	54
7.1.7.	CDNI MI SourceMetadata Payload Type	54
7.1.8.	CDNI MI Source Payload Type	55
7.1.9.	CDNI MI LocationACL Payload Type	55
7.1.10.	CDNI MI LocationRule Payload Type	55
7.1.11.	CDNI MI Footprint Payload Type	55
7.1.12.	CDNI MI TimeWindowACL Payload Type	55
7.1.13.	CDNI MI TimeWindowRule Payload Type	56
7.1.14.	CDNI MI TimeWindow Payload Type	56
7.1.15.	CDNI MI ProtocolACL Payload Type	56
7.1.16.	CDNI MI ProtocolRule Payload Type	56
7.1.17.	CDNI MI DeliveryAuthorization Payload Type	56
7.1.18.	CDNI MI Cache Payload Type	57
7.1.19.	CDNI MI Auth Payload Type	57
7.1.20.	CDNI MI Grouping Payload Type	57
7.2.	CDNI Metadata Footprint Types Registry	57

7.3. CDNI Metadata Protocol Types Registry	58
8. Security Considerations	58
8.1. Authentication and Integrity	59
8.2. Confidentiality and Privacy	59
8.3. Securing the CDNI Metadata interface	60
9. Acknowledgements	60
10. Contributing Authors	60
11. References	61
11.1. Normative References	61
11.2. Informative References	63
Authors' Addresses	64

1. Introduction

Content Delivery Networks Interconnection (CDNI) [RFC6707] enables a downstream Content Delivery Network (dCDN) to service content requests on behalf of an upstream CDN (uCDN).

The CDNI metadata interface is discussed in [RFC7336] along with four other interfaces that can be used to compose a CDNI solution (CDNI Control interface, CDNI Request Routing Redirection interface, CDNI Footprint & Capabilities Advertisement interface and CDNI Logging interface). [RFC7336] describes each interface and the relationships between them. The requirements for the CDNI metadata interface are specified in [RFC7337].

The CDNI metadata associated with a piece of content (or with a set of content) provides a dCDN with sufficient information for servicing content requests on behalf of an uCDN, in accordance with the policies defined by the uCDN.

This document defines the CDNI metadata interface which enables a dCDN to obtain CDNI metadata from an uCDN so that the dCDN can properly process and respond to:

- o Redirection requests received over the CDNI Request Routing Redirection interface [I-D.ietf-cdni-redirection].
- o Content requests received directly from User Agents.

Specifically, this document specifies:

- o A data structure for mapping content requests and redirection requests to CDNI metadata objects (Section 3 and Section 4.1).
- o An initial set of CDNI Generic metadata objects (Section 4.2).
- o A HTTP web service for the transfer of CDNI metadata (Section 6).

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

Additionally, the following terms are used throughout this document and are defined as follows:

- o Object - a collection of properties.
- o Property - a key and value pair where the key is a property name and the value is the property value or another object.

This document uses the phrase "[Object] A contains [Object] B" for simplicity when a strictly accurate phrase would be "[Object] A contains or references (via a Link object) [Object] B".

1.2. Supported Metadata Capabilities

Only the metadata for a small set of initial capabilities is specified in this document. This set provides the minimum amount of metadata for basic CDN interoperability while still meeting the requirements set forth by [RFC7337].

The following high-level functionality can be configured via the CDNI metadata objects specified in Section 4:

- o Acquisition Source: Metadata for allowing a dCDN to fetch content from a uCDN.
- o Delivery Access Control: Metadata for restricting (or permitting) access to content based on any of the following factors:
 - * Location
 - * Time Window
 - * Delivery Protocol
- o Delivery Authorization: Metadata for authorizing dCDN user agent requests.
- o Cache Control: Metadata for controlling cache behavior of the dCDN.

The metadata encoding described by this document is extensible in order to allow for future additions to this list.

The set of metadata specified in this document covers the initial capabilities above. It is only intended to support CDN interconnection for the delivery of content by a dCDN using HTTP/1.1 [RFC7230] and for a dCDN to be able to acquire content from a uCDN using either HTTP/1.1 or HTTP/1.1 over TLS [RFC2818].

Supporting CDN interconnection for the delivery of content using unencrypted HTTP/2 [RFC7540] (as well as for a dCDN to acquire content using unencrypted HTTP/2 or HTTP/2 over TLS) requires the registration of these protocol names in the CDNI Metadata Protocol Types registry Section 7.3.

Delivery of content using HTTP/1.1 over TLS or HTTP/2 over TLS SHOULD follow the guidelines set forth in [RFC7525]. Offline configuration of TLS parameters between CDNs is beyond the scope of this document.

2. Design Principles

The CDNI metadata interface was designed to achieve the following objectives:

1. Cacheability of CDNI metadata objects;
2. Deterministic mapping from redirection requests and content requests to CDNI metadata properties;
3. Support for DNS redirection as well as application-specific redirection (for example HTTP redirection);
4. Minimal duplication of CDNI metadata; and
5. Leveraging of existing protocols.

Cacheability can decrease the latency of acquiring metadata while maintaining its freshness, and therefore decrease the latency of serving content requests and redirection requests, without sacrificing accuracy. The CDNI metadata interface uses HTTP and its existing caching mechanisms to achieve CDNI metadata cacheability.

Deterministic mappings from content to metadata properties eliminates ambiguity and ensures that policies are applied consistently by all dCDNs.

Support for both HTTP and DNS redirection ensures that the CDNI metadata meets the same design principles for both HTTP and DNS based redirection schemes.

Minimal duplication of CDNI metadata improves storage efficiency in the CDNs.

Leveraging existing protocols avoids reinventing common mechanisms such as data structure encoding (by leveraging I-JSON [RFC7493]) and data transport (by leveraging HTTP [RFC7230]).

3. CDNI Metadata object model

The CDNI metadata object model describes a data structure for mapping redirection requests and content requests to metadata properties. Metadata properties describe how to acquire content from an uCDN, authorize access to content, and deliver content from a dCDN. The object model relies on the assumption that these metadata properties can be grouped based on the hostname of the content and subsequently on the resource path (URI) of the content. The object model associates a set of CDNI metadata properties with a Hostname to form a default set of metadata properties for content delivered on behalf of that Hostname. That default set of metadata properties can be overridden by properties that apply to specific paths within a URI.

Different Hostnames and URI paths will be associated with different sets of CDNI metadata properties in order to describe the required behaviour when a dCDN surrogate or request router is processing User Agent requests for content at that Hostname and URI path. As a result of this structure, significant commonality could exist between the CDNI metadata properties specified for different Hostnames, different URI paths within a Hostname and different URI paths on different Hostnames. For example the definition of which User Agent IP addresses should be grouped together into a single network or geographic location is likely to be common for a number of different Hostnames; although a uCDN is likely to have several different policies configured to express geo-blocking rules, it is likely that a single geo-blocking policy could be applied to multiple Hostnames delivered through the CDN.

In order to enable the CDNI metadata for a given Hostname and URI Path to be decomposed into reusable sets of CDNI metadata properties, the CDNI metadata interface splits the CDNI metadata into separate objects. Efficiency is improved by enabling a single CDNI metadata object (that is shared across Hostname and/or URI paths) to be retrieved and stored by a dCDN once, even if it is referenced by the CDNI metadata for multiple Hostnames and/or URI paths.

Important Note: Any CDNI metadata object A that contains another CDNI metadata object B can include a Link object specifying a URI that can be used to retrieve object B, instead of embedding object B within object A. The remainder of this document uses the phrase "[Object] A

contains [Object] B" for simplicity when a strictly accurate phrase would be "[Object] A contains or references (via a Link object) [Object] B". It is generally a deployment choice for the uCDN implementation to decide when to embed CDNI metadata objects and when to reference separate resources via Link objects.

Section 3.1 introduces a high level description of the HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects, and describes the relationships between them.

Section 3.2 introduces a high level description of the CDNI GenericMetadata object which represents the level at which CDNI metadata override occurs between HostMetadata and PathMetadata objects.

Section 4 describes in detail the specific CDNI metadata objects and properties specified by this document which can be contained within a CDNI GenericMetadata object.

3.1. HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects

The relationships between the HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects are described in Figure 1.

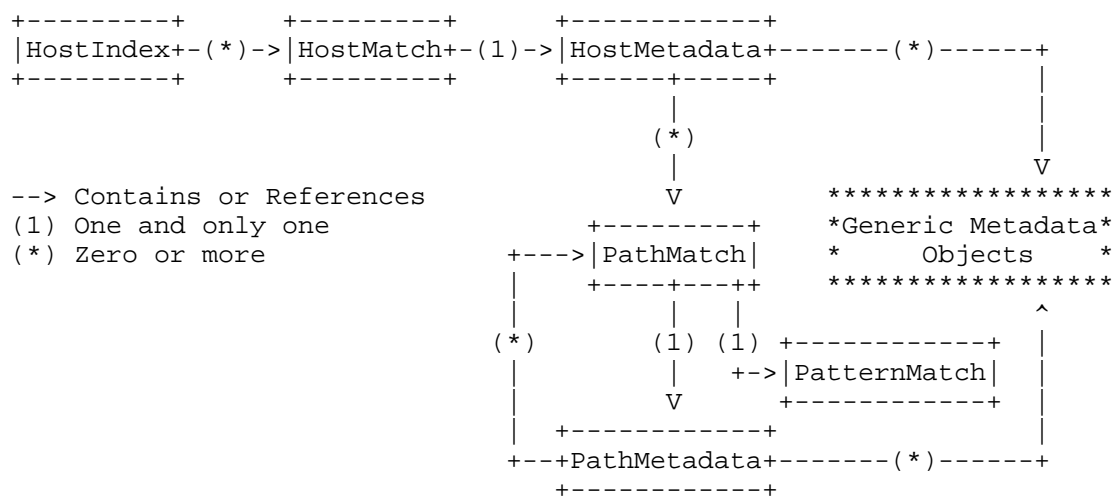


Figure 1: Relationships between CDNI Metadata Objects (Diagram Representation)

A HostIndex object (see Section 4.1.1) contains an array of HostMatch objects (see Section 4.1.2) that contain Hostnames (and/or IP addresses) for which content requests might be delegated to the dCDN. The HostIndex is the starting point for accessing the uCDN CDNI metadata data store. It enables the dCDN to deterministically discover which CDNI metadata objects it requires in order to deliver a given piece of content.

The HostIndex links Hostnames (and/or IP addresses) to HostMetadata objects (see Section 4.1.3) via HostMatch objects. A HostMatch object defines a Hostname (or IP address) to match against a requested host and contains a HostMetadata object.

HostMetadata objects contain the default GenericMetadata objects (see Section 4.1.7) required to serve content for that host. When looking up CDNI metadata, the dCDN looks up the requested Hostname (or IP address) against the HostMatch entries in the HostIndex, from there it can find HostMetadata which describes the default metadata properties for each host as well as PathMetadata objects (see Section 4.1.6), via PathMatch objects (see Section 4.1.4). PathMatch objects define patterns, contained inside PatternMatch objects (see Section 4.1.5), to match against the requested URI path. PatternMatch objects contain the pattern strings and flags that describe the URI path that a PathMatch applies to. PathMetadata objects contain the GenericMetadata objects that apply to content requests matching the defined URI path pattern. PathMetadata properties override properties previously defined in HostMetadata or less specific PathMatch paths. PathMetadata objects can contain additional PathMatch objects to recursively define more specific URI paths to which GenericMetadata properties might be applied.

A GenericMetadata object contains individual CDNI metadata objects which define the specific policies and attributes needed to properly deliver the associated content. For example, a GenericMetadata object could describe the source from which a CDN can acquire a piece of content. The GenericMetadata object is an atomic unit that can be referenced by HostMetadata or PathMetadata objects.

For example, if "example.com" is a content provider, a HostMatch object could include an entry for "example.com" with the URI of the associated HostMetadata object. The HostMetadata object for "example.com" describes the metadata properties which apply to "example.com" and could contain PathMatches for "example.com/movies/" and "example.com/music/", which in turn reference corresponding PathMetadata objects that contain the properties for those more specific URI paths. The PathMetadata object for "example.com/movies/" describes the properties which apply to that URI path. It could also contain a PathMatch object for

"example.com/movies/hd/*" which would reference the corresponding PathMetadata object for the "example.com/movies/hd/" path prefix.

The relationships in Figure 1 are also represented in tabular format in Table 1 below.

Data Object	Objects it contains or references
HostIndex	0 or more HostMatch objects.
HostMatch	1 HostMetadata object.
HostMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.
PathMatch	1 PatternMatch object. 1 PathMetadata object.
PatternMatch	Does not contain or reference any other objects.
PathMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.

Table 1: Relationships between CDNI Metadata Objects
(Table Representation)

3.2. Generic CDNI Metadata Objects

The HostMetadata and PathMetadata objects contain other CDNI metadata objects that contain properties which describe how User Agent requests for content should be processed, for example where to acquire the content from, authorization rules that should be applied, geo-blocking restrictions, and so on. Each such CDNI metadata object is a specialization of a CDNI GenericMetadata object. The GenericMetadata object abstracts the basic information required for metadata override and metadata distribution, from the specifics of any given property (i.e., property semantics, enforcement options, etc.).

The GenericMetadata object defines the properties contained within it as well as whether or not the properties are "mandatory-to-enforce". If the dCDN does not understand or support a "mandatory-to-enforce" property, the dCDN MUST NOT serve the content. If the property is not "mandatory-to-enforce", then that GenericMetadata object can be safely ignored and the content request can be processed in accordance with the rest of the CDNI metadata.

Although a CDN MUST NOT serve content to a User Agent if a "mandatory-to-enforce" property cannot be enforced, it could still be "safe-to-redistribute" that metadata to another CDN without modification. For example, in the cascaded CDN case, a transit CDN (tCDN) could pass through "mandatory-to-enforce" metadata to a dCDN.

For metadata which does not require customization or translation (i.e., metadata that is "safe-to-redistribute"), the data representation received off the wire MAY be stored and redistributed without being understood or supported by the transit CDN. However, for metadata which requires translation, transparent redistribution of the uCDN metadata values might not be appropriate. Certain metadata can be safely, though perhaps not optimally, redistributed unmodified. For example, source acquisition address might not be optimal if transparently redistributed, but it might still work.

Redistribution safety MUST be specified for each GenericMetadata property. If a CDN does not understand or support a given GenericMetadata property that is not "safe-to-redistribute", the CDN MUST set the "incomprehensible" flag to true for that GenericMetadata object before redistributing the metadata. The "incomprehensible" flag signals to a dCDN that the metadata was not properly transformed by the transit CDN. A CDN MUST NOT attempt to use metadata that has been marked as "incomprehensible" by a uCDN.

Transit CDNs MUST NOT change the value of "mandatory-to-enforce" or "safe-to-redistribute" when propagating metadata to a dCDN. Although a transit CDN can set the value of "incomprehensible" to true, a transit CDN MUST NOT change the value of "incomprehensible" from true to false.

Table 2 describes the action to be taken by a transit CDN (tCDN) for the different combinations of "mandatory-to-enforce" (MtE) and "safe-to-redistribute" (StR) properties, when the tCDN either does or does not understand the metadata in question:

MtE	StR	Metadata Understood by tCDN	Action
False	True	True	Can serve and redistribute.
False	True	False	Can serve and redistribute.
False	False	False	Can serve. MUST set "incomprehensible" to True when redistributing.
False	False	True	Can serve. Can redistribute after transforming the metadata (if the CDN knows how to do so safely), otherwise MUST set "incomprehensible" to True when redistributing.
True	True	True	Can serve and redistribute.
True	True	False	MUST NOT serve but can redistribute.
True	False	True	Can serve. Can redistribute after transforming the metadata (if the CDN knows how to do so safely), otherwise MUST set "incomprehensible" to True when redistributing.
True	False	False	MUST NOT serve. MUST set "incomprehensible" to True when redistributing.

Table 2: Action to be taken by a tCDN for the different combinations of MtE and StR properties

Table 3 describes the action to be taken by a dCDN for the different combinations of "mandatory-to-enforce" (MtE) and "incomprehensible" (Incomp) properties, when the dCDN either does or does not understand the metadata in question:

MtE	Incomp	Metadata Understood by dCDN	Action
False	False	True	Can serve.
False	True	True	Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.
False	False	False	Can serve.
False	True	False	Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.
True	False	True	Can serve.
True	True	True	MUST NOT serve.
True	False	False	MUST NOT serve.
True	True	False	MUST NOT serve.

Table 3: Action to be taken by a dCDN for the different combinations of MtE and Incomp properties

3.3. Metadata Inheritance and Override

In the metadata object model, a HostMetadata object can contain multiple PathMetadata objects (via PathMatch objects). Each PathMetadata object can in turn contain other PathMetadata objects. HostMetadata and PathMetadata objects form an inheritance tree where each node in the tree inherits or overrides the property values set by its parent.

GenericMetadata objects of a given type override all GenericMetadata objects of the same type previously defined by any parent object in the tree. GenericMetadata objects of a given type previously defined by a parent object in the tree are inherited when no object of the same type is defined by the child object. For example, if HostMetadata for the host "example.com" contains GenericMetadata objects of type LocationACL and TimeWindowACL, while a PathMetadata object which applies to "example.com/movies/*" defines an alternate GenericMetadata object of type TimeWindowACL, then:

- o the TimeWindowACL defined in the PathMetadata would override the TimeWindowACL defined in the HostMetadata for all User Agent requests for content under "example.com/movies/", and
- o the LocationACL defined in the HostMetadata would be inherited for all User Agent requests for content under "example.com/movies/".

A single HostMetadata or PathMetadata object MUST NOT contain multiple GenericMetadata objects of the same type. If an array of GenericMetadata contains objects of duplicate types, the receiver MUST ignore all but the first object of each type.

4. CDNI Metadata objects

Section 4.1 provides the definitions of each metadata object type introduced in Section 3. These metadata objects are described as structural metadata objects as they provide the structure for host and URI path-based inheritance and identify which GenericMetadata objects apply to a given User Agent content request.

Section 4.2 provides the definitions for a base set of core metadata objects which can be contained within a GenericMetadata object. These metadata objects govern how User Agent requests for content are handled. GenericMetadata objects can contain other GenericMetadata as properties; these can be referred to as sub-objects). As with all CDNI metadata objects, the value of the GenericMetadata sub-objects can be either a complete serialized representation of the sub-object, or a Link object that contains a URI that can be dereferenced to retrieve the complete serialized representation of the property sub-object.

Section 6.5 discusses the ability to extend the base set of GenericMetadata objects specified in this document with additional standards-based or vendor specific GenericMetadata objects that might be defined in the future in separate documents.

dCDNs and tCDNs MUST support parsing of all CDNI metadata objects specified in this document. A dCDN does not have to implement the underlying functionality represented by non-structural GenericMetadata objects (though that might restrict the content that a given dCDN will be able to serve). uCDNs as generators of CDNI metadata only need to support generating the CDNI metadata that they need in order to express the policies required by the content they are describing. See Section 6.4 for more details on the specific encoding rules for CDNI metadata objects.

Note: In the following sections, the term "mandatory-to-specify" is used to convey which properties MUST be included for a given structural or GenericMetadata object. When mandatory-to-specify is specified as "Yes" for an individual property, it means that if the object containing that property is included in a metadata response, then the mandatory-to-specify property MUST also be included (directly or by reference) in the response, e.g., a HostMatch property object without a host to match against does not make sense,

therefore, the host property is mandatory-to-specify inside a HostMatch object.

4.1. Definitions of the CDNI structural metadata objects

Each of the sub-sections below describe the structural objects introduced in Section 3.1.

4.1.1. HostIndex

The HostIndex object is the entry point into the CDNI metadata hierarchy. It contains an array of HostMatch objects. An incoming content request is checked against the Hostname (or IP address) specified by each of the listed HostMatch objects to find the HostMatch object which applies to the request.

Property: hosts

Description: Array of HostMatch objects. Hosts (HostMatch objects) MUST be evaluated in the order they appear and the first HostMatch object that matches the content request being processed MUST be used.

Type: Array of HostMatch objects

Mandatory-to-Specify: Yes.

Example HostIndex object containing two HostMatch objects, where the first HostMatch object is embedded and the second HostMatch object is referenced:

```
{
  "hosts": [
    {
      <Properties of embedded HostMatch object>
    },
    {
      "type": "MI.HostMatch",
      "href": "https://metadata.ucdn.example/hostmatch1234"
    }
  ]
}
```

4.1.2. HostMatch

The HostMatch object contains a Hostname or IP address to match against content requests. The HostMatch object also contains a HostMetadata object to apply if a match is found.

Property: host

Description: Hostname or IP address and optional port to match against the requested host, i.e., the [RFC3986] host and port. In order for a Hostname or IP address in a content request to match the Hostname or IP address in the host property the value from the content request when converted to lowercase MUST be identical to the value of the host property when converted to lowercase. All implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986]. IPv6 addresses MUST be encoded in one of the IPv6 address formats specified in [RFC5952] although receivers MUST support all IPv6 address formats specified in [RFC4291]. Hostnames MUST conform to the Domain Name System (DNS) syntax defined in [RFC1034] and [RFC1123]. Internationalized Domain Names (IDN) must first be transformed to the the A-label form [RFC5890] as per [RFC5891].

Type: Endpoint

Mandatory-to-Specify: Yes.

Property: host-metadata

Description: CDNI metadata to apply when delivering content that matches this host.

Type: HostMetadata

Mandatory-to-Specify: Yes.

Example HostMatch object with an embedded HostMetadata object:

```
{
  "host": "video.example.com",
  "host-metadata" : {
    <Properties of embedded HostMetadata object>
  }
}
```

Example HostMatch object referencing (via a Link object, see Section 4.3.1) a HostMetadata object:

```
{
  "host": "video.example.com",
  "host-metadata" : {
    "type": "MI.HostMetadata",
    "href": "https://metadata.ucdn.example/host1234"
  }
}
```

4.1.3. HostMetadata

A HostMetadata object contains the CDNI metadata properties for content served for a particular host (defined in the HostMatch object) and possibly child PathMatch objects.

Property: metadata

Description: Array of host related metadata.

Type: Array of GenericMetadata objects

Mandatory-to-Specify: Yes.

Property: paths

Description: Path specific rules. Path patterns (PathMatch objects) MUST be evaluated in the order they appear and the first (and only the first) PathMatch object that matches the content request being processed MUST be used.

Type: Array of PathMatch objects

Mandatory-to-Specify: No.

Example HostMetadata object containing a number of embedded GenericMetadata objects that will describe the default metadata for the host and an embedded PathMatch object that contains a path for which metadata exists that overrides the default metadata for the host:

```
{
  "metadata": [
    {
      <Properties of 1st embedded GenericMetadata object>
    },
    {
      <Properties of 2nd embedded GenericMetadata object>
    },
    ...
    {
      <Properties of Nth embedded GenericMetadata object>
    }
  ],
  "paths": [
    {
      <Properties of embedded PathMatch object>
    }
  ]
}
```

4.1.4. PathMatch

A PathMatch object contains PatternMatch object with a path to match against a resource's URI path, as well as how to handle URI query parameters. The PathMatch also contains a PathMetadata object with GenericMetadata to apply if the resource's URI matches the pattern within the PatternMatch object.

Property: path-pattern

Description: Pattern to match against the requested resource's URI.

Type: PatternMatch

Mandatory-to-Specify: Yes.

Property: path-metadata

Description: CDNI metadata to apply when delivering content that matches the associated PatternMatch.

Type: PathMetadata

Mandatory-to-Specify: Yes.

Example PathMatch object referencing the PathMetadata object to use for URIs that match the case-sensitive URI path pattern `"/movies/*"` (contained within an embedded PatternMatch object):

```
{
  "path-pattern": {
    "pattern": "/movies/*",
    "case-sensitive": true
  },
  "path-metadata": {
    "type": "MI.PathMetadata",
    "href": "https://metadata.ucdn.example/host1234/pathDCE"
  }
}
```

4.1.5. PatternMatch

A PatternMatch object contains the pattern string and flags that describe the pattern expression.

Property: pattern

Description: A pattern for matching against the URI path, i.e., against the [RFC3986] path-absolute. The pattern can contain the wildcards `*` and `?`, where `*` matches any sequence of [RFC3986] pchar or `"/` characters (including the empty string) and `?` matches exactly one [RFC3986] pchar character. The three literals `$`, `*` and `?` MUST be escaped as `$$`, `$*` and `$?` (where `$` is the designated escape character). All other characters are treated as literals.

Type: String

Mandatory-to-Specify: Yes.

Property: case-sensitive

Description: Flag indicating whether or not case-sensitive matching should be used. Note: Case-insensitivity applies to ALPHA characters in the URI path prior to percent-decoding [RFC3986].

Type: Boolean

Mandatory-to-Specify: No. Default is case-insensitive match.

Example PatternMatch object that matches the case-sensitive URI path pattern `"/movies/*"`. All query parameters will be ignored when

matching URIs requested from surrogates by content clients against this path pattern:

```
{
  "pattern": "/movies/*",
  "case-sensitive": true
}
```

Example PatternMatch object that matches the case-sensitive URI path pattern "/movies/*". Only the query parameter "sessionid" will be evaluated when matching URIs requested from surrogates by content clients against this path pattern:

```
{
  "pattern": "/movies/*",
  "case-sensitive": true
}
```

4.1.6. PathMetadata

A PathMetadata object contains the CDNI metadata properties for content requests that match against the associated URI path (defined in a PathMatch object).

Note that if DNS-based redirection is employed, then a dCDN will be unable to evaluate any metadata at the PathMetadata level or below because only the hostname of the content request is available at request routing time. dCDNs SHOULD still process all PathMetadata for the host before responding to the redirection request to detect if any unsupported metadata is specified. If any metadata not supported by the dCDN is marked as "mandatory-to-enforce", the dCDN SHOULD NOT accept the content redirection request, in order to avoid receiving content requests that it will not be able to satisfy/serve.

Property: metadata

Description: Array of path related metadata.

Type: Array of GenericMetadata objects

Mandatory-to-Specify: Yes.

Property: paths

Description: Path specific rules. Path patterns (PathMatch objects) MUST be evaluated in the order they appear and the first (and only the first) PathMatch object that matches the content request being processed MUST be used.

Type: Array of PathMatch objects

Mandatory-to-Specify: No.

Example PathMetadata object containing a number of embedded GenericMetadata objects that describe the metadata to apply for the URI path defined in the parent PathMatch object, as well as a more specific PathMatch object.

```
{
  "metadata": [
    {
      <Properties of 1st embedded GenericMetadata object>
    },
    {
      <Properties of 2nd embedded GenericMetadata object>
    },
    ...
    {
      <Properties of Nth embedded GenericMetadata object>
    }
  ],
  "paths": [
    {
      <Properties of embedded PathMatch object>
    }
  ]
}
```

4.1.7. GenericMetadata

A GenericMetadata object is a wrapper for managing individual CDNI metadata properties in an opaque manner.

Property: generic-metadata-type

Description: Case-insensitive CDNI metadata object type.

Type: String containing the CDNI Payload Type [RFC7736] of the object contained in the generic-metadata-value property (see Table 4).

Mandatory-to-Specify: Yes.

Property: generic-metadata-value

Description: CDNI metadata object.

Type: Format/Type is defined by the value of generic-metadata-type property above. Note: generic-metadata-values MUST NOT name any properties "href" (see Section 4.3.1).

Mandatory-to-Specify: Yes.

Property: mandatory-to-enforce

Description: Flag identifying whether or not the enforcement of the property metadata is required.

Type: Boolean

Mandatory-to-Specify: No. Default is to treat metadata as mandatory to enforce (i.e., a value of True).

Property: safe-to-redistribute

Description: Flag identifying whether or not the property metadata can be safely redistributed without modification.

Type: Boolean

Mandatory-to-Specify: No. Default is allow transparent redistribution (i.e., a value of True).

Property: incomprehensible

Description: Flag identifying whether or not any CDN in the chain of delegation has failed to understand and/or failed to properly transform this metadata object. Note: This flag only applies to metadata objects whose safe-to-redistribute property has a value of False.

Type: Boolean

Mandatory-to-Specify: No. Default is comprehensible (i.e., a value of False).

Example GenericMetadata object containing a metadata object that applies to the applicable URI path and/or host (within a parent PathMetadata and/or HostMetadata object, respectively):


```
{
  "mandatory-to-enforce": true,
  "safe-to-redistribute": true,
  "incomprehensible": false,
  "generic-metadata-type": <CDNI Payload Type of this metadata object>,
  "generic-metadata-value":
    {
      <Properties of this metadata object>
    }
}
```

4.2. Definitions of the initial set of CDNI Generic Metadata objects

The objects defined below are intended to be used in the GenericMetadata object generic-metadata-value field as defined in Section 4.1.7 and their generic-metadata-type property MUST be set to the appropriate CDNI Payload Type as defined in Table 4.

4.2.1. SourceMetadata

Source metadata provides the dCDN with information about content acquisition, i.e., how to contact an uCDN Surrogate or an Origin Server to obtain the content to be served. The sources are not necessarily the actual Origin Servers operated by the CSP but might be a set of Surrogates in the uCDN.

Property: sources

Description: Sources from which the dCDN can acquire content, listed in order of preference.

Type: Array of Source objects (see Section 4.2.1.1)

Mandatory-to-Specify: No. Default is to use static configuration, out-of-band from the metadata interface.

Example SourceMetadata object (which contains two Source objects) that describes which servers the dCDN should use for acquiring content for the applicable URI path and/or host:

```
{
  "generic-metadata-type": "MI.SourceMetadata",
  "generic-metadata-value":
    {
      "sources": [
        {
          "endpoints": [
            "a.service123.ucdn.example",
            "b.service123.ucdn.example"
          ],
          "protocol": "http/1.1"
        },
        {
          "endpoints": ["origin.service123.example"],
          "protocol": "http/1.1"
        }
      ]
    }
}
```

4.2.1.1. Source

A Source object describes the source to be used by the dCDN for content acquisition (e.g., a Surrogate within the uCDN or an alternate Origin Server), the protocol to be used, and any authentication method to be used when contacting that source.

Endpoints within a Source object MUST be treated as equivalent/equal. A uCDN can specify an array of sources in preference order within a SourceMetadata object, and then for each preference ranked Source object, a uCDN can specify an array of endpoints that are equivalent (e.g., a pool of servers that are not behind a load balancer).

Property: acquisition-auth

Description: Authentication method to use when requesting content from this source.

Type: Auth (see Section 4.2.7)

Mandatory-to-Specify: No. Default is no authentication required.

Property: endpoints

Description: Origins from which the dCDN can acquire content. If multiple endpoints are specified they are all equal, i.e.,

the list is not in preference order (e.g., a pool of servers behind a load balancer).

Type: Array of Endpoint objects (See Section 4.3.3)

Mandatory-to-Specify: Yes.

Property: protocol

Description: Network retrieval protocol to use when requesting content from this source.

Type: Protocol (see Section 4.3.2)

Mandatory-to-Specify: Yes.

Example Source object that describes a pair of endpoints (servers) the dCDN can use for acquiring content for the applicable host and/or URI path:

```
{
  "endpoints": [
    "a.servicel23.ucdn.example",
    "b.servicel23.ucdn.example"
  ],
  "protocol": "http/1.1"
}
```

4.2.2.2. LocationACL Metadata

LocationACL metadata defines which locations a User Agent needs to be in, in order to be able to receive the associated content.

A LocationACL which does not include a locations property results in an action of allow all, meaning that delivery can be performed regardless of the User Agent's location, otherwise a CDN MUST take the action from the first footprint to match against the User Agent's location. If two or more footprints overlap, the first footprint that matches against the User Agent's location determines the action a CDN MUST take. If the locations property is included but is empty, or if none of the listed footprints matches the User Agent's location, then the result is an action of deny.

Although the LocationACL, TimeWindowACL (see Section 4.2.3), and ProtocolACL (see Section 4.2.4) are independent GenericMetadata objects, they can provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the LocationACL and denied based on the TimeWindowACL. The dCDN MUST use

the logical AND of all ACLs (where 'allow' is true and 'deny' is false) to determine whether or not a request should be allowed.

Property: locations

Description: Access control list which allows or denies (blocks) delivery based on the User Agent's location.

Type: Array of LocationRule objects (see Section 4.2.2.1)

Mandatory-to-Specify: No. Default is allow all locations.

Example LocationACL object that allows the dCDN to deliver content to any location/IP address:

```
{
  "generic-metadata-type": "MI.LocationACL",
  "generic-metadata-value":
    {
    }
}
```

Example LocationACL object (which contains a LocationRule object which itself contains a Footprint object) that only allows the dCDN to deliver content to User Agents in the USA:

```
{
  "generic-metadata-type": "MI.LocationACL",
  "generic-metadata-value":
    {
      "locations": [
        {
          "action": "allow",
          "footprints": [
            {
              "footprint-type": "countrycode",
              "footprint-value": ["us"]
            }
          ]
        }
      ]
    }
}
```

4.2.2.1. LocationRule

A LocationRule contains or references an array of Footprint objects and the corresponding action.

Property: footprints

Description: Array of footprints to which the rule applies.

Type: Array of Footprint objects (see Section 4.2.2.2)

Mandatory-to-Specify: Yes.

Property: action

Description: Defines whether the rule specifies locations to allow or deny.

Type: Enumeration [allow|deny] encoded as a lowercase string

Mandatory-to-Specify: No. Default is deny.

Example LocationRule object (which contains a Footprint object) that allows the dCDN to deliver content to clients in the USA:

```
{
  "action": "allow",
  "footprints": [
    {
      "footprint-type": "countrycode",
      "footprint-value": ["us"]
    }
  ]
}
```

4.2.2.2. Footprint

A Footprint object describes the footprint to which a LocationRule can be applied to, e.g., an IPv4 address range or a geographic location.

Property: footprint-type

Description: Registered footprint type (see Section 7.2). The footprint types specified by this document are: "ipv4cidr" (IPv4CIDR, see Section 4.3.5), "ipv6cidr" (IPv6CIDR, see Section 4.3.6), "asn" (Autonomous System Number, see

Section 4.3.7) and "countrycode" (Country Code, see Section 4.3.8).

Type: Lowercase String

Mandatory-to-Specify: Yes.

Property: footprint-value

Description: Array of footprint values conforming to the specification associated with the registered footprint type. Footprint values can be simple strings (e.g., IPv4CIDR, IPv6CIDR, ASN, and CountryCode), however, other Footprint objects can be defined in the future, along with a more complex encoding (e.g., GPS coordinate tuples).

Type: Array of footprints

Mandatory-to-Specify: Yes.

Example Footprint object describing a footprint covering the USA:

```
{
  "footprint-type": "countrycode",
  "footprint-value": ["us"]
}
```

Example Footprint object describing a footprint covering the IP address ranges 192.0.2.0/24 and 198.51.100.0/24:

```
{
  "footprint-type": "ipv4cidr",
  "footprint-value": ["192.0.2.0/24", "198.51.100.0/24"]
}
```

Example Footprint object describing a footprint covering the IP address ranges 2001:db8::/32:

```
{
  "footprint-type": "ipv6cidr",
  "footprint-value": ["2001:db8::/32"]
}
```

Example Footprint object describing a footprint covering the autonomous system 64496:

```
{
  "footprint-type": "asn",
  "footprint-value": ["as64496"]
}
```

4.2.3. TimeWindowACL

TimeWindowACL metadata defines time-based restrictions.

A TimeWindowACL which does not include a times property results in an action of allow all, meaning that delivery can be performed regardless of the time of the User Agent's request, otherwise a CDN MUST take the action from the first window to match against the current time. If two or more windows overlap, the first window that matches against the current time determines the action a CDN MUST take. If the times property is included but is empty, or if none of the listed windows matches the current time, then the result is an action of deny.

Although the LocationACL (see Section 4.2.2), TimeWindowACL, and ProtocolACL (see Section 4.2.4) are independent GenericMetadata objects, they can provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the LocationACL and denied based on the TimeWindowACL. The dCDN MUST use the logical AND of all ACLs (where 'allow' is true and 'deny' is false) to determine whether or not a request should be allowed.

Property: times

Description: Access control list which allows or denies (blocks) delivery based on the time of a User Agent's request.

Type: Array of TimeWindowRule objects (see Section 4.2.3.1)

Mandatory-to-Specify: No. Default is allow all time windows.

Example TimeWindowACL object (which contains a TimeWindowRule object which itself contains a TimeWindow object) that only allows the dCDN to deliver content to clients between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC:

```
{
  "generic-metadata-type": "MI.TimeWindowACL",
  "generic-metadata-value":
    {
      "times": [
        {
          "action": "allow",
          "windows": [
            {
              "start": 946717200,
              "end": 946746000
            }
          ]
        }
      ]
    }
}
```

4.2.3.1. TimeWindowRule

A TimeWindowRule contains or references an array of TimeWindow objects and the corresponding action.

Property: windows

Description: Array of time windows to which the rule applies.

Type: Array of TimeWindow objects (see Section 4.2.3.2)

Mandatory-to-Specify: Yes.

Property: action

Description: Defines whether the rule specifies time windows to allow or deny.

Type: Enumeration [allow|deny] encoded as a lowercase string

Mandatory-to-Specify: No. Default is deny.

Example TimeWindowRule object (which contains a TimeWindow object) that only allows the dCDN to deliver content to clients between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC:


```
{
  "action": "allow",
  "windows": [
    {
      "start": 946717200,
      "end": 946746000
    }
  ]
}
```

4.2.3.2. TimeWindow

A TimeWindow object describes a time range which can be applied by an TimeWindowACL, e.g., start 946717200 (i.e., 09:00 01/01/2000 UTC), end: 946746000 (i.e., 17:00 01/01/2000 UTC).

Property: start

Description: The start time of the window.

Type: Time (see Section 4.3.4)

Mandatory-to-Specify: Yes.

Property: end

Description: The end time of the window.

Type: Time (see Section 4.3.4)

Mandatory-to-Specify: Yes.

Example TimeWindow object that describes a time window from 09:00 01/01/2000 UTC to 17:00 01/01/2000 UTC:

```
{
  "start": 946717200,
  "end": 946746000
}
```

4.2.4. ProtocolACL Metadata

ProtocolACL metadata defines delivery protocol restrictions.

A ProtocolACL which does not include a protocol-acl property results in an action of allow all, meaning that delivery can be performed regardless of the protocol in the User Agent's request, otherwise a CDN MUST take the action from the first protocol to match against the

request protocol. If two or more request protocols overlap, the first protocol that matches the request protocol determines the action a CDN MUST take. If the protocol-acl property is included but is empty, or if none of the listed protocol matches the request protocol, then the result is an action of deny.

Although the LocationACL, TimeWindowACL, and ProtocolACL are independent GenericMetadata objects, they can provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the ProtocolACL and denied based on the TimeWindowACL. The dCDN MUST use the logical AND of all ACLs (where 'allow' is true and 'deny' is false) to determine whether or not a request should be allowed.

Property: protocol-acl

Description: Description: Access control list which allows or denies (blocks) delivery based on delivery protocol.

Type: Array of ProtocolRule objects (see Section 4.2.4.1)

Mandatory-to-Specify: No. Default is allow all protocols.

Example ProtocolACL object (which contains a ProtocolRule object) that only allows the dCDN to deliver content using HTTP/1.1:

```
{
  "generic-metadata-type": "MI.ProtocolACL",
  "generic-metadata-value":
    {
      "protocol-acl": [
        {
          "action": "allow",
          "protocols": ["http/1.1"]
        }
      ]
    }
}
```

4.2.4.1. ProtocolRule

A ProtocolRule contains or references an array of Protocol objects and the corresponding action.

Property: protocols

Description: Array of protocols to which the rule applies.

Type: Array of Protocols (see Section 4.3.2)

Mandatory-to-Specify: Yes.

Property: action

Description: Defines whether the rule specifies protocols to allow or deny.

Type: Enumeration [allow|deny] encoded as a lowercase string

Mandatory-to-Specify: No. Default is deny.

Example ProtocolRule object (which contains a ProtocolRule object) that allows the dCDN to deliver content using HTTP/1.1:

```
{
  "action": "allow",
  "protocols": ["http/1.1"]
}
```

4.2.5. DeliveryAuthorization Metadata

Delivery Authorization defines authorization methods for the delivery of content to User Agents.

Property: delivery-auth-methods

Description: Options for authorizing content requests. Delivery for a content request is authorized if any of the authorization methods in the list is satisfied for that request.

Type: Array of Auth objects (see Section 4.2.7)

Mandatory-to-Specify: No. Default is no authorization required.

Example DeliveryAuthorization object (which contains an Auth object):

```
{
  "generic-metadata-type": "MI.DeliveryAuthorization",
  "generic-metadata-value":
    {
      "delivery-auth-methods": [
        {
          "auth-type": <CDNI Payload Type of this Auth object>,
          "auth-value":
            {
              <Properties of this Auth object>
            }
        }
      ]
    }
}
```

4.2.6. Cache

A Cache object describes the cache control parameters to be applied to the content by intermediate caches.

Cache keys are generated from the URI of the content request [RFC7234]. In some cases, a CDN or content provider might want certain path segments or query parameters to be excluded from the cache key generation. The Cache object provides guidance on what parts of the path and query string to include.

Property: exclude-path-pattern

Description: A pattern for matching against the URI path, i.e., against the [RFC3986] path-absolute. The pattern can contain the wildcards * and ?, where * matches any sequence of [RFC3986] pchar or "/" characters (including the empty string) and ? matches exactly one [RFC3986] pchar character. The three literals \$, * and ? MUST be escaped as \$\$, \$* and \$? (where \$ is the designated escape character). All other characters are treated as literals. Cache key generation MUST only include the portion of the path-absolute that matches the wildcard portions of the pattern. Note: Inconsistency between the PatternMatch pattern Section 4.1.5 and the exclude-path-pattern can result in inefficient caching.

Type: String

Mandatory-to-Specify: No. Default is to use the full URI path-absolute to generate the cache key.

Property: include-query-strings

Description: Allows a Surrogate to specify the URI query string parameters [RFC3986] to include when comparing the requested URI against the URIs in its cache for equivalence. Matching query parameters MUST be case-insensitive. If all query parameters should be ignored, then the list MUST be specified and MUST be empty. If a query parameter appears multiple times in the query string, each instance value MUST be aggregated prior to comparison. For consistent cache key generation, query parameters SHOULD be evaluated in the order specified in this array.

Type: Array of String

Mandatory-to-Specify: No. Default is to consider all query string parameters when comparing URIs.

Example Cache object that instructs the dCDN to use the full URI path and ignore all query parameters:

```
{
  "generic-metadata-type": "MI.Cache",
  "generic-metadata-value":
  {
    "include-query-strings": []
  }
}
```

Example Cache object that instructs the dCDN to exclude the "CDNX" path prefix and only include the (case-insensitive) query parameters named "mediaid" and "providerid":

```
{
  "generic-metadata-type": "MI.Cache",
  "generic-metadata-value":
  {
    "exclude-path-pattern": "/CDNX/*",
    "include-query-strings": ["mediaid", "providerid"]
  }
}
```

Example Cache object that instructs the dCDN to exclude the "CDNX" path prefix, but includes all query parameters:

```
{
  "generic-metadata-type": "MI.Cache",
  "generic-metadata-value":
  {
    "exclude-path-pattern": "/CDNX/*"
  }
}
```

4.2.7. Auth

An Auth object defines authentication and authorization methods to be used during content acquisition and content delivery, respectively.

Note: This document does not define any Auth methods. Individual Auth methods are being defined separately (e.g., URI Signing [I-D.ietf-cdni-uri-signing]). The GenericMetadata which contain Auth objects is defined herein for convenience and so as not to be specific to any particular Auth method.

Property: auth-type

Description: Auth type (The CDNI Payload Type [RFC7736] of the GenericMetadata object contained in the auth-value property).

Type: String

Mandatory-to-Specify: Yes.

Property: auth-value

Description: An object conforming to the specification associated with the Auth type.

Type: GenericMetadata Object

Mandatory-to-Specify: Yes.

Example Auth object:

```
{
  "generic-metadata-type": "MI.Auth",
  "generic-metadata-value":
  {
    "auth-type": <CDNI Payload Type of this Auth object>,
    "auth-value":
    {
      <Properties of this Auth object>
    }
  }
}
```

4.2.8. Grouping

A Grouping object identifies a group of content to which a given asset belongs.

Property: ccid

Description: Content Collection identifier for an application-specific purpose such as logging aggregation.

Type: String

Mandatory-to-Specify: No. Default is an empty string.

Example Grouping object that specifies a Content Collection Identifier for the content associated with the Grouping object's parent HostMetadata and PathMetadata:

```
{
  "generic-metadata-type": "MI.Grouping",
  "generic-metadata-value":
  {
    "ccid": "ABCD"
  }
}
```

4.3. CDNI Metadata Simple Data Type Descriptions

This section describes the simple data types that are used for properties of CDNI metadata objects.

4.3.1. Link

A Link object can be used in place of any of the objects or properties described above. Link objects can be used to avoid duplication if the same metadata information is repeated within the

metadata tree. When a Link object replaces another object, its href property is set to the URI of the resource and its type property is set to the CDNI Payload Type of the object it is replacing.

dCDNs can detect the presence of a Link object by detecting the presence of a property named "href" within the object. This means that GenericMetadata types MUST NOT contain a property named "href" because doing so would conflict with the ability for dCDNs to detect Link objects being used to reference a GenericMetadata object.

Property: href

Description: The URI of the addressable object being referenced.

Type: String

Mandatory-to-Specify: Yes.

Property: type

Description: The CDNI Payload type of the object being referenced.

Type: String

Mandatory-to-Specify: No. If the container specifies the type (e.g., the HostIndex object contains an array of HostMatch objects, so a Link object in the list of HostMatch objects must reference a HostMatch), then it is not necessary to explicitly specify a type.

Example Link object referencing a HostMatch object:

```
{
  "type": "MI.HostMatch",
  "href": "https://metadata.ucdn.example/hostmatch1234"
}
```

Example Link object referencing a HostMatch object, without an explicit type, inside a HostIndex object:


```
{
  "hosts": [
    {
      <Properties of embedded HostMatch object>
    },
    {
      "href": "https://metadata.ucdn.example/hostmatch1234"
    }
  ]
}
```

4.3.1.1. Link Loop Prevention

When following a Link, CDNI metadata clients SHOULD verify that the CDNI Payload Type of the object retrieved matches the expected CDNI Payload Type, as indicated by the link object. For GenericMetadata objects, type checks will prevent self references; however, incorrect linking can result in circular references for structural metadata objects, specifically, PathMatch and PathMetadata objects Figure 1. To prevent the circular references, CDNI metadata clients SHOULD verify that no duplicate Links occur for PathMatch or PathMetadata objects.

4.3.2. Protocol

Protocol objects are used to specify registered protocols for content acquisition or delivery (see Section 7.3).

Type: String

Example:

"http/1.1"

4.3.3. Endpoint

A Hostname (with optional port) or an IP address (with optional port).

All implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986]. IPv6 addresses MUST be encoded in one of the IPv6 address formats specified in [RFC5952] although receivers MUST support all IPv6 address formats specified in [RFC4291]. Hostnames MUST conform to the Domain Name System (DNS) syntax defined in [RFC1034] and [RFC1123]. Internationalized Domain Names (IDN) must first be transformed to the A-label form [RFC5890] as per [RFC5891].

Type: String

Example Hostname:

"metadata.ucdn.example"

Example IPv4 address:

"192.0.2.1"

Example IPv6 address (with port number):

"[2001:db8::1]:81"

4.3.4. Time

A time value expressed in seconds since the Unix epoch (i.e., zero hours, zero minutes, zero seconds, on January 1, 1970) Coordinated Universal Time (UTC) [POSIX].

Type: Integer

Example Time representing 09:00:00 01/01/2000 UTC:

946717200

4.3.5. IPv4CIDR

An IPv4address CIDR block encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986] followed by a / followed by an unsigned integer representing the leading bits of the routing prefix (i.e., IPv4 CIDR notation). Single IP addresses can be expressed as /32.

Type: String

Example IPv4 CIDR:

"192.0.2.0/24"

4.3.6. IPv6CIDR

An IPv6address CIDR block encoded in one of the IPv6 address formats specified in [RFC5952] followed by a / followed by an unsigned integer representing the leading bits of the routing prefix (i.e., IPv6 CIDR notation). Single IP addresses can be expressed as /128.

Type: String

Example IPv6 CIDR:

"2001:db8::/32"

4.3.7. ASN

An Autonomous System Number encoded as a string consisting of the characters "as" (in lowercase) followed by the Autonomous System number [RFC6793].

Type: String

Example ASN:

"as64496"

4.3.8. CountryCode

An ISO 3166-1 alpha-2 code [ISO3166-1] in lowercase.

Type: String

Example Country Code representing the USA:

"us"

5. CDNI Metadata Capabilities

CDNI metadata is used to convey information pertaining to content delivery from uCDN to dCDN. For optional metadata, it can be useful for the uCDN to know if the dCDN supports the underlying functionality described by the metadata, prior to delegating any content requests to the dCDN. If some metadata is "mandatory-to-enforce", and the dCDN does not support it, any delegated requests for content that requires that metadata will fail. The uCDN will likely want to avoid delegating those requests to that dCDN. Likewise, for any metadata which might be assigned optional values, it could be useful for the uCDN to know which values a dCDN supports, prior to delegating any content requests to that dCDN. If the optional value assigned to a given piece of content's metadata is not supported by the dCDN, any delegated requests for that content can fail, so again the uCDN is likely to want to avoid delegating those requests to that dCDN.

The CDNI Footprint and Capabilities Interface (FCI) provides a means of advertising capabilities from dCDN to uCDN [RFC7336]. Support for optional metadata types and values can be advertised using the FCI.

6. CDNI Metadata interface

This section specifies an interface to enable a dCDN to retrieve CDNI metadata objects from a uCDN.

The interface can be used by a dCDN to retrieve CDNI metadata objects either:

- o Dynamically as required by the dCDN to process received requests. For example in response to a query from an uCDN over the CDNI Request Routing Redirection interface (RI) [I-D.ietf-cdni-redirection] or in response to receiving a request for content from a User Agent. Or;
- o In advance of being required. For example in the case of pre-positioned CDNI metadata acquisition, initiated through the "CDNI Control interface / Triggers" (CI/T) interface [I-D.ietf-cdni-control-triggers].

The CDNI metadata interface is built on the principles of HTTP web services. In particular, this means that requests and responses over the interface are built around the transfer of representations of hyperlinked resources. A resource in the context of the CDNI metadata interface is any object in the object model (as described in Section 3 and Section 4).

To retrieve CDNI metadata, a CDNI metadata client (i.e., a client in the dCDN) first makes a HTTP GET request for the URI of the HostIndex which provides the CDNI metadata client with an array of Hostnames for which the uCDN can delegate content delivery to the dCDN. The CDNI metadata client can then obtain any other CDNI metadata objects by making a HTTP GET requests for any linked metadata objects it requires.

CDNI metadata servers (i.e., servers in the uCDN) are free to assign whatever structure they desire to the URIs for CDNI metadata objects and CDNI metadata clients MUST NOT make any assumptions regarding the structure of CDNI metadata URIs or the mapping between CDNI metadata objects and their associated URIs. Therefore any URIs present in the examples in this document are purely illustrative and are not intended to impose a definitive structure on CDNI metadata interface implementations.

6.1. Transport

The CDNI metadata interface uses HTTP as the underlying protocol transport [RFC7230].

The HTTP Method in the request defines the operation the request would like to perform. A server implementation of the CDNI metadata interface MUST support the HTTP GET and HEAD methods.

The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses that contain a response body SHOULD include an ETag to enable validation of cached versions of returned resources.

As the CDNI metadata interface builds on top of HTTP, CDNI metadata server implementations MAY make use of any HTTP feature when implementing the CDNI metadata interface, for example, a CDNI metadata server MAY make use of HTTP's caching mechanisms to indicate that the returned response/representation can be reused without re-contacting the CDNI metadata server.

6.2. Retrieval of CDNI Metadata resources

In the general case, a CDNI metadata server makes CDNI metadata objects available via a unique URIs and thus, in order to retrieve CDNI metadata, a CDNI metadata client first makes a HTTP GET request for the URI of the HostIndex which provides an array of Hostnames for which the uCDN can delegate content delivery to the dCDN.

In order to retrieve the CDNI metadata for a particular request the CDNI metadata client processes the received HostIndex object and finds the corresponding HostMetadata entry (by matching the hostname in the request against the hostnames listed in the HostMatch objects). If the HostMetadata is linked (rather than embedded), the CDNI metadata client then makes a GET request for the URI specified in the href property of the Link object which points to the HostMetadata object itself.

In order to retrieve the most specific metadata for a particular request, the CDNI metadata client inspects the HostMetadata for references to more specific PathMetadata objects (by matching the URI path in the request against the path-patterns in any PathMatch objects listed in the HostMetadata object). If any PathMetadata are found to match (and are linked rather than embedded), the CDNI metadata client makes another GET request for the PathMetadata. Each PathMetadata object can also include references to yet more specific metadata. If this is the case, the CDNI metadata client continues requesting PathMatch and PathMetadata objects recursively. The CDNI metadata client repeats this approach of processing metadata objects and retrieving (via HTTP GETs) any linked objects until it has all the metadata objects it requires in order to process the redirection request from an uCDN or the content request from a User Agent.

In cases where a dCDN is not able to retrieve the entire set of CDNI metadata associated with a User Agent request, or it has retrieved that metadata but it is stale according to standard HTTP caching rules and cannot be revalidated, for example because the uCDN is unreachable or returns a HTTP 4xx or 5xx status in response to some or all of the dCDN's CDNI metadata requests, the dCDN MUST NOT serve the requested content.

Where a dCDN is interconnected with multiple uCDNs, the dCDN needs to determine which uCDN's CDNI metadata should be used to handle a particular User Agent request.

When application level redirection (e.g., HTTP 302 redirects) is being used between CDNs, it is expected that the dCDN will be able to determine the uCDN that redirected a particular request from information contained in the received request (e.g., via the URI). With knowledge of which uCDN routed the request, the dCDN can choose the correct uCDN from which to obtain the HostIndex. Note that the HostIndexes served by each uCDN can be unique.

In the case of DNS redirection there is not always sufficient information carried in the DNS request from User Agents to determine the uCDN that redirected a particular request (e.g., when content from a given host is redirected to a given dCDN by more than one uCDN) and therefore dCDNs will have to apply local policy when deciding which uCDN's metadata to apply.

6.3. Bootstrapping

The URI for the HostIndex object of a given uCDN needs to be configured in the dCDN. All other objects/resources are then discoverable from the HostIndex object by following any links in the HostIndex object and through the referenced HostMetadata and PathMetadata objects and their GenericMetadata sub-objects.

Manual configuration of the URI for the HostIndex object is outside the scope of this document.

6.4. Encoding

CDNI metadata objects MUST be encoded as I-JSON objects [RFC7493] containing a dictionary of (key,value) pairs where the keys are the property names and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e., dependent on the CDNI Payload Type of the returned resource). Likewise, the values associated with each

property (dictionary key) are dependent on the specific object being encoded (i.e., dependent on the CDNI Payload Type of the returned resource).

Dictionary keys (properties) in I-JSON are case sensitive. By convention, any dictionary key (property) defined by this document (for example, the names of CDNI metadata object properties) MUST be lowercase.

6.5. Extensibility

The set of GenericMetadata objects can be extended with additional (standards based or vendor specific) metadata objects through the specification of new GenericMetadata objects. The GenericMetadata object defined in Section 4.1.7 specifies a type field and a type-specific value field that allows any metadata to be included in either the HostMetadata or PathMetadata arrays.

As with the initial GenericMetadata types defined in Section 4.2, future GenericMetadata types MUST specify the information necessary for constructing and decoding the GenericMetadata object.

Any document which defines a new GenericMetadata type MUST:

1. Specify and register the CDNI Payload Type [RFC7736] used to identify the new GenericMetadata type being specified.
2. Define the set of properties associated with the new GenericMetadata object. GenericMetadata MUST NOT contain a property named "href" because doing so would conflict with the ability to detect Link objects (see Section 4.3.1).
3. Define a name, description, type, and whether or not the property is mandatory-to-specify.
4. Describe the semantics of the new type including its purpose and example of a use case to which it applies including an example encoded in I-JSON.
5. Describe the security and privacy consequences, for both the user-agent and the CDN, of the new GenericMetadata object.
6. Describe any relation to, conflict with, or obsolescence of other existing CDNI metadata objects.

Note: In the case of vendor specific extensions, vendor-identifying CDNI Payload Type names will decrease the possibility of GenericMetadata type collisions.

6.6. Metadata Enforcement

At any given time, the set of GenericMetadata types supported by the uCDN might not match the set of GenericMetadata types supported by the dCDN.

In cases where a uCDN sends metadata containing a GenericMetadata type that a dCDN does not support, the dCDN MUST enforce the semantics of the "mandatory-to-enforce" property. If a dCDN does not understand or is unable to perform the functions associated with any "mandatory-to-enforce" metadata, the dCDN MUST NOT service any requests for the corresponding content.

Note: Ideally, uCDNs would not delegate content requests to a dCDN that does not support the "mandatory-to-enforce" metadata associated with the content being requested. However, even if the uCDN has a priori knowledge of the metadata supported by the dCDN (e.g., via the FCI or through out-of-band negotiation between CDN operators), metadata support can fluctuate or be inconsistent (e.g., due to miscommunication, mis-configuration, or temporary outage). Thus, the dCDN MUST always evaluate all metadata associated with redirection and content requests and reject any requests where "mandatory-to-enforce" metadata associated with the content cannot be enforced.

6.7. Metadata Conflicts

It is possible that new metadata definitions will obsolete or conflict with existing GenericMetadata (e.g., a future revision of the CDNI metadata interface could redefine the Auth GenericMetadata object or a custom vendor extension could implement an alternate Auth metadata option). If multiple metadata (e.g., MI.Auth.v2, vendor1.Auth, and vendor2.Auth) all conflict with an existing GenericMetadata object (i.e., MI.Auth) and all are marked as "mandatory-to-enforce", it could be ambiguous which metadata should be applied, especially if the functionality of the metadata overlap.

As described in Section 3.3, metadata override only applies to metadata objects of the same exact type found in HostMetadata and nested PathMetadata structures. The CDNI metadata interface does not support enforcement of dependencies between different metadata types. It is the responsibility of the CSP and the CDN operators to ensure that metadata assigned to a given piece of content do not conflict.

Note: Because metadata is inherently ordered in HostMetadata and PathMetadata arrays, as well as in the PathMatch hierarchy, multiple conflicting metadata types MAY be used, however, metadata hierarchies SHOULD ensure that independent PathMatch root objects are used to prevent ambiguous or conflicting metadata definitions.

6.8. Versioning

The version of CDNI metadata objects is conveyed inside the CDNI Payload Type that is included in either the HTTP Content-Type header, for example: "Content-Type: application/cdni; ptype=MI.HostIndex", when retrieved via a link, or in the link type (Section 4.3.1), generic-metadata-type (Section 4.1.7), or auth-type (Section 4.2.7) properties in the JSON payload. The CDNI Payload Type uniquely identifies the specification defining that object, including any relation to, conflicts with, or obsolescence of other metadata. There is no explicit version mapping requirement, however, for ease of understanding, metadata creators SHOULD make new versions of metadata easily visible via the CDNI Payload Type, e.g., by appending a version string. Note: A version string is optional on the first version, e.g., MI.HostIndex, but could be added for subsequent versions, e.g., MI.HostIndex.v2, MI.HostIndex.v3, etc.

Except when referenced by a Link object, nested metadata objects (i.e., structural metadata below the HostIndex; Source objects; Location, TimeWindow, and Protocol Rule objects; and Footprint and TimeWindow objects) can be serialized into a JSON payload without explicit CDNI Payload Type information. The type is inferred from the outer structural metadata, generic metadata, or auth object CDNI Payload Type. To avoid ambiguity when revising nestable metadata objects, any outer metadata object(s) MUST be reversioned and allocated new CDNI Payload Type(s) at the same time. For example, the MI.HostIndex object defined in this document contains an array of MI.HostMatch objects, which each in turn contains a MI.HostMetadata object. If a new MI.HostMetadata.v2 object were required, the outer MI.HostIndex and MI.HostMatch objects would need to be revised, e.g., to MI.HostIndex.v2 and MI.HostMatch.v2, respectively. Similarly, if a new MI.TimeWindowRule.v2 object was required, the outer MI.TimeWindowACL object would need to be revised, e.g., to MI.TimeWindowACL.v2; the MI.TimeWindowRule.v2 object, though, could still contain MI.TimeWindow objects, if so specified.

HTTP requests sent to a metadata server SHOULD include an Accept header with the CDNI Payload Type of the expected object. Metadata clients can specify multiple CDNI Payload Types in the Accept header, for example, if a metadata client is capable of processing two different versions of the same type of object (defined by different CDNI Payload Types) it might decide to include both in the Accept header.

6.9. Media Types

All CDNI metadata objects use the Media Type "application/cdni". The CDNI Payload Type for each object then contains the object name of that object as defined by this document, prefixed with "MI.". Table 4 lists the CDNI Payload Type for the metadata objects (resources) specified in this document.

Data Object	CDNI Payload Type
HostIndex	MI.HostIndex
HostMatch	MI.HostMatch
HostMetadata	MI.HostMetadata
PathMatch	MI.PathMatch
PatternMatch	MI.PatternMatch
PathMetadata	MI.PathMetadata
SourceMetadata	MI.SourceMetadata
Source	MI.Source
LocationACL	MI.LocationACL
LocationRule	MI.LocationRule
Footprint	MI.Footprint
TimeWindowACL	MI.TimeWindowACL
TimeWindowRule	MI.TimeWindowRule
TimeWindow	MI.TimeWindow
ProtocolACL	MI.ProtocolACL
ProtocolRule	MI.ProtocolRule
DeliveryAuthorization	MI.DeliveryAuthorization
Cache	MI.Cache
Auth	MI.Auth
Grouping	MI.Grouping

Table 4: CDNI Payload Types for CDNI Metadata objects

6.10. Complete CDNI Metadata Example

A dCDN requests the HostIndex and receive the following object with a CDNI payload type of "MI.HostIndex":

```
{
  "hosts": [
    {
      "host": "video.example.com",
      "host-metadata": {
        "type": "MI.HostMetadata",
        "href": "https://metadata.ucdn.example/host1234"
      }
    },
    {
      "host": "images.example.com",
      "host-metadata": {
        "type": "MI.HostMetadata",
        "href": "https://metadata.ucdn.example/host5678"
      }
    }
  ]
}
```

If the incoming request has a Host header with "video.example.com" then the dCDN would fetch the HostMetadata object from "https://metadata.ucdn.example/host1234" expecting a CDNI payload type of "MI.HostMetadata":

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.SourceMetadata",
      "generic-metadata-value": {
        "sources": [
          {
            "endpoint": ["acq1.ucdn.example"],
            "protocol": "http/1.1"
          },
          {
            "endpoint": ["acq2.ucdn.example"],
            "protocol": "http/1.1"
          }
        ]
      }
    },
    {
      "generic-metadata-type": "MI.LocationACL",
      "generic-metadata-value": {
        "locations": [
          {
            "footprints": [
              {

```

```

        "footprint-type": "ipv4cidr",
        "footprint-value": ["192.0.2.0/24"]
    },
    {
        "footprint-type": "ipv6cidr",
        "footprint-value": ["2001:db8::/32"]
    },
    {
        "footprint-type": "countrycode",
        "footprint-value": ["us"]
    },
    {
        "footprint-type": "asn",
        "footprint-value": ["as64496"]
    }
],
"action": "deny"
}
]
}
},
{
    "generic-metadata-type": "MI.ProtocolACL",
    "generic-metadata-value": {
        "protocol-acl": [
            {
                "protocols": [
                    "http/1.1"
                ],
                "action": "allow"
            }
        ]
    }
}
],
"paths": [
    {
        "path-pattern": {
            "pattern": "/video/trailers/*"
        },
        "path-metadata": {
            "type": "MI.PathMetadata",
            "href": "https://metadata.ucdn.example/host1234/pathABC"
        }
    },
    {
        "path-pattern": {
            "pattern": "/video/movies/*"
        }
    }
]

```

```

    },
    "path-metadata": {
      "type": "MI.PathMetadata",
      "href": "https://metadata.ucdn.example/host1234/pathDEF"
    }
  ]
}

```

Suppose the path of the requested resource matches the `"/video/movies/*"` pattern, the next metadata requested would be for `"https://metadata.ucdn.example/host1234/pathDCE"` with an expected CDNI payload type of `"MI.PathMetadata"`:

```

{
  "metadata": [],
  "paths": [
    {
      "path-pattern": {
        "pattern": "/videos/movies/hd/*"
      },
      "path-metadata": {
        "type": "MI.PathMetadata",
        "href": "https://metadata.ucdn.example/host1234/pathDEF/path123"
      }
    }
  ]
}

```

Finally, if the path of the requested resource also matches the `"/videos/movies/hd/*"` pattern, the dCDN would also fetch the following object from `"https://metadata.ucdn.example/host1234/pathDEF/path123"` with CDNI payload type `"MI.PathMetadata"`:

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.TimeWindowACL",
      "generic-metadata-value": {
        "times": [
          "windows": [
            {
              "start": "1213948800",
              "end": "1327393200"
            }
          ],
          "action": "allow"
        }
      }
    ]
  }
}
```

The final set of metadata which applies to the requested resource includes a SourceMetadata, a LocationACL, a ProtocolACL, and a TimeWindowACL.

7. IANA Considerations

7.1. CDNI Payload Types

This document requests the registration of the following CDNI Payload Types under the IANA CDNI Payload Type registry:

Payload Type	Specification
MI.HostIndex	RFCthis
MI.HostMatch	RFCthis
MI.HostMetadata	RFCthis
MI.PathMatch	RFCthis
MI.PatternMatch	RFCthis
MI.PathMetadata	RFCthis
MI.SourceMetadata	RFCthis
MI.Source	RFCthis
MI.LocationACL	RFCthis
MI.LocationRule	RFCthis
MI.Footprint	RFCthis
MI.TimeWindowACL	RFCthis
MI.TimeWindowRule	RFCthis
MI.TimeWindow	RFCthis
MI.ProtocolACL	RFCthis
MI.ProtocolRule	RFCthis
MI.DeliveryAuthorization	RFCthis
MI.Cache	RFCthis
MI.Auth	RFCthis
MI.Grouping	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

7.1.1. CDNI MI HostIndex Payload Type

Purpose: The purpose of this payload type is to distinguish HostIndex MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.1

7.1.2. CDNI MI HostMatch Payload Type

Purpose: The purpose of this payload type is to distinguish HostMatch MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.2

7.1.3. CDNI MI HostMetadata Payload Type

Purpose: The purpose of this payload type is to distinguish HostMetadata MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.3

7.1.4. CDNI MI PathMatch Payload Type

Purpose: The purpose of this payload type is to distinguish PathMatch MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.4

7.1.5. CDNI MI PatternMatch Payload Type

Purpose: The purpose of this payload type is to distinguish PatternMatch MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.5

7.1.6. CDNI MI PathMetadata Payload Type

Purpose: The purpose of this payload type is to distinguish PathMetadata MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.6

7.1.7. CDNI MI SourceMetadata Payload Type

Purpose: The purpose of this payload type is to distinguish SourceMetadata MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.1

7.1.1.8. CDNI MI Source Payload Type

Purpose: The purpose of this payload type is to distinguish Source MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.1.1

7.1.1.9. CDNI MI LocationACL Payload Type

Purpose: The purpose of this payload type is to distinguish LocationACL MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.2

7.1.1.10. CDNI MI LocationRule Payload Type

Purpose: The purpose of this payload type is to distinguish LocationRule MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.2.1

7.1.1.11. CDNI MI Footprint Payload Type

Purpose: The purpose of this payload type is to distinguish Footprint MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.2.2

7.1.1.12. CDNI MI TimeWindowACL Payload Type

Purpose: The purpose of this payload type is to distinguish TimeWindowACL MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.3

7.1.13. CDNI MI TimeWindowRule Payload Type

Purpose: The purpose of this payload type is to distinguish TimeWindowRule MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.3.1

7.1.14. CDNI MI TimeWindow Payload Type

Purpose: The purpose of this payload type is to distinguish TimeWindow MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.3.2

7.1.15. CDNI MI ProtocolACL Payload Type

Purpose: The purpose of this payload type is to distinguish ProtocolACL MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.4

7.1.16. CDNI MI ProtocolRule Payload Type

Purpose: The purpose of this payload type is to distinguish ProtocolRule MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.4.1

7.1.17. CDNI MI DeliveryAuthorization Payload Type

Purpose: The purpose of this payload type is to distinguish DeliveryAuthorization MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.5

7.1.18. CDNI MI Cache Payload Type

Purpose: The purpose of this payload type is to distinguish Cache MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.6

7.1.19. CDNI MI Auth Payload Type

Purpose: The purpose of this payload type is to distinguish Auth MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.7

7.1.20. CDNI MI Grouping Payload Type

Purpose: The purpose of this payload type is to distinguish Grouping MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.8

7.2. CDNI Metadata Footprint Types Registry

The IANA is requested to create a new "CDNI Metadata Footprint Types" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI Metadata Footprint Types" namespace defines the valid Footprint object type values used by the Footprint object in Section 4.2.2.2. Additions to the Footprint type namespace conform to the "Specification Required" policy as defined in [RFC5226]. The designated expert will verify that new type definitions do not duplicate existing type definitions and prevent gratuitous additions to the namespace. New registrations are required to provide a clear description of how to interpret new footprint types.

The following table defines the initial Footprint Registry values:

Footprint Type	Description	Specification
ipv4cidr	IPv4 CIDR address block	RFCthis
ipv6cidr	IPv6 CIDR address block	RFCthis
asn	Autonomous System (AS) Number	RFCthis
countrycode	ISO 3166-1 alpha-2 code	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

7.3. CDNI Metadata Protocol Types Registry

The IANA is requested to create a new "CDNI Metadata Protocol Types" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI Metadata Protocol Types" namespace defines the valid Protocol object values in Section 4.3.2, used by the SourceMetadata and ProtocolACL objects. Additions to the Protocol namespace conform to the "Specification Required" policy as defined in [RFC5226], where the specification defines the Protocol Type and the protocol to which it is associated. The designated expert will verify that new protocol definitions do not duplicate existing protocol definitions and prevent gratuitous additions to the namespace.

The following table defines the initial Protocol values corresponding to the HTTP and HTTPS protocols:

Protocol Type	Description	Type Specification	Protocol Specifications
http/1.1	Hypertext Transfer Protocol -- HTTP/1.1	RFCthis	RFC7230
https/1.1	HTTP/1.1 Over TLS	RFCthis	RFC7230, RFC2818

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

8. Security Considerations

8.1. Authentication and Integrity

A malicious metadata server, proxy server, or attacker, impersonating an authentic uCDN metadata interface without being detected, could provide false metadata to a dCDN that either:

- o Denies service for one or more pieces of content to one or more User Agents;
- o Directs dCDNs to contact malicious origin servers instead of the actual origin servers, and substitute legitimate content with malware or slanderous alternate content; or
- o Removes delivery restrictions (e.g., LocationACL, TimeWindowACL, ProtocolACL, or Auth metadata), allowing access to content that would otherwise be denied, and thus possibly violating license restrictions and incurring unwarranted delivery costs.

Unauthorized access to metadata could also enable a malicious metadata client to continuously issue metadata requests in order to overload a uCDN's metadata server(s).

Unauthorized access to metadata could further result in leakage of private information. A malicious metadata client could request metadata in order to gain access to origin servers, as well as information pertaining to content restrictions.

An implementation of the CDNI metadata interface MUST use mutual authentication and message authentication codes to prevent unauthorized access to and undetected modification of metadata (see Section 8.3).

8.2. Confidentiality and Privacy

Unauthorized viewing of metadata could result in leakage of private information. Content provider origin and policy information is conveyed through the CDNI metadata interface. A third party could intercept metadata transactions in order to gain access to origin servers, as well as information pertaining to content restrictions and usage patterns.

Note: The distribution of metadata by a uCDN to dCDNs could introduce privacy concerns for some content providers, e.g., dCDNs accepting content requests for a content provider's content might be able to obtain additional information and usage patterns relating to the users of a content provider's services. Content providers with concerns about divulging information to dCDNs can instruct their uCDN partners not to use CDNI when delivering their content.

An implementation of the CDNI metadata interface MUST use strong encryption to prevent unauthorized interception or monitoring of metadata (see Section 8.3).

8.3. Securing the CDNI Metadata interface

An implementation of the CDNI metadata interface MUST support TLS transport as per [RFC2818] and [RFC7230].

TLS MUST be used by the server-side (dCDN) and the client-side (uCDN) of the CDNI metadata interface, including authentication of the remote end, unless alternate methods are used for ensuring the security of the information in the CDNI metadata interface requests and responses (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

The use of TLS for transport of the CDNI metadata interface messages allows:

- o The dCDN and uCDN to authenticate each other.

and, once they have mutually authenticated each other, it allows:

- o The dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI metadata requests and responses from an authorized CDN);
- o CDNI metadata interface requests and responses to be transmitted with confidentiality; and
- o The integrity of the CDNI metadata interface requests and responses to be protected during the exchange.

When TLS is used, the general TLS usage guidance in [RFC7525] MUST be followed.

9. Acknowledgements

The authors would like to thank David Ferguson, Francois Le Faucheur, Jan Seedorf and Matt Miller for their valuable comments and input to this document.

10. Contributing Authors

[RFC Editor Note: Please move the contents of this section to the Authors' Addresses section prior to publication as an RFC.]

Grant Watson
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: gwatson@velocix.com

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose, 95134
USA

Email: kleung@cisco.com

11. References

11.1. Normative References

- [ISO3166-1] The International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes", ISO 3166-1:2013, 2013.
- [POSIX] Institute of Electrical and Electronics Engineers, "Information Technology Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language]", IEEE P1003.1, 1990.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<http://www.rfc-editor.org/info/rfc1034>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<http://www.rfc-editor.org/info/rfc1123>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<http://www.rfc-editor.org/info/rfc4291>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<http://www.rfc-editor.org/info/rfc5890>>.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <<http://www.rfc-editor.org/info/rfc5891>>.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <<http://www.rfc-editor.org/info/rfc5952>>.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <<http://www.rfc-editor.org/info/rfc6707>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<http://www.rfc-editor.org/info/rfc7493>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<http://www.rfc-editor.org/info/rfc7525>>.

11.2. Informative References

- [I-D.ietf-cdni-control-triggers]
Murray, R. and B. Niven-Jenkins, "CDNI Control Interface / Triggers", draft-ietf-cdni-control-triggers-15 (work in progress), May 2016.
- [I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-20 (work in progress), August 2016.
- [I-D.ietf-cdni-uri-signing]
Leung, K., Faucheur, F., Brandenburg, R., Downey, B., and M. Fisher, "URI Signing for CDN Interconnection (CDNI)", draft-ietf-cdni-uri-signing-09 (work in progress), June 2016.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.
- [RFC6793] Vohra, Q. and E. Chen, "BGP Support for Four-Octet Autonomous System (AS) Number Space", RFC 6793, DOI 10.17487/RFC6793, December 2012, <<http://www.rfc-editor.org/info/rfc6793>>.
- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<http://www.rfc-editor.org/info/rfc7234>>.
- [RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <<http://www.rfc-editor.org/info/rfc7336>>.
- [RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <<http://www.rfc-editor.org/info/rfc7337>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <<http://www.rfc-editor.org/info/rfc7736>>.

Authors' Addresses

Ben Niven-Jenkins
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: ben@velocix.com

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: rmurray@velocix.com

Matt Caulfield
Cisco Systems
1414 Massachusetts Avenue
Boxborough, MA 01719
USA

Phone: +1 978 936 9307
Email: mcaulfie@cisco.com

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.j.ma@ericsson.com

CDNI
Internet-Draft
Intended status: Informational
Expires: March 30, 2013

W. Jin
M. Li
B. Khasnabish
ZTE Corporation
September 26, 2012

Content De-duplication for CDNi Optimization
draft-jin-cdni-content-deduplication-optimization-03

Abstract

Recent explosive growth of content delivery/distribution networks (CDNs) and their interconnection are causing unintended repetition of content storage in the same dCDN. This can be avoided by using a suitable de-duplication mechanism. This document explores the scenarios which create the problems, and then discusses the approaches to eliminate the duplicated transmission of the same content from uCDN(s) to dCDN in CDNi networks. To implement the optimization, some enhancements to the CDNi metadata model and interface are required.

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 March 30, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. Deployment Scenarios	4
2.1. Impact of Content Duplication on the Network System	4
2.2. Current Data Deduplication Technologies	5
2.3. Content Duplication Scenario Involved in this Draft	5
2.3.1. Scenario 1	5
2.3.2. Scenario 2	6
2.3.3. Scenario 3	7
3. Content Naming for CDNi	8
3.1. Uniqueness	8
3.2. Ownership	9
4. CDNi Content De-duplication Optimization Implementation	9
4.1. Constant URL	9
4.2. Content Naming Mechanism	10
4.3. Content ID	11
4.4. Description of Content De-duplication	12
4.4.1. Pre-Positioned Content Acquisition	13
4.4.2. Dynamic Content Acquisition	14
4.4.3. Content Purge and Invalidate	16
5. Security Considerations	19
6. IANA Considerations	19
7. Acknowledgments	19
8. References	19
8.1. Normative References	19
8.2. Informative References	19
Authors' Addresses	20

1. Introduction

In some CDNi deployment, the dCDN occasionally caches the same content copy multiple times from the same Content Service Provider (CSP). For example, the CSP may have the agreement with two authoritative CDNs, and both could be the upstream CDNs of the same dCDN. Cascading of CDNs may result in a similar scenario as well. The top-layer uCDN establishes connections with two intermediate-layer uCDNs respectively, and both connect to the same bottom-layer dCDN. In such scenarios, the dCDN may receive the content via 'push' from one of the uCDN via pre-position procedure, and then may also request for downloading the same content from another uCDN via another pre-position procedure or upon user's content request.

Storing the same content multiple times not only wastes the dCDN's the memory or storage resources, it also causes wastage of transmission bandwidth that is used to deliver the same content repeatedly. Therefore, it is necessary to avoid delivering the same content from different uCDNs to dCDN repeatedly. In this draft, we list a set of scenarios which may cause repeated delivery of the same content. A feasible solution for content de-duplication is then discussed.

In order to address the content repetition problem, several issues need to be considered.

- * How to detect content repetition by dCDN.

- * How to avoid content repetition, when one or more uCDNs selects one dCDN to deliver the same content to multiple User Agents.

This document provides detailed analysis on the issues of content repetition. We realize that there is a need to develop an optimized mechanism to de-duplicate the content in CDNi network. In order to implement such optimization, enhancement to CDNi metadata model and interface may be required.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document reuses the terminology defined in:

[draft-ietf-cdni-problem-statement-08],

[draft-ietf-cdni-requirements-03],

[draft-ietf-cdni-framework-01], and

[draft-ietf-cdni-use-cases-09].

Resource Id: a metadata object (e.g., partial or whole URL, or other format) which is generated by uCDN and identifies the storage of the content in the uCDN.

Content Id: a metadata object (e.g. a URN) which uniquely identifies the content in the scope of CDNi.

2. Deployment Scenarios

This section illustrates several CDNi deployment scenarios that typically lead to duplicated content in the same server.

2.1. Impact of Content Duplication on the Network System

Along with the explosive growth of digital information, the space occupied by data is increasing as well. In the past decade, the capacity of storage system provided by many industries has developed from dozens of gigabytes to hundreds of terabytes or even more. With the exponential growth of data, enterprises are facing with an increasing number of time points for quick data backup and recovery. The cost to manage and store data, as well as the space and consumption of data center, is also growing into a more and more serious situation. Survey exposes that up to 60% of the data stored in the application system is redundant and the proportion continues to grow along with the time moving forward.

As for CDN, the duplication of the content delivered will:

1. increase the complexity of CDN management. An increasing number of content tables to be maintained are needed, thus reducing the efficiency of content search;
2. demand larger CDN storage capacity which would be a waste of facility and investment;
3. lead to inaccurate statistics of hot content, which consequently results in the inaccuracy of the arithmetic for the hot content dispatching in the CDN;
4. increase the latency for the CDN content access. Such cases as local content could not be hit and the same content has to be acquired from other CDNs would often occur.

2.2. Current Data Deduplication Technologies

At present, data deduplication technologies are widely used in the storage backup and archiving system, in which the deduplication module is responsible for comparing and analyzing the data content, finding out redundant data and sending corresponding metadata back to the storage service interface. Finally, non-duplicated data will be stored into the storage medium. Main technologies can be divided into:

1. Identical Data Detection: identical data includes identical files and identical data block. Whole File Detection (WFD) technology uses Hash for data mining. Fixed-sized partition (FSP) detection technology, content-defined chunking (CDC) detection technology and sliding block technology are used for the lookup and deletion of duplicated data;
2. Resembling Data Detection: according to the resemblance characteristics of the data itself, shingle technology, bloom filter technology and mode match technology are used to find out the duplicated data that can not be detected by Identical Data Detection technologies.

Above technologies are applied under the prerequisite that the content is completed downloaded and stored. However, for CDN, comparison of the content after downloading is a waste of transmission traffic and computation used for comparison. What!_s more, with a widespread deployment of CDNi system, content duplication issue will not be caused by above reasons. Instead, it results from the redirection procedures used in CDNi during which URLs pointed to the same content would be changed. In this case, dCDN would download the same content several times due to the different URLs that in fact point to the same content. Therefore, in CDNi, current data deduplication technologies can not be used to address the issue and scenarios proposed in this draft.

2.3. Content Duplication Scenario Involved in this Draft

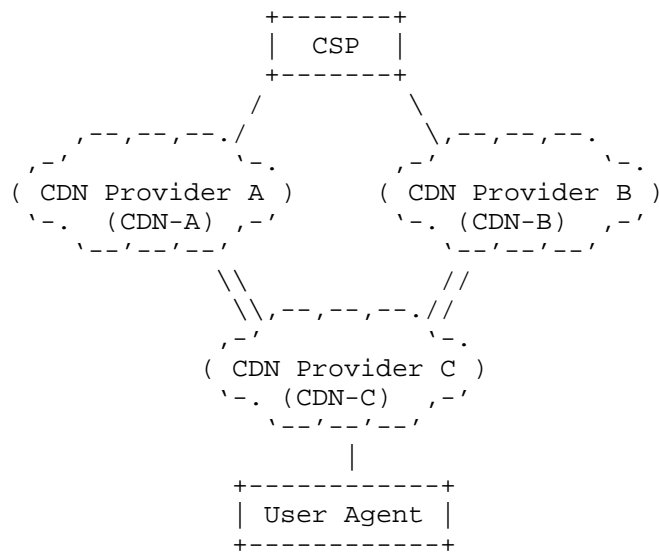
In this document, the content duplication results from the redirection procedures used in CDNi during which URLs pointed to the same content would be changed. In this case, dCDN would download the same content several times due to the different URLs that in fact point to the same content.

2.3.1. Scenario 1

As depicted in Figure 1, both CDN-A and CDN-B establish interconnections with CDN-C that acts as a dCDN. Thus, CDN-C will

cache the content for CDN-A and CDN-B. When both CDN provider A and CDN provider B have agreements with the same CSP for content delivery, CDN-C may be required by CDN-A and CDN-B separately to retrieve and cache the same content from the CSP. CDN-C is likely to suffer from content repetition troubles.

As the location of the content in a CDN is normally assigned by CDN itself, the URLs of the same content are likely different between CDNs. So it is not enough to determine whether the content to be retrieved and cached is the same only by the URL of the content.



=== CDN Interconnect

Figure 1 Interconnected CDNs with the same CSP and with one dCDN

2.3.2. Scenario 2

Now we consider the case of cascaded CDNs, as depicted in Figure 2. Note that the top-layer Upstream CDN-A has direct contract with CSP and interconnects with two middle-layer CDNs (CDN-B and CDN-C) that have the same bottom-layer Downstream CDN (CDN-D) interconnected to them. Consequently, there are two possible delivery paths for CDN-D to cache the contents of CSP, one is CDN-A -> CDN-B -> CDN-D, and the other is CDN-A -> CDN-C -> CDN-D. CDN-D may need to cache the same content by upstream CDNs(CDN-B and CDN-C) on different paths. If the URL of the content is changed by CDN-B or CDN-C, CDN-D cannot be aware of the contents to be cached and therefore this may lead to storing of duplicated contents.

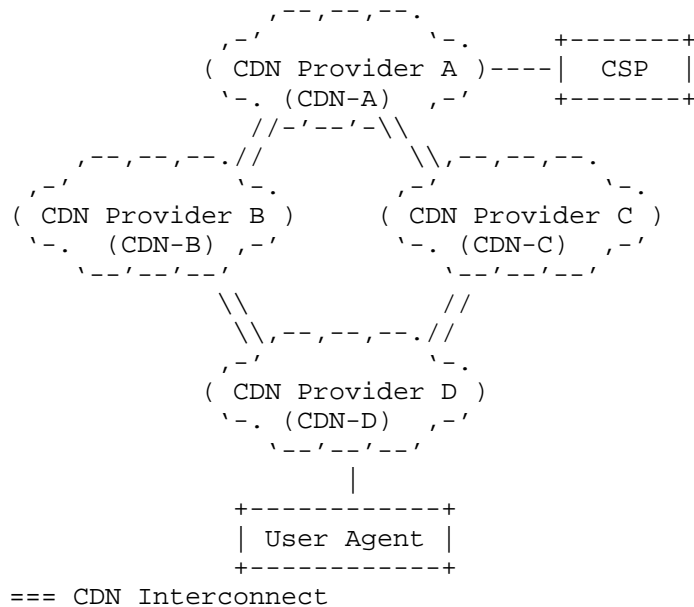


Figure 2 Cascaded CDNs

2.3.3. Scenario 3

As depicted in Figure 3, two interconnected CDNs - CDN-A(uCDN) and CDN-B(dCDN) - both have contracts with CSP. CDN-B plays two roles at the same time: downstream CDN of CDN-A and Authoritative CDN of CSP. When an end-user of CDN-A initiates content request from the CSP, CDN-A decides that CDN-B should be the serving CDN. Then CDN-A redirects the request to CDN-B. If CDN B does not have a local copy of the requested content yet (cache miss), CDN B ingests the content from CDN A. Normally CDN-B, as Authoritative CDN, is very likely to have already cached this content from original server. If CDN-B cannot identify the requested contents as the same content, this same content will be repeatedly retrieved and cached.

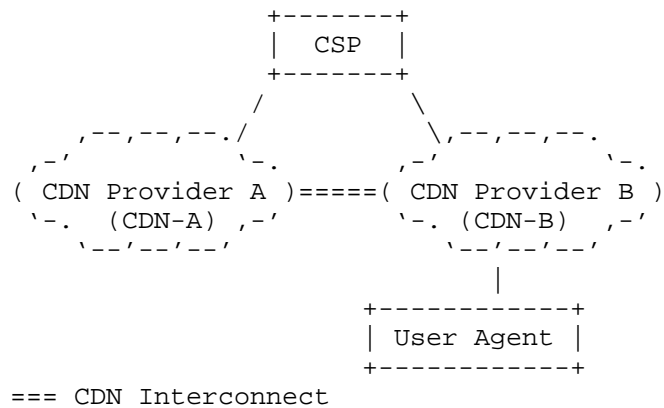


Figure 3 Interconnected CDNs with the same CSP

3. Content Naming for CDNi

It is well known that CDNs have their own content naming mechanisms, most of which are independent and separated from one another due to the use of different algorithms such as Hash algorithms. It implies that for the same content distributed by two CDNs, the corresponding content identifiers are likely to be quite different. [I-D.ietf-cdni-requirements-03] treats the information regarding CDN content naming as intra-CDN information and the CDNI solution MUST not require intra-CDN information to be exposed to other CDNs for effective and efficient delivery of the content. Therefore, establishing a uniform content naming mechanism is urgently needed for CDNi network. This mechanism which can be implemented by CDNI Metadata Distribution Protocol may have the following properties.

3.1. Uniqueness

CDNi content naming mechanism must guarantee the uniqueness of the content identification. Although URL is widely used for identifying network resource, it is not quite suitable for content identification in CDNi network where content de-duplication needs to be taken into consideration. Although the method of URL match is commonly used by many cache systems to detect the repetitive files with same name for avoiding content repetition, it will probably fail for CDNi case. This is due to the fact that different forwarding mechanisms are used in different CDNs involved. The user-originated requests are always snooped by the DPI (deep packet inspection) devices before transmitted to the original server, whereas the requests received by dCDN are always redirected by one or more uCDNs. Since there is no guarantee that the URLs will not be changed through the redirection

process, a new type of object needs to be defined to represent the uniqueness of content identifier.

For the CSP's the contents that are distributed into different interconnected CDNs, the related metadata objects may be somewhat different in many cases. For example, in Figure 2, when both CDN-A and CDN-B delegate the delivery of the same CSP's content, the content metadata such as (a) content description, (b) access policy, and (c) security policy may be not be exactly the same in both cases. Such metadata information is not suitable for content identification. Consequently, we need to define a new type of metadata object that helps uniquely identify the same content.

3.2. Ownership

CDNi content naming mechanism should embody the ownership of content identification. Typically, a CDN provider may have contracts with many CSPs for delivering their contents, as well as operate its own Content delivery network. However, it may happen that a lot of contents published by these CSPs may be very similar, and many of them may even be exactly the same. Therefore, the problem is whether these identical copies are from the same CSP or how the interconnected CDNs can verify that these contents are identical. (Note: Such copies are pointed by the same content identification only if they are from the same content source.) For a traditional (non-interconnected) CDN, there is no problem to distinguish them via its intra content naming mechanism. When a CDN interconnects with other CDNs, the condition becomes more complicated due to the lack of awareness of CSP's content when the CDN acts as a dCDN.

4. CDNi Content De-duplication Optimization Implementation

4.1. Constant URL

In general, URL-based mechanism can be used to implement content de-duplication in CDNi network. As referred in section 2, the URL description for a content may be different from uCDN to uCDN and is likely to be changed in the redirection process. An agreement to configure a specific URL between a pair of interconnected CDN is used in the draft [I-D.ietf-cdni-framework-01], however this method is not flexible enough for supporting de-duplication in complex CDNi network. So a feasible proposal is to specify a mechanism for CDNi network to guarantee that CSP's same contents cached in different uCDNs are identified by the same URL and that the URL is unchanged or at least the path part remains the same in the redirection process. The main problem of this mechanism is lack of resilience and we prefer an alternative mechanism as introduced in section 4.2 below.

4.2. Content Naming Mechanism

This section provides a detailed description of CDNi content naming mechanism using CDNi Metadata Protocol.

In general, CSP's content as well as its copies cached in interconnected CDNs are delivered to numerous End-Users. The draft [I-D.ietf-cdni-framework-01] assigns each copy an identifier in the form of an URL that is embedded with "CDN-Domain" which is used to distinguish whether a download request is from an end-user or from an uCDN. We use the term Resource Identifier to represent the storage pointing to the content copies in interconnected CDNs. However, unlike the usage in the draft [I-D.ietf-cdni-framework-01], in this document CDNi content naming mechanism specifies Resource Identifiers as the one which is only related to contents in uCDNs. Taking the example in section 2.3, we use Resource Identifier A to point to content originated through uCDN A.

Although the Resource Identifier is able to identify a content, the uniqueness of content identification can't be guaranteed, as Resource Identifier is used to identify the storage of the content in the uCDN and therefore will be changed during redirection processes between different uCDNs. In order to resolve it, we introduce the term Content Identifier that is assigned to associate with Resource Identifier to uniquely identify the content and is similar to the URN usage. Note that the Content Identifier MUST be globally unique. Figure 4 shows a metadata model that can be used for maintaining the relationship between the two types of Identifiers. Using this model, the dCDN is able to uniquely identify and route requests towards the same targeted content.

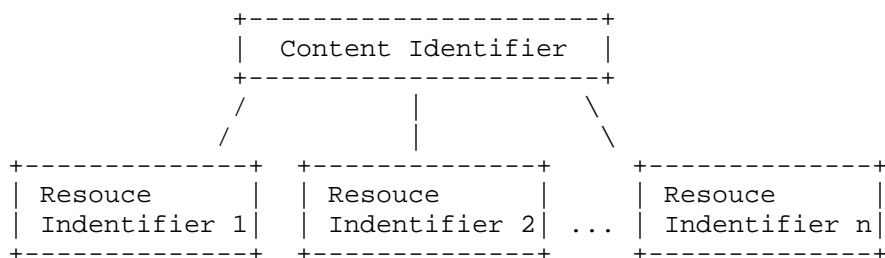


Figure 4 Metadata Model for Maintaining Relationship among Multi-IDs

Note: We need to develop an authoritative 'Entity' for creating and maintaining the Content Identifier.

4.3. Content ID

Actually, EIDR (Entertainment Identifier Registry), an industry non-profit organization, has already started the research and standardization of the globally unique content identifier and its generating mechanism. As stated in the whitepaper of [UNIVERSAL UNIQUE IDENTIFIERS IN MOVIE AND TELEVISION SUPPLY CHAIN MANAGEMENT], EIDR offers an inexpensive mechanism for uniquely identifying the complete range of audiovisual assets relevant to commerce including micro-assets such as clips and newer types of objects such as encodings. The EIDR data model can be readily extended to cover new and emerging objects and relationships as the industry evolves over time. EIDR naming system meets the requirements of coverage, flexibility, extensibility, scalability, cost-effectiveness, interoperability, etc.

The EIDR registry assigns a unique universal identifier for all registered assets. EIDR is an opaque ID with all information about the registered asset stored in the central registry. Its structure consists of a standard registry prefix, the unique suffix for each asset and a check digit. The suffix of an asset ID is of the form XXXX-XXXX-XXXX-XXXX-XXXX-C, where X is a hexadecimal digit and C is the ISO 7064 Mod 37, 36 check character.

Standard Prefix for EIDR Registry	Unique Suffix for each asset	check digit
10.5240/	XXXX-XXXX-XXXX-XXXX-XXXX	-C

Figure 5 Structure of Content ID

The content provider submits content items for registration to the registry system, along with core metadata and information. The system uses a sophisticated system to insure that the object submitted to the registry has not already been registered and then generates an EIDR for the content. All above is done before the content is injected into CDNi system. After that, the content identifier, used as metadata of the content, is injected into CDNi system and transmitted between uCDN and dCDN via such interface as Control Interface, Metadata Interface.

The detail information of EIDR can be referred to in the whitepaper of [UNIVERSAL UNIQUE IDENTIFIERS IN MOVIE AND TELEVISION SUPPLY CHAIN MANAGEMENT].

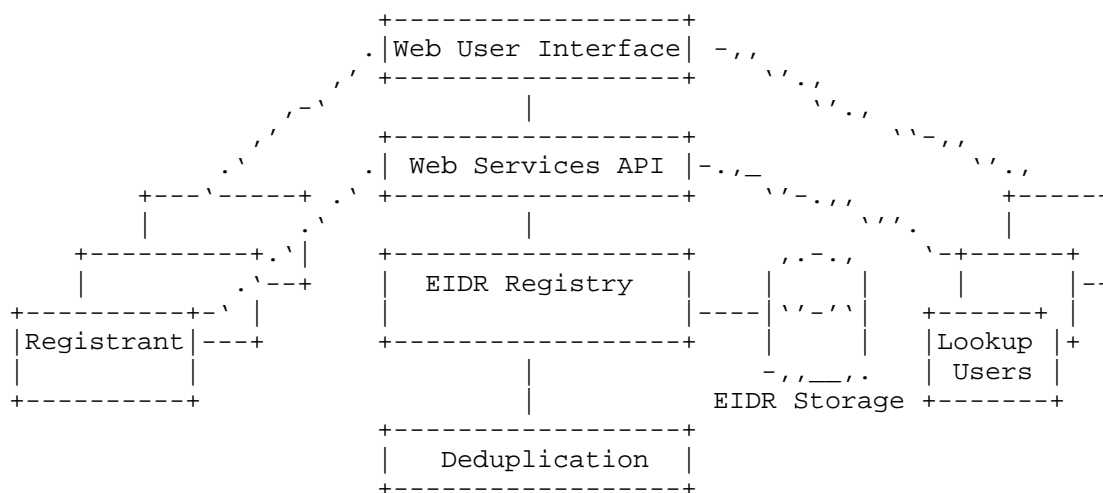


Figure 6 Entertainment Identifier Registry diagram

From the analysis of above section, the content identifier specified in EIDR is in line with the requirements of content deduplication proposed in this draft. In this document, it is suggested that we introduce content identifier format and a mechanism for generating it into CDNi with an objective to address content duplication issue.

4.4. Description of Content De-duplication

In this section the details of the solutions that use CDNi Content Naming mechanism for content de-duplication are discussed.

Content Identifier can be defined as a metadata object. The metadata object can be distributed with/without the actual content item from uCDN to dCDN for storage in the dCDN, if the uCDN wants to pre-position the content item to the dCDN before the actual user request. The content Identifier metadata object binds with the Resource Identifier to identify the storage of the content in the uCDN and forms a content identification model for the content item. By using this content identification model, an interconnected CDN is able to detect content repetition. The content status must be synchronously updated by the interconnected CDN. According to content status, the interconnected CDN can determine whether the resource copy is cached or not.

We present several procedures for optimized implementation of avoidance of CDNi content repetition.

4.4.1. Pre-Positioned Content Acquisition

The following flow illustrates how the two uCDNs successively pre-position the same content in the dCDN. In this flow, the content to be pre-positioned in the dCDN is identified by different Resource Identifiers corresponding to uCDN A and uCDN B.

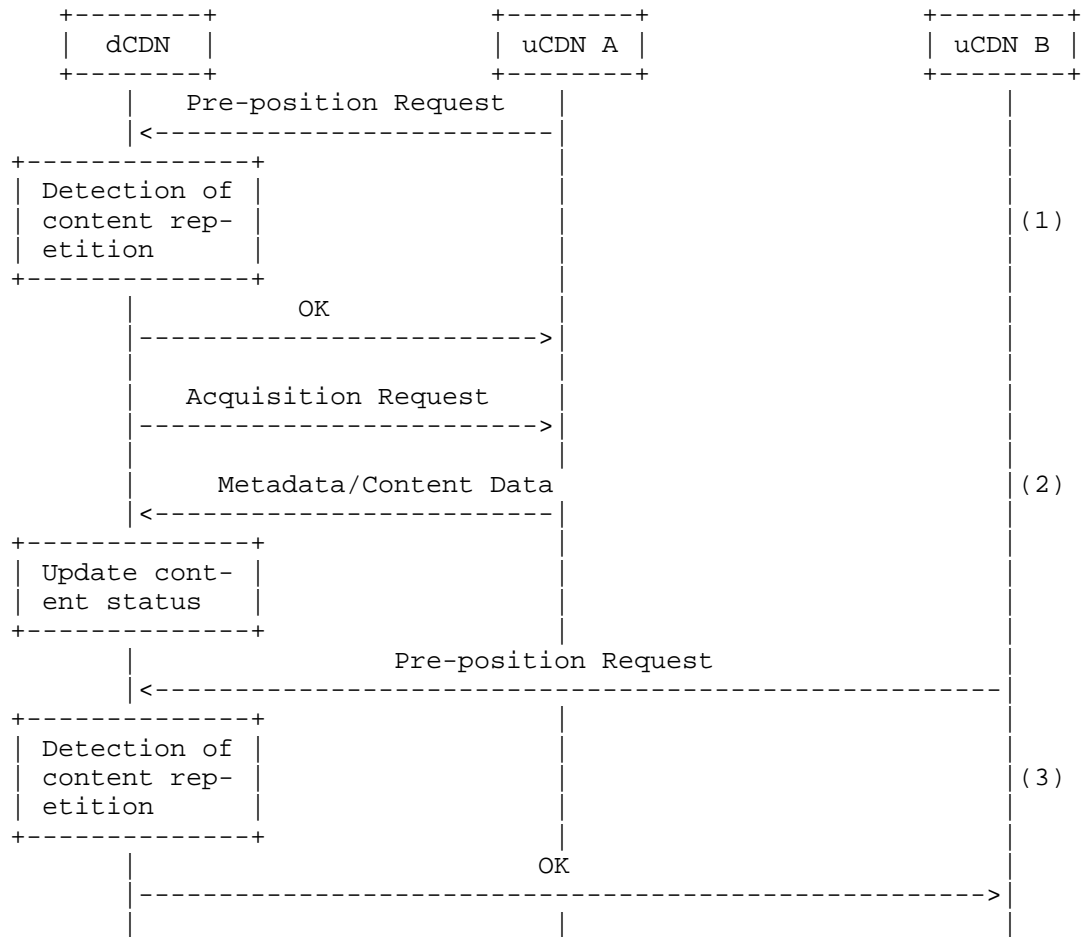


Figure 7 Acquisition of Pre-Positioned Content

The steps that are illustrated in the figure are as follows:

1. The uCDN A requests that the dCDN pre-positions a particular content item identified by its Resource Identifier and Content

Identifier. This message is sent via Trigger Interface. Receiving the message, the dCDN uses the Content Identifier to look up its stored content identification model to see whether the same content item is cached. With the result that the metadata does not exist, the dCDN replies to uCDN A with an OK message to notify that no such copy has been cached and content pre-position is required.

2. The dCDN acquires metadata of the content or the content itself from uCDN A. Once the content is pre-positioned, dCDN updates the content status and maintains the binding relation between Resource Identifier and Content Identifier metadata.

3. The uCDN B requests that dCDN pre-positions the same content item identified by its Resource Identifier and Content Identifier. Receiving the message, the dCDN uses the Content Identifier to look up its stored content identification model. Because such metadata exists, dCDN determines that the content is already cached. Then, dCDN replies to uCDN A with an OK message to notify that the same copy has been cached and the pre-position request should be cancelled. The dCDN locally binds the new Resource Identifier provided by uCDN B with the Content Identifier.

4.4.2. Dynamic Content Acquisition

The following flows illustrates how the dCDN performs content de-duplication in cases of a cache miss and a cache hit without content pre-positioning.

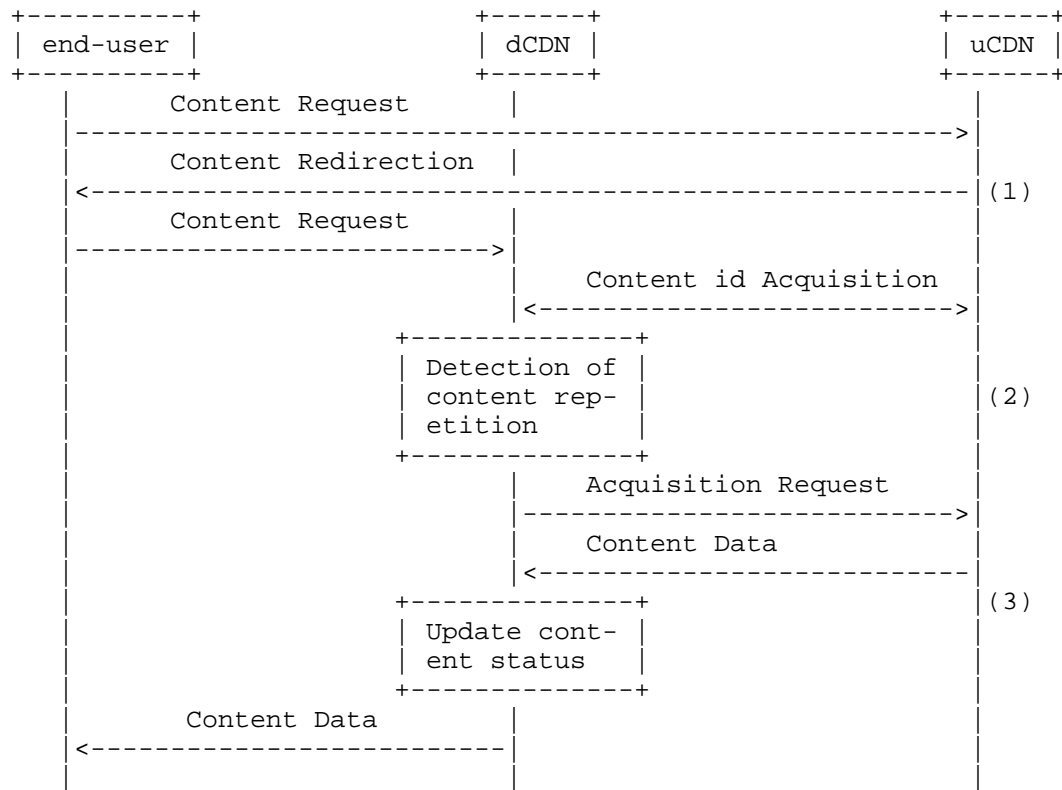


Figure 8 Dynamic Content Acquisition (cache miss case)

The steps that are illustrated in the figure are as follows:

1. A content request originated from an end-user is received at uCDN. The uCDN processes the request and recognizes that the end-user is best served by a dCDN. So uCDN redirects the request to the dCDN by sending redirection response to the end-user who then requests the content from the dCDN. The uCDN encapsulates Resource Identifier of the requested content item in the redirection response.
2. Receiving the request, the dCDN uses the Resource Identifier pointing to the requested resource to fetch the corresponding Content Identifier from the uCDN. The dCDN then uses the Content Identifier to look up its stored content identification model to check whether the content item is cached. With the result that such metadata does not exist, the case of a cache miss is determined by the dCDN, and therefore content needs to be downloaded from the uCDN before delivered to the end-user.

3. The dCDN acquires the requested content from uCDN A. Once the content is cached, dCDN updates the content status and maintains the binding relation between Resource Identifier and the Content Identifier metadata. The dCDN then delivers content data to the end-user.

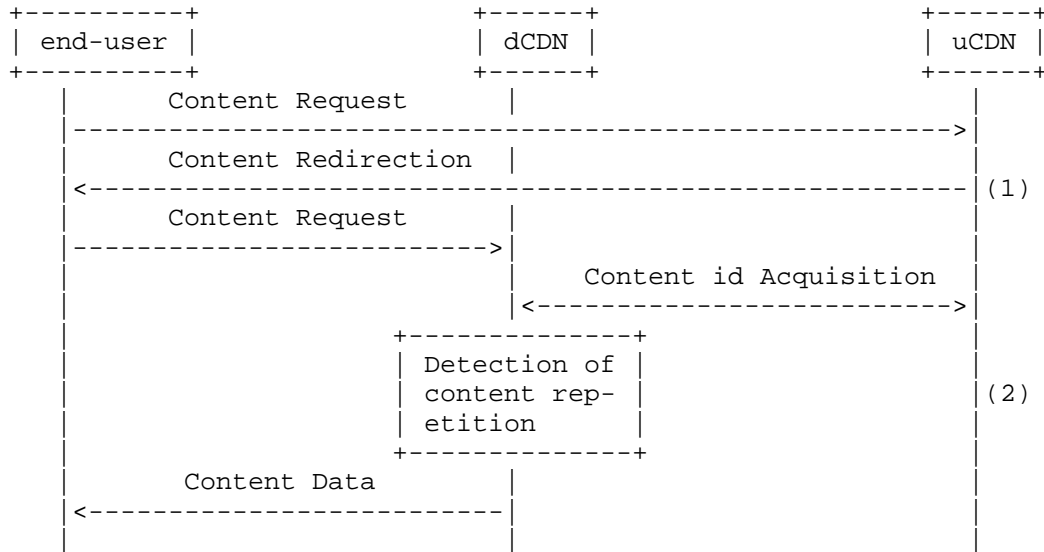


Figure 9 Dynamic Content Acquisition (cache hit case)

The steps that are illustrated in the figure are as follows:

Steps 1 and 2 are exactly the same as steps 1 and 2 of Figure 6, except that this time dCDN determines the case of a cache hit according to the existence of such record in the corresponding Content Identifier metadata.

This flow differs from the one in Figure 6 only in terms of not triggering dynamic content acquisition (step 3), since the content has already been cached by dCDN.

4.4.3. Content Purge and Invalidate

In general, the dCDN would assign separate location for each of its uCDN to store triggers. However, when it comes to complex CDNi deployment as discussed in this draft, dCDN is likely to receive multiple trigger operations coming from different uCDNs on the same content. If one of the uCDNs requests to invalidate certain content,

after receiving this Invalidated Trigger, dCDN finds the content using the Content Identifier of this content and marks Invalid in the Trigger Status Resource corresponding to the requesting uCDN. By doing so, this content is unavailable to this uCDN before it re-validates this content. The access to this content by other uCDNs will not be impacted.

If one of the uCDNs requests to purge certain content, after receiving this Purge Trigger, dCDN finds the content using the Content Identifier of this content and marks Invalid in the Trigger Status Resource corresponding to the requesting uCDN. By doing this, this uCDN is not able to make any operation on this content, while the content itself is not deleted from the cache of dCDN. The access to this content by other uCDNs will not be impacted. Only when all the uCDNs connected with this dCDN request to purge the same content and dCDN accepts all these requests will the content be purged from dCDN.

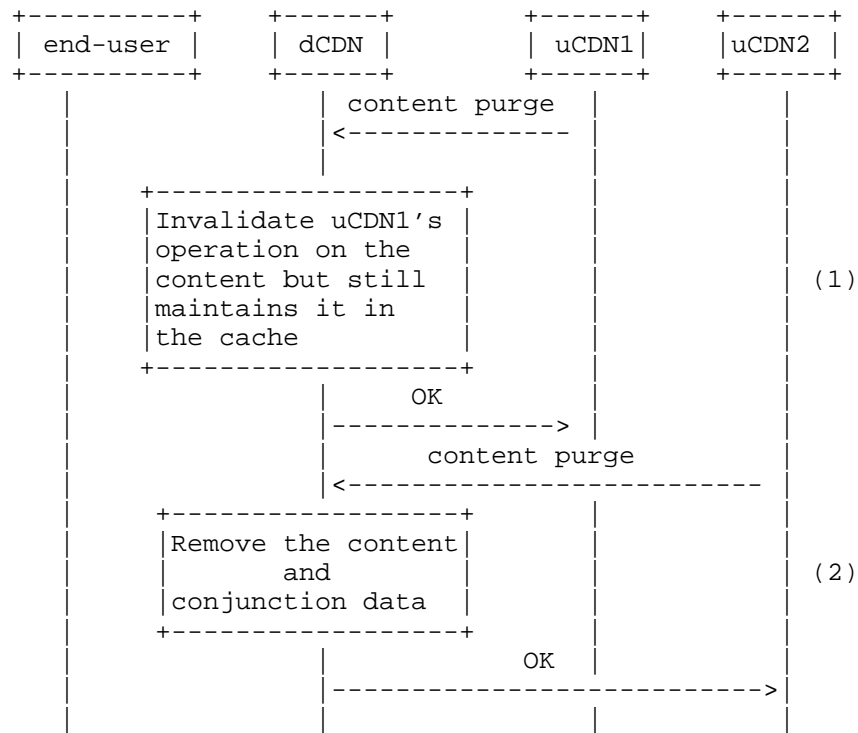


Figure 10 Removal of Content

A premise is that the content copy to be purged has already been cached in the dCDN from the uCDN. The Content Identifier of the content is linked with two Resource IDs from uCDN1 and uCDN2. The steps illustrated in the figure are as follows:

1. The uCDN1 requests that the dCDN remove some content resource identified by the Content Identifier due to the deployment policy or expiration of content's life-time. dCDN then invalidates uCDN's operation on the requested content but still maintains it in the cache so that other uCDNs that also have conjunction with this content can operate on this content normally. It then replies an OK response to uCDN1
2. If uCDN2 also requests that dCDN remove the same content, dCDN removes the content and all the conjunction metadata. It then replies an OK response to uCDN2.

5. Security Considerations

To be added later

6. IANA Considerations

This document has no IANA Considerations.

7. Acknowledgments

The authors would like to thank Francois Le Faucheur, Kevin Ma, Theodore Zahariadis, Ben Niven-Jenkins, and Ramki Krishnam for valuable inputs, suggestions, and discussions.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2. Informative References

[Data Deduplication Techniques]

Ao, L., Shu, JW., and MQ. Li, "Data Deduplication Techniques", May 2010.

[EIDR WHITEPAPER]

EIDR, EIDR., "UNIVERSAL UNIQUE IDENTIFIERS IN MOVIE AND TELEVISION SUPPLY CHAIN MANAGEMENT", October 2010.

[I-D.ietf-cdni-framework]

Peterson, L. and B. Davie, "Framework for CDN Interconnection", April 2012.

[I-D.ietf-cdni-problem-statement]

Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", May 2012.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", December 2011.

[I-D.murray-cdni-triggers]

Murray, R., Niven-Jenkins, B., and Velocix, "CDN Interconnect Triggers", March 2013.

[I-D.narten-iana-considerations-rfc2434bis]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs",
draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

Authors' Addresses

WeiYi Jin
ZTE Corporation
Nanjing, 210012
China

Phone: +86 025-52871364
Email: jin.weiya@zte.com.cn

Mian Li
ZTE Corporation
Nanjing, 210012
China

Phone: +86 025-88014641
Email: li.mian@zte.com.cn

Bhumip Khasnabish
ZTE Corporation
New Jersey, 07960
USA

Phone: +001-781-752-8003
Email: bhumip.khasnabish@zteusa.com

CDNI
Internet-Draft
Intended status: Informational
Expires: September 29, 2013

W. Jin
M. Li
B. Khasnabish
ZTE Corporation
March 28, 2013

Content De-duplication for CDNI Optimization
draft-jin-cdni-content-deduplication-optimization-04

Abstract

Recent explosive growth of content delivery/distribution networks (CDNs) and their interconnection are causing unintended repetition of content storage in the same dCDN. This can be avoided by using a suitable de-duplication mechanism. This document explores the scenarios which create the problem, and then discusses the approaches to eliminate the duplicated transmission of the same content from uCDN(s) to dCDN in CDNI networks. To implement the optimization, some enhancement to the CDNI metadata model and interface is required.

We realize that for business-specific purposes the same content may be encrypted/packaged with different keys for different providers. The impact of DRM (Digital Rights Management) technology on de-duplication will be discussed in a future version of this draft.

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 September 29, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. Deployment Scenarios	4
2.1. Impact of Content Duplication on the Network System	4
2.2. Current Data Deduplication Technologies	5
2.3. Content Duplication Scenario Involved in this Draft	5
2.3.1. Scenario 1	5
2.3.2. Scenario 2	6
2.3.3. Scenario 3	7
3. Content Naming for CDNi	8
3.1. Uniqueness	8
3.2. Ownership	9
4. CDNi Content De-duplication Optimization Implementation	9
4.1. Constant URL	9
4.2. Content Naming Mechanism	10
4.3. Content ID	11
4.4. Description of Content De-duplication	12
4.4.1. Pre-Positioned Content Acquisition	13
4.4.2. Dynamic Content Acquisition	14
4.4.3. Content Purge and Invalidate	16
5. Security Considerations	18
6. IANA Considerations	18
7. Acknowledgments	18
8. References	18
8.1. Normative References	18
8.2. Informative References	18
Authors' Addresses	19

1. Introduction

In some CDNi deployment, the dCDN occasionally caches the same content copy multiple times from the same Content Service Provider (CSP). For example, the CSP may have the agreement with two authoritative CDNs, and both could be the upstream CDNs of the same dCDN. Cascading of CDNs may result in a similar scenario as well. The top-layer uCDN establishes connections with two intermediate-layer uCDNs respectively, and both connect to the same bottom-layer dCDN. In such scenarios, the dCDN may receive the content via 'push' from one of the uCDN via pre-position procedure, and then may also request for downloading the same content from another uCDN via another pre-position procedure or upon user's content request.

Copies of the same content may be transferred to one dCDN, which results in waste of the dCDN's memory or storage, and transmission bandwidth that is used to deliver the copies repeatedly. Therefore, it is necessary to avoid delivering the same content from different uCDNs to dCDN repeatedly. In this draft, we list a set of scenarios which may cause repeated delivery of the same content. A feasible solution for content de-duplication is then discussed.

In order to address the content repetition problem, several issues need to be considered.

- * How to detect content repetition by dCDN.

- * How to avoid content repetition, when one or more uCDNs select(s) one dCDN to deliver the same content to multiple User Agents.

This document provides detailed analysis on the issues of content repetition. We realize that there is a need to develop an optimized mechanism to de-duplicate the content in CDNi network. In order to implement such optimization, enhancement to CDNi metadata model and interface may be required.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document reuses the terminology defined in:

[RFC 6707],

[RFC 6770],

[draft-ietf-cdni-framework], and

[draft-ietf-cdni-requirements].

Resource Id: a metadata object (e.g., partial or whole URL, or other format) which is generated by uCDN and identifies the storage of the content in the uCDN.

Content Id: a metadata object (e.g. a URN) which uniquely identifies the content in the scope of CDNi.

2. Deployment Scenarios

This section illustrates several CDNi deployment scenarios that typically lead to duplicated content in the same server.

2.1. Impact of Content Duplication on the Network System

Along with the explosive growth of digital information, the storage space required by data is increasing as well. In the past decade, the capacity of storage system provided by many industries has developed from dozens of gigabytes to hundreds of terabytes or even more. With the exponential growth of data, enterprises are facing with much more frequent data backup and recovery. The cost to manage and store data, as well as the space and consumption of data center, is also growing into a more and more serious situation. Survey exposes that up to 60% of the data stored in the application system is redundant and the proportion continues to grow along with the time moving forward. This also leads to extra load of the network in transmitting repeat copies of the same content.

As for CDN, the duplication of the content delivered will:

1. increase the complexity of CDN management. An increasing number of content tables need to be maintained, which will reduce the efficiency of content search;
2. demand larger CDN storage capacity, which would be a waste of facility and investment;
3. lead to inaccurate statistics of hot content, which consequently results in the inaccuracy of the arithmetic for the hot content dispatching in the CDN;
4. increase the latency for the CDN content access. Such cases as local content could not be hit and the same content has to be acquired from other CDNs would occur more frequently.

2.2. Current Data Deduplication Technologies

At present, data deduplication technologies are widely used in the storage backup and archiving system, in which the deduplication module is responsible for comparing and analyzing the data content, finding out redundant data and sending corresponding metadata back to the storage service interface. Finally, non-duplicated data will be stored into the storage medium. Main technologies can be divided into:

1. Identical Data Detection: identical data includes identical files and identical data block. Whole File Detection (WFD) technology uses Hash for data mining. Fixed-sized partition (FSP) detection technology, content-defined chunking (CDC) detection technology and sliding block technology are used for duplicated data lookup and deletion;

2. Resembling Data Detection: according to the resemblance characteristics of the data itself, shingle technology, bloom filter technology and mode match technology are used to find out the duplicated data that can not be detected by Identical Data Detection technologies.

Above technologies are applied under the prerequisite that the content is completed downloaded and stored. However, for CDN, comparison of the content after downloading is a waste of transmission traffic and computation. In addition, with a widespread deployment of CDNi system, content duplication issue will not result from above reasons. Instead, it results from the redirection procedures used in CDNi during which URLs pointed to the same content is changed. In this case, dCDN would download the same content several times due to different URLs that in fact point to the same content. Therefore, in CDNi, current data deduplication technologies can not be used to address the issue and scenarios proposed in this draft.

2.3. Content Duplication Scenario Involved in this Draft

In this document, the content duplication results from the redirection procedures used in CDNi during which URLs pointed to the same content is changed. In this case, dCDN would download the same content several times due to different URLs that in fact point to the same content.

2.3.1. Scenario 1

As depicted in Figure 1, both CDN-A and CDN-B establish interconnections with CDN-C that acts as a dCDN. Thus, CDN-C will

cache the content for CDN-A and CDN-B. When both CDN provider A and CDN provider B have agreements with the same CSP for content delivery, CDN-C may be required by CDN-A and CDN-B separately to retrieve and cache the same content from the CSP. CDN-C is likely to suffer from content repetition problems.

As the location of the content in a CDN is normally assigned by CDN itself, the URLs of the same content are likely to be different between CDNs. So it is not enough to determine whether the content to be retrieved and cached is the same only by the URLs of the content.

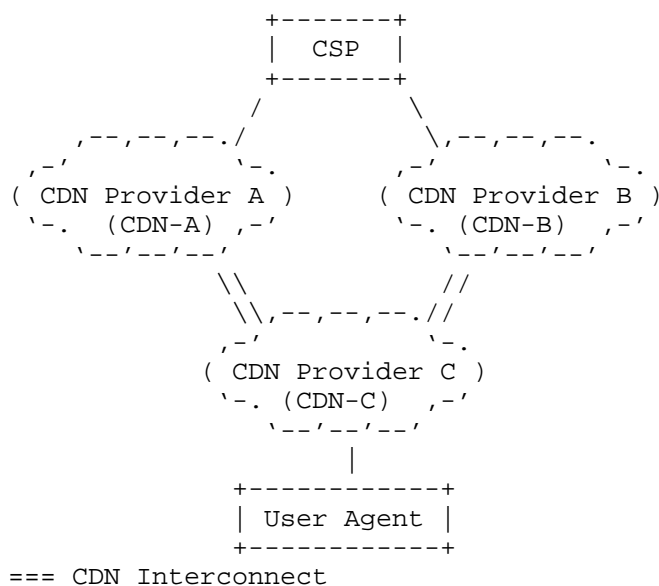


Figure 1 Interconnected CDNs with the same CSP and with one dCDN

2.3.2. Scenario 2

Now we consider the case of cascaded CDNs, as depicted in Figure 2. Note that the top-layer Upstream CDN-A has direct contract with CSP and interconnects with two middle-layer CDNs (CDN-B and CDN-C) that have the same bottom-layer Downstream CDN (CDN-D) interconnected to them. Consequently, there are two possible delivery paths for CDN-D to cache the content of CSP. One is CDN-A -> CDN-B -> CDN-D, and the other is CDN-A -> CDN-C -> CDN-D. CDN-D may cache the same content by upstream CDNs (CDN-B and CDN-C) on different paths. If the URL of the content is changed by CDN-B or CDN-C, CDN-D is not able to be aware of the content to be cached and therefore this may lead to

duplicate storage of the same content.

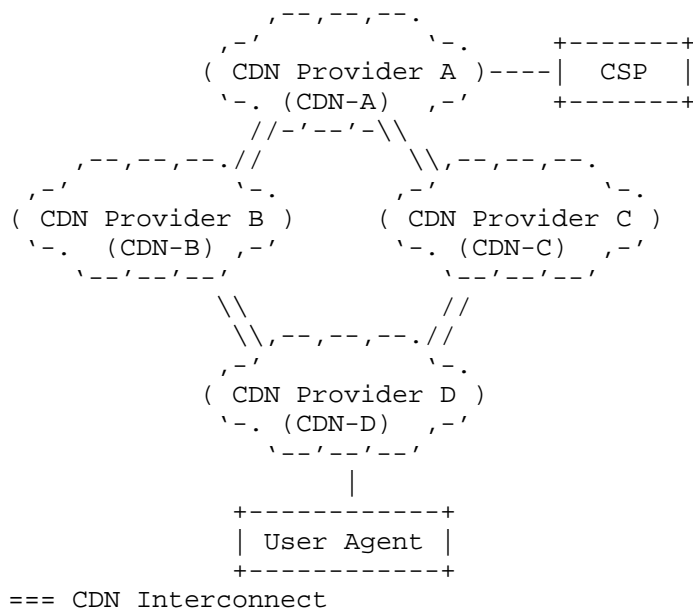


Figure 2 Cascaded CDNs

2.3.3. Scenario 3

As depicted in Figure 3, both two interconnected CDNs - CDN-A(uCDN) and CDN-B(dCDN) - have contracts with CSP. CDN-B plays two roles at the same time: downstream CDN of CDN-A and Authoritative CDN of CSP. When an end-user of CDN-A served by CDN-B initiates content request from the CSP, CDN-A decides that CDN-B should be the serving CDN. Then CDN-A redirects the request to CDN-B. If CDN B does not have a local copy of the requested content (cache miss), CDN B ingests the content from CDN A. When CSP pushes the same content to CDN-B and if CDN-B cannot identify the duplication, this same content will be repeatedly retrieved and cached.

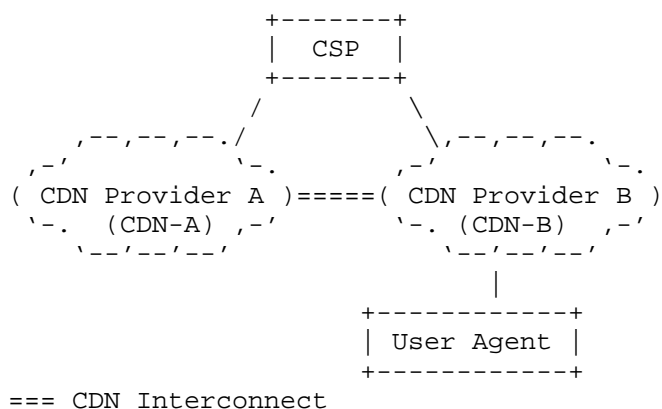


Figure 3 Interconnected CDNs with the same CSP

3. Content Naming for CDNi

It is well known that CDNs have their own content naming mechanisms, most of which are independent and separated from one another due to the use of different algorithms such as Hash algorithms. It implies that for the same content distributed by two CDNs, the corresponding content identifiers are likely to be quite different. [I-D.ietf-cdni-requirements] treats the information regarding CDN content naming as intra-CDN information and the CDNI solution MUST not require intra-CDN information to be exposed to other CDNs for effective and efficient delivery of the content. Therefore, establishing a uniform content naming mechanism is urgently needed for CDNi network. This mechanism which can be implemented by CDNI Metadata Distribution Protocol may have the following properties.

3.1. Uniqueness

CDNi content naming mechanism must guarantee the uniqueness of the content identification. Although URL is widely used for identifying network resource, it is not quite suitable for content identification in CDNi network where content de-duplication needs to be taken into consideration. Although the method of URL match is commonly used by many cache systems to detect the repetitive files with same name for avoiding content repetition, it will probably fail for CDNi case. This is due to the fact that different forwarding mechanisms are used in different CDNs involved. The user-originated requests are always snooped by the DPI (deep packet inspection) devices before transmitted to the original server, whereas the requests received by dCDN are always redirected by one or more uCDNs. Since there is no guarantee that the URLs will not be changed through the redirection

process, a new type of object needs to be defined to represent the uniqueness of content identifier.

For the CSP's the contents that are distributed into different interconnected CDNs, the related metadata objects may be somewhat different in many cases. For example, in Figure 2, when both CDN-A and CDN-B delegate the delivery of the same CSP's content, the content metadata such as (a) content description, (b) access policy, and (c) security policy may be not be exactly the same in both cases. Such metadata information is not suitable for content identification. Consequently, we need to define a new type of metadata object that helps uniquely identify the same content.

3.2. Ownership

CDNi content naming mechanism should embody the ownership of content identification. Typically, a CDN provider may have contracts with many CSPs for delivering their contents, as well as operate its own Content delivery network. However, it may happen that a lot of contents published by these CSPs may be very similar, and many of them may even be exactly the same. Therefore, the problem is whether these identical copies are from the same CSP or how the interconnected CDNs can verify that these contents are identical. (Note: Such copies are pointed by the same content identification only if they are from the same content source.) For a traditional (non-interconnected) CDN, there is no problem to distinguish them via its intra content naming mechanism. When a CDN interconnects with other CDNs, the condition becomes more complicated due to the lack of awareness of CSP's content when the CDN acts as a dCDN.

4. CDNi Content De-duplication Optimization Implementation

4.1. Constant URL

In general, URL-based mechanism can be used to implement content de-duplication in CDNi network. As referred in section 2, the URL description for a content may be different from uCDN to uCDN and is likely to be changed in the redirection process. An agreement to configure a specific URL between a pair of interconnected CDN is used in the draft [I-D.ietf-cdni-framework-01], however this method is not flexible enough for supporting de-duplication in complex CDNi network. So a feasible proposal is to specify a mechanism for CDNi network to guarantee that CSP's same contents cached in different uCDNs are identified by the same URL and that the URL is unchanged or at least the path part remains the same in the redirection process. The main problem of this mechanism is lack of resilience and we prefer an alternative mechanism as introduced in section 4.2 below.

4.2. Content Naming Mechanism

This section provides a detailed description of CDNi content naming mechanism using CDNi Metadata Protocol.

In general, CSP's content as well as its copies cached in interconnected CDNs are delivered to numerous End-Users. The draft [I-D.ietf-cdni-framework-01] assigns each copy an identifier in the form of an URL that is embedded with "CDN-Domain" which is used to distinguish whether a download request is from an end-user or from an uCDN. We use the term Resource Identifier to represent the storage pointing to the content copies in interconnected CDNs. However, unlike the usage in the draft [I-D.ietf-cdni-framework-01], in this document CDNi content naming mechanism specifies Resource Identifiers as the one which is only related to contents in uCDNs. Taking the example in section 2.3, we use Resource Identifier A to point to content originated through uCDN A.

Although the Resource Identifier is able to identify a content, the uniqueness of content identification can't be guaranteed, as Resource Identifier is used to identify the storage of the content in the uCDN and therefore will be changed during redirection processes between different uCDNs. In order to resolve it, we introduce the term Content Identifier that is assigned to associate with Resource Identifier to uniquely identify the content and is similar to the URN usage. Note that the Content Identifier MUST be globally unique. Figure 4 shows a metadata model that can be used for maintaining the relationship between the two types of Identifiers. Using this model, the dCDN is able to uniquely identify and route requests towards the same targeted content.

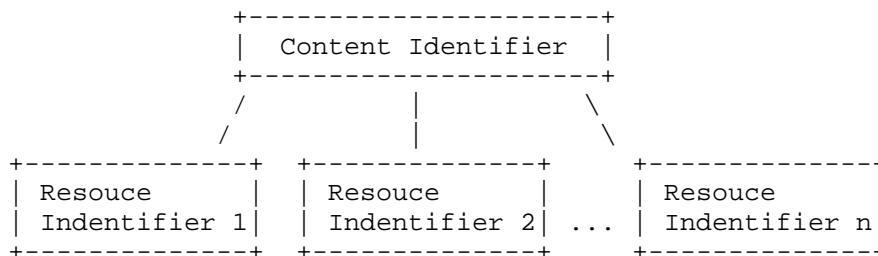


Figure 4 Metadata Model for Maintaining Relationship among Multi-IDs

Note: We need to develop an authoritative 'Entity' for creating and maintaining the Content Identifier.

4.3. Content ID

Actually, EIDR (Entertainment Identifier Registry), an industry non-profit organization, has already started the research and standardization of the globally unique content identifier and its generating mechanism. As stated in the whitepaper of [UNIVERSAL UNIQUE IDENTIFIERS IN MOVIE AND TELEVISION SUPPLY CHAIN MANAGEMENT], EIDR offers an inexpensive mechanism for uniquely identifying the complete range of audiovisual assets relevant to commerce including micro-assets such as clips and newer types of objects such as encodings. The EIDR data model can be readily extended to cover new and emerging objects and relationships as the industry evolves over time. EIDR naming system meets the requirements of coverage, flexibility, extensibility, scalability, cost-effectiveness, interoperability, etc.

The EIDR registry assigns a unique universal identifier for all registered assets. EIDR is an opaque ID with all information about the registered asset stored in the central registry. Its structure consists of a standard registry prefix, the unique suffix for each asset and a check digit. The suffix of an asset ID is of the form XXXX-XXXX-XXXX-XXXX-XXXX-C, where X is a hexadecimal digit and C is the ISO 7064 Mod 37, 36 check character.

Standard Prefix for EIDR Registry	Unique Suffix for each asset	check digit
10.5240/	XXXX-XXXX-XXXX-XXXX-XXXX	-C

Figure 5 Structure of Content ID

The content provider submits content items for registration to the registry system, along with core metadata and information. The system uses a sophisticated system to insure that the object submitted to the registry has not already been registered and then generates an EIDR for the content. All the above need to be done before the content is injected into CDNi system. After that, the content identifier, used as metadata of the content, is injected into CDNi system and transmitted between uCDN and dCDN via such interface as Control Interface, Metadata Interface.

The detail information of EIDR can be referred to in the whitepaper of [UNIVERSAL UNIQUE IDENTIFIERS IN MOVIE AND TELEVISION SUPPLY CHAIN MANAGEMENT].

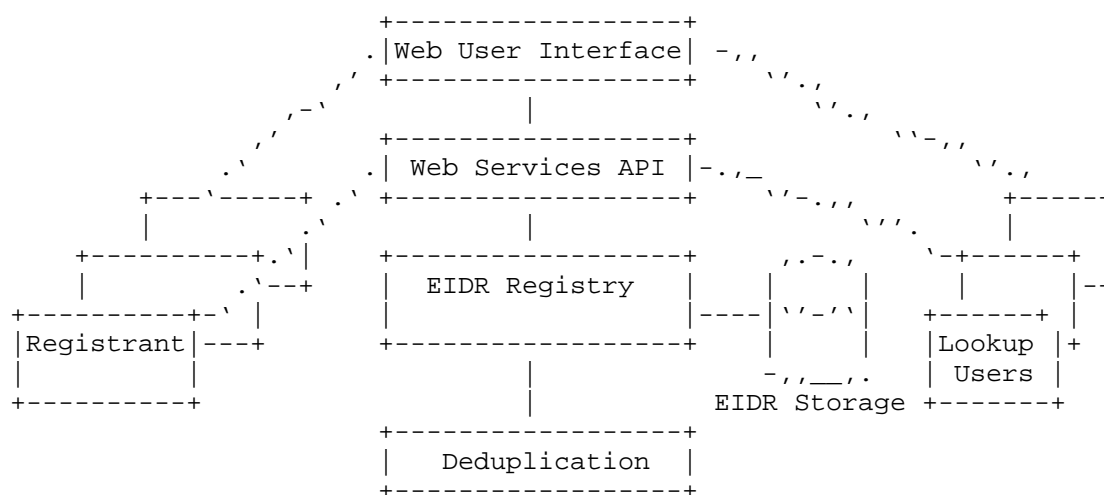


Figure 6 Entertainment Identifier Registry diagram

From the analysis of above section, the content identifier specified in EIDR is in line with the requirements of content deduplication proposed in this draft. In this document, it is suggested that we introduce content identifier format and a mechanism for generating it into CDNi to address content duplication issue.

4.4. Description of Content De-duplication

In this section the details of the solutions that use CDNi Content Naming mechanism for content de-duplication are discussed.

Content Identifier can be defined as a metadata object. The metadata object can be distributed with/without the actual content item from uCDN to dCDN for storage in the dCDN, if the uCDN wants to pre-position the content item to the dCDN before the actual user request. The content Identifier metadata object binds with the Resource Identifier to identify the storage of the content in the uCDN and forms a content identification model for the content item. By using this content identification model, an interconnected CDN is able to detect content repetition. The content status must be synchronously updated by the interconnected CDN. According to content status, the interconnected CDN can determine whether the resource copy is cached or not.

We present several procedures for optimized implementation for preventing CDNi content repetition.

4.4.1. Pre-Positioned Content Acquisition

The following flow illustrates how the two uCDNs successively pre-position the same content in the dCDN. In this flow, the content to be pre-positioned in the dCDN is identified by different Resource Identifiers corresponding to uCDN A and uCDN B.

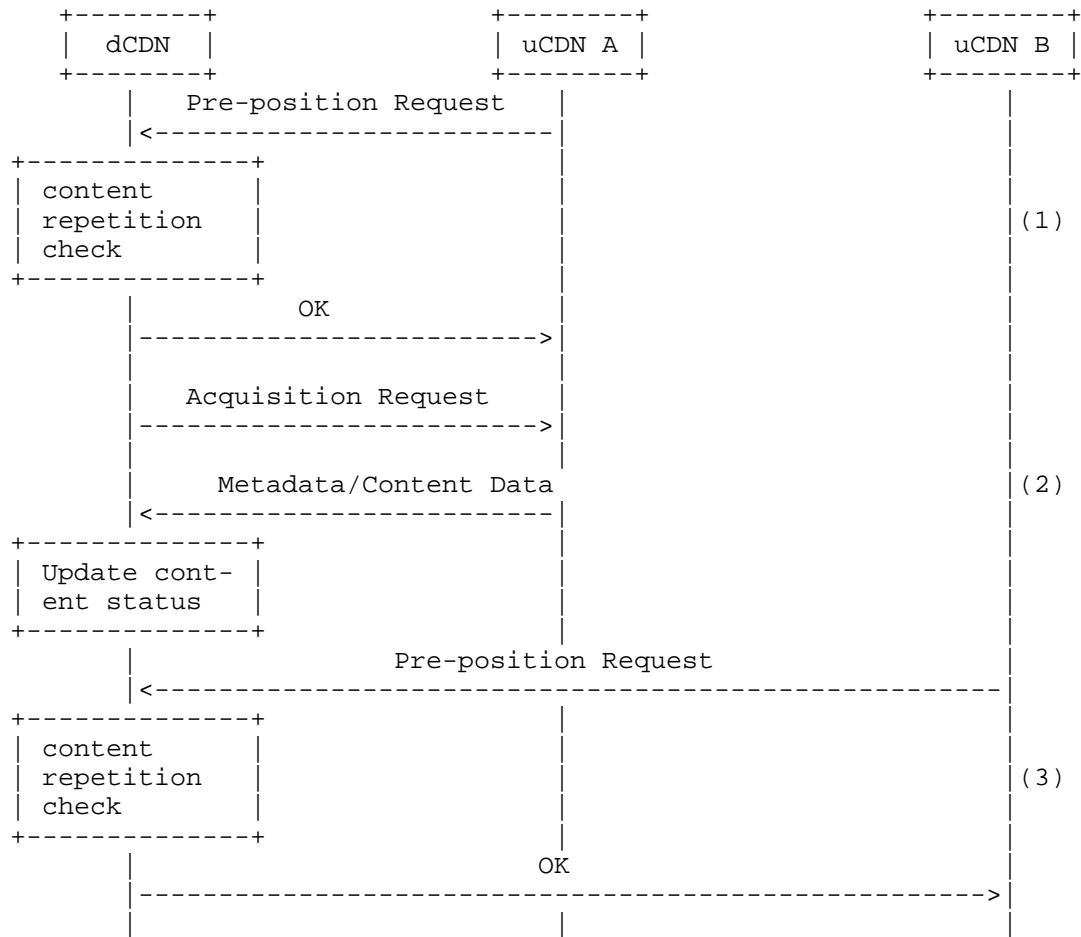


Figure 7 Acquisition of Pre-Positioned Content

The steps that are illustrated in the figure are as follows:

1. The uCDN A requests that the dCDN pre-positions a particular content item identified by its Resource Identifier and Content

Identifier. This message is sent via Trigger Interface. On reception of this message, the dCDN checks whether the same content item is cached by looking up its stored content identification model with the Content Identifier. If the metadata does not exist, the dCDN replies to uCDN A with an OK message to notify that no such copy has been cached and content pre-position is required.

2. The dCDN acquires metadata of the content or the content itself from uCDN A. Once the content is pre-positioned, dCDN updates the content status and maintains the binding between Resource Identifier and Content Identifier metadata.

3. The uCDN B requests that dCDN pre-positions the same content item identified by its Resource Identifier and Content Identifier. On reception of this message, the dCDN looks up its stored content identification model with the Content Identifier. As such metadata exists, dCDN determines that the content is already cached. Then, dCDN replies to uCDN A with an OK message to notify that the same copy has already been cached and the pre-position request should be cancelled. The dCDN locally binds the new Resource Identifier provided by uCDN B with the Content Identifier.

4.4.2. Dynamic Content Acquisition

The following flows illustrates how the dCDN performs content de-duplication in cases of a cache miss and a cache hit without content pre-positioning.

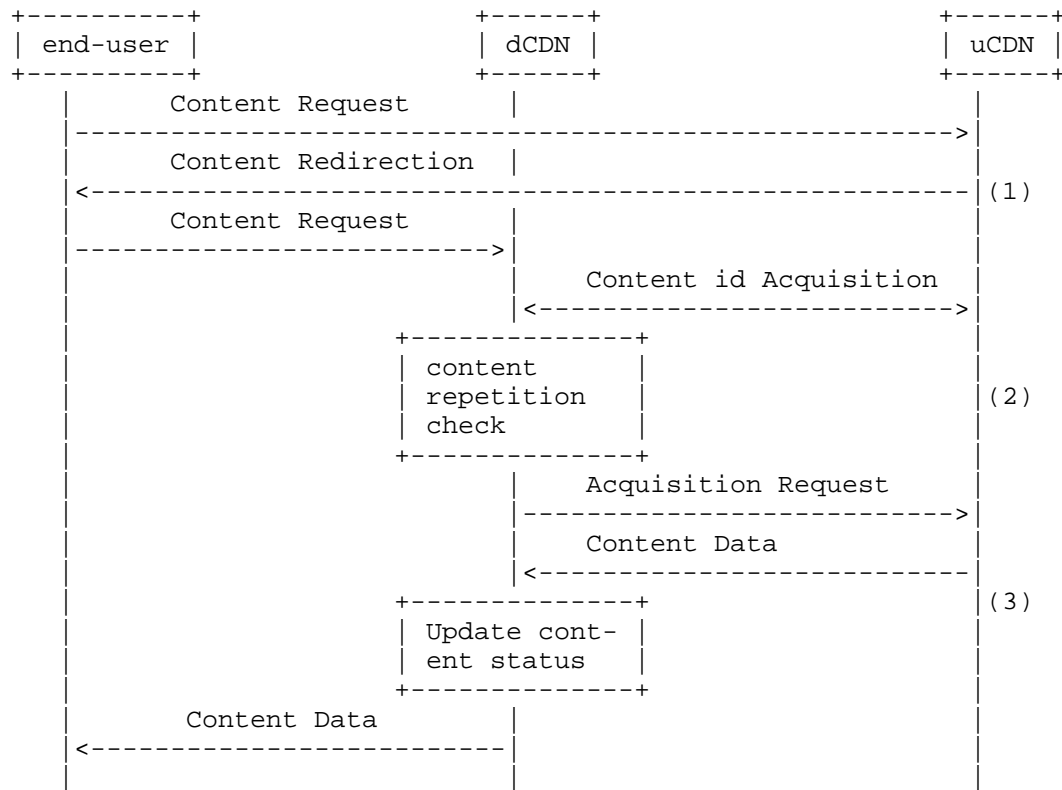


Figure 8 Dynamic Content Acquisition (cache miss case)

The steps that are illustrated in the figure are as follows:

1. A content request originated from an end-user is received by uCDN. The uCDN processes the request and recognizes that the end-user is best served by a dCDN. So uCDN redirects the request to the dCDN by sending redirection response to the end-user who then requests the content from the dCDN. The uCDN encapsulates Resource Identifier of the requested content item in the redirection response.
2. On reception of this request, the dCDN fetches the corresponding Content Identifier from the uCDN by using the Resource Identifier pointing to the requested resource. The dCDN checks whether the same content item is cached by looking up its stored content identification model with the Content Identifier. If such metadata does not exist, the case of a cache miss is determined by the dCDN, and therefore content needs to be downloaded from the uCDN before delivered to the end-user.

3. The dCDN acquires the requested content from uCDN A. Once the content is cached, dCDN updates the content status and maintains the binding relation between Resource Identifier and the Content Identifier metadata. The dCDN then delivers content data to the end-user.

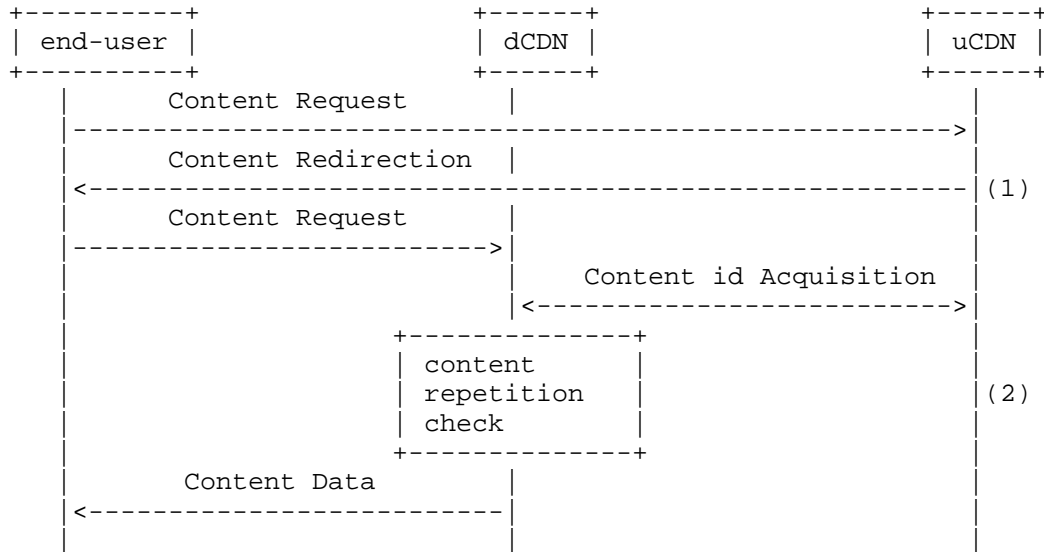


Figure 9 Dynamic Content Acquisition (cache hit case)

The steps that are illustrated in the figure are as follows:

Steps 1 and 2 are exactly the same as steps 1 and 2 of Figure 6, except that in this figure, dCDN determines the case of a cache hit according to the existence of such record in the corresponding Content Identifier metadata.

This flow differs from that in Figure 6 only in terms of not triggering dynamic content acquisition (step 3), since the content has already been cached by dCDN.

4.4.3. Content Purge and Invalidate

In general, the dCDN would assign separate location for each of its uCDN to store triggers. However, when it comes to complex CDNi deployment as discussed in this draft, dCDN is likely to receive multiple trigger operations coming from different uCDNs on the same content. If one of the uCDNs requests to invalidate certain content,

after receiving the Invalidated Trigger, dCDN will first identify the content using the Content Identifier and mark Invalid in the Trigger Status Resource corresponding to the requesting uCDN. By doing so, this content is unavailable to this uCDN before it is re-validated. The access to this content by other uCDNs will not be impacted.

If one of the uCDNs requests to purge certain content, after receiving the Purge Trigger, dCDN will identify the content using the Content Identifier and mark Invalid in the Trigger Status Resource corresponding to the requesting uCDN. By doing this, this uCDN is not able to make any further operation on this content, while the content itself is not deleted from the cache of dCDN. The access to this content by other uCDNs will not be impacted. Only when all the uCDNs binded to this content request to purge the same content and dCDN accepts all these requests will the content be purged from dCDN.

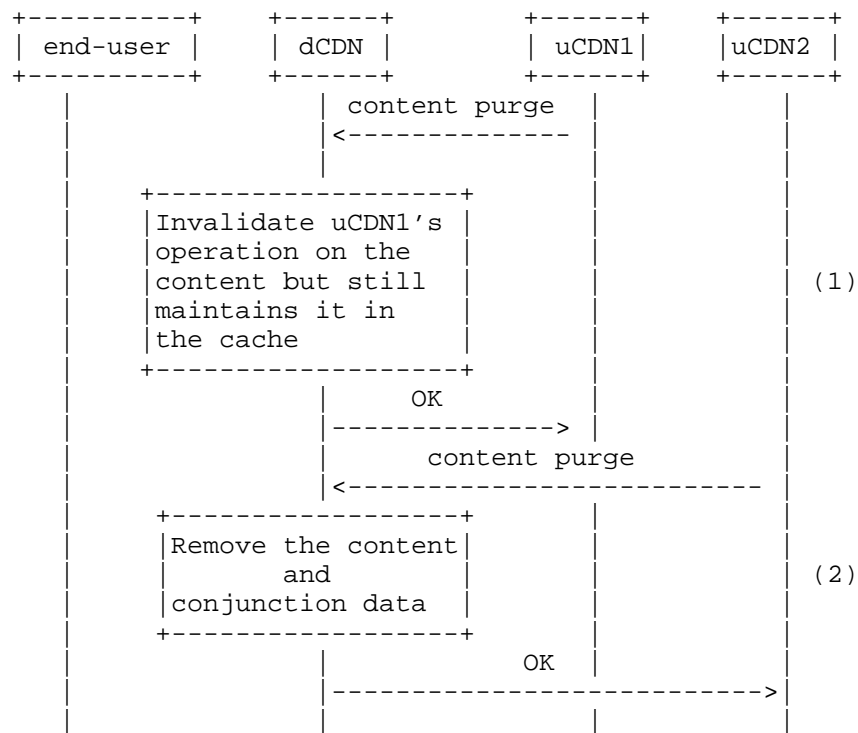


Figure 10 Removal of Content

A premise is that the content copy to be purged has already been cached in the dCDN from the uCDN. The Content Identifier of the

content is linked with two Resource IDs from uCDN1 and uCDN2 respectively. The steps illustrated in the figure are as follows:

1. The uCDN1 requests the dCDN to remove some content resource identified by the Content Identifier due to the deployment policy or expiration of content's life-time. As not only uCDN1 has been binded to this content, dCDN then only invalidates uCDN's operation on the requested content but still maintains it in the cache so that other uCDNs (e.g. uCDN2) binded to this content can still operate on this content. It then replies an OK response to uCDN1.

2. If uCDN2 also requests dCDN to remove the same content, dCDN will remove the content and all the conjunction metadata. It then replies an OK response to uCDN2.

5. Security Considerations

To be discussed/aded later.

6. IANA Considerations

This document has no IANA Considerations.

7. Acknowledgments

The authors would like to thank Francois Le Faucheur, Kevin Ma, Theodore Zahariadis, Ben Niven-Jenkins, Ram Krishnan, and Marcin Pilarski for valuable inputs, suggestions, and discussions.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2. Informative References

[Data Deduplication Techniques]
Ao, L., Shu, JW., and MQ. Li, "Data Deduplication Techniques", May 2010.

[EIDR WHITEPAPER]
EIDR, EIDR., "UNIVERSAL UNIQUE IDENTIFIERS IN MOVIE AND

TELEVISION SUPPLY CHAIN MANAGEMENT", October 2010.

[I-D.ietf-cdni-framework]

Peterson, L. and B. Davie, "Framework for CDN Interconnection", February 2013.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", December 2012.

[I-D.murray-cdni-triggers]

Murray, R., Niven-Jenkins, B., and Velocix, "CDN Interconnect Triggers", March 2013.

[I-D.narten-iana-considerations-rfc2434bis]

Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

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

Authors' Addresses

WeiYi Jin
ZTE Corporation
Nanjing, 210012
China

Phone: +86 025-52871364
Email: jin.weiya@zte.com.cn

Mian Li
ZTE Corporation
Nanjing, 210012
China

Phone: +86 025-88014641
Email: li.mian@zte.com.cn

Bhumip Khasnabish
ZTE Corporation
New Jersey, 07960
USA

Phone: +001-781-752-8003
Email: bhumip.khasnabish@zteusa.com, vumipl@gmail.com

CDNI
Internet Draft
Intended status: Informational
Expires: January 30 2013

R. Krishnan
Brocade Communications
M. Li
B. Khasnabish
ZTE Corporation
C. Ge
China Telecom
September 26, 2012

Best practices and Requirements for delivering Long Tail personalized
content delivery over CDN Interconnections

draft-krishnan-cdni-long-tail-04.txt

Status of this Memo

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

This Internet-Draft is submitted in full conformance with the
provisions of BCP 78 and BCP 79. This document may not be modified,
and derivative works of it may not be created, and it may not be
published except as an Internet-Draft.

This Internet-Draft is submitted in full conformance with the
provisions of BCP 78 and BCP 79. This document may not be modified,
and derivative works of it may not be created, except to publish it
as an RFC and to translate it into languages other than English.

This document may contain material from IETF Documents or IETF
Contributions published or made publicly available before November
10, 2008. The person(s) controlling the copyright in some of this
material may not have granted the IETF Trust the right to allow
modifications of such material outside the IETF Standards Process.
Without obtaining an adequate license from the person(s) controlling
the copyright in such materials, this document may not be modified
outside the IETF Standards Process, and derivative works of it may
not be created outside the IETF Standards Process, except to format
it for publication as an RFC or to translate it into languages other
than English.

Internet-Drafts are working documents of the Internet Engineering
Task Force (IETF), its areas, and its working groups. Note that
other groups may also distribute working documents as Internet-
Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on March 26, 2009.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

The content desire of users is evolving from most popular to long tail personalized content. This document discusses the best practices and requirements for delivering long tail personalized content in CDN Interconnection scenarios.

Table of Contents

1. Introduction.....	3
2. Conventions used in this document.....	3

3. No Caching in CDNs.....	4
4. Benefits of HTTP Adaptive Streaming.....	6
5. Other techniques for delivering long tail personalized content.	7
6. Acknowledgements.....	8
7. References.....	8
7.1. Normative References.....	8
7.2. Informative References.....	8

1. Introduction

Typically, the CDNI interface between CDNs is a long-haul backbone network where bandwidth is premium. For user content requests from the downstream CDN (dCDN), a cache in the dCDN addresses the CDNI bandwidth challenge by being able to serve the content from the dCDN and avoiding accessing the content from the upstream CDN (uCDN). The cache has limited storage space/processing power and relies on the fact that the same piece of content is of interest to a lot of users.

Most popular content is of interest to a lot of users; examples are latest movies, latest catch-up episodes etc. A single copy of the content is delivered across CDNI to the cache; the content is delivered to multiple users from the cache. Thus, most popular content is very amenable to caching.

Long tail personalized content is of interest to only a few users; examples are documentaries, very old movies etc. Long tail personalized content is typically not shared by many users and caching of long tail personalized content could lead to cache thrashing issues. Thus, long tail personalized content is not amenable to caching.

This document discusses the best practices and requirements for delivering long tail personalized content in CDN Interconnection scenarios.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document reuses the terminology defined in:

[I-D.ietf-cdni-problem-statement-06],

[I-D.ietf-cdni-requirements-03],

[I-D.ietf-cdni-framework-00], and

[I-D.ietf-cdni-use-cases-08].

3. No Caching in CDNs

Long tail personalized content is typically not shared by many users and not amenable to caching. Avoiding caching in the CDNs has the following benefits 1) Better cache utilization 2) Avoid unnecessary HTTP redirection.

Each CDN has a local monitoring server which monitors the end user content usage in the CDN. By monitoring the content usage, each CDN determines whether or not the content should be cached locally in the CDN. Through the CDNI interface, each dCDN propagates this information to the uCDN(s). Thus, the uCDN(s) determine the dCDNs in which the content should be cached/not cached. This results in the following CDNI metadata interface requirement and request routing interface changes which are described in this draft.

An example interconnected CDN topology is depicted in Figure 1; CDN-A and CDN-B are uCDNs which have a relationship with the Content Service Provider(CSP). CDN-C, where the end users are connected, is a dCDN and has a local monitoring server.

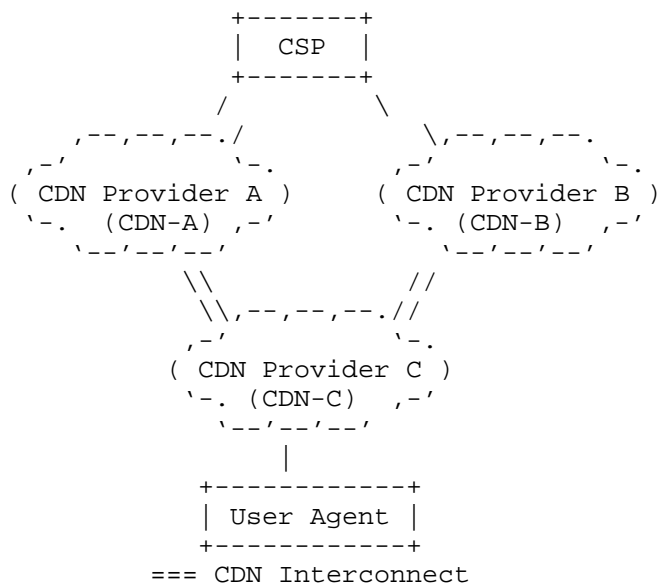


Figure 1: Interconnected CDNs with one dCDN

Metadata interface requirement

The CDNI Metadata Distribution interface shall provide indication by the dCDN to the uCDN whether the content should be cached or not cached in the dCDN. This information should be on a per URL basis. The default behavior would be to cache the content in the dCDN

Referring to the example in Fig. 2, Section 3 [I-D.ietf-cdni-framework]; it shows Operator A as the uCDN and Operator B as the dCDN, where the former has a relationship with a content provider and the latter being the best CDN to deliver content to the end-user. Referring to the HTTP example in Fig. 3, Section 3.2 [I-D.ietf-cdni-framework];

Request routing interface changes

Step 2: A Request Router for Operator A (which is the uCDN) processes the HTTP request. The HTTP URL metadata is looked up in a metadata database. For long tail personalized content, the metadata database lookup result indicates that the content should not be cached by the dCDN. The Request Router for Operator A recognizes that the end-user is best served by the uCDN without

any caching the in dCDN and returns a 302 redirect message with the URL of Operator A delivery node. The end-user proceeds to retrieve the data from Operator A delivery node. This is illustrated in Figure 2 below.

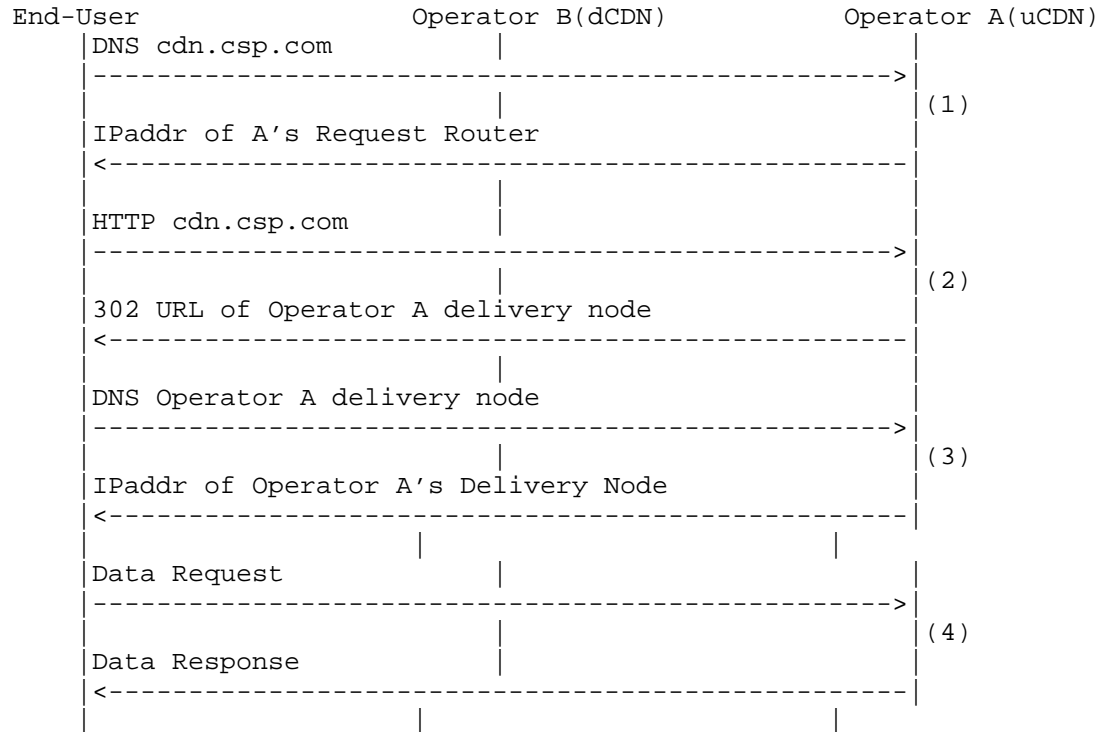


Figure 2: Request Routing interface for long tail personalized content

Logging and Auditing requirements

Work in progress

4. Benefits of HTTP Adaptive Streaming

As discussed before, long tail personalized content is not amenable to caching. Also, there is heavy asymmetric usage of the network between peak and quiet hours, where the peak hour load is much higher than the quiet hour load. These create unique bandwidth challenges across CDNI. HTTP Adaptive Streaming (HAS), which can adapt to

network congestion, is ideally suited for delivering long tail personalized content across interconnected CDNs.

5. Other techniques for delivering long tail personalized content

Approach 1

If the uCDN has a charging agreement with the dCDN that the dCDN pays fixed monthly money to uCDN (no matter how much traffic they exchange each month) and the CDN has enough storage capacity, the cache control of the long tail content is not that necessary, but let each CDN decide whether to cache the content or not locally. If the user request is redirected to dCDN but the dCDN does not cache the content, the dCDN can acquire the content from its uCDN.

Approach 2

If static control is desired for long tail content, the CSP can assign a second-level domain name for such kind of content, e.g. `nocache.example.com/contentID`, so that when this content is injected into CDNI system, CDN would determine whether to cache it or not according to this domain name.

Approach 3

So far, what has been discussed is streaming delivery of long tail personalized content. Caching in the end user device is another technique which can be used to address the bandwidth challenges created by streaming delivery of long tail personalized content over CDNI. This introduces a new model for long tail personalized content delivery. The various components of this model can be defined as 1)End user chooses the content to watch 2) The content is downloaded in the background and cached in the end user device 3)End user is notified of content availability. This model is typically applicable for long form content where the overhead in managing a background download is justifiable.

Caching in the end user device can have potential DRM issues which can be addressed using the following techniques 1) The content can be accessed by the end user only for playback 2) The content has a time expiry after which it destructs itself 3) In the case of end user device loss, the content destructs itself.

6. Acknowledgements

The authors would like to thank Francois Le Faucheur, Kevin Ma, Jin Weiyi and Ben Niven-Jenkins for their input.

7. References

7.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [2] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

7.2. Informative References

- [I-D.ietf-cdni-framework]L. Peterson et al., "Framework for CDN Interconnection", April 2012.
- [I-D.ietf-cdni-problem-statement]B. Niven-Jenkins et al., "Content Distribution Network Interconnection (CDNI) Problem Statement", May 2012.
- [I-D.ietf-cdni-requirements]K. Leung et al., "Content Distribution Network Interconnection (CDNI) Requirements", December 2011.
- [I-D.ietf-cdni-use-cases]Bertrand, G. et al., "Use Cases for Content Delivery Network Interconnection", June 2012.

Authors' Addresses

Ram Krishnan
Brocade Communications
San Jose, 95134, USA

Phone: +001-408-406-7890

Email: ramk@brocade.com

Mian Li

ZTE Corporation

Nanjing, 210012

China

Phone:

Email: li.mian@zte.com.cn

Bhumip Khasnabish

ZTE Corporation

New Jersey, 07960, USA

Phone: +001-781-752-8003

Email: bhumip.khasnabish@zteusa.com

Chen Ge

China Telecom

109 West Zhongshan Ave

Guangzhou, Tianhe District, China

Phone:

Email: cheng@gsta.com

CDNI
Internet Draft
Intended status: Informational
Expires: November 2013

R. Krishnan
Brocade Communications
M. Li
B. Khasnabish
ZTE USA
C. Ge
China Telecom
May 26, 2013

Best practices and Requirements for delivering Long Tail
personalized content delivery over CDN Interconnections

draft-krishnan-cdni-long-tail-05.txt

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on November, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

Abstract

The content desire of users is evolving from most popular to long tail personalized content. This document discusses the best practices and requirements for delivering long tail personalized content in CDN Interconnection scenarios.

Table of Contents

1. Introduction.....	2
2. Conventions used in this document.....	3
3. No Caching in CDNs.....	3
4. Benefits of HTTP Adaptive Streaming.....	6
5. Other techniques for delivering long tail personalized content.....	6
6. Acknowledgements.....	7
7. References.....	7
7.1. Normative References.....	7
7.2. Informative References.....	7

1. Introduction

Typically, the CDNI interface between CDNs is a long-haul backbone network where bandwidth is premium. For user content requests from the downstream CDN (dCDN), a cache in the dCDN addresses the CDNI bandwidth challenge by being able to serve the content from the dCDN and avoiding accessing the content from the upstream CDN (uCDN). The cache has limited storage space/processing power and relies on the fact that the same piece of content is of interest to a lot of users.

Most popular content is of interest to a lot of users; examples are latest movies, latest catch-up episodes etc. A single copy of the content is delivered across CDNI to the cache; the content is

delivered to multiple users from the cache. Thus, most popular content is very amenable to caching.

Long tail personalized content is of interest to only a few users; examples are documentaries, very old movies etc. Long tail personalized content is typically not shared by many users and caching of long tail personalized content could lead to cache thrashing issues. Thus, long tail personalized content is not amenable to caching.

This document discusses the best practices and requirements for delivering long tail personalized content in CDN Interconnection scenarios. This will be pursued further in the second phase of the CDNI work.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document reuses the terminology defined in:

[I-D.ietf-cdni-problem-statement-08],

[I-D.ietf-cdni-requirements-06],

[I-D.ietf-cdni-framework-03], and

[I-D.ietf-cdni-use-cases-10].

3. No Caching in CDNs

Long tail personalized content is typically not shared by many users and not amenable to caching. Avoiding caching in the CDNs has the following benefits 1) Better cache utilization 2) Avoid unnecessary HTTP redirection.

Each CDN has a local monitoring server which monitors the end user content usage in the CDN. By monitoring the content usage, each CDN determines whether or not the content should be cached locally in the CDN. Through the CDNI interface, each dCDN propagates this information to the uCDN(s). Thus, the uCDN(s) determine the dCDNs in which the content should be cached/not cached. This results in the following CDNI metadata interface requirement and request routing interface changes which are described in this draft.

An example interconnected CDN topology is depicted in Figure 1; CDN-A and CDN-B are uCDNs which have a relationship with the Content Service Provider(CSP). CDN-C, where the end users are connected, is a dCDN and has a local monitoring server.

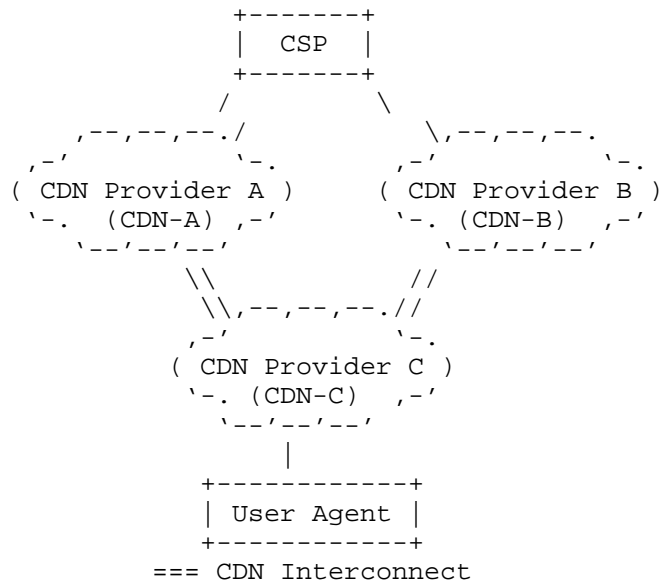


Figure 1: Interconnected CDNs with one dCDN

Metadata interface requirement

The CDNI Metadata Distribution interface shall provide indication by the dCDN to the uCDN whether the content should be cached or not cached in the dCDN. This information should be on a per URL basis. The default behavior would be to cache the content in the dCDN

Referring to the example in Fig. 2, Section 3 [I-D.ietf-cdni-framework]; it shows Operator A as the uCDN and Operator B as the dCDN, where the former has a relationship with a content provider and the latter being the best CDN to deliver content to the end-user. Referring to the HTTP example in Fig. 3, Section 3.2 [I-D.ietf-cdni-framework];

Request routing interface changes

Step 2: A Request Router for Operator A (which is the uCDN) processes the HTTP request. The HTTP URL metadata is looked up in a metadata database. For long tail personalized content, the metadata database lookup result indicates that the content should not be cached by the dCDN. The Request Router for Operator A recognizes that the end-user is best served by the uCDN without any caching in dCDN and returns a 302 redirect message with the URL of Operator A delivery node. The end-user proceeds to retrieve the data from Operator A delivery node. This is illustrated in Figure 2 below.

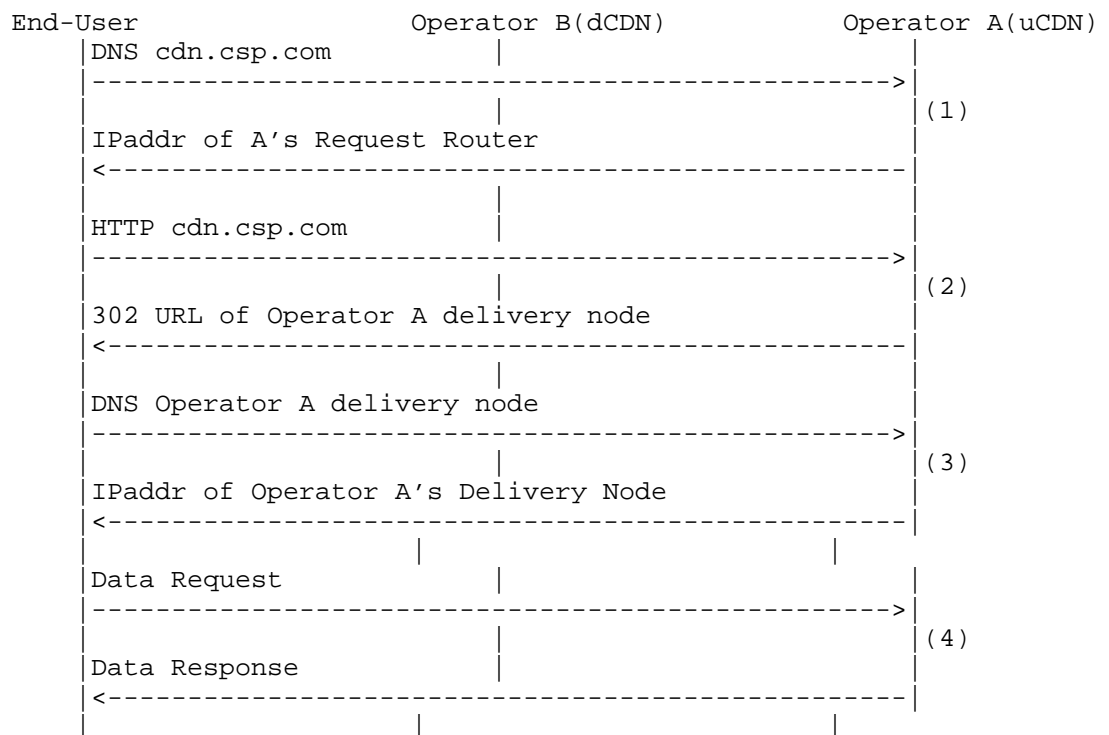


Figure 2: Request Routing interface for long tail personalized content

Logging and Auditing requirements

Work in progress

4. Benefits of HTTP Adaptive Streaming

As discussed before, long tail personalized content is not amenable to caching. Also, there is heavy asymmetric usage of the network between peak and quiet hours, where the peak hour load is much higher than the quiet hour load. These create unique bandwidth challenges across CDNI. HTTP Adaptive Streaming (HAS), which can adapt to network congestion, is ideally suited for delivering long tail personalized content across interconnected CDNs.

5. Other techniques for delivering long tail personalized content

Approach 1

If the uCDN has a charging agreement with the dCDN that the dCDN pays fixed monthly money to uCDN (no matter how much traffic they exchange each month) and the CDN has enough storage capacity, the cache control of the long tail content is not that necessary, but let each CDN decide whether to cache the content or not locally. If the user request is redirected to dCDN but the dCDN does not cache the content, the dCDN can acquire the content from its uCDN.

Approach 2

If static control is desired for long tail content, the CSP can assign a second-level domain name for such kind of content, e.g. `nocache.example.com/contentID`, so that when this content is injected into CDNI system, CDN would determine whether to cache it or not according to this domain name.

Approach 3

So far, what has been discussed is streaming delivery of long tail personalized content. Caching in the end user device is another technique which can be used to address the bandwidth challenges created by streaming delivery of long tail personalized content over CDNI. This introduces a new model for long tail personalized content delivery. The various components of this model can be defined as 1)End user chooses the content to watch 2) The content is downloaded in the background and cached in the end user device 3)End user is notified of content availability. This model is typically applicable for long form content where the overhead in managing a background download is justifiable.

Caching in the end user device can have potential DRM issues which can be addressed using the following techniques 1) The content can be accessed by the end user only for playback 2) The content has a

time expiry after which it destructs itself 3) In the case of end user device loss, the content destructs itself.

6. Acknowledgements

The authors would like to thank Francois Le Faucheur, Kevin Ma, Jin Weiyi and Ben Niven-Jenkins for their input.

7. References

7.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [2] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

7.2. Informative References

- [I-D.ietf-cdni-framework]L. Peterson et al., "Framework for CDN Interconnection", <http://www.ietf.org/id/draft-ietf-cdni-framework-03.txt>, February 2013.
- [I-D.ietf-cdni-problem-statement]B. Niven-Jenkins et al., "Content Distribution Network Interconnection (CDNI) Problem Statement", <http://tools.ietf.org/id/draft-ietf-cdni-problem-statement-08.txt>, June 2012, and <http://datatracker.ietf.org/doc/rfc6707/>, Sep. 2012.
- [I-D.ietf-cdni-requirements]K. Leung et al., "Content Distribution Network Interconnection (CDNI) Requirements", <http://www.ietf.org/id/draft-ietf-cdni-requirements-06.txt>, April 2013.
- [I-D.ietf-cdni-use-cases]Bertrand, G. et al., "Use Cases for Content Delivery Network Interconnection", <http://tools.ietf.org/id/draft-ietf-cdni-use-cases-10.txt>, August 2012, <http://datatracker.ietf.org/doc/rfc6770/>, November, 2012

Authors' Addresses

Ram Krishnan
Brocade Communications
San Jose, 95134, USA

Phone: +001-408-406-7890
Email: ramk@brocade.com

Mian Li
ZTE Corporation
Nanjing, 210012
China

Phone:
Email: li.mian@zte.com.cn

Bhumip Khasnabish
ZTE Corporation
New Jersey, 07960, USA

Phone: +001-781-752-8003
Email: bhumip.khasnabish@zteusa.com, vumipl@gmail.com

Chen Ge
China Telecom
109 West Zhongshan Ave
Guangzhou, Tianhe District, China

Phone:
Email: cheng@gsta.com

CDNI
Internet-Draft
Intended status: Standards Track
Expires: April 17, 2013

K. Leung
F. Le Faucheur
M. Caulfield
Cisco Systems
Oct 14, 2012

URI Signing for CDN Interconnection (CDNI)
draft-leung-cdni-uri-signing-01

Abstract

This document describes how the concept of URI signing supports the content access control requirements of CDNI and proposes a candidate URI signing scheme.

The proposed URI signing method specifies the information needed to be included in the URI and the algorithm used to authorize and to validate access request for the content referenced by the URI.

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 April 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. URI Signing Overview	4
2. Authorization Attributes in URI Signing	5
3. URI Signing and Validation	6
4. Considerations for CDNI Interfaces	9
4.1. CDNI Capabilities Advertisement	9
4.2. CDNI Metadata Interface	9
4.3. CDNI Logging Interface	10
5. URI Signing Operation	10
5.1. HTTP Redirection	10
5.2. DNS Redirection	13
6. HTTP Adaptive Bit Rate	16
7. IANA Considerations	16
8. Security Considerations	17
9. Acknowledgements	17
10. References	18
10.1. Normative References	18
10.2. Informative References	18
Authors' Addresses	18

1. Introduction

The overall problem space for CDN Interconnection is described in [RFC6707].

The CDNI Problem Statement [RFC6707], the CDN requirements document [I-D.ietf-cdni-requirements] and the CDNI Framework document [I-D.ietf-cdni-framework] discuss the need for the interconnected CDNs to be able to implement an access control mechanism that enforces the Content Service Provider (CSP) distribution policy.

Specifically, [I-D.ietf-cdni-framework] states:

"The CSP may also trust the CDN operator to perform actions such as ..., and to enforce per-request authorization performed by the CSP using techniques such as URI signing."

In particular, the following requirement is listed in [I-D.ietf-cdni-requirements]:

"META-17 [HIGH] The CDNI Metadata Distribution interface shall allow signaling of authorization checks and validation that are to be performed by the surrogate before delivery. For example, this could potentially include:

- * need to validate URI signed information (e.g. Expiry time, Client IP address)."

This document proposes a URI signing scheme that allows Surrogates in interconnected CDNs to enforce a per-request authorization performed by the CSP. Splitting the role of performing per-request authorization by CSP and the role of validation of this authorization by the CDN allows any arbitrary distribution policy to be enforced across CDNs without the need of CDNs to have any awareness of the actual CSP distribution policy.

1.1. Terminology

This document uses the terminology defined in [RFC6707].

This document also uses the terminology of [RFC2104] including the following terms (reproduced here for convenience):

- o MAC: message authentication code
- o HMAC: hash-based message authentication code (HMAC)

- o HMAC-SHA1: HMAC instantiation using SHA1 as the cryptographic hash function
- o HMAC-MD5: HMAC instantiation using MD5 as the cryptographic hash function

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

- o URI Signature: message digest that is computed with an algorithm that uses the key, the Original URI and request attributes as inputs to the hash function. This digest is conveyed inside the Signed URI.
- o Original URI: the URI before URI signing is applied.
- o Signed URI: the URI containing the Original URI, the attributes and the URI Signature.

1.2. URI Signing Overview

URI Signing is an authorization method for content delivery. This is based on embedding the URI with information that can be validated to ensure the request has legitimate access to the content. There are two parts: 1) attributes that convey authorization restrictions (e.g. source IP address and time period), and 2) message digest that confirms the integrity and authenticity of the URI provided by the URI creator. The authorization attributes can be anything agreed upon between the entity that creates the URI and the entity that validates the URI. A key is used by the HMAC algorithm of the URI signing function to generate the message digest (i.e. sign the URI). A key is also used by the HMAC algorithm of the URI signature validating function to validate the message digest (i.e. URI signature). The two functions may or may not use the same key.

Two types of keys can be used for URI Signing: 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 URI, respectively. A symmetric key is used for both functions. Regardless of the type of key, the entity that validates the URI has to obtain the key. There are very different requirements for key distribution with asymmetric keys and with symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent another party from getting access to the key, since it could then generate valid Signed URIs for unauthorized requests. Key distribution for asymmetric keys does not require confidentiality since public keys can typically be distributed openly (because they cannot be used for URI signing) and private keys are kept by the URI signing function.

URI Signing operates in the following way. After request authorisation, the CSP computed a Signed URO from the Original URI and provides the signed URI to the user out of band. The user request for the Signed URI is handled by the CDN which is responsible for validating the URI Signature before delivering the content.

2. Authorization Attributes in URI Signing

This section identifies the set of attributes that may be needed to enforce the CSP distribution policy. These attributes can therefore be covered by the URI Signature hash and can be embedded (by the signing function) in the as query component of the Signed URI (to enable subsequent signature validation by the signature validating function).

In order to provide flexibility in distribution policies to be enforced, the exact subset of attributes used for URI signature in a given request is a deployment decision. The defined keyword for each query string attribute is specified in parenthesis below.

- o Version (VER) - An integer used for identifying the version of URI signing method with its set of capabilities.
- o Expiry Time (ET) - Time in seconds when URI Signature expires since midnight 1/1/1970 UTC (i.e. UNIX epoch).
- o Client IP (CIP) - IP address of the client, in a dotted decimal format.
- o Key Owner (KO) - Identifier of the owner of the key used for URI signing, in an integer format.
- o Key ID (KN) - A number that is used as an index, within the set of keys of a given Key Owner, to the key used for URI signing, in an integer format.
- o Hash Function (HF) - A string used for identifying the hash function to compute the URI signature (e.g. "MD5", "SHA1").
- o Algorithm (ALG) - An integer used for identifying the algorithm to compute the URI signature.
- o Client ID (CID) - Identifier of the client such as IMSI, MSISDN, MEID, MAC address, etc.

The query string attributes are embedded within the Signed URI to be used for the content request in order to provide to the signature

validating function the information needed to enforce the distribution policy and to validate the URI Signature. Each of the attributes is further described below.

The Version attribute indicates which version of URI signing scheme is used (including which attributes and algorithms are supported). The present document specifies Version 0. More versions may be defined in the future.

The Expiry Time attribute ensures that the content authorization expires after a predetermined time. This limits the time window for content access and prevents replay of the request beyond the authorized time window.

The Client IP attribute is used to restrict content access to a particular End User, based on its client IP address for whom the content access was authorized.

The Key Owner and Key ID attributes are used to identify the key that is to be retrieved as input to the HMAC algorithm to compute the message digest for validating the signed URI.

The Hash function attribute indicates the HMAC hash function to be used for message digest computation.

The Algorithm indicates the specific algorithm for computation of the URI Signature. For example, this indicates whether the scheme component of the URI is to be covered by the signature computation or not.

The Client ID attribute is used to restrict content access to a particular user associated with this identifier. For example, it could be the information about the subscriber, device, or network access interface.

3. URI Signing and Validation

The keyword for embedding the actual URI Signature in the URI query string is "US".

The following steps are taken for signing a URI for the algorithms defined in this document. Note that some steps may be skipped if the attribute is not needed to enforce the distribution policy. The entire URI (i.e. scheme, authority, path, query, and fragment as defined in URI Generic Syntax [RFC3986]) is protected by the URI signature when the algorithm (i.e. "ALG") is set to 1. The scheme is removed from the URI when the algorithm is set to 2. This allows

the URI signature to be validated correctly in the case when a client performs a fallback to HTTP for a content referenced by an URI with RTSP scheme.

1. Check if the Original URI already contains a query string. If not, append a "?" character. If yes, append an "&" character.
2. Append the string "VER=0". This represents the version of URI Signing specified in this document.
3. Append the string "&ET=".
4. Get the current time in seconds since epoch (as an integer). Add the validity time in seconds as an integer.
5. Append this integer.
6. Append the string "&CIP=".
7. Append the client's IP address in dotted decimal format.
8. Append the string "&KO=".
9. Append the numeric value of the key owner corresponding to the key being used.
10. Append the string "&KN=".
11. Append the key ID number corresponding to the key being used.
12. Append the string "&HF=".
13. Append the string for the type of hash function.
14. Append the string "&ALG=".
15. Append the integer for the type of algorithm. If algorithm is "1", no additional logic needed by default. If algorithm is "2", remove the scheme part of the URI.
16. Append the string "&US=".
17. Store this as the message on which to compute the hash-based message authentication code (e.g. `http://example.com/content.mov?VER=0&ET=1209422976&CIP=171.71.50.123&KO=1&KN=2&HF=1&ALG=1&US=`).

18. For symmetric key, compute the message digest (i.e. URI signature) using the algorithm with key and message as inputs to the hash function. For asymmetric keys, after the message digest computation (as described previously only using the public key), use the public key again to encrypt the message digest.
19. Convert the message digest to its equivalent human readable hexadecimal value (e.g. f08b56f46075813e44b2d4888628a471).
20. Append this hexadecimal value to the previously created message. This is the complete Signed URI.

The following steps are taken for validating a Signed URI. Note that some steps are to be skipped if the corresponding attribute is not embedded in the Signed URI. The absence of a given attribute indicates enforcement of its purpose is not necessary in the distribution policy.

1. Check if the Signed URI contains a query string. If not, it is not a Signed URI. If the CDNI Metadata for the corresponding content indicate that access control is to be enforced via URI Signing, then the request is denied.
2. Extract the value from "US=" part of URI. This value is the URI signature.
3. Extract the values from "KO=" and "KN=" part of URI. Use these values to locate the key value and also key type (i.e. asymmetric or symmetric)
4. Extract the value from "HF=" part of URI. The value is the type of hash function.
5. Extract the value from "ALG=" part of URI. The value is the type of algorithm.
6. Store URI excluding the part after "US=" as the message on which to compute the hash-based message authentication code.
7. If the extracted algorithm value is "1", keep message without change. If algorithm value is "2", remove the scheme part of the URI in the message.
8. Compute the message digest (i.e. URI signature) using the algorithm with key and message as inputs to the hash function (based on the extracted hash function value).

9. For symmetric key, compare this computed digest with the received URI Signature. For asymmetric keys, decrypt the URI Signature with the public key. Then compare the computed digest with the decrypted URI Signature. If the comparison is not a match, the request is denied. Otherwise, continue with next step. Note that the request is denied if any of the following validations failed.
10. Validate that the request came from the same IP address as indicated in the "CIP=".
11. Validate that the request arrived before expiration time as indicated in the "ET=" based on the current time.

4. Considerations for CDNI Interfaces

The CDNI Interfaces need enhancements to support URI Signing. A Downstream CDN that supports URI Signing needs to be able to advertise this capability to the Upstream CDN. The Upstream CDN selects a Downstream CDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the Upstream CDN need to be able to distribute via the CDNI Metadata interface the information necessary to allow the Downstream CDN to validate a Signed URI . Events that pertain to URI Signing (e.g. request denial or delivery after access authorization) need to be included in the logs communicated through the CDNI Logging interface.

4.1. CDNI Capabilities Advertisement

The Downstream CDN advertises its capability to support URI Signing via the CDNI Request Routing/Footprint & Capabilities Advertisement interface. The supported version of URI Signing needs to be included. TBD: to be taken into account by Footprint & Capabilities design team working on this area.

- o URI Signing support and its version

4.2. CDNI Metadata Interface

The following CDNI metadata are specified for URI Signing. Note that the Key Owner and Key ID information are not needed if only one key is provided by CSP or Upstream CDN for the content or set of contents covered by the CDNI metadata. Also, the CDNI metadata for HMAC algorithm is not needed when the Algorithm attribute is embedded in the signed URI. TBD: CDNI Metadata Interface is work in progress.

- o Content access control indication.
- o Type of access control. Specifically, access to content is subject to URI Signing
- o Key value along with its key index (i.e. Key Owner and Key ID) and type (asymmetric or symmetric) used for validating URI signature
- o List of Downstream CDNs authorized for key distribution (i.e. trust relationship between CSP and CDNs) [Editor's Note: is this needed?]
- o Algorithm for HMAC to be used for validation.

4.3. CDNI Logging Interface

The Downstream CDN reports that enforcement of the access control was applied to the request for content delivery. TBD: CDNI Logging interface is work in progress.

- o URI signature validation events (e.g. invalid client IP address, expired signed URI, incorrect URI signature, successful validation)
- o Delivery log with confirmation of access control enforcement (i.e. Delivery CDN enforced URI Signing before content delivery)

5. URI Signing Operation

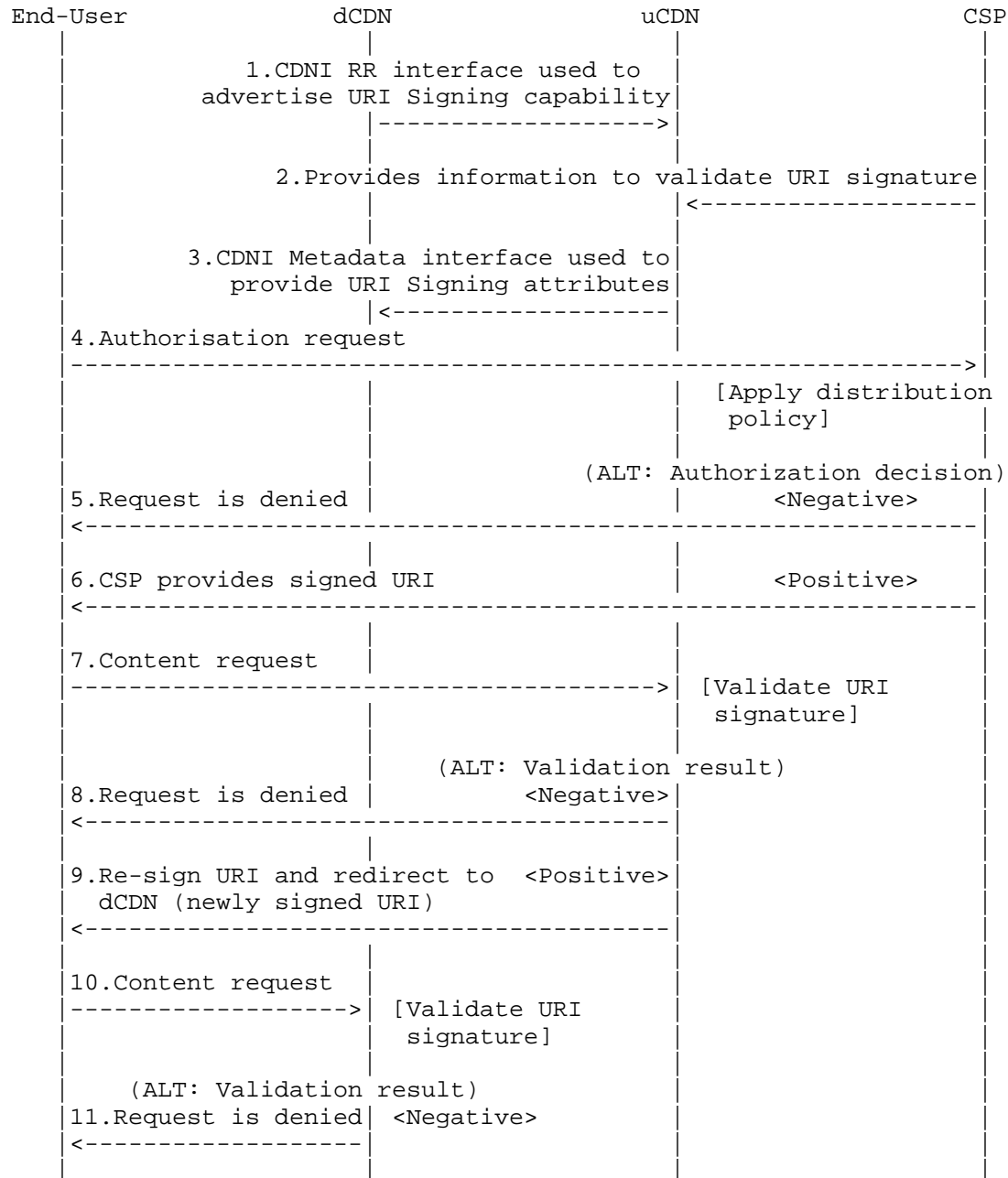
URI Signing supports both the HTTP-based and DNS-based request routing. HMAC [RFC2104] defines a hash-based message authentication code allowing two parties that share a symmetric key or asymmetric keys to establish the integrity and authenticity of a set of information (e.g. a message) through a cryptographic hash function.

5.1. HTTP Redirection

For HTTP-based request routing, HMAC is applied to a set of information that is unique to a given end user content request using key information that is specific to a pair of adjacent CDNI hops (e.g. between the CSP and the Authoritative CDN, between the Authoritative CDN and a Downstream CDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI signing scheme described below is based on the following

steps (assuming HTTP redirection, iterative request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



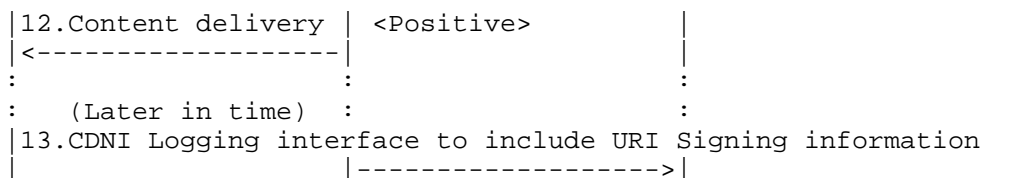


Figure 1: HTTP-based Request Routing with URI Signing

1. Using the CDNI Request Routing/Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.
2. CSP provides to the Authoritative CDN the information needed to validate URI signatures from that CSP. For example, this information may include a hashing function, algorithm, and a key value.
3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate URI signatures from the Authoritative CDN for the given CSP. For example, this information may include a hashing algorithm and private key corresponding to the trust relationship between the Authoritative CDN and the Downstream CDN.
4. On receipt of a given authorisation request on the CSP portal, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy.
5. If the authorization decision is negative, the CSP rejects the request.
6. If the authorization decision is positive, the CSP computes a Signed URI that is based on unique parameters of that request and conveys it to the end user as the URI to use to request the content.
7. On receipt of the corresponding content request, the authoritative CDN validates the URI Signature in the URI using the information provided by the CSP.
8. If the validation is negative, the authoritative CDN rejects the request
9. If the validation is positive, the authoritative CDN computes a Signed URI that is based on unique parameters of that request

and provides to the end user as the URI to use to further request the content from the Downstream CDN

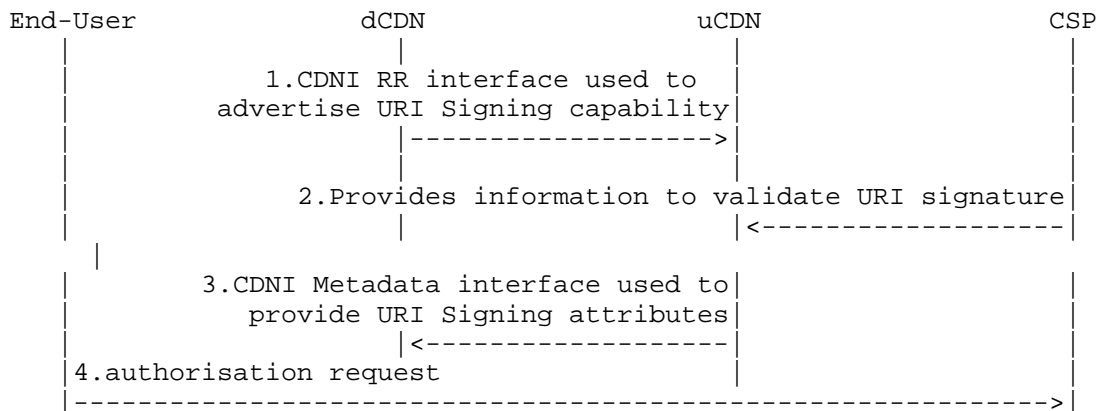
10. On receipt of the corresponding content request, the Downstream CDN validates the URI Signature in the Signed URI using the information provided by the Authoritative CDN in the CDNI Metadata
11. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.
12. If the validation is positive, the Downstream CDN serves the request and delivers the content.
13. At a later time, Downstream CDN reports logging events that includes URI signing information.

With HTTP-based request routing, URI Signing matches well the general chain of trust model of CDNI both with symmetric key and asymmetric keys because the key information only need to be specific to a pair of adjacent CDNI hops.

5.2. DNS Redirection

For DNS-based request routing, HMAC is applied to a set of information that is unique to a given end user content request using a secret key shared between CSP and the Delivery CDN. The Delivery CDN needs to obtain the key information to validate the Signed URL, which is computed by the CSP based on its distribution policy.

The URI signing scheme described below is based on the following steps (assuming iterative DNS request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



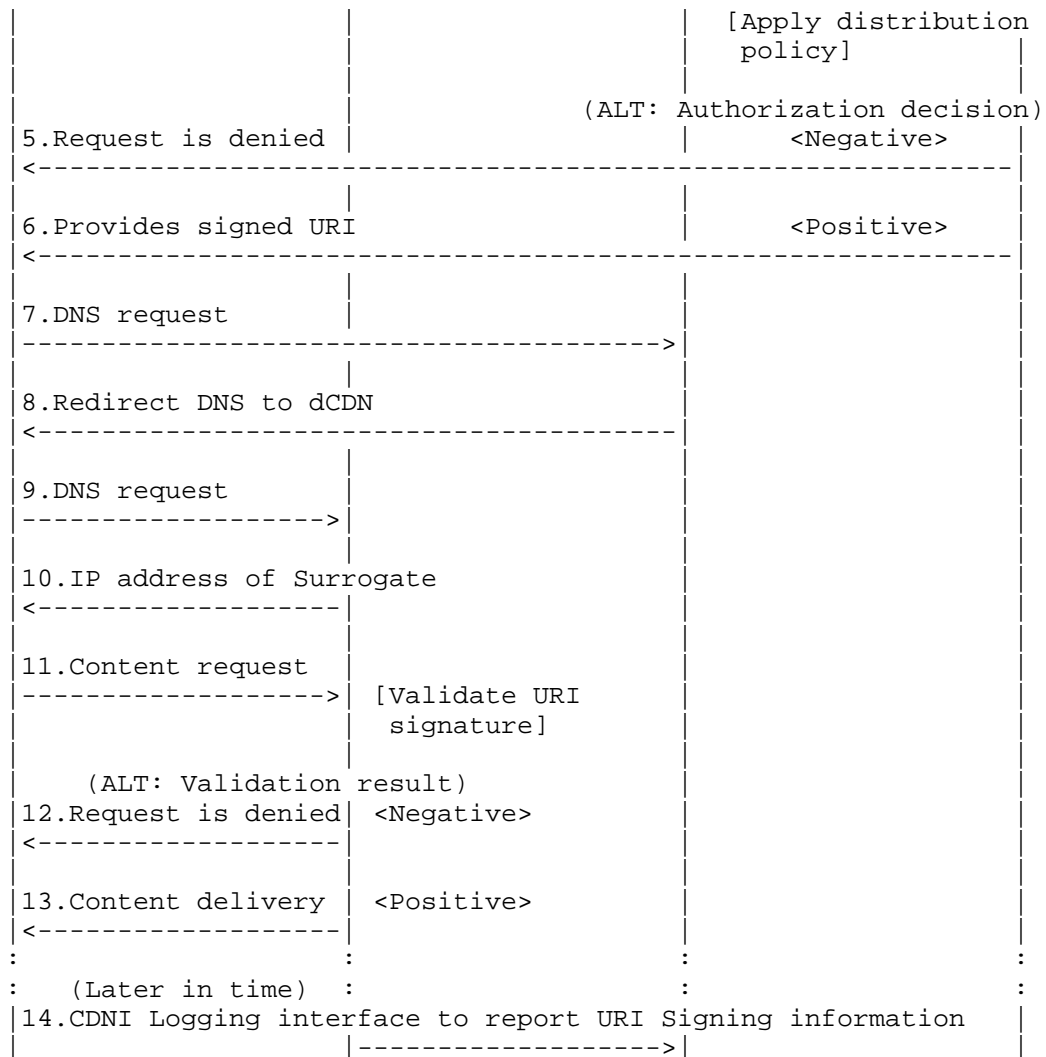


Figure 2: DNS-based Request Routing with URI Signing

1. Using the CDNI Request Routing interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.
2. CSP provides to the Authoritative CDN the information needed to validate cryptographic signatures from that CSP. For example, this information may include a hash function, algorithm, and a key.

3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate cryptographic signatures from the CSP (i.e. private key between CSP and participating CDNs). This requires a relationship between CSP and Downstream CDN. The CDNI metadata specifies CDNs with trust relationships according to the CSP. The set of Downstream CDNs is limited by this criteria.
4. On receipt of a given authorisation request on the CSP portal, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy.
5. If the authorization decision is negative, the CSP rejects the request
6. If the authorization decision is positive, the CSP computes a cryptographic signature that is based on unique parameters of that request and includes it in the URI provided to the end user to request the content.
7. End user sends DNS request to the authoritative CDN.
8. On receipt of the DNS request, the authoritative CDN redirects the request to the Downstream CDN.
9. End user sends DNS request to the Downstream CDN.
10. On receipt of the DNS request, the Downstream CDN responds with IP address of one of its Surrogates.
11. On receipt of the corresponding content request, the Downstream CDN validates the cryptographic signature in the URI using the information provided by the Authoritative CDN in the CDNI Metadata
12. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.
13. If the validation is positive, the Downstream CDN serves the request and delivers the content.
14. At a later time, Downstream CDN reports logging events that includes URI signing information.

With DNS-based request routing, URI Signing matches well the general chain of trust model of CDNI when used with asymmetric keys because the only key information that need to be distributed across multiple CDNI hops including non-adjacent hops is the public key, that is

generally not confidential.

With DNS-based request routing, URI Signing does match well the general chain of trust model of CDNI when used with symmetric keys because the symmetric key information needs to be distributed across multiple CDNI hops including non-adjacent hops. This raises a security concern for applicability of URI Signing with Symmetric keys in case of DNS-based inter-CDN request routing.

6. HTTP Adaptive Bit Rate

TBD - HTTP ABR calls for specific support by URI Signing ("flexible URI signing") as discussed in [I-D.brandenburg-cdni-has]. This will be added in a future version of this document.

7. IANA Considerations

This document requests IANA to create a new registry for CDNI URI Signing. The following query string attribute names (a.k.a. keywords) are assigned for the authorization attributes used in CDNI URI Signing. There is no intention to claim any query string attribute for URI beyond the CDNI URI Signing context. That means the entities that sign the URI or validate the URI signature comply to the keywords specified in the query string for the URI Signing function only when URI Signing is used and only in the context of CDNI.

- o US (URI signature)
- o VER (Version)
- o ET (Expiry time)
- o CIP (Client IP address)
- o KO (Key owner)
- o KN (Key ID)
- o HF (Hash Function)
- o ALG (Algorithm)
- o CID (Client ID)

This document requests IANA to create a registry for each of the

defined query string attribute and assign the following values for the authorization attribute:

VER: 0 (Base)

HF: "MD5", "SHA1", "SHA256"

ALG: 1 (Full URI), 2 (URI without scheme)

CID: "MAC:<value>", "IMSI:<value>", "MSISDN:<value>", "MEID:<value>", "NAI:<value>" (TBD)

8. Security Considerations

A symmetric key needs to be shared by the entity that produces the URI signature and the entity that validates the URI signature. In the case of DNS-based request routing, CSP that signed the URI may not have a relationship with the Downstream CDN that validates the signed URI. In this case, the Upstream CDN shall select only the Downstream CDN with a relationship with CSP. Otherwise, asymmetric keys should be used for DNS-based request routing. The Downstream CDN only needs to use the CSP's public key to validate the signed URI. Asymmetric keys method does not require a trust relationship between the two entities participating in URI Signing (i.e. signing function and signature validating function).

For HTTP-based request routing, the two entities participating in URI Signing are always the adjacent Upstream CDN and Downstream CDN because of the hop by hop nature of the redirection. Therefore, either symmetric key or asymmetric keys can be used because the adjacent Upstream CDN and Downstream CDN have a relationship.

The following security threats are identified (TBD):

- o Client IP address spoofing
- o Illegitimate client behind a NAT
- o Replay of request

9. Acknowledgements

TBD

10. References

10.1. Normative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-01 (work in progress), July 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.
- [I-D.ietf-cdni-use-cases]
Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-10 (work in progress), August 2012.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

10.2. Informative References

- [I-D.brandenburg-cdni-has]
Brandenburg, R., Deventer, O., Faucheur, F., and K. Leung, "Models for adaptive-streaming-aware CDN Interconnection", draft-brandenburg-cdni-has-03 (work in progress), July 2012.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

Authors' Addresses

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose 95134
USA

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

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

Matt Caulfield
Cisco Systems
1414 Massachusetts Avenue
Boxborough, MA 01719
USA

Phone: +1 978 936 9307
Email: mcaulfie@cisco.com

CDNI
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2014

K. Leung
F. Le Faucheur
Cisco Systems
B. Downey
Verizon Labs
R. van Brandenburg
TNO
S. Leibrand
Limelight Networks
March 4, 2014

URI Signing for CDN Interconnection (CDNI)
draft-leung-cdni-uri-signing-05

Abstract

This document describes how the concept of URI signing supports the content access control requirements of CDNI and proposes a URI signing scheme.

The proposed URI signing method specifies the information needed to be included in the URI and the algorithm used to authorize and to validate access requests for the content referenced by the URI. Some of the information may be accessed by the CDN via configuration or CDNI metadata.

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 September 5, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Background on URI Signing	4
1.3. CDNI URI Signing Overview	6
1.4. URI Signing in a non-CDNI context	8
2. Signed URI Information Elements	8
2.1. Enforcement Information Elements	10
2.2. Signature Computation Information Elements	11
2.3. URI Signature Information Elements	12
2.4. URI Signing Package Attribute	13
3. Creating the Signed URI	14
3.1. Calculating the URI Signature	14
3.2. Packaging the URI Signature	17
4. Validating a URI Signature	18
4.1. Information element validation	18
4.2. Signature validation	19
5. Relationship with CDNI Interfaces	21
5.1. CDNI Control Interface	22
5.2. CDNI Footprint & Capabilities Advertisement Interface	22
5.3. CDNI Request Routing Redirection Interface	22
5.4. CDNI Metadata Interface	23
5.5. CDNI Logging Interface	24
6. URI Signing Message Flow	24
6.1. HTTP Redirection	25
6.2. DNS Redirection	27
7. HTTP Adaptive Streaming	30
8. IANA Considerations	30
9. Security Considerations	31
10. Privacy	33
11. Acknowledgements	33
12. References	33
12.1. Normative References	33
12.2. Informative References	33
Authors' Addresses	34

1. Introduction

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of interconnected CDNs (CDNI). The primary goal of URI Signing is to make sure that only authorized User Agents (UAs) are able to access the content, with a Content Service Provider (CSP) being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as DRM, are more appropriate.

The overall problem space for CDN Interconnection (CDNI) is described in CDNI Problem Statement [RFC6707]. In this document, along with the CDNI Requirements [I-D.ietf-cdni-requirements] document and the CDNI Framework [I-D.ietf-cdni-framework] the need for interconnected CDNs to be able to implement an access control mechanism that enforces the CSP's distribution policy is described.

Specifically, CDNI Framework [I-D.ietf-cdni-framework] states:

"The CSP may also trust the CDN operator to perform actions such as ..., and to enforce per-request authorization performed by the CSP using techniques such as URI signing."

In particular, the following requirement is listed in CDNI Requirements [I-D.ietf-cdni-requirements]:

"MI-16 [HIGH] The CDNI Metadata Distribution interface shall allow signaling of authorization checks and validation that are to be performed by the surrogate before delivery. For example, this could potentially include:

- * need to validate URI signed information (e.g. Expiry time, Client IP address)."

This document proposes a URI Signing scheme that allows Surrogates in interconnected CDNs to enforce a per-request authorization performed by the CSP. Splitting the role of performing per-request authorization by CSP and the role of validation of this authorization by the CDN allows any arbitrary distribution policy to be enforced across CDNs without the need of CDNs to have any awareness of the actual CSP distribution policy.

1.1. Terminology

This document uses the terminology defined in CDNI Problem Statement [RFC6707].

This document also uses the terminology of Keyed-Hashing for Message Authentication (HMAC) [RFC2104] including the following terms (reproduced here for convenience):

- o MAC: message authentication code.
- o HMAC: Hash-based message authentication code (HMAC) is a specific construction for calculating a MAC involving a cryptographic hash function in combination with a secret key.
- o HMAC-SHA1: HMAC instantiation using SHA-1 as the cryptographic hash function.
- o HMAC-MD5: HMAC instantiation using MD5 as the cryptographic hash function.

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

- o URI Signature: Message digest or digital signature that is computed with an algorithm for protecting the URI.
- o Original URI: The URI before URI Signing is applied.
- o Signed URI: Any URI that contains a URI Signature.
- o Target CDN URI: Embedded URI created by the CSP to direct UA towards the Upstream CDN. The Target CDN URI can be signed by the CSP and verified by the Upstream CDN.
- o Redirection URI: URI created by the Upstream CDN to redirect UA towards the Downstream CDN. The Redirection URI can be signed by the Upstream CDN and verified by the Downstream CDN. In a cascaded CDNI scenario, there can be more than one Redirection URI.

1.2. Background on URI Signing

The next section provides an overview of how URI Signing works in a CDNI environment. As background information, URI Signing is first explained in terms of a single CDN delivering content on behalf of a CSP.

A CSP and CDN are assumed to have a trust relationship that enables the CSP to authorize access to a content item by including a set of attributes in the URI before redirecting a UA to the CDN. Using these attributes, it is possible for a CDN to check an incoming content request to see whether it was authorized by the CSP (e.g. based on the UA's IP address or a time window). Of course, the

attributes need to be added to the URI in a way that prevents a UA from changing the attributes, thereby leaving the CDN to think that the request was authorized by the CSP when in fact it wasn't. For this reason, a URI Signing mechanism includes in the URI a message digest or digital signature that allows a CDN to check the authenticity of the URI. The message digest or digital signature can be calculated based on a shared secret between the CSP and CDN or using CSP's asymmetric public/private key pair, respectively.

Figure 1, shown below, presents an overview of the URI Signing mechanism in the case of a CSP with a single CDN. When the UA browses for content on CSP's website (#1), it receives HTML web pages with embedded content URIs. Upon requesting these URIs, the CSP redirects to a CDN, creating a Target CDN URI (#2) (alternatively, the Target CDN URI itself is embedded in the HTML). The Target CDN URI is the Signed URI which may include the IP address of the UA and/or a time window and always contains the URI Signature which is generated by the CSP using the shared secret or a private key. Once the UA receives the response with the embedded URI, it sends a new HTTP request using the embedded URI to the CDN (#3). Upon receiving the request, the CDN checks to see if the Signed URI is authentic by verifying the URI signature. In addition, it checks whether the IP address of the HTTP request matches that in the Signed URI and if the time window is still valid. After these values are confirmed to be valid, the CDN delivers the content (#4).

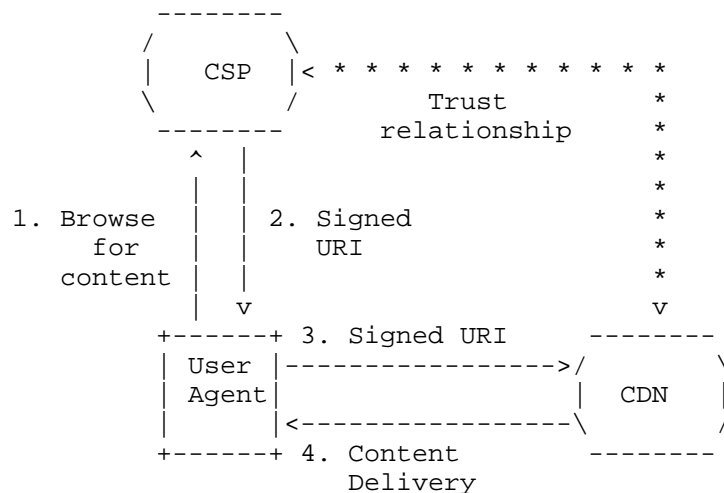


Figure 1: Figure 1: URI Signing in a CDN Environment

1.3. CDNI URI Signing Overview

In a CDNI environment, URI Signing operates the same way in the initial steps #1 and #2 but the later steps involve multiple CDNs in the process of delivering the content. The main difference from the single CDN case is a redirection step between the Upstream CDN and the Downstream CDN. In step #3, UA may send HTTP request or DNS request. Depending on whether HTTP-based or DNS-based request routing is used, the Upstream CDN responds by directing the UA towards the Downstream CDN using either a Redirection URI (which is a Signed URI generated by the Upstream CDN) or a DNS reply, respectively (#4). Once the UA receives the response, it sends the Redirection URI/Target CDN URI to the Downstream CDN (#5). The received URI is validated by the Downstream CDN before delivering the content (#6). This is depicted in the figure below. Note: The CDNI call flows are covered in Detailed URI Signing Operation (Section 6).

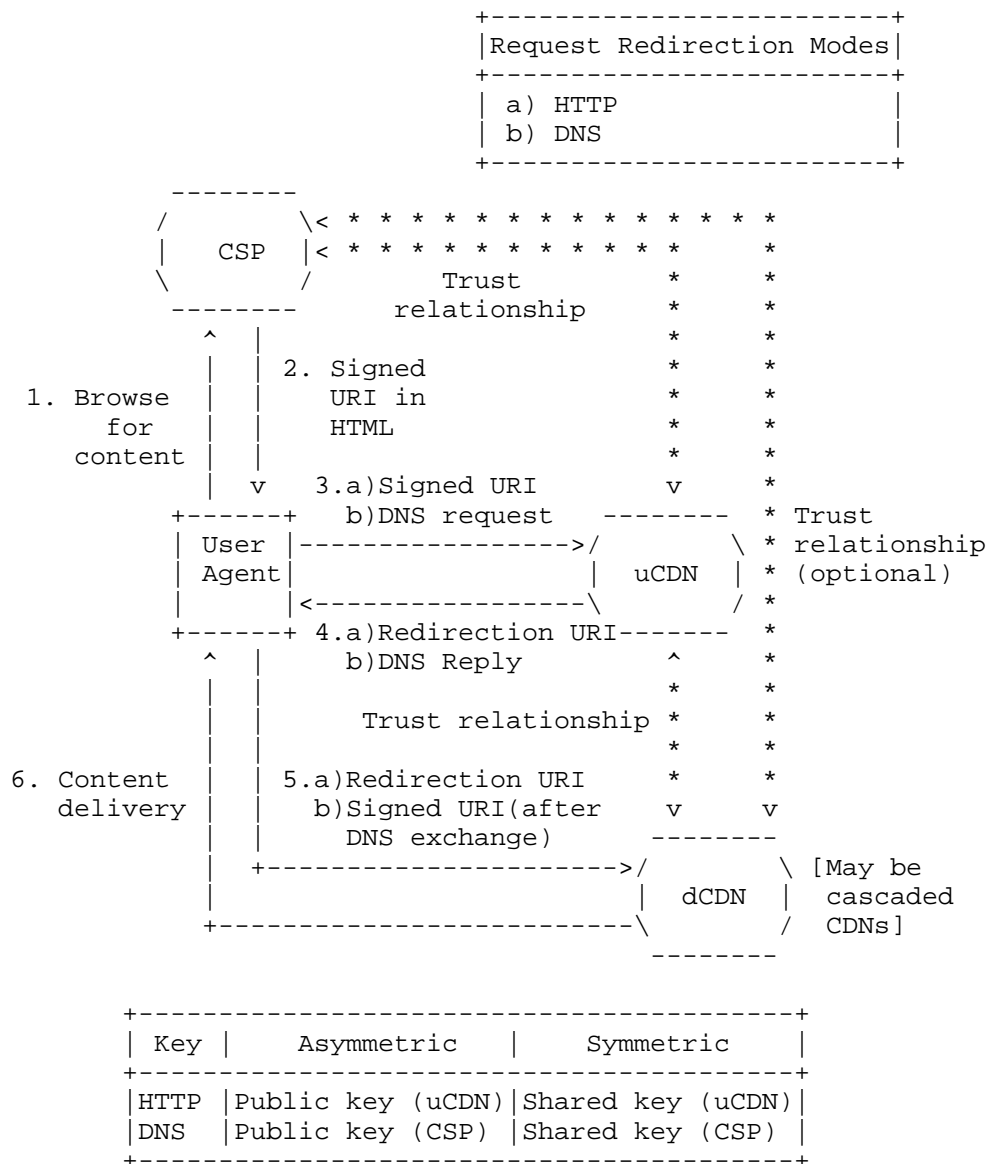


Figure 2: URI Signing in a CDNI Environment

The trust relationships between CSP, Upstream CDN, and Downstream CDN have direct implications for URI Signing. In the case shown in Figure 2, the CDN that the CSP has a trust relationship with is the Upstream CDN. The delivery of the content may be delegated to the Downstream CDN, which has a relationship with the Upstream CDN but

may have no relationship with the CSP.

In CDNI, there are two methods for request routing: DNS-based and HTTP-based. For DNS-based request routing, the Signed URI (i.e. Target CDN URI) provided by the CSP reaches the Downstream CDN directly. In the case where the Downstream CDN does not have a trust relationship with the CSP, this means that only an asymmetric public/private key method can be used for computing the URI Signature because the CSP and Downstream CDN are not able to exchange symmetric shared secret keys. Since the CSP is unlikely to have relationships with all the Downstream CDNs that are delegated to by the Upstream CDN, the CSP may choose to allow the Authoritative CDN to redistribute the shared key to a subset of their Downstream CDNs .

For HTTP-based request routing, the Signed URI (i.e. Target CDN URI) provided by the CSP reaches the Upstream CDN. After this URI has been verified to be correct by the Upstream CDN, the Upstream CDN creates and signs a new Redirection URI to redirect the UA to the Downstream CDN. Since this new URI also has a new URI Signature, this new signature can be based around the trust relationship between the Upstream CDN and Downstream CDN, and the relationship between the Downstream CDN and CSP is not relevant. Given the fact that such a relationship between Upstream CDN and Downstream CDN always exists, both asymmetric public/private keys and symmetric shared secret keys can be used for URI Signing. Note that the signed Redirection URI SHOULD maintain the same level of security as the original Signed URI.

1.4. URI Signing in a non-CDNI context

While the URI signing scheme defined in this document was primarily created for the purpose of allowing URI Signing in CDNI scenarios, e.g. between a uCDN and a dCDN or between a CSP and a dCDN, there is nothing in the defined URI Signing scheme that precludes it from being used in a non-CDNI context. As such, the described mechanism could be used in a single-CDN scenario such as shown in Figure 1 in Section 1.2, for example to allow a CSP that uses different CDNs to only have to implement a single URI Signing mechanism.

2. Signed URI Information Elements

The concept behind URI Signing is based on embedding in the Target CDN URI/Redirection URI a number of information elements that can be validated to ensure the UA has legitimate access to the content. These information elements are appended, in an encapsulated form, to the original URI.

For the purposes of the URI signing mechanism described in this document, three types of information elements may be embedded in the URI:

- o Enforcement Information Elements: Information Elements that are used to enforce a distribution policy defined by the CSP. Examples of enforcement attributes are IP address of the UA and time window.
- o Signature Computation Information Elements: Information Elements that are used by the CDN to verify the URI signature embedded in the received URI. In order to verify a URI Signature, the CDN requires some information elements that describe how the URI Signature was generated. Examples of Signature Computation Elements include the used HMACs hash function and/or the key identifier.
- o URI Signature Information Elements: The information elements that carry the actual message digest or digital signature representing the URI signature used for checking the integrity and authenticity of the URI. A typical Signed URI will only contain one embedded URI Signature Information Element.

In addition, the this document specifies the following URI attribute:

- o URI Signing Package Attribute: The URI attribute that encapsulates all the URI Signing information elements in an encoded format. Only this attribute is exposed in the Signed URI as a URI query parameter.

If the UA or another entity needs to add one or more attributes to the Signed URI for purposes other than URI Signing, those attributes MUST be appended after the URI Signing Packacke Attribute. Any attributes appended in such way after the URI Signature has been calculated are not validated for the purpose of content access authorization. Note that adding any such attributes to the Signed URI before the URI Signing Packacke Attribute will cause the URI Signing validation to fail.

Two types of keys can be used for URI Signing: asymmetric keys and symmetric keys. Asymmetric keys are based on a public/private key pair mechanism and always contain a private key only known to the entity signing the URI (either CSP or uCDN) and a public key for the verification of the Signed URI. With symmetric keys, the same key is used by both the signing entity for signing the URI as well as by the validating entity for validating the Signed URI. Regardless of the type of keys used, the validating entity has to obtain the key (either the public or the symmetric key). There are very different

requirements for key distribution (out of scope of this document) with asymmetric keys and with symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent another party from getting access to the key, since it could then generate valid Signed URIs for unauthorized requests. Key distribution for asymmetric keys does not require confidentiality since public keys can typically be distributed openly (because they cannot be used for URI signing) and private keys are kept by the URI signing function.

2.1. Enforcement Information Elements

This section identifies the set of information elements that may be needed to enforce the CSP distribution policy. New information elements may be introduced in the future to extend the capabilities of the distribution policy.

In order to provide flexibility in distribution policies to be enforced, the exact subset of information elements used in the URI Signature of a given request is a deployment decision. The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to enforce the distribution policy:

- o Expiry Time (ET) [optional] - Time when the Signed URI expires. This is represented as an integer denoting the number of seconds since midnight 1/1/1970 UTC (i.e. UNIX epoch). The request is rejected if the received time is later than this timestamp. Note: The time, including time zone, on the entities that generate and validate the signed URI need to be in sync (e.g. NTP is used).
- o Client IP (CIP) [optional] - IP address of the client for which this Signed URI is generated. This is represented in dotted decimal format for IPv4 or canonical text representation for IPv6 address [RFC5952] . The request is rejected if sourced from a client with a different IP address.

The Expiry Time Information Element ensures that the content authorization expires after a predetermined time. This limits the time window for content access and prevents replay of the request beyond the authorized time window.

The Client IP Information Element is used to restrict content access to a particular User Agent, based on its IP address for whom the content access was authorized.

Note: See the Security Considerations (Section 9) section on the

limitations of using an expiration time and client IP address for distribution policy enforcement.

2.2. Signature Computation Information Elements

This section identifies the set of information elements that may be needed to verify the URI (signature). New information elements may be introduced in the future if new URI signing algorithms are developed.

The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to validate the URI by recreating the URI Signature.

- o Version (VER) [optional] - An integer used for identifying the version of URI signing method. If this Information Element is not present in the URI Signing Package Attribute, the default version is 1.
- o Key ID (KID) [optional] - A string used for obtaining the key (e.g. database lookup, URI reference) which is needed to validate the URI signature.
- o Hash Function (HF) [optional] - A string used for identifying the hash function to compute the URI signature (e.g. "MD5", "SHA-1", "SHA-256", "SHA-3") with HMAC. If this Information Element is not present in the URI Signing Package Attribute, the default hash function is SHA-1.
- o Digital Signature Algorithm (DSA) [optional] - Algorithm used to calculate the Digital Signature (e.g. "RSA", "DSA", "EC-DSA"). If this Information Element is not present in the URI Signing Package Attribute, the default is EC-DSA.

The Version Information Element indicates which version of URI signing scheme is used (including which attributes and algorithms are supported). The present document specifies Version 1. If the Version attribute is not present in the Signed URI, then the version is obtained from the CDNI metadata, else it is considered to have been set to the default value. More versions may be defined in the future.

The Key ID Information Element is used to retrieve the key which is needed as input to the algorithm for validating the Signed URI. The method used for obtaining the actual key from the reference included in the Key ID Information Element is outside the scope of this

document.

The Hash Function Information Element indicates the hash function to be used for HMAC-based message digest computation. The Hash Function Information Element is used in combination with the Message Digest Information Element defined in section Section 2.3.

The Digital Signature Algorithm Information Element indicates the digital signature function to be in the case asymmetric keys are used. The Digital Signature Algorithm Information Element is used in combination with the Digital Signature Information Element defined in section Section 2.3.

2.3. URI Signature Information Elements

This section identifies the set of information elements that carry the URI Signature that is used for checking the integrity and authenticity of the URI.

The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to carry the actual URI Signature.

- o Message Digest (MD) [mandatory for symmetric key] - A string used for the message digest generated by the URI signing entity.
- o Digital Signature (DS) [mandatory for asymmetric keys] - A string used for the digital signature provided by the URI signing entity.

The Message Digest attribute contains the message digest used to validate the Signed URI when symmetric keys are used.

The Digital Signature attribute contains the digital signature used to verify the Signed URI when asymmetric keys are used.

In the case of symmetric key, HMAC algorithm is used for the following reasons: 1) Ability to use hash functions (i.e. no changes needed) with well understood cryptographic properties that perform well and for which code is freely and widely available, 2) Easy to replace the embedded hash function in case faster or more secure hash functions are found or required, 3) Original performance of the hash function is maintained without incurring a significant degradation, and 4) Simple way to use and handle keys.

In the case of asymmetric keys, Elliptic Curve Digital Signature Algorithm (EC DSA) - a variant of DSA - is used because of the

following reasons: 1) Key size is small while still offering good security, 2) Key is easy to store, and 3) Computation is faster than DSA or RSA.

2.4. URI Signing Package Attribute

The URI Signing Package Attribute is an encapsulation container for the URI Signing Information Elements defined in the previous sections. The URI Signing Information Elements are encoded and stored in this attribute. URI Signing Package Attribute is appended to the Original URI to create the Signed URI.

The primary advantage of the URI Signing Package Attribute is that it avoids having to expose the URI Signing Information Elements directly in the query string of the URI, thereby reducing the potential for a namespace collision space within the URI query string. A side benefit of the attribute is the obfuscation performed by the URI Signing Package Attribute hides the information (e.g. client IP address) from view of the common user, who is not aware of the encoding scheme. Obviously, this is not a security method since anyone who knows the encoding scheme is able to obtain the clear text. Note that any parameters appended to the query string after the URI Signing Package Attribute are not validated and hence do not affect URI Signing.

The following attribute is used to carry the encoded set of URI Signing attributes in the Signed URI.

- o URI Signing Package (URISigningPackage) - The encoded attribute containing all the CDNI URI Signing Information Elements used for URI Signing.

The URI Signing Package Attribute contains the URI Signing Information Elements in the Base-64 encoding with URL and Filename Safe Alphabet (a.k.a. "base64url") as specified in the Base-64 Data Encoding [RFC4648] document. The URI Signing Package Attribute is the only URI Signing attribute exposed in the Signed URI. The attribute MUST be the last parameter in the query string of the URI when the Signed URI is generated. However, a client or CDN may append other query parameters unrelated to URI Signing to the Signed URI. Such additional query parameters SHOULD NOT use the same name as the URI Signing Package Attribute to avoid namespace collision and potential failure of the URI Signing validation.

The parameter name of the URI Signing Package Attribute shall be defined in the CDNI Metadata interface. If the CDNI Metadata interface does not include a parameter name for the URI Signing Package Attribute, the parameter name is set by configuration ((out

of scope of this document).

3. Creating the Signed URI

The following procedure for signing a URI defines the algorithms in this version of URI Signing. Note that some steps may be skipped if the CSP does not enforce a distribution policy and the Enforcement Information Elements are therefore not necessary. A URI (as defined in URI Generic Syntax [RFC3986]) contains the following parts: scheme name, authority, path, query, and fragment. The entire URI except the "scheme name" part is protected by the URI signature. This allows the URI signature to be validated correctly in the case when a client performs a fallback to another scheme (e.g. HTTP) for a content item referenced by a URI with a specific scheme (e.g. RTSP). The benefit is that the content access is protected regardless of the type of transport used for delivery. If the CSP wants to ensure a specific protocol is used for content delivery, that information is passed by CDNI metadata. Note: Support for changing of the URL scheme requires that the default port is used, or that the protocols must both run on the same non-standard port.

The process of generating a Signed URI can be divided into two sets of steps: calculating the URI Signature and packaging the URI Signature and appending it to the Original URI. Note it is possible to use some other algorithm and implementation as long as the same result is achieved. An example for the Original URI, "http://example.com/content.mov", is used to clarify the steps.

3.1. Calculating the URI Signature

Calculate the URI Signature by following the procedure below.

1. Copy the Original URI, excluding the "scheme name" part, into a buffer to hold the message for performing the operations below.
2. Check if the URI already contains a query string. If not, append a "?" character. If yes, append an "&" character.
3. If the version is the default value, skip this step. Append the string "VER=". Append the string for the version number.
4. If time window enforcement is not needed, step 4 can be skipped.
 - A. If an attribute was added to the URI, append an "&" character. Append the string "ET=". Note in the case of re-signing a URI, the attribute is carried over from the received Signed URI.

- B. Get the current time in seconds since epoch (as an integer). Add the validity time in seconds as an integer. Note in the case of re-signing a URI, the value MUST remain the same as the received Signed URI.
 - C. Convert this integer to a string and append to the message.
5. If client IP enforcement is not needed, step 5 can be skipped.
- A. If an attribute was added to the URI, append an "&" character. Append the string "CIP=". Note in the case of re-signing a URI, the attribute is carried over from the received Signed URI.
 - B. Convert the client's IP address in dotted decimal notation format (i.e. for IPv4 address) or canonical text representation (for IPv6 address [RFC5952]) to a string and append to the message. Note in the case of re-signing an URI, the value MUST remain the same as the received Signed URI.
6. Depending on the type of key used to sign the URI, compute the message digest or digital signature for symmetric key or asymmetric keys, respectively.
- A. For symmetric key, HMAC is used.
 - 1. Obtain the shared key to be used for signing the URI.
 - 2. If the key identifier is provided by the CDNI metadata, skip this step. If an attribute was added to the URI, append an "&" character. Append the string "KID=". Append the key identifier (e.g. "example:keys:123") needed by the entity to locate the shared key for validating the URI signature.
 - 3. If the hash function for the HMAC uses the default value (SHA-1), skip this step. If an attribute was added to the URI, append an "&" character. Append the string "HF=". Append the string for the type of hash function. Note that re-signing a URI MUST use the same hash function as the received Signed URI or one of the allowable hash functions designated by the CDNI metadata.
 - 4. If an attribute was added to the URI, append an "&" character. Append the string "MD=". The message now contains the complete section of the URI that is protected (e.g. "://example.com/

content.mov?ET=1209422976&CIP=10.0.0.1&
KID=example:keys:123&MD=").

5. Compute the message digest using the HMAC algorithm with the shared key (e.g. "secretkey" and message as the two inputs to the hash function which is specified by the "HF" attribute.
6. Convert the message digest to its equivalent hexadecimal format.
7. Append the string for the message digest (e.g. "://example.com/
content.mov?ET=1209422976&CIP=10.0.0.1&
KID=example:keys:123&
MD=da58513e8b309c1e8a9695baceba629d180b50b8").

B. For asymmetric keys, EC DSA is used.

1. Generate the EC private and public key pair (e.g. private key is "8b5b417336492707a83836b02ceee55b3847be5ec1521e4949977b224950e708", public key is "04840b1be11cfd1404c2fc588d30150a4103cadcc4172e786bcafl5d7feeb6d246f7d8a91fa055cb10efb2f52860d1dlb2f339244e9ad79a23e10ed9b720f6157f"). Store the EC public key in a location that's reachable for any entity that needs to validate the URI signature.
2. If the key identifier is provided by the CDNI metadata, skip this step. If an attribute was added to the URI, append an "&" character. Append the string "KID=". Append the key identifier (e.g. "http://example.com/public/keys/123") needed by the entity to locate the shared key for validating the URI signature. Note the Key ID URI contains only the "scheme name", "authority", and "path" parts (i.e. query string is not allowed).
3. If the digital signature algorithm uses the default value (EC-DSA), skip this step. If an attribute was added to the URI, append an "&" character. Append the string "DSA=". Append the string denoting the digital signature function used.
4. If an attribute was added to the URI, append an "&" character. Append the string "DS=". The message now contains the complete section of the URI that is protected. (e.g. "://example.com/
content.mov?ET=1209422976&CIP=10.0.0.1&KID=http://

example.com/public/keys/123&DS=").

5. Compute the message digest using SHA-1 (without a key) for the message (e.g. message digest is "b95cb62f1d30ad03969619e9574a925fbfe9aeaf"). Note: The reason the digital signature calculated in the next step is calculated over the SHA-1 message digest, instead of over the cleartype message, is to reduce the length of the digital signature, and thereby the length of the URI Signing Package Attribute and the resulting Signed URI.
6. Compute the digital signature, using the EC-DSA algorithm by default or another algorithm if specified by the DSA Information Element, with the private EC key and message digest obtained in previous step as inputs.
7. Convert the digital signature to its equivalent hexadecimal format.
8. Append the string for the digital signature. In the case where EC-DSA algorithm is used, this string contains the values for the 'r' and 's' parameters, delimited by ':' (e.g. "://example.com/content.mov?ET=1209422976&CIP=10.0.0.1&KID=http://example.com/public/keys/123&DS=r:CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D:s:57ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E")

3.2. Packaging the URI Signature

Apply the URI Signing Package Attribute by following the procedure below to generate the Signed URI.

1. Remove the Original URI portion from the message to obtain all the URI Signing Information Elements, including the URI signature (e.g. "ET=1209422976&CIP=10.0.0.1&KID=example:keys:123&&MD=da58513e8b309c1e8a9695baceba629d180b50b8").
2. Compute the URI Signing Package Attribute using Base-64 Data Encoding [RFC4648] on the message (e.g. "RVQ9MTIwOTQyMjk3NiZDSVA9MTAuMCAwLjEmS0lEPWV4YWlwbGU6a2V5czoxMjMmJk1EPWRhNTg1MTNlOGIzMd1jMWU4YTk2OTViYWNlYmE2MjlkMTgwYjUwYjg="). Note: This is the value for the URI Signing Package Attribute.

3. Copy the entire Original URI into a buffer to hold the message.
4. Check if the Original URI already contains a query string. If not, append a "?" character. If yes, append an "&" character.
5. Append the parameter name used to indicate the URI Signing Package Attribute, as communicated via the CDNI Metadata interface, followed by an "=". If none is communicated by the CDNI Metadata interface, it defaults to "URISigningPackage". For example, if the CDNI Metadata interface specifies "SIG", append the string "SIG=" to the message.
6. Append the URI Signing token to the message (e.g. "http://example.com/content.mov?URISigningPackage=RVQ9MTIwOTQyMjk3NiZDSVA9MTAuMC4wLjEmS0lEPWV4YWlwbGU6a2V5czoxMjMmJk1EPWRhNTg1MTNlOGIzMd1jMWU4YTk2OTViYWNlYmE2MjlkMTgwYjUwYjg="). Note: this is the completed Signed URI.

4. Validating a URI Signature

The process of validating a Signed URI can be divided into two sets of steps: validation of the information elements embedded in the Signed URI and validation of the URI Signature. Note it is possible to use some other algorithm and implementation as long as the same result is achieved.

4.1. Information element validation

Extract and validate the information elements embedded in the URI. Note that some steps are to be skipped if the corresponding URI Signing Information Element is not embedded in the Signed URI. The absence of a given Enforcement Information Element indicates enforcement of its purpose is not necessary in the CSP's distribution policy.

1. Extract the value from 'URISigningPackage' attribute. This value is the encoded URI Signing Package Attribute. If there are multiple instances of this attribute, the first one is used and the remaining ones are ignored. This ensures that the Signed URI can be validated despite a client appending another instance of the 'URISigningPackage' attribute.
2. Decode the string using Base-64 Data Encoding [RFC4648] (or another encoding method specified by configuration or CDNI metadata) to obtain all the URI Signing Information Elements (e.g. "ET=1209422976&CIP=10.0.0.1&KID=example:keys:123&&").

MD=da58513e8b309c1e8a9695baceba629d180b50b8").

3. Extract the value from "VER" if the information element exists in the query string. Determine the version of the URI Signing algorithm used to process the Signed URI. If the CDNI Metadata interface is used, check to see if the used version of the URI Signing algorithm is among the allowed set of URI Signing versions specified by the metadata. If this is not the case, the request is denied. If the attribute is not in the URI, then obtain the version number in another manner (e.g. configuration, CDNI metadata or default value).
4. Extract the value from "CIP" if the information element exists in the query string. Validate that the request came from the same IP address as indicated in the "CIP" attribute. If the IP address is incorrect, then the request is denied.
5. Extract the value from "ET" if the information element exists in the query string. Validate that the request arrived before expiration time based on the "ET" attribute. If the time expired, then the request is denied.
6. Extract the value from "MD" if the information element exists in the query string. The existence of this information element indicates a symmetric key is used.
7. Extract the value from "DS" if the information element exists in the query string. The existence of this information element indicates a asymmetric key is used.
8. If neither "MD" or "DS" attribute is in the URI, then no URI Signature exists and the request is denied. If both the "MD" and the "DS" information elements are present, the Signed URI is considered to be malformed and the request is denied.

4.2. Signature validation

Validate the URI Signature for the Signed URI.

1. Copy the Original URI, excluding the "scheme name" part, into a buffer to hold the message for performing the operations below.
2. Remove the "URISigningPackage" attribute from the message. Remove any subsequent part of the query string after the "URISigningPackage" attribute.
3. Append the decoded value from "URISigningPackage" attribute (which contains all the URI Signing Information Elements).

4. Depending on the type of key used to sign the URI, validate the message digest or digital signature for symmetric key or asymmetric keys, respectively.
 - A. For symmetric key, HMAC algorithm is used.
 - a. Extract the value from the "KID" information element, if it exists. Use the key identifier (e.g. "example:keys:123") to locate the shared key, which may be one of the keys available to use (i.e. set by configuration or CDNI metadata). If the information element is not in the URI Signing Package Attribute, then obtain the key in another manner (e.g. configuration or CDNI metadata). If the "KID" information element is present but its value is not in the allowable KID set as listed in the CDNI metadata, the request is denied.
 - b. Extract the value from the "HF" information element, if it exists. Determine the type of hash function (e.g. "MD5", "SHA-1", "SHA-256", "SHA-3") to use for HMAC. If the information element is not in the URI, the default hash function is SHA-1. If the "HF" information element is present but its value is not in the allowable "HF" set as listed in the CDNI metadata, the request is denied.
 - c. Extract the value from the "MD" information element. This is the received message digest.
 - d. Convert the message digest to binary format. This will be used to compare with the computed value later.
 - e. Remove the value part of the "MD" information element (but not the '=' character) from the message. The message is ready for validation of the message digest (e.g. "://example.com/content.mov?ET=1209422976&CIP=10.0.0.1&KID=example:keys:123&MD=").
 - f. Compute the message digest using the HMAC algorithm with the shared key and message as the two inputs to the hash function which is specified by the "HF" attribute.
 - g. Compare the result with the received message digest to validate the Signed URI.

- B. For asymmetric keys, a digital signature function is used.
- a. Extract the value from the "KID" information element, if it exists. Use the key identifier (e.g. "http://example.com/public/keys/123") to obtain the EC public key, which may be one of the keys available to use (i.e. set by configuration or CDNI metadata). If the information element is not in the URI, then obtain the key in another manner (e.g. configuration or CDNI metadata).
 - b. Extract the value from the "DSA" information element, if it exists. Determine the type of digital signature function (e.g. "RSA", "DSA", "EC-DSA") to use for calculating the Digital Signature. If the information element is not in the URI, the default digital signature function is EC-DSA. If the "DSA" information element is present but its value is not in the allowable "EC-DSA" set as listed in the CDNI metadata, the request is denied.
 - c. Extract the value from the "DS" information element. This is the digital signature.
 - d. Convert the digital signature to binary format. This will be used for verification later.
 - e. Remove the value part of the "DS" information element (but not the '=' character) from the message. The message is ready for validation of the digital signature (e.g. "://example.com/content.mov?ET=1209422976&CIP=10.0.0.1&KID=http://example.com/public/keys/123&DS=").
 - f. Compute the message digest using SHA-1 (without a key) for the message.
 - g. Verify the digital signature using the digital signature function (e.g. EC-DSA) with the public key, received digital signature, and message digest (obtained in previous step) as inputs. This validates the Signed URI.

5. Relationship with CDNI Interfaces

Some of the CDNI Interfaces need enhancements to support URI Signing. As an example: A Downstream CDN that supports URI Signing needs to be able to advertise this capability to the Upstream CDN. The Upstream

CDN needs to select a Downstream CDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the Upstream CDN needs to be able to distribute via the CDNI Metadata interface the information necessary to allow the Downstream CDN to validate a Signed URI . Events that pertain to URI Signing (e.g. request denial or delivery after access authorization) need to be included in the logs communicated through the CDNI Logging interface (Editor's Note: Is this within the scope of the CDNI Logging interface?).

5.1. CDNI Control Interface

URI Signing has no impact on this interface.

5.2. CDNI Footprint & Capabilities Advertisement Interface

The Downstream CDN advertises its capability to support URI Signing via the CDNI Footprint & Capabilities Advertisement interface (FCI). The supported version of URI Signing needs to be included to allow for future extensibility.

[Editor's Note: To be discussed with FCI authors]

5.3. CDNI Request Routing Redirection Interface

[Editor's Note: Debate the approach of dCDN providing the Signed URI vs. uCDN performing the signing function. List the pros/cons of each approach for the CDNI Request Routing Redirection interface (RI). Offer recommendation?]

The two approaches:

1. Downstream CDN provides the Signed URI
 - * Key distribution is not necessary
 - * Downstream CDN can use any scheme for Signed URI as long as the security level meets the CSP's expectation
2. Upstream CDN signs the URI
 - * Consistency with interative request routing method
 - * URI Signing works even when Downstream CDN does not have the signing function (which may be the case when the Downstream CDN operates only as a delivering CDN)

- * Upstream CDN can act as a conversion gateway for the requesting routing interface between Upstream CDN and CSP and request routing interface between Upstream CDN and Downstream CDN since these two interfaces may not be the same

5.4. CDNI Metadata Interface

The following CDNI Metadata objects are specified for URI Signing.

- o URI Signing enforcement flag. Specifically, this flag indicates if the access to content is subject to URI Signing. URI Signing requires the Downstream CDN to ensure that the URI must be signed and validated before content delivery. Otherwise, Downstream CDN does not perform validation regardless if URI is signed or not.
- o Designated key identifier used for URI Signing computation when the Signed URI does not contain the Key ID information element
- o Allowable Key ID set that the Signed URI's Key ID information element can reference
- o Designated hash function used for URI Signing computation when the Signed URI does not contain the Hash Function information element
- o Allowable Hash Function set that the Signed URI's Hash Function information element can reference
- o Designated digital signature function used for URI Signing computation when the Signed URI does not contain the Digital Signature Algorithm information element.
- o Allowable digital signature function set that the Signed URI's Digital Signature Algorithm information element can reference.
- o Designated version used for URI Signing computation when the Signed URI does not contain the VER attribute
- o Allowable version/algorithm set that the Signed URI's VER attribute can reference
- o Allowable set of Downstream CDNs that participate in URI Signing based on the symmetric key
- o Overwrite the default encoding method for URI Signing Attribute Set attribute? [Editor's Note: Do we need this?]
- o Overwrite the default name for the URL Signing Attribute Set attribute? [Editor's Note: Do we need this?]

Note that the Key ID information is not needed if only one key is provided by the CSP or the Upstream CDN for the content item or set of content items covered by the CDNI Metadata object. In the case of asymmetric keys, it's easy for any entity to sign the URI for content with a private key and provide the public key in the Signed URI. This just confirms that the URI Signer authorized the delivery. But it's necessary for the URI Signer to be the content owner. So, the CDNI Metadata interface MUST provide the public key for the content or information to authorize the received Key ID attribute.

5.5. CDNI Logging Interface

The Downstream CDN reports that enforcement of the access control was applied to the request for content delivery.

The following CDNI Logging field for URI Signing SHOULD be supported in the HTTP Request Logging Record as specified in CDNI Logging Interface [I-D.ietf-cdni-logging].

- o s-uri-signing:
 - * format: 1DIGIT
 - * field value: this characterises the uri signing validation performed by the Surrogate on the request. The allowed values are:
 - + "0" : no uri signature validation performed
 - + "1" : uri signature validation performed and validated
 - + "2" : uri signature validation performed and rejected
 - * occurrence: there MUST be zero or exactly one instance of this field.

[Editor's note: Need to log these URI signature validation events (e.g. invalid client IP address, expired signed URI, incorrect URI signature, successful validation)?]

TBD: CDNI Logging interface is work in progress.

6. URI Signing Message Flow

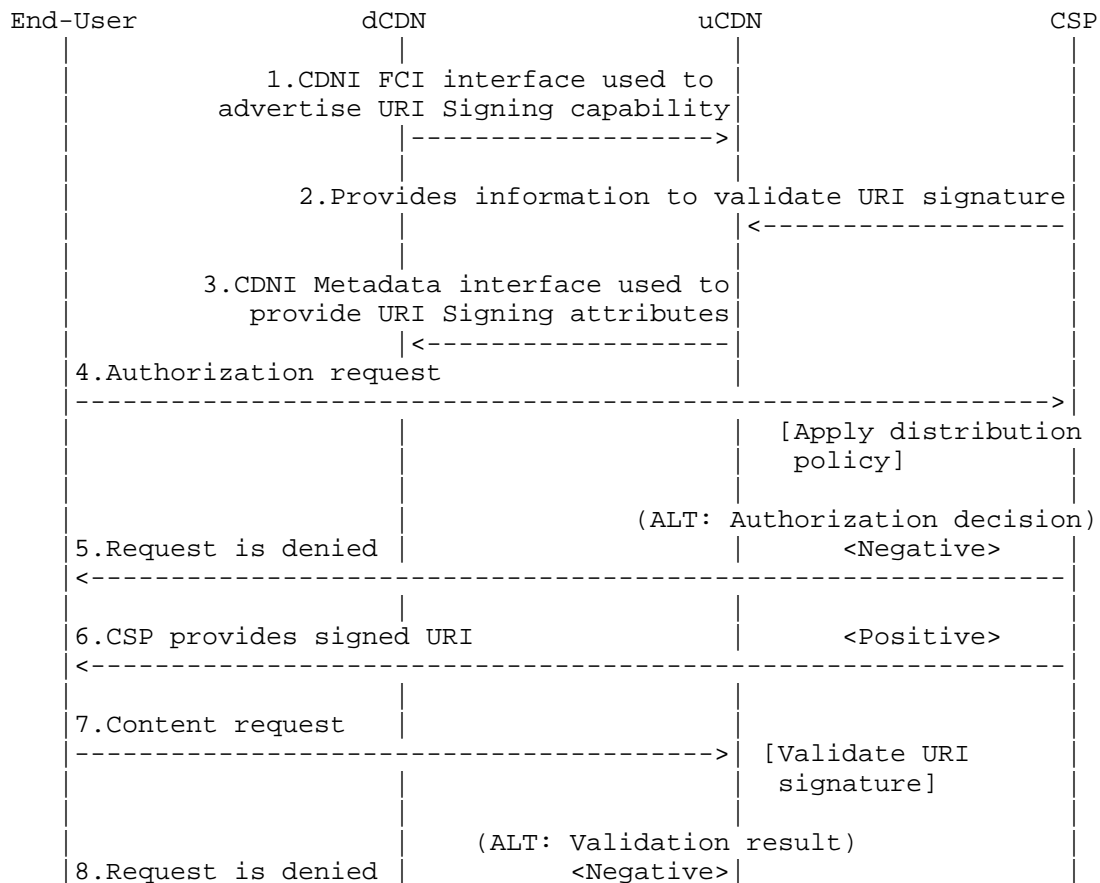
URI Signing supports both HTTP-based and DNS-based request routing. HMAC [RFC2104] defines a hash-based message authentication code allowing two parties that share a symmetric key or asymmetric keys to

establish the integrity and authenticity of a set of information (e.g. a message) through a cryptographic hash function.

6.1. HTTP Redirection

For HTTP-based request routing, HMAC is applied to a set of information that is unique to a given end user content request using key information that is specific to a pair of adjacent CDNI hops (e.g. between the CSP and the Authoritative CDN, between the Authoritative CDN and a Downstream CDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI signing scheme described below is based on the following steps (assuming HTTP redirection, iterative request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



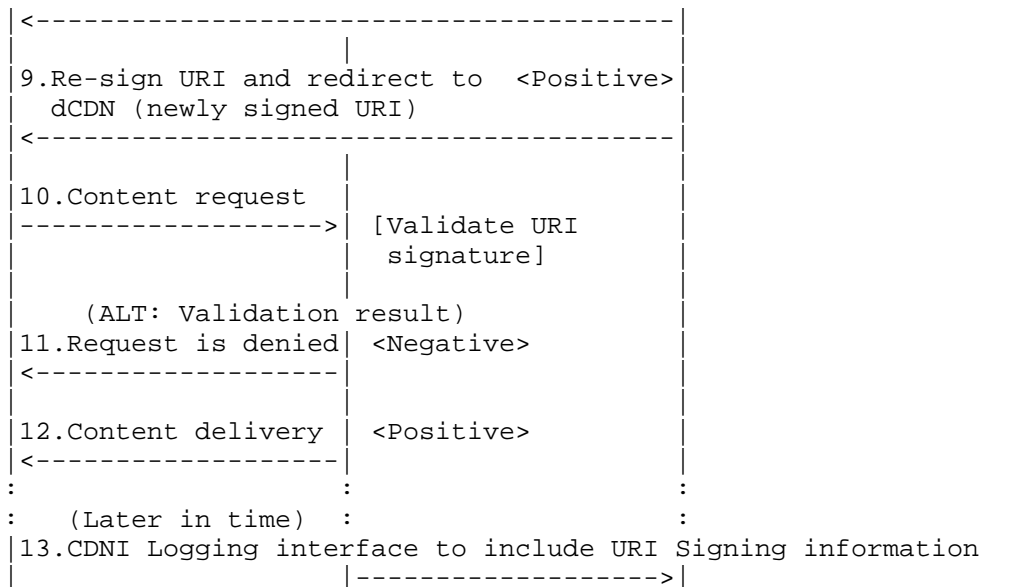


Figure 3: HTTP-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.
2. CSP provides to the Authoritative CDN the information needed to validate URI signatures from that CSP. For example, this information may include a hashing function, algorithm, and a key value.
3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate URI signatures from the Authoritative CDN for the given CSP. For example, this information may include the URI query string parameter name for the URI Signing Package Attribute, a hashing algorithm and/or a key corresponding to the trust relationship between the Authoritative CDN and the Downstream CDN.
4. When a UA requests a piece of protected content from the CSP, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy
5. If the authorization decision is negative, the CSP rejects the request.

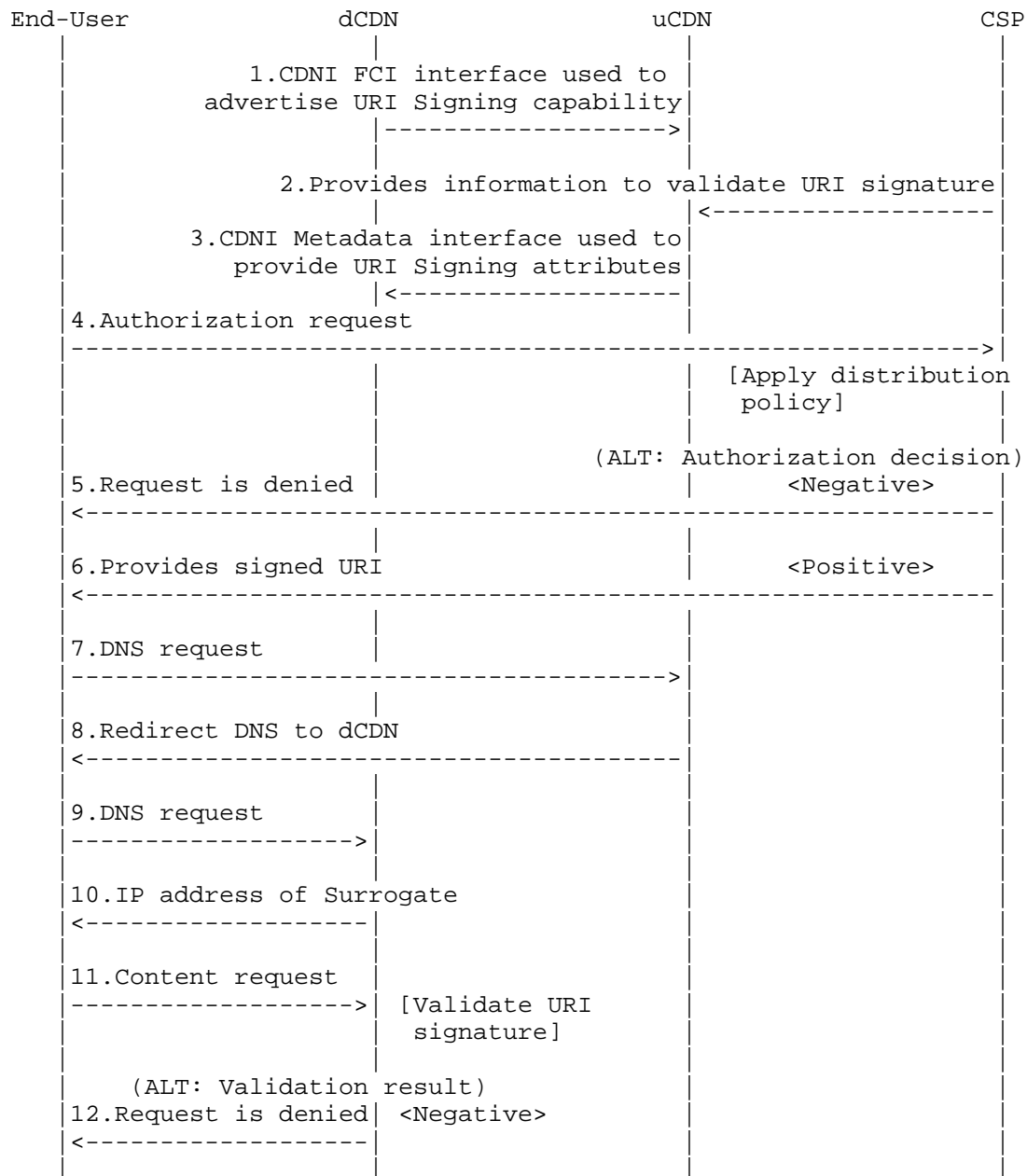
6. If the authorization decision is positive, the CSP computes a Signed URI that is based on unique parameters of that request and conveys it to the end user as the URI to use to request the content.
7. On receipt of the corresponding content request, the authoritative CDN validates the URI Signature in the URI using the information provided by the CSP.
8. If the validation is negative, the authoritative CDN rejects the request
9. If the validation is positive, the authoritative CDN computes a Signed URI that is based on unique parameters of that request and provides to the end user as the URI to use to further request the content from the Downstream CDN
10. On receipt of the corresponding content request, the Downstream CDN validates the URI Signature in the Signed URI using the information provided by the Authoritative CDN in the CDNI Metadata
11. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.
12. If the validation is positive, the Downstream CDN serves the request and delivers the content.
13. At a later time, Downstream CDN reports logging events that includes URI signing information.

With HTTP-based request routing, URI Signing matches well the general chain of trust model of CDNI both with symmetric key and asymmetric keys because the key information only need to be specific to a pair of adjacent CDNI hops.

6.2. DNS Redirection

For DNS-based request routing, the CSP and Authoritative CDN must agree on a trust model appropriate to the security requirements of the CSP's particular content. Use of asymmetric public/private keys allows for unlimited distribution of the public key to Downstream CDNs. However, if a shared secret key is preferred, then the CSP may want to restrict the distribution of the key to a (possibly empty) subset of trusted Downstream CDNs. Authorized Delivery CDNs need to obtain the key information to validate the Signed UR, which is computed by the CSP based on its distribution policy.

The URI signing scheme described below is based on the following steps (assuming iterative DNS request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



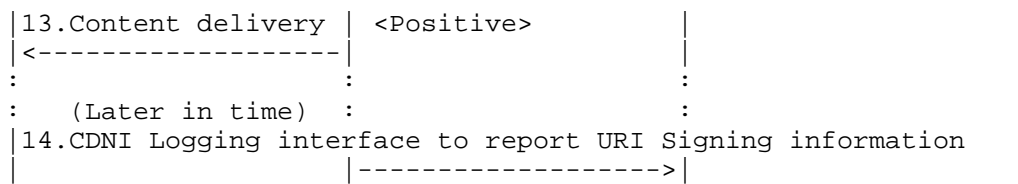


Figure 4: DNS-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.
2. CSP provides to the Authoritative CDN the information needed to validate cryptographic signatures from that CSP. For example, this information may include a hash function, algorithm, and a key.
3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate cryptographic signatures from the CSP (e.g. the URI query string parameter name for the URI Signing Package Attribute). In the case of symmetric key, the Authoritative CDN checks if the Downstream CDN is allowed by CSP to obtain the shared secret key.
4. When a UA requests a piece of protected content from the CSP, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy.
5. If the authorization decision is negative, the CSP rejects the request
6. If the authorization decision is positive, the CSP computes a cryptographic signature that is based on unique parameters of that request and includes it in the URI provided to the end user to request the content.
7. End user sends DNS request to the authoritative CDN.
8. On receipt of the DNS request, the authoritative CDN redirects the request to the Downstream CDN.
9. End user sends DNS request to the Downstream CDN.
10. On receipt of the DNS request, the Downstream CDN responds with IP address of one of its Surrogates.

11. On receipt of the corresponding content request, the Downstream CDN validates the cryptographic signature in the URI using the information provided by the Authoritative CDN in the CDNI Metadata
12. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.
13. If the validation is positive, the Downstream CDN serves the request and delivers the content.
14. At a later time, Downstream CDN reports logging events that includes URI signing information.

With DNS-based request routing, URI Signing matches well the general chain of trust model of CDNI when used with asymmetric keys because the only key information that need to be distributed across multiple CDNI hops including non-adjacent hops is the public key, that is generally not confidential.

With DNS-based request routing, URI Signing does not match well the general chain of trust model of CDNI when used with symmetric keys because the symmetric key information needs to be distributed across multiple CDNI hops including non-adjacent hops. This raises a security concern for applicability of URI Signing with symmetric keys in case of DNS-based inter-CDN request routing.

7. HTTP Adaptive Streaming

The authors note that in order to perform URI signing for individual content segments of HTTP Adaptive Bitrate content, specific URI signing mechanisms are needed. Such mechanisms are currently out-of-scope of this document. More details on this topic is covered in Models for HTTP-Adaptive-Streaming-Aware CDNI [RFC6983].

8. IANA Considerations

[Editor's note: (Is there a need to) register default value for URI Signing Package Attribute URI query string parameter name (i.e. URISigningPackage) to be used for URI Signing? Need anything from IANA?]

[Editor's note: To do: Convert to proper IANA Registry format]

This document requests IANA to create three new registries for the Information Elements and their defined values to be used for URI

Signing.

The following Enforcement Information Element names are allocated:

- o ET (Expiry time)
- o CIP (Client IP address)

The following Signature Computation Information Element names are allocated:

- o VER (Version): 1(Base)
- o KID (Key ID)
- o HF (Hash Function): "MD5", "SHA-1", "SHA-256", "SHA-3"
- o DSA (Digital Signature Algorithm): "RSA", "DSA", "EC-DSA"

The following URI Signature Information Element names are allocated:

- o MD (Message Digest)
- o DS (Digital Signature)

The IANA is requested to allocate a new entry to the CDNI Logging Field Names Registry as specified in CDNI Logging Interface [I-D.ietf-cdni-logging] in accordance to the "Specification Required" policy [RFC5226]

- o s-url-signing
- o [Editor's note: logging error codes are needed for URI Signing?]

The IANA is requested to allocate a new entry to the CDNI Metadata Field Names Registry as specified in CDNI Metadata Interface [I-D.ietf-cdni-metadata] in accordance to the "Specification Required" policy [RFC5226]

- o URI Signing Package URI query parameter name 1 Token
- o More metadata...

9. Security Considerations

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of interconnected

CDNs (CDNI). The primary goal of URI Signing is to make sure that only authorized UAs are able to access the content, with a Content Service Provider (CSP) being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as DRM, are more appropriate.

In general, it holds that the level of protection against illegitimate access can be increased by including more Enforcement Information Elements in the URI. The current version of this document includes elements for enforcing Client IP Address and Expiration Time, however this list can be extended with other, more complex, attributes that are able to provide some form of protection against some of the vulnerabilities highlighted below.

That said, there are a number of aspects that limit the level of security offered by URI signing and that anybody implementing URI signing should be aware of.

Replay attacks: Any (valid) Signed URI can be used to perform replay attacks. The vulnerability to replay attacks can be reduced by picking a relatively short window for the Expiration Time attribute, although this is limited by the fact that any HTTP-based request needs a window of at least a couple of seconds to prevent any sudden network issues from preventing legitimate UAs access to the content. One way to reduce exposure to replay attacks is to include in the URI a unique one-time access ID. Whenever the Downstream CDN receives a request with a given unique access ID, it adds that access ID to the list of 'used' IDs. In the case an illegitimate UA tries to use the same URI through a replay attack, the Downstream CDN can deny the request based on the already-used access ID.

Illegitimate client behind a NAT: In cases where there are multiple users behind the same NAT, all users will have the same IP address from the point of view of the Downstream CDN. This results in the Downstream CDN not being able to distinguish between the different users based on Client IP Address and illegitimate users being able to access the content. One way to reduce exposure to this kind of attack is to not only check for Client IP but also for other attributes that can be found in the HTTP headers.

TBD: ...

The shared key between CSP and Authoritative CDN may be distributed to Downstream CDNs - including cascaded CDNs. Since this key can be used to legitimately sign a URL for content access authorization,

it's important to know the implications of a compromised shared key.

[Editor's note: Threat model cover in the Security section - Prevent client from spoofing URI (Ray) - Security implications - The scope of protection by URI Signing - Protects against DoS (network bandwidth and other nodes besides the edge cache); limits the time window.]

10. Privacy

The privacy protection concerns described in CDNI Logging Interface [I-D.ietf-cdni-logging] apply when the client's IP address (CIP attribute) is embedded in the Signed URI. This means that, when anonymization is enabled, the URI Signing Package Attribute MUST be removed from the logging record.

11. Acknowledgements

The authors would like to thank the following people for their contributions in reviewing this document and providing feedback: Kevin Ma, Ben Niven-Jenkins, Thierry Magnien, Dan York, Bhaskar Bhupalam, Matt Caulfield, and Samuel Rajakumar .

12. References

12.1. Normative References

- [I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-10 (work in progress), March 2014.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

12.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L., Davie, B., and R. Brandenburg, "Framework for CDN Interconnection", draft-ietf-cdni-framework-10 (work in progress), March 2014.

- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M.,
Leung, K., and K. Ma, "CDN Interconnect Metadata",
draft-ietf-cdni-metadata-06 (work in progress),
February 2014.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network
Interconnection (CDNI) Requirements",
draft-ietf-cdni-requirements-17 (work in progress),
January 2014.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-
Hashing for Message Authentication", RFC 2104,
February 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, January 2005.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data
Encodings", RFC 4648, October 2006.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6
Address Text Representation", RFC 5952, August 2010.
- [RFC6770] Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma,
K., and G. Watson, "Use Cases for Content Delivery Network
Interconnection", RFC 6770, November 2012.
- [RFC6983] van Brandenburg, R., van Deventer, O., Le Faucheur, F.,
and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware
Content Distribution Network Interconnection (CDNI)",
RFC 6983, July 2013.

Authors' Addresses

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose 95134
USA

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

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

Bill Downey
Verizon Labs
60 Sylvan Road
Waltham, Massachusetts 02451
USA

Phone: +1 781 466 2475
Email: william.s.downey@verizon.com

Ray van Brandenburg
TNO
Brassersplein 2
Delft, 2612CT
the Netherlands

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

Scott Leibrand
Limelight Networks
222 S Mill Ave
Tempe, AZ 85281
USA

Phone: +1 360 419 5185
Email: sleibrand@llnw.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 6, 2013

K. Ma
Azuki Systems, Inc.
August 5, 2012

Content Distribution Network Interconnection (CDNI) Capabilities
Interface
draft-ma-cdni-capabilities-00

Abstract

The interconnection of Content Distribution Networks (CDNs) is predicated on the ability of downstream CDNs (dCDNs) to handle end-user requests in a functionally equivalent manner to the upstream CDN (uCDN). The uCDN must be able to assess the ability of the dCDN to handle individual requests. A CDN interconnection (CDNI) interface is needed to facilitate the advertisement of capabilities by the dCDN to the uCDN. This document describes an approach to implementing a CDNI capabilities advertisement interface.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 February 6, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. Capabilities	4
2.1. Delivery Protocol	6
2.2. Acquisition Protocol	7
2.3. Metadata Bundle	9
2.4. Redirection Mode	10
3. Capability Advertisement	11
3.1. Capability Update	12
3.2. Capability Removal	13
4. IANA Considerations	14
5. Security Considerations	14
6. Acknowledgements	15
7. Appendix A: Capability Aggregation	15
7.1. Downstream CDN Aggregation	15
7.2. Internal Request Router Aggregation	17
7.3. Internal Capability Aggregation	18
8. References	20
8.1. Normative References	20
8.2. Informative References	20
Author's Address	20

1. Introduction

The need for capabilities advertisement in CDNI is described in the CDNI requirements document [I-D.ietf-cdni-requirements]. Requirement REQ-2 describes the need to allow dCDNs to communicate capabilities to the uCDN. The uCDN aggregates the capabilities information for all dCDNs for use in making request routing decisions. Per requirement REQ-3, the uCDN should further use the aggregated capabilities in advertisements to CDNs further upstream. This concept of aggregation can apply to both organizationally different dCDNs (e.g., other CDN providers, or different business units within a larger organization) or logical entities within the same CDN (e.g., using multiple request routers for scalability reasons, to segregate surrogates based on specific protocol support, or to segregate surrogates based on software version or feature level, etc.).

Appendix A contains more detailed descriptions of the different capabilities management scenarios, but it is important to note that it is the ability of the dCDN to service each request in a functionally equivalent manner as the uCDN that is important, not the physical layout of resources through which it services the request. The aggregation of resource knowledge by the dCDN into a simple set of capabilities which can be advertised to the uCDN enables efficient decision making at each delegation point in the CDN interconnection hierarchy.

It is assumed that an authoritative request router in each CDN will be responsible for aggregating and advertising capabilities information in a dCDN, and receiving and aggregating capabilities information in the uCDN. As there is no centralized CDNI controller defined in the CDNI architecture, the request router seems the most logical place for capabilities aggregation to occur, as it is the request router that needs such information to make delegation decisions. The protocol defined herein may be implemented as part of an entity other than an authoritative request router, but for the purposes of this discussion, the authoritative request router is assumed to be the centralized capabilities aggregation point.

Though there is an obvious need for the ability to exchange and update capability information in real-time, it is assumed that capabilities do not change very often. There is also an assumption that the capabilities are not by themselves useful for making delegation decisions. Capability information is assumed to be input into business logic. It is the business logic which provides the algorithms for delegation decision making. The definition of business logic occurs outside the scope of CDNI and outside the timescale of capability advertisement. It may be the case that the business logic anticipates and reacts to changes in dCDN

capabilities. However, it may also be the case that business logic is tailored through offline processes as dCDN capabilities change. The capabilities interface is agnostic to the business processes employed by a given uCDN. The capabilities that are advertised over the capabilities interface may be used by the uCDN at its leisure to implement delegation rules. Setting proper defaults in the business logic should prevent any unwanted delegation from occurring when dCDN capabilities change, however, that is beyond the scope of this discussion.

1.1. Terminology

[Ed. note: Insert terminology reference]

2. Capabilities

There is a basic set of capabilities that must be advertised by the dCDN for the uCDN to be able to determine if the dCDN is functionally able to handle a given request. Mandatory capabilities include:

- o Delivery Protocol
- o Acquisition Protocol
- o Metadata Bundle
- o Redirection Mode

The following sections describe each of the capabilities in further detail, however, all of the capabilities can be described using the same general format. A capability object can be described as:

CapabilityObject:

 Name: Identifier for the capability

 Values: List of supported options for the capability

 Footprint: Optional list of footprint restrictions

The list of valid capability options for a given capability will be specific to the given capability type. Though the degenerate case may exist where the range of option values is a single item, it is anticipated that all capability types will have more than one capability option values. For consistency in the model, all capability types are implemented with lists of values. To optimize actions on the entire range of capability option values for a given capability type, the capability option value "ALL" is reserved and MUST be supported by all capability types. For completeness, the capability option value "NONE" is also reserved and MUST be supported

by all capability types. Reserved values SHOULD NOT be combined with any other valid capability option values for a given capability type. Reserved values MUST NOT be combined with each other for a given capability type. The reserved values MUST be given precedence over all other capability option values for a given capability type, when specified in the same capabilities advertisement message.

The footprint restriction list is optional. Footprint restriction lists override in their entirety all previously advertised footprint restriction lists for the specified capability option values for the given capability type. If no footprint restriction list is specified, it SHALL be understood that all of the capability option values specified have global reach, overriding any previous capability advertisements for the specified capability option values. The footprint restriction list supports either IP prefix or ASN values. IP prefix and ASN restrictions MUST NOT be mixed within a given footprint list. In the case of interconnecting private network CDNs, IP prefix or ASN-based footprint advertisement is the likely method for describing the coverage areas.

[Ed. note: Though other methods exist for defining footprints (e.g., GPS coordinates, country/zip/area code, etc.) only IP prefix and ASN are considered here. Need to evaluate use cases for other methods of footprint definition.]

Note: Further optimization of the capabilities object to provide quality information about a given footprint is certainly possible, however, it is not critical to the basic interconnection of CDNs. The ability to transfer quality information in capabilities advertisements may be desirable and is noted here for completeness, however, the specifics of such mechanisms are outside the scope of this document.

Multiple capability objects of the same capability type are allowed within a given capability advertisement message as long as the capability option values do not overlap, i.e., a given capability option value MUST NOT show up in multiple capability objects within a single capability advertisement message. If multiple capability objects for a given capability type exist, those capability objects SHOULD have different footprint restrictions; capability objects of a given capability type with identical footprint restrictions SHOULD be combined.

Note: The examples shown below use an XML representation, however, other representations (e.g., JSON) may also be acceptable.

2.1. Delivery Protocol

The delivery protocol refers to the protocol over which the client has requested content. If a dCDN does not support the protocol requested by the client, then the dCDN is not a viable candidate for delegation. The delivery protocol is specified in the URI scheme (as defined in RTC3986 [RFC3986]) in the client request URL.

[Ed. note: The proposal to allow only URI schemes in the specification of delivery protocol is somewhat restrictive in that some CDNs may only support a subset of features defined for that protocol (e.g., different HTTP methods, chunking, ranges, etc.). Need to consider a scalable way to define feature bundles without enumerating every feature of every known protocol.]

[Ed. note: The proposal to allow only URI schemes in the specification of delivery protocol is somewhat restrictive in that it does not differentiate between application layer protocols (e.g., HLS, DASH, etc.) which run over top of HTTP. While support for HTTP should be sufficient to support application layer protocols that run over top of HTTP, there is significant interest in being able to optimize the application layer protocols. Need to consider a way to enhance the protocol definition to include application layer protocols, possibly as a separate capability type.]

The delivery protocol capability object MUST support a list of protocols for a given footprint. The delivery protocol capability SHOULD support optional footprint restriction information. The following XML-encoded example shows two lists of protocols with different footprints.

```
<Capabilities>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
      <Protocol>RTSP</Protocol>
      <Protocol>MMS</Protocol>
    </Protocols>
  </DeliveryProtocols>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>RTMP</Protocol>
      <Protocol>HTTPS</Protocol>
    </Protocols>
    <Footprint>
      <Prefixes>
        <Prefix>10.1.0.0/16</Prefix>
        <Prefix>10.10.10.0/24</Prefix>
      </Prefixes>
    </Footprint>
  </DeliveryProtocols>
</Capabilities>
```

In the above example, the three protocols HTTP, RTSP, and MMS are supported globally, while the protocols RTMP and HTTPS are only supported in a restricted footprint (in this case, specified by IP address prefix).

A protocol MUST not appear in multiple delivery protocol lists, within a given capability advertisement message. Redefinition of footprint restrictions in subsequent capability advertisement messages MAY occur and SHALL override all previous capability advertisements for the specified delivery protocol.

2.2. Acquisition Protocol

The acquisition protocol refers to the protocol over which the dCDN may acquire content from the uCDN. If a dCDN does not support any of the protocols offered by the uCDN, then the dCDN is not a viable candidate for delegation. The acquisition protocol is disseminated to the dCDN in the URI scheme (as defined in RTC3986 [RFC3986]) provided by the uCDN via the CDNI Metadata Interface.

[Ed. note: The proposal to allow only URI schemes in the specification of acquisition protocol is somewhat restrictive in that some CDNs may only support a subset of features defined for that protocol (e.g., different HTTP methods, chunking, ranges, etc.). Need to consider a scalable way to define feature bundles without enumerating every feature of every known protocol.]

[Ed. note: The proposal to allow only URI schemes in the specification of acquisition protocol is somewhat restrictive in that it does not differentiate between application layer protocols (e.g., HLS, DASH, etc.) which run over top of HTTP. While support for HTTP should be sufficient to support application layer protocols that run over top of HTTP, there is significant interest in being able to optimize application layer protocols. Need to consider a way to enhance the protocol definition to include application layer protocols, possibly as a separate capability type.]

The acquisition protocol capability object MUST support a list of protocols for a given footprint. The acquisition protocol capability SHOULD support optional footprint restriction information. The following XML-encoded example shows two lists of protocols with different footprints.

```
<Capabilities>
  <AcquisitionProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
      <Protocol>FTP</Protocol>
    </Protocols>
  </AcquisitionProtocols>
  <AcquisitionProtocols>
    <Protocols>
      <Protocol>SFTP</Protocol>
      <Protocol>HTTPS</Protocol>
    </Protocols>
    <Footprint>
      <ASNs>
        <ASN>0</ASN>
        <ASN>65535</ASN>
      </ASNs>
    </Footprint>
  </AcquisitionProtocols>
</Capabilities>
```

In the above example, the two protocols HTTP and FTP are supported globally, while the protocols SFTP and HTTPS are only supported in a restricted footprint (in this case, specified by ASN).

A protocol MUST not appear in multiple acquisition protocol lists, within a given capability advertisement message. Redefinition of footprint restrictions in subsequent capability advertisement messages MAY occur and SHALL override all previous capability advertisements for the specified acquisition protocol.

2.3. Metadata Bundle

Metadata bundles are groupings of one or more metadata definitions which the dCDN is capable of understanding. There will be a core set of metadata defined which all CDNI implementations will be required to support, however, future revisions of CDNI may include additional mandatory to implement metadata, and some operators may require the use of vendor specific metadata. If a dCDN is not capable of understanding a piece of metadata which the uCDN feels is mandatory for delivery, then the dCDN is not a viable candidate for delegation. Metadata bundle naming is to be defined in the CDNI Metadata interface.

[Ed. note: Support for individual options within a given piece of metadata within a given bundle (e.g., URL signing method 3, within the CDNI core metadata version 1 bundle) may also need to be advertised, however, this needs to be coordinated with TBD aspects the CDNI Metadata interface specification.]

The metadata bundle capability object MUST support a list of protocols for a given footprint. The metadata bundle capability SHOULD support optional footprint restriction information. The following XML-encoded example shows two lists of bundles with different footprints.

```
<Capabilities>
  <MetadataBundles>
    <Bundles>
      <Bundle>cdni.core.v1</Bundle>
    </Bundles>
  </MetadataBundles>
  <MetadataBundle>
    <Bundles>
      <Bundle>vendor.azuki.has.v1</Bundle>
      <Bundle>private.rw.color.v1</Bundle>
    </Bundles>
    <Footprint>
      <Prefixes>
        <Prefix>10.1.2.0/24</Prefix>
      </Prefixes>
    </Footprint>
  </MetadataBundle>
</Capabilities>
```

In the above example, core version 1 CDNI metadata cdni.core.v1 is supported globally, while the vendor specific metadata bundles vendor.azuki.has.v1 and private.rw.color.v1 are only supported in a restricted footprint (in this case, specified by IP Prefix).

A bundle MUST not appear in multiple metadata bundle lists, within a given capability advertisement message. Redefinition of footprint restrictions in subsequent capability advertisement messages MAY occur and SHALL override all previous capability advertisements for the specified metadata bundle.

2.4. Redirection Mode

The redirection mode refers to the method or method(s) employed by the request routing interface. The CDNI framework [I-D.ietf-cdni-framework] document describes four possible request routing modes:

- o DNS iterative (DNS-I)
- o DNS recursive (DNS-R)
- o HTTP iterative (HTTP-I)
- o HTTP recursive (HTTP-R)

If a dCDN supports only a specific mode or subset of modes that does not overlap with the modes supported by the uCDN, then the dCDN is not a viable candidate for delegation.

[Ed. note: The list of request routing modes may not be an exhaustive list of request routing modes that will eventually be supported. This needs to be coordinated with TBD aspects the CDNI Request Routing interface specification.]

The redirection mode capability object MUST support a list of redirection modes for a given footprint. The redirection mode capability SHOULD support optional footprint restriction information. The following XML-encoded example shows two lists of modes with different footprints.

```
<Capabilities>
  <RedirectionModes>
    <Modes>
      <Mode>DNS-I</Mode>
      <Mode>HTTP-I</Mode>
    </Modes>
  </RedirectionModes>
  <RedirectionModes>
    <Modes>
      <Mode>DNS-R</Mode>
      <Mode>HTTP-R</Mode>
    </Modes>
    <Footprint>
      <ASNs>
        <ASN>64511</ASN>
      </ASNs>
    </Footprint>
  </RedirectionModes>
</Capabilities>
```

In the above example, iterative redirection is supported globally, while recursive redirection is only supported in a restricted footprint (in this case, specified by ASN).

A mode MUST not appear in multiple redirection mode lists, within a given capability advertisement message. Redefinition of footprint restrictions in subsequent capability advertisement messages MAY occur and SHALL override all previous capability advertisements for the specified redirection mode.

3. Capability Advertisement

The capabilities advertisement protocol is an HTTP-based protocol using the POST and DELETE methods. Capability removal is distinguished from capability advertisement by the HTTP request method. Capability advertisement uses the HTTP POST method, while capability removal uses the HTTP DELETE method. The dCDN request router asynchronously issues capability advertisement messages to the uCDN request router. It is assumed that the dCDN request router has been configured with the uCDN request router address either through the CDNI Control interface bootstrapping function, or through some other out-of-band configuration.

Note: The examples shown below use an XML representation, however, other representations (e.g., JSON) are also acceptable.

3.1. Capability Update

A capabilities advertisement message may combine objects from one or more capability types. On an initial advertisement, the dCDN SHOULD send a value for every capability type.

```
POST /CDNI/RI/capabilities HTTP/1.1
Host: rr0.u.example.com
Accept: */*
Content-Length: 530
Content-Type: application/x-www-form-urlencoded
```

```
<Capabilities>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
    </Protocols>
  </DeliveryProtocols>
  <AcquisitionProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
      <Protocol>FTP</Protocol>
    </Protocols>
  </AcquisitionProtocols>
  <MetadataBundles>
    <Bundles>
      <Bundle>cdni.core.v1</Bundle>
      <Bundle>private.rw.color.v1</Bundle>
    </Bundles>
  </MetadataBundles>
  <RedirectionModes>
    <Modes>
      <Mode>HTTP-I</Mode>
    </Modes>
  </RedirectionModes>
</Capabilities>
```

In the above example, the dCDN issues a post to the uCDN request router rr0.u.example.com with a capability advertisement message specifying support for HTTP-based delivery, HTTP or FTP-based content acquisition, iterative HTTP redirection, and support for both core and color metadata. A subsequent capabilities advertisement message may be used to update this information without respecifying all the fields. In the following example, the dCDN updates its ability to deliver content via HTTPS while restricting its previously advertised support for delivering content via HTTP to only its internal network. All previous advertisements for acquisition protocol, metadata bundles, and redirection modes still are unaffected.

```
POST /CDNI/RI/capabilities HTTP/1.1
Host: rr0.u.example.com
Accept: */*
Content-Length: 356
Content-Type: application/x-www-form-urlencoded
```

```
<Capabilities>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>HTTPS</Protocol>
    </Protocols>
  </DeliveryProtocols>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
    </Protocols>
    <Footprint>
      <Prefixes>
        <Prefix>10.0.0.0/8</Prefix>
      </Prefixes>
    </Footprint>
  </DeliveryProtocols>
</Capabilities>
```

3.2. Capability Removal

In the following example, the dCDN issues a DELETE command to the uCDN request router rr0.u.example.com with a capability advertisement message containing all four capability types specifying the reserved capability option value ALL. This removes capabilities information for all capability options for all capability types, effectively allowing the dCDN to remove itself from delegation consideration.

```
DELETE /CDNI/RI/capabilities HTTP/1.1
Host: rr0.u.example.com
Accept: */*
Content-Length: 442
Content-Type: application/x-www-form-urlencoded
```

```
<Capabilities>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>ALL</Protocol>
    </Protocols>
  </DeliveryProtocols>
  <AcquisitionProtocols>
    <Protocols>
      <Protocol>ALL</Protocol>
    </Protocols>
  </AcquisitionProtocols>
  <MetadataBundles>
    <Bundles>
      <Bundle>ALL</Bundle>
    </Bundles>
  </MetadataBundles>
  <RedirectionModes>
    <Modes>
      <Mode>ALL</Mode>
    </Modes>
  </RedirectionModes>
</Capabilities>
```

4. IANA Considerations

This memo includes no request to IANA.

[Ed. note: Should there be a registry for capability names?]

5. Security Considerations

There are a number of security concerns associated with the capabilities interface. As the capabilities interface essentially provides configuration information which the uCDN uses to make request routing decisions. Injection of fake capability advertisement messages, or interception and discard of real capability advertisement messages may be used for denial of service (e.g., by falsely advertising or deleting capabilities or preventing capability advertisements from reaching the uCDN). dCDN capability advertisements MUST be authenticated by the uCDN to prevent

unauthorized capability advertisement message injection. uCDN request router capability interface servers MUST be authenticated by the dCDN to prevent unauthorized interception of capability advertisement messages. TLS with client authentication SHOULD be used for all capability interface implementations. Deployments in controlled environments where physical security and IP address white-listing is employed MAY choose not to use TLS.

6. Acknowledgements

The authors would like to thank Ray van Brandenburg and Gilles Bertrand for their expeditious reviews and invaluable comments.

7. Appendix A: Capability Aggregation

The following sections show examples of three aggregation scenarios. In each case, CDN-U is the ultimate uCDN and CDN-P is the penultimate CDN which must perform capabilities aggregation.

7.1. Downstream CDN Aggregation

Figure A1 shows five organizationally different CDNs: CDN-U, CDN-P, and CDNs A, B, and C, the dCDNs of CDN-P which are being aggregated. Given the setup shown in Figure A1, we can construct a number of use cases, based on the coverage areas of each dCDN (i.e., CDNs P, A, B, and C). Note: In all cases, the reachability of the uCDN (i.e., CDN-U) is a don't care as it is assumed that the uCDN knows its own coverage area and is likely to favor itself in most situations, and if it has decided that it needs to delegate to a dCDN, then the only relevant question is if the dCDN can handle the request.

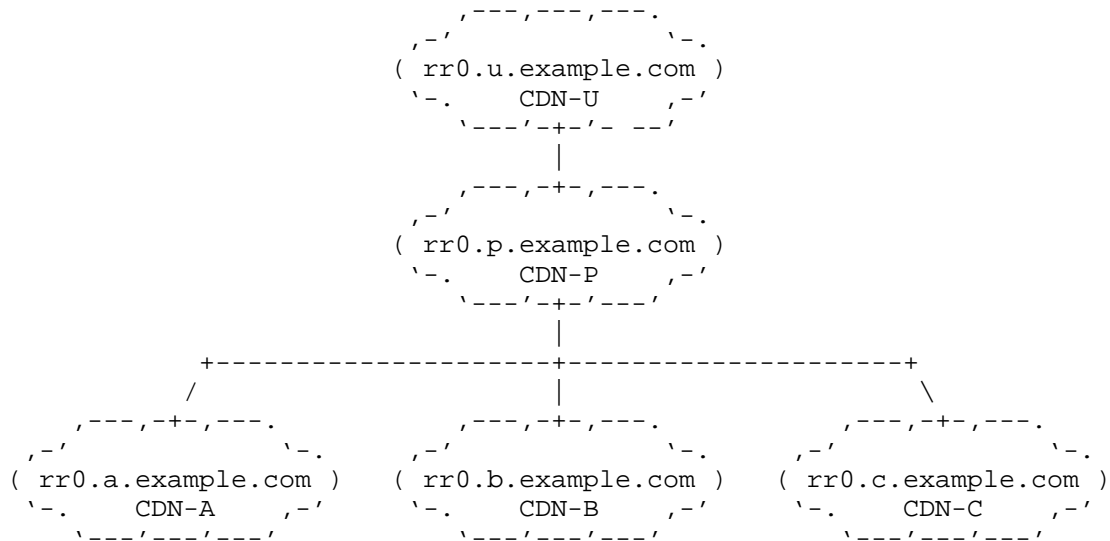


Figure A1: CDNI dCDN Request Router Aggregation

- o None of the four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, each CDN is likely to advertise footprint information with its capabilities, specifying its reachability. When CDN-P advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and CDNs A, B, and C. Note: CDN-P MAY exclude any dCDN, and consequently its footprint, per its own internal aggregation decision criteria.
- o All four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, none of the CDNs is likely to advertise any footprint information as none have any footprint restrictions. When CDN-P advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.
- o Some of the four dCDNs (CDNs P, A, B, and C) have global reachability and some do not. In this case, even though some dCDNs do not have global reachability, the aggregate of some dCDNs having global reachability and some not should still be global reachability (for the given capability). When CDN-P advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one dCDN has global reach as being supported with global reachability. It is up to the CDN-P request router to properly select a dCDN to process individual client requests and not choose a dCDN whose restricted footprint makes it unsuitable for delivering the requested content.

7.2. Internal Request Router Aggregation

Figure A2 shows CDN-U and CDN-P where CDN-P internally has four request routers: the authoritative request router rr0, and three other request routers rr1, rr2, and rr3. The use of multiple request routers may be used to distribute request routing load across resources, possibly in different geographic regions covered by CDN-P. Similar to Figure A1, the setup shown in Figure A2 requires the authoritative request router rr0 in CDN-P to aggregate capabilities information from downstream request routers rr1, rr2, and rr3. The primary difference between the scenario is that the request routers in Figure A2 are logically within the same CDN-P organization. The same reachability scenarios apply to Figure A2 as with Figure A1.

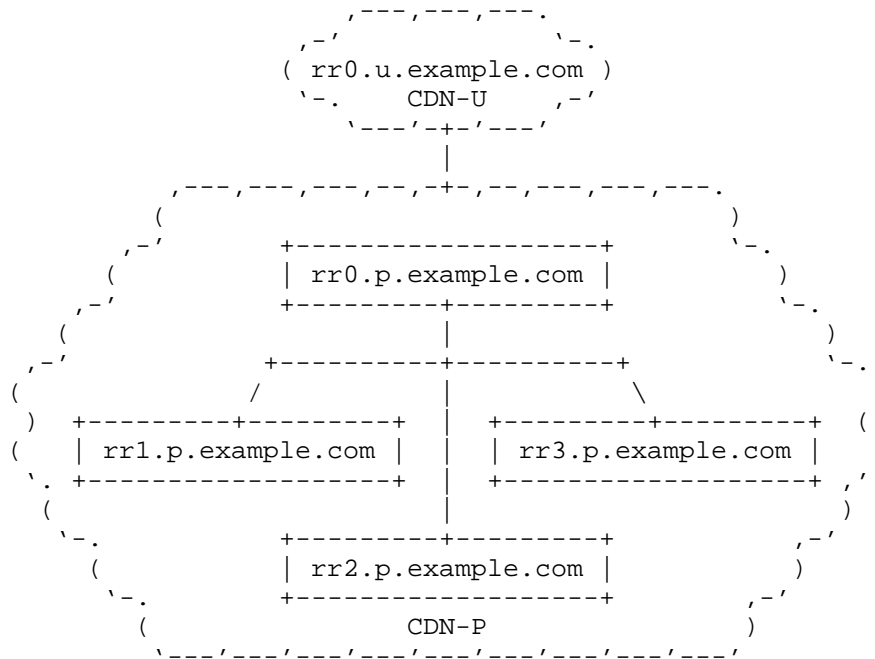


Figure A2: Local CDN Request Router Aggregation

- o None of the four CDN-P request routers have global reachability. In this case, each request router is likely to advertise footprint information with its capabilities, specifying its reachability. When rr0 advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and rr1, rr2, and rr3.
- o All four CDN-P request routers have global reachability. In this case, none of the request routers is likely to advertise any

footprint information as none has any footprint restrictions. When rr0 advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.

- o Some of the four CDN-P request routers have global reachability and some do not. In this case, even though some request routers do not have global reachability, the aggregate of some request routers having global reachability and some not should still be global reachability (for the given capability). When rr0 advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one request router has global reach as being supported with global reachability. It is up to the authoritative request router rr0 to properly select from the other request routers for any given request, and not choose a request router whose restricted footprint makes it unsuitable for delivering the requested content.

7.3. Internal Capability Aggregation

Figure A3 shows CDN-U and CDN-P where the delivery network of CDN-P is segregated by delivery protocol (e.g., RTSP, HTTP, and RTMP). Figure A3 differs from Figures A1 and A2 in that request router rr0 of CDN-P is not aggregating the capabilities advertisements of multiple other downstream request routers, but rather it is managing the disparate capabilities across resources within its own local CDN. Though not every delivery node has the same protocol capabilities, the aggregate delivery protocol capabilities advertised by CDN-A may include all delivery protocols. Note, Figure A3 should not be construed to imply anything about the coverage areas for each delivery protocol. They may all support the same delivery footprint, or they may have different delivery footprints. It is the responsibility of the request router rr0 to properly assign protocol-appropriate delivery nodes to individual content requests. If certain protocols have limited reachability, CDN-P may advertise footprint restrictions for each protocol.

It should be noted that though the delivery protocol capability was selected for this example, the concept of internal capability aggregation applies to all capabilities as discussed below.

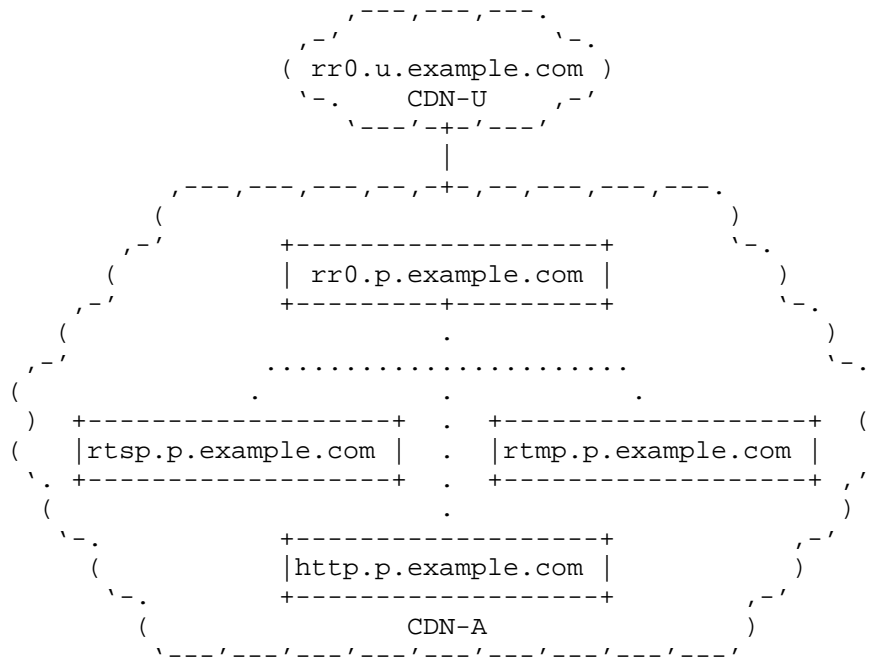


Figure A3: Local CDN Capability Segregation

Another situation in which physical footprint may not matter in an aggregated view has to do with feature support (e.g., new CDNI metadata features or new redirection modes). Situations often arise when phased roll-out of software upgrades, or staging network segregation result in only certain portions of a CDN's resources supporting the new feature set. The dCDN has a few options in this case:

- o Enforce atomic update: The dCDN does not advertise support for the new capability until all resources have been upgraded to support the new capability.
- o Transparent segregation: The dCDN advertises support for the new capability, and when requests are received that require the new capability, the dCDN request router properly selects a resource which supports that capability.
- o Advertised segregation: The dCDN advertises support for the new capability with a footprint restriction allowing the uCDN to make delegation decisions based on the dCDN's limit support.

The level of aggregation employed by the dCDN is likely to vary as

business relationships dictate, however, the capability interface should support all possible modes of operation.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

8.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L., Ed. and B. Davie, Ed., "Framework for CDN Interconnection draft-ietf-cdni-framework-01", July 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements draft-ietf-cdni-requirements-03", June 2012.

Author's Address

Kevin J. Ma
Azuki Systems, Inc.
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.ma@azukisystems.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 24, 2016

K. Ma
Ericsson
J. Seedorf
NEC
April 22, 2016

CDNI Footprint & Capabilities Advertisement Interface
draft-ma-cdni-capabilities-09

Abstract

Content Distribution Network Interconnection (CDNI) is predicated on the ability of downstream CDNs (dCDNs) to handle end-user requests in a functionally equivalent manner to the upstream CDN (uCDN). The uCDN must be able to assess the ability of the dCDN to handle individual requests. The CDNI Footprint & Capabilities Advertisement interface (FCI) is provided for the advertisement of capabilities and the footprints to which they apply by the dCDN to the uCDN. This document describes an approach to implementing the CDNI FCI.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 October 24, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. CDNI FCI Capability Advertisement	4
2.1. CDNI FCI Capability Initialization	5
3. CDNI FCI Capabilities Service	5
3.1. CDNI FCI Map	5
3.1.1. Media Type	5
3.1.2. HTTP Method	5
3.1.3. Accept Input Parameters	6
3.1.4. Capabilities	6
3.1.5. Uses	6
3.1.6. Response	6
3.1.7. CDNI FCI Capabilities	7
3.1.7.1. Delivery Protocol	7
3.1.7.2. Acquisition Protocol	9
3.1.7.3. Redirection Mode	11
3.1.7.4. Logging Capabilities	13
3.1.7.5. Metadata Capabilities	14
4. CDNI FCI Capabilities Filtering Service	15
4.1. Filtered CDNI FCI Map	15
4.1.1. Media Type	15
4.1.2. HTTP Method	15
4.1.3. Accept Input Parameters	15
4.1.4. Capabilities	15
4.1.5. Uses	15
4.1.6. Response	16
4.1.7. Example	16
5. Footprint via ALTO Network Map	16
5.1. ALTO Network Maps	16
5.2. Example ALTO Network Map for CDNI FCI Footprint	16
5.3. Example of ALTO Network Map Footprint in FCI Map	17

6.	IANA Considerations	18
6.1.	ALTO Media Types	19
6.1.1.	ALTO CDNI FCI Map Media Type	19
6.1.2.	ALTO CDNI FCI Map Filter Media Type	20
6.2.	CDNI Footprint Types	22
7.	Security Considerations	22
7.1.	Securing the CDNI Footprint & Capabilities Advertisement interface	22
8.	Acknowledgements	23
9.	References	23
9.1.	Normative References	23
9.2.	Informative References	24
Appendix A.	Capability Aggregation	25
A.1.	Downstream CDN Aggregation	25
A.2.	Internal Request Router Aggregation	27
A.3.	Internal Capability Aggregation	28
Authors' Addresses	30

1. Introduction

The need for footprint and capabilities advertisement in Content Distribution Network Interconnection (CDNI) is described in the CDNI requirements document [RFC7337]. Requirements FCI-1 and FCI-2 describe the need to allow dCDNs to communicate capabilities to the uCDN. Requirement FCI-3 describes how a uCDN may aggregate the footprint and capabilities information for all cascaded dCDNs and use the aggregated information in advertisements to CDNs further upstream. This concept of aggregation can apply to both organizationally different dCDNs (e.g., other CDN providers, or different business units within a larger organization) or logical entities within the same CDN (e.g., using multiple request routers for scalability reasons, to segregate surrogates based on specific protocol support, or to segregate surrogates based on software version or feature level, etc.).

Appendix A contains more detailed descriptions of different footprint and capabilities management scenarios, but it is important to note that it is the ability of the dCDN to service each request in a functionally equivalent manner as the uCDN that is important, not the physical layout of resources through which it services the request. The aggregation of resource knowledge by the dCDN into a simple set of capabilities and their affective footprints, that is then advertised to the uCDN, enables efficient decision making at each delegation point in the CDN interconnection hierarchy.

It is assumed that an authoritative request router in each CDN will be responsible for aggregating and advertising capabilities information in a dCDN and/or receiving and aggregating capabilities

information in the uCDN. The CDNI Footprint & Capabilities Advertisement interface (FCI) along with the CDNI Request Routing Redirection interface (RI) [I-D.ietf-cdni-redirection] make up the CDNI Request Routing Interface. As there is no other centralized CDNI controller, the authoritative request router seems the most logical place for capabilities aggregation to occur, as it is the request router that needs such information to make delegation decisions. The protocol defined herein may be implemented as part of an entity other than an authoritative request router, but for the purposes of this discussion, the authoritative request router is assumed to be the centralized capabilities aggregation point.

Though there is an obvious need for the ability to exchange and update footprint and capability information in real-time, it is assumed that capabilities do not change very often. It is also assumed that the capabilities are not by themselves useful for making delegation decisions. Capability information is assumed to be input into business logic. It is the business logic which provides the algorithms for delegation decision making. The definition of business logic occurs outside the scope of CDNI and outside the timescale of footprint and capability advertisement [I-D.ietf-cdni-footprint-capabilities-antics]. It may be the case that the business logic anticipates and reacts to changes in dCDN capabilities, however, it may also be the case that business logic is tailored through offline processes as dCDN capabilities change. The FCI is agnostic to the business processes employed by any given uCDN. The footprints and capabilities that are advertised over the FCI may be used by the uCDN at its discretion, to implement delegation rules. Setting proper defaults in the business logic should prevent any unwanted delegation from occurring when dCDN capabilities change, however, that is beyond the scope of this discussion.

1.1. Terminology

This document uses the terminology defined in section 1.1 of the CDNI Framework [RFC7336] document.

2. CDNI FCI Capability Advertisement

The FCI is implemented as an ALTO [RFC7285] Service. The ALTO protocol defines an HTTP-based transport through which ALTO service information may be retrieved using either a GET or POST method. The uCDN request router may at any time query the dCDN ALTO FCI Service for the full set of dCDN capability information. The uCDN may use a separate FCI Filter Service to retrieve a subset of the dCDN capability information.

[Ed.: Need to update this with ALTO asynchronous update support.]

[Ed.: Need to update this with ALTO incremental update support.]

2.1. CDNI FCI Capability Initialization

In lieu of any out-of-band pre-configured capability information, when the FCI is first brought up between a uCDN and a dCDN, the uCDN SHOULD assume that the dCDN has no CDNI capabilities. If an out-of-band capability baseline has been exchanged, the uCDN MAY use that information to initialize its capabilities database. In either case, the uCDN SHOULD verify the initial state of the dCDN (as a temporary outage may affect availability in the dCDN).

The dCDN MUST support sending its entire set of capabilities to the uCDN through the ALTO service interface

3. CDNI FCI Capabilities Service

As described in Requirement FCI-2, there is a basic set of capabilities that must be supported by the FCI for the uCDN to be able to determine if the dCDN is functionally able to handle a given request. [I-D.ietf-cdni-footprint-capabilities-semantics] lists mandatory capabilities types:

- o Delivery Protocol
- o Acquisition Protocol
- o Redirection Mode
- o CDNI Logging Capabilities
- o CDNI Metadata Capabilities

To be consistent with the base ALTO service definitions, we use the JSON object definition notation as specified in the ALTO protocol [RFC7285].

3.1. CDNI FCI Map

3.1.1. Media Type

The media type of CDNI FCI Map is "application/alto-cdni-fcimap+json"

3.1.2. HTTP Method

A CDNI FCI Map resource is requested using the HTTP GET method.

3.1.3. Accept Input Parameters

None.

3.1.4. Capabilities

None.

3.1.5. Uses

None.

3.1.6. Response

The data component of a CDNI FCI Map resource is named "fcimap" which is a JSON object of type FCIMapData:

```
object {
  FCIMapData fcimap<0..*>;
} InfoResourceFCIMap : ResponseEntityBase;

object {
  FCICapability capabilities<1..*>;
} FCIMapData;

object {
  JSONString capability-type;
  JSONValue capability-value
  FCIFootprint footprints<0..*>;
} FCICapability;

object {
  JSONString footprint-type;
  JSONString footprint-value<1..*>;
} FCIFootprint;
```

The FCIMapData object contains a CDNI Payload Type [RFC7736] "ptype" which identifies the capability and a "value" object containing the associated Capability Advertisement Object (e.g., delivery-protocols, acquisition-protocols, or redirection-modes, as defined in [I-D.ietf-cdni-footprint-capabilities-semantics]). The FCIMapData object may also contain an optional list of FCIFootprint objects "footprints". The FCIFootprint object specifies a "footprint-type" (as defined by the CDNI Metadata Footprint Types registry, e.g., ipv4cidr, ipv6cidr, asn, or countrycode [I-D.ietf-cdni-metadata]) which identifies the contents and encoding of the individual footprint entries contained in the associated "footprint-value" array.

The "footprints" list MUST NOT contain multiple FCIFootprint objects of the same type. Footprint restriction information MAY be specified using multiple different footprint-types. If no footprint restriction list is specified (or an empty list is specified), it SHALL be understood that all footprint types MUST be reset to "global" coverage.

Note: Further optimization of the footprint object to provide quality information for a given footprint is certainly possible, however, it is not necessary for the basic interconnection of CDNs. The ability to transfer quality information in capabilities advertisements may be desirable and is noted here for completeness, however, the specifics of such mechanisms are outside the scope of this document.

Multiple FCIMapData objects with the same capability type are allowed within a given CDNI FCI Map response as long as the capability option footprint-value do not overlap, i.e., a given capability option value MUST NOT show up in multiple FCIMapData objects within a single CDNI FCI Map response. If multiple FCIMapData objects for a given capability type exist, those capability objects MUST have different footprint restrictions. Capability objects of a given capability type with identical footprint restrictions MUST be combined into a single capability object.

3.1.7. CDNI FCI Capabilities

3.1.7.1. Delivery Protocol

The delivery protocol refers to the protocol over which an end user (EU) has requested content. If a dCDN does not support the protocol requested by the client, then the dCDN is not a viable candidate for delegation.

Though the delivery protocol is specified in the URI scheme (as defined in [RFC3986]) of the client request URL, protocol feature subsets or augmented protocol feature sets MAY be defined and SHOULD correspond with the protocols listed in the CDNI Metadata Protocol Types registry, e.g., http1.1 or https1.1 [I-D.ietf-cdni-metadata].

The following example shows two lists of delivery protocols with different footprints.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 627
Content-Type: application/alto-fcimap+json
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.DeliveryProtocol"
        "capability-value": {
          "delivery-protocols": [
            "http1.1"
          ]
        }
      },
      { "capability-type": "FCI.DeliveryProtocol"
        "capability-value": {
          "delivery-protocols": [
            "https1.1"
          ]
        }
      },
      "footprints": [
        { "footprint-type": "ipv4cidr",
          "footprint-value": [
            "10.1.0.0/16",
            "10.10.10.0/24"
          ]
        }
      ]
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by IPv4 prefix).

A given protocol MUST NOT appear in multiple FCIMapData FCI.DeliveryProtocol object values.

3.1.7.2. Acquisition Protocol

The acquisition protocol refers to the protocol over which an end user (EU) has requested content. If a dCDN does not support the protocol requested by the client, then the dCDN is not a viable candidate for delegation.

Though the acquisition protocol is specified in the URI scheme (as defined in [RFC3986]) of the client request URL, protocol feature subsets or augmented protocol feature sets MAY be defined and SHOULD correspond with the protocols listed in the CDNI Metadata Protocol Types registry, e.g., http1.1 or https1.1 [I-D.ietf-cdni-metadata].

The following example shows two lists of acquisition protocols with different footprints.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 620
Content-Type: application/alto-fcimap+json
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.AcquisitionProtocol"
        "capability-value": {
          "acquisition-protocols": [
            "http1.1"
          ]
        }
      },
      { "capability-type": "FCI.AcquisitionProtocol"
        "capability-value": {
          "acquisition-protocols": [
            "https1.1"
          ]
        }
      },
      "footprints": [
        { "footprint-type": "asn",
          "footprint-value": [
            "AS0",
            "AS65535"
          ]
        }
      ]
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by Autonomous System number).

A given protocol MUST NOT appear in multiple FCIMapData FCI.AcquisitionProtocol value objects.

3.1.7.3. Redirection Mode

The redirection mode refers to the method(s) employed by request routers to perform request redirection. The CDNI framework [RFC7336] document describes four possible request routing modes:

- o DNS iterative (DNS-I)
- o DNS recursive (DNS-R)
- o HTTP iterative (HTTP-I)
- o HTTP recursive (HTTP-R)

[I-D.ietf-cdni-footprint-capabilities-semantics] defines the "CDNI Capabilities Redirection Modes" registry and the initial supported redirection mode values shown in parentheses above.

If a dCDN supports only a specific mode or subset of modes that does not overlap with the modes supported by the uCDN, then the dCDN might not be a viable candidate for delegation.

The following example shows two lists of redirection modes with different footprints.


```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json
```

```
HTTP/1.1 200 OK
Content-Length: 767
Content-Type: application/alto-fcimap+json
```

```
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.RedirectionMode",
        "capability-value": {
          "redirection-modes": [
            "DNS-I",
            "HTTP-I"
          ]
        }
      },
      { "capability-type": "FCI.RedirectionMode",
        "capability-value": {
          "redirection-modes": [
            "DNS-R",
            "HTTP-R"
          ]
        }
      },
      "footprints": [
        { "footprint-type": "countrycode",
          "footprint-value": [
            "SE"
          ]
        },
        { "footprint-type": "ipv6cidr",
          "footprint-value": [
            "9876:5432::1/36"
          ]
        }
      ]
    }
  }
}
```

In the above example, iterative redirection is supported globally, while recursive redirection is only supported in a restricted

footprint (in this case, specified by both Country Code and IPv6 prefix).

A given mode MUST NOT appear in multiple FCIMapData FCI.RedirectionMode object values.

3.1.7.4. Logging Capabilities

[I-D.ietf-cdni-logging] describes the "cdni_http_request_v1" logging record-types and optional vs. mandatory-to-implement logging fields for that record-type. It also allows new logging record-types and logging fields to be defined which would be optional for a dCDN to implement.

If a dCDN does not support certain logging parameters which may affect billing agreements or legal requirements of the uCDN, then the dCDN is not a viable candidate for delegation.

3.1.7.4.1. CDNI Logging Capability Object Serialization

The following shows an example of CDNI Logging Capability Object Serialization, for a CDN that supports the optional Content Collection ID logging field (but not the optional Session ID logging field) for the "cdni_http_request_v1" record type.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Logging",
      "capability-value": {
        "record-type": "cdni_http_request_v1",
        "fields": [ "s-ccid" ]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

The next example shows the CDNI Logging Capability Object Serialization, for a CDN that supports all optional fields for the "cdni_http_request_v1" record type.

```

{
  "capabilities": [
    {
      "capability-type": "FCI.Logging",
      "capability-value": {
        "record-type": "cdni_http_request_v1"
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}

```

3.1.7.5. Metadata Capabilities

[I-D.ietf-cdni-metadata] describes GenericMetadata types which may be optional for a dCDN to implement, but which, if present, are mandatory-to-enforce. It also allows for new GenericMetadata to be defined which would be optional for a dCDN to implement.

If a dCDN does not support certain GenericMetadata types which are designated mandatory-to-enforce and may affect the correctness or security of the content being delivered, then the dCDN is not a viable candidate for delegation.

3.1.7.5.1. CDNI Metadata Capability Object Serialization

The following shows an example of CDNI Metadata Capability Object Serialization, for a CDN that supports only the SourceMetadata GenericMetadata type (i.e., it can acquire and deliver content, but cannot enforce and security policies, e.g., time, location, or protocol ACLs).

```

{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": ["MI.SourceMetadata"]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}

```

The next example shows the CDNI Metadata Capability Object Serialization, for a CDN that supports only structural metadata (i.e., it can parse metadata as a transit CDN, but cannot enforce security policies or deliver content).

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": []
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

4. CDNI FCI Capabilities Filtering Service

4.1. Filtered CDNI FCI Map

4.1.1. Media Type

Since a Filtered CDNI FCI Map is still a CDNI FCI Map, it uses the media type defined for CDNI FCI Map (see Section 3.1.1).

4.1.2. HTTP Method

A Filtered CDNI FCI Map is requested using the HTTP POST method.

4.1.3. Accept Input Parameters

TBD.

4.1.4. Capabilities

None.

4.1.5. Uses

TBD.

4.1.6. Response

The format is the same as unfiltered CDNI FCI Map (see Section 3.1.6).

4.1.7. Example

TBD.

5. Footprint via ALTO Network Map

5.1. ALTO Network Maps

The ALTO Protocol offers an information service "ALTO map service" that provides information to ALTO clients in the form of Network Map and Cost Map [RFC7285]. As an alternative to the explicit definition of a CDNI Footprint Type (e.g., ipv4cidr, ipv6cidr, as, countrycode), a reference to an ALTO network map can be used to define an FCI footprint. To enable such referencing to ALTO network maps, a new CDNI Footprint Type "altonetworkmap" is defined (see also Section 6.2).

The first altonetworkmap entry must be a URI for accessing the ALTO server that hosts the ALTO network map (as defined in the ALTO protocol specification [RFC7285]). All subsequent altonetworkmap entries must be of type PIDName (as defined in [RFC7285], where the PIDName corresponds to a PID in the ALTO network map referenced by the preceding URI. Parsing and processing of an ALTO network map follows the ALTO protocol specification [RFC7285].

5.2. Example ALTO Network Map for CDNI FCI Footprint

An ALTO client can retrieve a network map of media type 'application/alto-networkmap+json' under a given URI (e.g., 'http://alto.example.com/fcifootprint001') with a GET request for a network map as specified in the ALTO protocol [RFC7285]. The following network map would convey to a uCDN that the given dCDN (which would provide the map) has three footprints called "south-france" and "germany", and provides the corresponding IPv4 address ranges for these footprints.

```
GET /networkmap HTTP/1.1
Host: http://alto.example.com/fcifootprint001
Accept: application/alto-networkmap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 319
Content-Type: application/alto-networkmap+json

{
  "meta" : {
    "vtag": [
      { "resource-id": "my-eu-netmap",
        "tag": "1266506139"
      }
    ]
  },
  "network-map" : {
    "south-france" : {
      "ipv4" : [ "192.0.2.0/24", "198.51.100.0/25" ]
    },
    "germany" : {
      "ipv4" : [ "192.0.3.0/24" ]
    }
  }
}
```

5.3. Example of ALTO Network Map Footprint in FCI Map

To reference an ALTO network map as an FCI footprint, set the footprint-type to "altonetworkmap", and set the first entry in the footprint-value to the URI of the ALTO server hosting the network map, followed by a list of PID names contained in the network map.

The following example shows two lists of delivery protocols (see Section 3.1.7.1), with the second having an ALTO network map footprint.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json
```

```
HTTP/1.1 200 OK
Content-Length: 618
Content-Type: application/alto-fcimap+json
```

```
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [
          "http1.1"
        ]
      },
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [
          "values": [
            "https1.1"
          ],
          "footprints": [
            { "footprint-type": "altonetworkmap",
              "footprint-value": [
                "http://alto.example.com/fcifootprint001",
                "germany",
                "south-france"
              ]
            }
          ]
        ]
      }
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by an ALTO network map for Germany and Southern France).

6. IANA Considerations

6.1. ALTO Media Types

This document requests the registration of the following media types:

Type	Subtype
application	alto-cdni-fcimap+json
application	alto-cdni-fcimapfilter+json

6.1.1. ALTO CDNI FCI Map Media Type

Type name: application

Subtype name: alto-cdni-fcimap+json

Required parameters: none

Optional parameters: none

Encoding considerations:

Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations:

Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of [RFC7285]. Additional security considerations for the CDNI Footprint & Capabilities Advertisement interface are discussed in Section 7.

Interoperability considerations:

[RFC7285] and RFCthis specify the format of conforming messages and the interpretation thereof. [RFC Editor: Please replace RFCthis with the published RFC number for this document.]

Published specification: RFCthis [RFC Editor: Please replace RFCthis with the published RFC number for this document.]

Applications that use this media type:

ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Fragment identifier considerations: N/A

Additional information: N/A

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information:

Kevin Ma <kevin.j.ma@ericsson.com>

Intended usage: LIMITED USE

Restrictions on usage:

This media type is intended only for use in CDNI Footprint & Capabilities Advertisement interface protocol message exchanges.

Author: IETF CDNI working group

Change controller: IETF CDNI working group

Provisional registration: no

6.1.2. ALTO CDNI FCI Map Filter Media Type

Type name: application

Subtype name: alto-cdni-fcimapfilter+json

Required parameters: none

Optional parameters: none

Encoding considerations:

Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations:

Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of

[RFC7285]. Additional security considerations for the CDNI Footprint & Capabilities Advertisement interface are discussed in Section 7.

Interoperability considerations:

[RFC7285] and RFCthis specify the format of conforming messages and the interpretation thereof. [RFC Editor: Please replace RFCthis with the published RFC number for this document.]

Published specification: RFCthis [RFC Editor: Please replace RFCthis with the published RFC number for this document.]

Applications that use this media type:

ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Fragment identifier considerations: N/A

Additional information: N/A

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information:

Kevin Ma <kevin.j.ma@ericsson.com>

Intended usage: LIMITED USE

Restrictions on usage:

This media type is intended only for use in CDNI Footprint & Capabilities Advertisement interface protocol message exchanges.

Author: IETF CDNI working group

Change controller: IETF CDNI working group

Provisional registration: no

6.2. CDNI Footprint Types

This document requests the following addition to the "CDNI Metadata Footprint Types" registry:

Footprint Type	Description	Specification
altonetworkmap	URI of an ALTO Server hosting an ALTO network map, followed by a list of PID-names	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

7. Security Considerations

There are a number of security concerns associated with the FCI. The FCI essentially provides configuration information which the uCDN uses to make request routing decisions. Injection of fake capability advertisement messages or the interception and discard of real capability advertisement messages may be used for denial of service (e.g., by falsely advertising or deleting capabilities or preventing capability advertisements from reaching the uCDN). FCI messages may also be monitored to detect when CDN performance degrades or outages occur. Such information may be considered private by the dCDN.

dCDN capability advertisements MUST be authenticated by the uCDN to prevent unauthorized capability injection. uCDN FCI servers MUST be authenticated by the dCDN to prevent unauthorized interception of ALTO messages. Encryption MUST be used to ensure confidentiality of the dCDN's private messages.

7.1. Securing the CDNI Footprint & Capabilities Advertisement interface

An implementation of the CDNI Footprint & Capabilities Advertisement interface MUST support TLS transport as per [RFC2818] and [RFC7230]. The use of TLS for transport of the CDNI metadata interface messages allows:

- o The dCDN and uCDN to authenticate each other.

and, once they have mutually authenticated each other, it allows:

- o The dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI FCI messages from an authorized CDN);

- o CDNI FCI messages to be transmitted with confidentiality; and
- o The integrity of the CDNI FCI messages to be protected during the exchange.

In an environment where any such protection is required, TLS MUST be used (including authentication of the remote end) by the server-side (uCDN) and the client-side (dCDN) of the CDNI Footprint & Capabilities Advertisement interface unless alternate methods are used for ensuring the confidentiality of the information in the CDNI FCI messages (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

When TLS is used, the general TLS usage guidance in [RFC7525] MUST be followed.

8. Acknowledgements

The authors would like to thank Jon Peterson, Ray van Brandenburg, Gilles Bertrand, and Scott Wainner for their timely reviews and invaluable comments.

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

9. References

9.1. Normative References

- [I-D.ietf-cdni-footprint-capabilities-semantic]
Seedorf, J., Peterson, J., Previdi, S., Brandenburg, R., and K. Ma, "CDNI Request Routing: Footprint and Capabilities Semantics", draft-ietf-cdni-footprint-capabilities-semantic-16 (work in progress), April 2016.
- [I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-25 (work in progress), April 2016.

- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma,
"CDN Interconnection Metadata", draft-ietf-cdni-
metadata-15 (work in progress), April 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
Protocol (HTTP/1.1): Message Syntax and Routing",
RFC 7230, DOI 10.17487/RFC7230, June 2014,
<<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S.,
Previdi, S., Roome, W., Shalunov, S., and R. Woundy,
"Application-Layer Traffic Optimization (ALTO) Protocol",
RFC 7285, DOI 10.17487/RFC7285, September 2014,
<<http://www.rfc-editor.org/info/rfc7285>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre,
"Recommendations for Secure Use of Transport Layer
Security (TLS) and Datagram Transport Layer Security
(DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May
2015, <<http://www.rfc-editor.org/info/rfc7525>>.

9.2. Informative References

- [I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing
Redirection interface for CDN Interconnection", draft-
ietf-cdni-redirection-18 (work in progress), April 2016.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818,
DOI 10.17487/RFC2818, May 2000,
<<http://www.rfc-editor.org/info/rfc2818>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March
2014, <<http://www.rfc-editor.org/info/rfc7159>>.

- [RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <<http://www.rfc-editor.org/info/rfc7336>>.
- [RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <<http://www.rfc-editor.org/info/rfc7337>>.
- [RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <<http://www.rfc-editor.org/info/rfc7736>>.

Appendix A. Capability Aggregation

The following sections show examples of three aggregation scenarios. In each case, CDN-U is the ultimate uCDN and CDN-P is the penultimate CDN which must perform capabilities aggregation.

A.1. Downstream CDN Aggregation

Figure A1 shows five organizationally different CDNs: CDN-U, CDN-P, and CDNS A, B, and C, the dCDNs of CDN-P which are being aggregated. Given the setup shown in Figure A1, we can construct a number of use cases, based on the coverage areas of each dCDN (i.e., CDNs P, A, B, and C). Note: In all cases, the reachability of the uCDN (i.e., CDN-U) is a don't care as it is assumed that the uCDN knows its own coverage area and is likely to favor itself in most situations, and if it has decided that it needs to delegate to a dCDN, then the only relevant question is if the dCDN can handle the request.

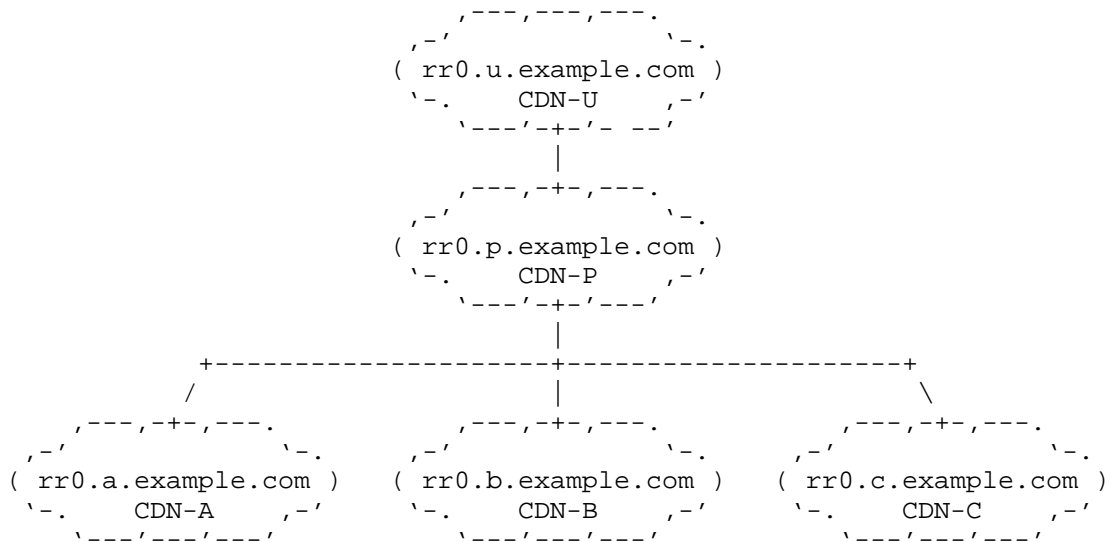


Figure A1: CDNI dCDN Request Router Aggregation

- o None of the four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, each CDN is likely to advertise footprint information with its capabilities, specifying its reachability. When CDN-P advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and CDNs A, B, and C. Note: CDN-P MAY exclude any dCDN, and consequently its footprint, per its own internal aggregation decision criteria.
- o All four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, none of the CDNs is likely to advertise any footprint information as none have any footprint restrictions. When CDN-P advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.
- o Some of the four dCDNs (CDNs P, A, B, and C) have global reachability and some do not. In this case, even though some dCDNs do not have global reachability, the aggregate of some dCDNs having global reachability and some not should still be global reachability (for the given capability). When CDN-P advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one dCDN has global reach as being supported with global reachability. It is up to the CDN-P request router to properly select a dCDN to process individual client requests and not choose a dCDN whose restricted footprint makes it unsuitable for delivering the requested content.

A.2. Internal Request Router Aggregation

Figure A2 shows CDN-U and CDN-P where CDN-P internally has four request routers: the authoritative request router rr0, and three other request routers rr1, rr2, and rr3. The use of multiple request routers may be used to distribute request routing load across resources, possibly in different geographic regions covered by CDN-P. Similar to Figure A1, the setup shown in Figure A2 requires the authoritative request router rr0 in CDN-P to aggregate capabilities information from downstream request routers rr1, rr2, and rr3. The primary difference between the scenario is that the request routers in Figure A2 are logically within the same CDN-P organization. The same reachability scenarios apply to Figure A2 as with Figure A1.

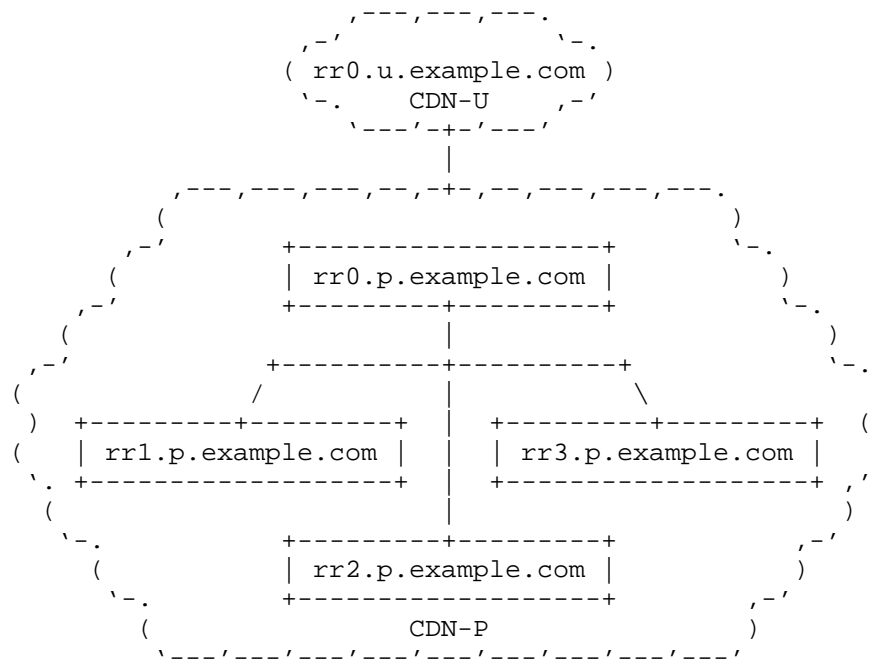


Figure A2: Local CDN Request Router Aggregation

- o None of the four CDN-P request routers have global reachability. In this case, each request router is likely to advertise footprint information with its capabilities, specifying its reachability. When rr0 advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and rr1, rr2, and rr3.
- o All four CDN-P request routers have global reachability. In this case, none of the request routers is likely to advertise any

footprint information as none has any footprint restrictions. When rr0 advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.

- o Some of the four CDN-P request routers have global reachability and some do not. In this case, even though some request routers do not have global reachability, the aggregate of some request routers having global reachability and some not should still be global reachability (for the given capability). When rr0 advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one request router has global reach as being supported with global reachability. It is up to the authoritative request router rr0 to properly select from the other request routers for any given request, and not choose a request router whose restricted footprint makes it unsuitable for delivering the requested content.

A.3. Internal Capability Aggregation

Figure A3 shows CDN-U and CDN-P where the delivery network of CDN-P is segregated by delivery protocol (e.g., RTSP, HTTP, and RTMP). Figure A3 differs from Figures A1 and A2 in that request router rr0 of CDN-P is not aggregating the capabilities advertisements of multiple other downstream request routers, but rather it is managing the disparate capabilities across resources within its own local CDN. Though not every delivery node has the same protocol capabilities, the aggregate delivery protocol capabilities advertised by CDN-A may include all delivery protocols. Note, Figure A3 should not be construed to imply anything about the coverage areas for each delivery protocol. They may all support the same delivery footprint, or they may have different delivery footprints. It is the responsibility of the request router rr0 to properly assign protocol-appropriate delivery nodes to individual content requests. If certain protocols have limited reachability, CDN-P may advertise footprint restrictions for each protocol.

It should be noted that though the delivery protocol capability was selected for this example, the concept of internal capability aggregation applies to all capabilities as discussed below.

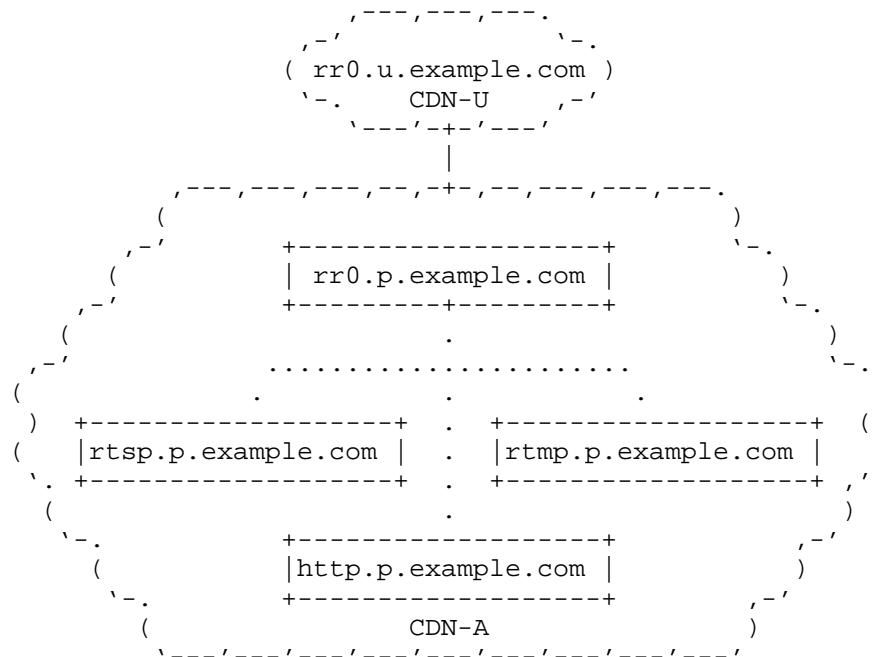


Figure A3: Local CDN Capability Segregation

Another situation in which physical footprint may not matter in an aggregated view has to do with feature support (e.g., new CDNI metadata features or new redirection modes). Situations often arise when phased roll-out of software upgrades, or staging network segregation result in only certain portions of a CDN's resources supporting the new feature set. The dCDN has a few options in this case:

- o Enforce atomic update: The dCDN does not advertise support for the new capability until all resources have been upgraded to support the new capability.
- o Transparent segregation: The dCDN advertises support for the new capability, and when requests are received that require the new capability, the dCDN request router properly selects a resource which supports that capability.
- o Advertised segregation: The dCDN advertises support for the new capability with a footprint restriction allowing the uCDN to make delegation decisions based on the dCDN's limit support.

The level of aggregation employed by the dCDN is likely to vary as business relationships dictate, however, the FCI should support all possible modes of operation.

Authors' Addresses

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.j.ma@ericsson.com

Jan Seedorf
NEC
Kurfuerstenanlage 36
Heidelberg 69115
Germany

Phone: +49 6221 4342 221
Fax: +49 6221 4342 155
Email: seedorf@neclab.eu

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 3, 2013

R. Murray
B. Niven-Jenkins
Velocix (Alcatel-Lucent)
August 30, 2012

CDN Interconnect Triggers
draft-murray-cdni-triggers-01

Abstract

This document proposes a mechanism for a CDN to trigger activity in an interconnected CDN that is configured to deliver content on its behalf. The upstream CDN can use this mechanism to request that the downstream CDN pre-positions metadata or content, or that it re-validate or purge metadata or content. The upstream CDN can monitor the status of activity that it has triggered in the downstream CDN.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 March 3, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Terminology	5
2. Model for CDNI Triggers	5
2.1. Timing of Triggered Activity	7
2.2. Trigger Results	7
3. Collections of Trigger Status Resources	8
4. CDNI Trigger interface	8
4.1. Creating Triggers	10
4.2. Checking Status	11
4.2.1. Polling Trigger Status Resource collections	11
4.2.2. Polling Trigger Status Resources	11
4.3. Deleting Triggers	11
4.4. Expiry of Trigger Status Resources	12
5. Properties of Triggers	12
5.1. Properties of Trigger Requests	12
5.1.1. Content URLs	13
5.2. Properties of Trigger Status Resources	13
5.3. Properties of Trigger Collections	14
5.4. Trigger Resource Simple Data Type Descriptions	15
5.4.1. TriggerType	15
5.4.2. TriggerStatus	15
5.4.3. URLs	15
5.4.4. UrlPatterns	15
5.4.5. AbsoluteTime	16
6. JSON Encoding of Objects	16
6.1. JSON Encoding of Embedded Types	17
6.1.1. TriggerType	17
6.1.2. TriggerStatus	17
6.1.3. Metadata and Content References	17
6.1.4. Pattern	18
6.1.5. Relationship	18
6.2. MIME Media Types	19
7. Examples	19
7.1. Creating Triggers	20
7.1.1. Preposition	20
7.1.2. Invalidate	21
7.2. Examining Trigger Status	23

7.2.1.	Collection of All Triggers	23
7.2.2.	Filtered Collections of Triggers	24
7.2.3.	Trigger Status Resources	25
7.2.4.	Polling for Change	28
7.2.5.	Cancelling or Removing a Trigger	30
8.	IANA Considerations	32
9.	Security Considerations	32
10.	Acknowledgements	32
11.	References	32
11.1.	Normative References	32
11.2.	Informative References	32
	Authors' Addresses	33

1. Introduction

[I-D.ietf-cdni-problem-statement] introduces the Problem scope for CDN Interconnection (CDNI) and lists the four categories of interfaces that may be used to compose a CDNI solution (Control, Metadata, Request Routing, Logging).

[I-D.ietf-cdni-framework] expands on the information provided in [I-D.ietf-cdni-problem-statement] and describes each of the interfaces and the relationships between them in more detail.

This draft concentrates on the "High" and "Medium" priority requirements for the CDNI Control Interface identified in section 4 of [I-D.ietf-cdni-requirements], reproduced here for convenience:

CNTL-1 [HIGH] The CDNI Control interface shall allow the Upstream CDN to request that the Downstream CDN (and, if cascaded CDNs are supported by the solution, that the potential cascaded Downstream CDNs) perform the following actions on an object or object set:

- * Mark an object(s) and/or its CDNI metadata as "stale" and revalidate them before they are delivered again.
- * Delete an object(s) and/or its CDNI metadata from the CDN surrogates and any storage. Only the object(s) and CDNI metadata that pertain to the requesting Upstream CDN are allowed to be purged.

CNTL-2 [HIGH] The CDNI Control interface shall allow the downstream CDN to report on the completion of these actions (by itself, and if cascaded CDNs are supported by the solution, by potential cascaded Downstream CDNs), in a manner appropriate for the action (e.g. synchronously or asynchronously).

CNTL-3 [HIGH] The CDNI Control interface shall support initiation and control by the Upstream CDN of pre-positioned CDNI metadata acquisition by the Downstream CDN.

CNTL-4 [MED] The CDNI Control interface should support initiation and control by the Upstream CDN of pre-positioned content acquisition by the Downstream CDN.

This document does not consider those parts of the control interface that relate to configuration, bootstrapping or authentication of CDN Interconnect interfaces.

- o Section 2 outlines the model for the Trigger Interface at a high level.
- o Section 3 describes collections of Trigger Resources.
- o Section 4 defines the RESTful web service provided by dCDN.

- o Section 5 lists properties of Trigger Requests and Status Resources.
- o Section 6 defines a JSON encoding for Trigger Requests and Status Resources.
- o Section 7 contains example messages.

1.1. Terminology

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

2. Model for CDNI Triggers

A trigger, sent from uCDN to dCDN, is a request for dCDN to do some work relating to data originating from uCDN.

The trigger may request action on either metadata or content, the following actions can be requested:

- o preposition - used to instruct dCDN to fetch metadata from uCDN, or content from any origin including uCDN.
- o invalidate - used to instruct dCDN to revalidate specific metadata or content before re-using it.
- o purge - used to instruct dCDN to delete specific metadata or content.

The CDNI trigger interface is a RESTful web service offered by dCDN. It allows creation and deletion of triggers, and tracking of the triggered activity. When dCDN accepts a trigger it creates a resource describing status of the triggered activity, a Trigger Status Resource. The uCDN may poll Trigger Status Resources to monitor progress.

Requests to invalidate and purge metadata or content apply to all variants of that data with a given URI.

The dCDN maintains a collection of Trigger Status Resources for each uCDN, each uCDN only has access to its own collection and the location of that collection is shared when CDN interconnection is established.

To trigger activity in dCDN, uCDN will POST to the collection of Trigger Status Resources. If dCDN accepts the trigger, it creates a new Trigger Status Resource and returns its location to uCDN. To monitor progress, uCDN may GET the Trigger Status Resource. To cancel a trigger, or remove a trigger from the collection once its activity has been completed, uCDN may DELETE the Trigger Status

Resource.

In addition to the collection of all Trigger Status Resources for uCDN, uCDN shall have access to filtered views of that collection. These filtered views are defined in Section 3 and include collections of active and completed triggers. These collections provide a mechanism for polling the status of multiple jobs.

Figure 1 is an example showing the basic message flow used by the uCDN to trigger activity in dCDN, and for uCDN to discover the status of that activity. Only successful triggering is shown. Examples of the messages are given in Section 7.

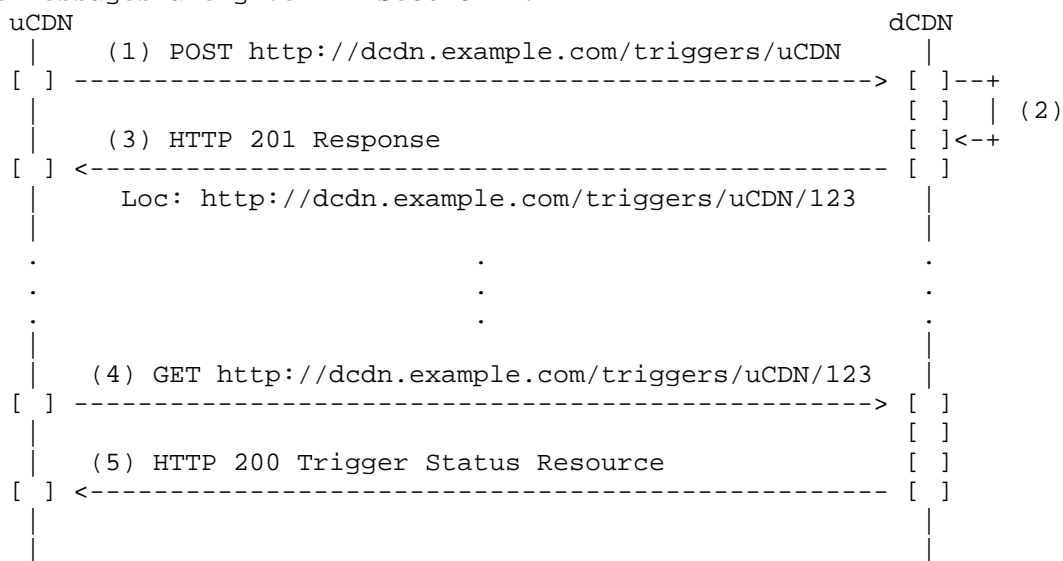


Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are:

1. uCDN triggers action in dCDN by posting to a collection of Trigger Status Resources, "http://dcdn.example.com/triggers/uCDN". The URL of this was given to uCDN when the trigger interface was established.
2. dCDN authenticates the request, validates the trigger and if it accepts the request, creates a new Trigger Status Resource.
3. dCDN responds to uCDN with an HTTP 201 response status, and the location of the Trigger Status Resource.
4. uCDN may repeatedly poll the Trigger Status Resource in dCDN.
5. dCDN responds with the Trigger Status Resource, describing progress or results of the triggered activity.

The remainder of this document describes the messages, Trigger Status Resources, and collections of Trigger Status Resources in more detail.

2.1. Timing of Triggered Activity

Timing of triggered activity is under dCDN control, including its start-time and pacing of the activity in the network.

Invalidate and purge triggers MUST be applied to all data acquired before the trigger was created in dCDN. The dCDN MAY apply the triggers to data acquired after trigger creation.

If uCDN triggers preposition as well as invalidate or purge of data in a single request, dCDN MUST ensure that data is not immediately invalidated or purged as it is prepositioned as a result of that same trigger. It MUST therefore also pass those triggers on to any downstream CDNs that have the data in a single Trigger Request to ensure that the further downstream CDN behaves in the same way.

Note that each CDN may perform preposition and invalidate/purge activity in parallel even if triggered by the same request, but only if it is able to ensure that newly prepositioned data is not affected. It may achieve this by, for example, only invalidating/purging data acquired before the trigger was created.

2.2. Trigger Results

Each Trigger Request may operate on multiple data items, and may request different actions. The trigger shall be reported as successful only if all actions can be completed successfully.

If any part of the trigger request fails, the trigger shall be reported as failed, and the error property in the Trigger Status Resource will be used to enumerate which actions failed and the reasons for failure.

If a dCDN is also acting as uCDN in a cascade, it MUST forward triggers to any downstream CDNs that may have data affected by the trigger. The trigger MUST NOT be reported as complete in a CDN until it is complete in all of its downstream CDNs. A trigger MAY be reported as failed as soon as it fails in a CDN or in any of its downstream CDNs.

[Editor's Note: behaviour in case of partial failure is an open issue to be addressed in a later version of this draft - should the CDN report failure immediately or attempt to complete all actions first, and exactly how are partial failures reported?]

3. Collections of Trigger Status Resources

As described in Section 2, Trigger Status Resources exist in dCDN to report the status of activity triggered by each uCDN.

A collection of Trigger Status Resources is a resource that contains a reference to each Trigger Status Resource in that collection.

To trigger activity in dCDN, uCDN creates a new Trigger Status Resource by posting to dCDN's collection of uCDN's Trigger Status Resources. The URL of each Trigger Status Resource is generated by the dCDN when it accepts the trigger, and returned to uCDN. This immediately enables uCDN to check the status of that trigger.

The dCDN must present a different set of Trigger Status Resources to each interconnected uCDN, only Trigger Status Resources belonging to a uCDN shall be visible to it. The dCDN may, for example, achieve this by offering different collection URLs to uCDNs, or by filtering the response based on the client uCDN.

The dCDN resource representing the collection of all uCDN's Trigger Status Resources is accessible to uCDN. This collection lists all uCDN triggers that have been accepted by dCDN, and have not yet been deleted by uCDN or expired and removed by dCDN.

In order to allow uCDN to check status of multiple jobs in a single request, dCDN shall also maintain collections representing filtered views of the collection of all Trigger Status Resources. The filtered collections are:

- o Pending - Trigger Status Resources for triggers that have been accepted, but not yet acted upon.
- o Active - Trigger Status Resources for triggered activity that is currently being processed in dCDN.
- o Complete - Trigger Status Resources representing activity that completed successfully.
- o Failed - Trigger Status Resources representing activity that failed.

4. CDNI Trigger interface

This section describes an interface to enable an upstream CDN to trigger defined activities in a downstream CDN. The interface is intended to be independent of the set of activities defined now, or that may be defined in future.

The CDNI Trigger interface is built on the principles of RESTful web services. Requests are made over HTTP, and the HTTP Method defines

the operation the request would like to perform. The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses from servers implementing the CDNI Triggers interface that contain a response body SHOULD include an ETag to enable validation of cached versions of returned resources.

Servers implementing the CDNI Trigger interface MUST support the HTTP GET, HEAD, POST and DELETE methods. The only representation specified in this document is JSON.

Trigger Requests are POSTed to a URI in dCDN. If the trigger is accepted by dCDN, it creates a Trigger Status Resource and returns its URI to dCDN in an HTTP 201 response. The triggered activity can then be monitored by uCDN using that resource and the collections described in Section 3.

The URI that Trigger Requests should be POSTed to needs to be either discovered by or configured in the upstream CDN. Performing a GET on that URI retrieves a collection of the URIs of all Trigger Status Resources. The URI of each Trigger Status Resource is also returned to uCDN when it is created. This means all Trigger Status Resources can be discovered, so CDNI Trigger servers are free to assign whatever structure they desire to the URIs for CDNI Trigger resources. CDNI Trigger clients MUST NOT make any assumptions regarding the structure of CDNI Trigger URIs or the mapping between CDNI Trigger objects and their associated URIs. Therefore any URIs present in the examples below are purely illustrative and are not intended to impose a definitive structure on CDNI Trigger interface implementations.

The CDNI Trigger interface builds on top of HTTP, so CDNI Trigger servers may make use of any HTTP feature when implementing the CDNI Trigger interface. For example, a CDNI Trigger server may make use of HTTP's caching mechanisms to indicate that the returned response/representation has not been modified since it was last returned, reducing the processing needed to determine whether the status of triggered activity has changed.

This specification is neutral with regard to the transport below the HTTP layer. For example, it is anticipated that decisions on use of HTTPS for other CDNI interfaces will be adopted for Triggers.

Discovery of the CDNI Triggers Interface is outside the scope of this document. It is anticipated that a common mechanism for discovery of all CDNI interfaces will be defined.

The dCDN must ensure that activity triggered by uCDN only affects metadata or content originating from that uCDN. Since only one CDN can be authoritative for a given item of metadata or content, this requirement means there cannot be any "loops" in trigger requests between CDNs.

4.1. Creating Triggers

To create a new trigger, uCDN makes an HTTP POST to the unfiltered collection of its triggers. The request body of that POST is a Trigger Request.

dCDN validates and authenticates that request, if it is malformed or uCDN does not have sufficient access rights it MAY reject the request immediately. In this case, it SHALL respond with an appropriate 4xx HTTP error code and no resource shall be created on dCDN.

If the request is accepted, uCDN SHALL create a new Trigger Status Resource. The HTTP response to dCDN SHALL have status code 201 and the URI of the Trigger Status Resource in the Location header field. The HTTP response MAY include the content of the newly created Trigger Status Resource, this is recommended particularly in cases where the trigger has completed immediately.

Once a Trigger Status Resource has been created dCDN MUST NOT re-use its location, even after that resource has been removed through deletion or expiry.

The "request" property of the Trigger Status Resource SHALL contain the information posted in the body of the Trigger Request. Note that this need not be a byte-for-byte copy. For example, in the JSON representation the dCDN may re-serialise the information differently.

If the trigger is queued by dCDN for later action, the "status" property of the Trigger Status Resource SHALL be "pending". Once trigger processing has started the "status" SHALL be "active".

A trigger may result in no activity in dCDN if, for example, it is an invalidate or purge request for data the dCDN has not acquired, or a prepopulate request for data it has already acquired. In this case, the "status" of the Trigger Status Resource shall be "complete" and the Trigger Status Resource shall be added to the dCDN collection of Complete Triggers.

Once created, Trigger Status Resources may be deleted by uCDN but not modified. The dCDN MUST reject PUT and POST requests from uCDN to Trigger Status Resources using HTTP status code 403.

4.2. Checking Status

The uCDN has two ways to check progress of activity it has triggered in dCDN, described in the following sections.

Because the triggers protocol is based on HTTP, Entity Tags may be used by the uCDN as cache validators, as defined in section 3.11 of [RFC2616], to check for change in status of a resource or collection of resources without re-fetching the whole resource or collection.

The dCDN should use the cache control headers for responses to GETs for Trigger Status Resources and Collections to indicate the frequency at which it recommends uCDN should poll for change.

4.2.1. Polling Trigger Status Resource collections

uCDN can fetch the collection of its Trigger Status Resources, or filtered views of that collection.

This makes it possible to poll status of all triggered activity in a single request. If dCDN moves a Trigger Status Resource from the Active to the Completed collection, uCDN may chose to fetch the result of that activity.

When polling in this way, uCDN may choose to use HTTP Entity Tags to monitor for change, rather than repeatedly fetching the whole collection.

4.2.2. Polling Trigger Status Resources

uCDN has a reference (URI provided by the dCDN) for each Trigger Status Resource it has created, it may fetch that resource at any time.

This may be used to retrieve progress information, and to fetch the result of triggered activity.

4.3. Deleting Triggers

The uCDN MAY delete Trigger Status Resources at any time, using the HTTP DELETE method.

Once deleted, the references to a Trigger Status Resource MUST be removed from all Trigger Status Resource collections. Subsequent requests for the resource shall be handled as required by HTTP, and so will receive responses with status 404 or 410.

If a "pending" Trigger Status Resource is deleted, dCDN SHOULD NOT

start processing of that activity. Deleting a "pending" trigger does not however guarantee that it is not started because, once it has triggered activity, uCDN cannot control the timing of that activity. Processing may, for example, start after the DELETE is sent by uCDN and before the DELETE is processed by dCDN.

If an "active" Trigger Status Resource is deleted, dCDN MAY stop processing the triggered activity. However, as with deletion of a "pending" trigger, dCDN does not guarantee this.

Deletion of a "complete" or "failed" Trigger Status Resource requires no processing in dCDN other than deletion of the resource.

4.4. Expiry of Trigger Status Resources

The dCDN MAY choose to automatically delete Trigger Status Resources some time after they become completed or failed. In this case, dCDN will remove the resource and respond to subsequent requests for it with HTTP status 404 or 410.

If dCDN performs this housekeeping, it MUST have reported the length of time after which completed Trigger Status Resources become stale via a property of the collection of all Trigger Status Resources. It is recommended that Trigger Status Resources are automatically deleted 24 hours after they become completed or failed.

To ensure it has access to the status of its completed and failed triggers, it is recommended that uCDN's polling interval is half the time after which records for completed activity will be considered stale.

5. Properties of Triggers

5.1. Properties of Trigger Requests

Properties of Trigger Requests are defined in the following subsections.

Property: type

Description: This property defines the type of the trigger:

Type: TriggerType

Mandatory: Yes

Property: metadata.urls

Description: The uCDN URL for the metadata the trigger applies to.

Type: URLs
Mandatory: No, but at least one of 'metadata.*' or 'content.*'
MUST be present and non-empty.

Property: content.urls
Description: URLs of content data the trigger applies to, see
Section 5.1.1.
Type: URLs
Mandatory: No, but at least one of 'metadata.*' or 'content.*'
MUST be present and non-empty.

Property: metadata.patterns
Description: The metadata the trigger applies to.
Type: UrlPatterns
Mandatory: No, but at least one of 'metadata.*' or 'content.*'
MUST be present and non-empty, and metadata.patterns MUST NOT
be present if the TriggerType is Preposition.

Property: content.patterns
Description: The content data the trigger applies to.
Type: UrlPatterns
Mandatory: No, but at least one of 'metadata.*' or 'content.*'
MUST be present and non-empty, and content.patterns MUST NOT be
present if the TriggerType is Preposition.

5.1.1. Content URLs

Once interfaces for metadata and request routing interfaces have been agreed, it will be possible to define a way to make reference to content. That form of reference will be used in Trigger Requests.

Some possible options for content references are:

- o Canonical URL - a reference to the content that is shared by all Interconnected CDNs. A potential problem is that the URL visible to dCDN may have been modified by uCDN during request redirection.
- o Origin URL - the URL from which dCDN acquired the content. May be different in each CDN as each dCDN may use its uCDN as the origin.
- o Metadata URL - some portion of the metadata served to uCDN by dCDN will describe content, it would be possible to refer to content fetched as a result of that metadata description.

If the Content URL is modified by uCDN, it is uCDN's responsibility to translate and pass-on Trigger Requests relating to that content using appropriately modified Content URLs.

5.2. Properties of Trigger Status Resources

Property: trigger
Description: The properties of trigger request that created this record.

Type: TriggerRequest
Mandatory: Yes

Property: ctime
Description: Time at which the request was received by dCDN.
Time is local to dCDN, there is no requirement to synchronise
clocks between interconnected CDNs.
Type: AbsoluteTime
Mandatory: Yes

Property: mtime
Description: Time at which the resource was last modified.
Time is local to dCDN, there is no requirement to synchronise
clocks between interconnected CDNs.
Type: AbsoluteTime
Mandatory: Yes

Property: etime
Description: Estimate of the time at which dCDN expects to
complete the activity. Time is local to dCDN, there is no
requirement to synchronise clocks between interconnected CDNs.
Type: AbsoluteTime
Mandatory: No

Property: status
Description: Current status of the triggered activity.
Type: TriggerStatus
Mandatory: Yes

Property: error
Description: Error indication.
Type: (To be decided - a set of standard error conditions needs
to be defined. The namespace for these errors codes should
allow vendor-defined error codes for extension of the protocol.
This may allow, for example, for the definition of more
specific error codes when two CDNs supplied by the same vendor
are interconnected.)
Mandatory: No, and only allowed when "status" is "Failed".

5.3. Properties of Trigger Collections

Property: links
Description: References to Trigger Status Resources in the
collection.
Type: List of Relationships.
Mandatory: Yes
Property: staleresourcetime

Description: The length of time for which dCDN guarantees to keep a completed Trigger Status Resource. After this time, dCDN MAY delete the resource and all references to it from collections.

Type: Integer, time in seconds.

Mandatory: Yes, in the collection of all Trigger Status Resources if dCDN deletes stale entries. If the property is present in the filtered collections, it MUST have the same value as in the collection of all Trigger Status Resources.

5.4. Trigger Resource Simple Data Type Descriptions

This section describes the simpler data types that are used for properties of Trigger Status resources.

5.4.1. TriggerType

This type defines the type of action being triggered, permitted actions are:

- o Preposition - a request for dCDN to acquire metadata or content.
- o Invalidate - a request for dCDN to invalidate metadata or content. After servicing this request the dCDN will not use the specified data without first re-validating it using, for example, an "If-None-Match" HTTP request. The dCDN need not erase the associated data.
- o Purge - a request for dCDN to erase metadata or content. After servicing the request, the specified data must not be held on dCDN.

5.4.2. TriggerStatus

This type describes the current status of a Trigger, possible values are:

- o Pending - the trigger has not yet been acted upon.
- o Active - the trigger is currently being acted upon.
- o Complete - the triggered activity completed successfully.
- o Failed - the triggered activity could not be completed.

5.4.3. URLs

This type describes a set of references to metadata or content, it is simply a list of absolute URLs.

5.4.4. UrlPatterns

This type describes a reference to a set of metadata or content. It is a set of 'pattern.string' properties each of which has an optional

'pattern.flags'.

The intention is to align this with the pattern matching capabilities of the CDNI metadata interface, once defined. This current definition is based on that in [I-D.cjlmw-cdni-metadata].

Property: pattern.string

Description: String to match against the URL of metadata or content, i.e. a [RFC3986] path-absolute.

Type: Pattern

Mandatory: Yes

Property: pattern.flags

Description: Flags to control the pattern match.

Type: PatternFlags

Mandatory: No (default Case-sensitive infix matching)

5.4.5. AbsoluteTime

Times are expressed in seconds since the UNIX epoch.

6. JSON Encoding of Objects

This encoding is based on that described in [I-D.cjlmw-cdni-metadata], but has been reproduced here while metadata work is in progress. Once that work is complete, the authors would look to align with the structure of the metadata draft and make reference to common definitions as appropriate.

The encoding for a CDNI Trigger object is a JSON object containing a dictionary of (key,value) pairs where the keys are the property names, and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource). Likewise, the values associated with each key are dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource).

The "triggers" property of the top level JSON object lists the requested actions.

Key: triggers

Description: List of triggers.

Type: List of JSON objects, each specifying a trigger type and a set of data to act upon.

Mandatory: Yes.

Object keys in JSON are case sensitive and therefore any dictionary key defined by this document (for example the names of CDNI Triggers object properties) MUST always be represented in lowercase.

In addition to the properties of the object, the following additional keys may be present.

Key: base

Description: Provides a prefix for any relative URLs in the object. This is similar to the XML base tag [XML-BASE]. If absent, all URLs in the remainder of the document must be absolute URLs.

Type: URI

Mandatory: No

Key: links

Description: The relationships of this object to other addressable objects.

Type: List of Relationships.

Mandatory: Yes

6.1. JSON Encoding of Embedded Types

6.1.1. TriggerType

Key: type

Description: One of "preposition", "invalidate" or "purge".

Type: string

Mandatory: Yes

6.1.2. TriggerStatus

Key: status

Description: One of "pending", "active", "failed", "complete"

Type: string

Mandatory: Yes

6.1.3. Metadata and Content References

Keys: metadata.urls, content.urls

Description: A list of URLs of the addressable objects being referenced.

Type: URLs

Mandatory: No

Keys: metadata.patterns, content.patterns
Description: A list of patterns to match against URLs of objects being referenced.
Type: list of Patterns
Mandatory: No

6.1.4. Pattern

JSON: A dictionary with two keys, "pattern.string" and "pattern.flags":
Key: pattern.string
Description: The string to match URLs against.
Type: string
Mandatory: Yes
Key: pattern.flags
Description: A number calculated by adding together the values associated with each flag that is set.

+ 1 - Case-insensitive
+ 2 - Prefix
+ 4 - Suffix
Type: integer
Mandatory: Yes

Example of case-insensitive prefix match against "http://www.example.com/trailers/":
{
 "pattern.string": "http://www.example.com/trailers",
 "pattern.flags": 3
}

6.1.5. Relationship

JSON: A dictionary with the following keys:

- o href - The URI of the of the addressable object being referenced.
- o rel - The Relationship between the referring object and the object it is referencing.
- o type - The MIME Media Type of the referenced object. See Section 6.2 for the MIME Media Types of objects specified in this document.
- o title - An optional title for the Relationship/link.

Note: The above structure follows the pattern of atom:link in [RFC4287].

Example Relationship to a CDNI Trigger Resource within a CDNI Trigger Collection:

```
{
  "href": "http://triggers.cdni.example.com/trigger/12345",
  "rel": "Trigger",
  "type": "application/vnd.cdni.control.trigger.status+json"
}
```

The format of relationship values is expected to align with other CDNI interfaces. For example, rather than use simple names (like "Trigger" in this case), there may be a namespace that allows well-known and proprietary values to co-exist.

6.2. MIME Media Types

Table 1 lists the MIME Media Type for each trigger object (resource) that is retrievable through the CDNI Trigger interface.

Note: A prefix of "vnd.cdni" is used, however it is expected that a more appropriate prefix will be used if the CDNI WG accepts this document.

Data Object	MIME Media Type
TriggerStatus	application/ vnd.cdni.control.trigger.status+json
TriggerCollection	application/ vnd.cdni.control.trigger.collection+json

Table 1: MIME Media Types for CDNI Trigger resources

7. Examples

The following sections provide examples of different CDNI Trigger objects encoded as JSON.

No authentication is shown in the following illustrative examples, it is anticipated that authentication mechanisms will be aligned with other CDNI Interfaces as and when those mechanisms are defined.

Discovery of the triggers interface is out of scope of this document. In an implementation, all URLs are under control of dCDN and the uCDN must not attempt to ascribe any meaning to individual elements of the path. In examples in this section, the following URLs are used as the location of the collections of triggers:

- o Collection of all Triggers belonging to one uCDN:
http://dcdn.example.com/triggers
- o Filtered collections:
 - Pending: http://dcdn.example.com/triggers/pending
 - Active: http://dcdn.example.com/triggers/active
 - Complete: http://dcdn.example.com/triggers/complete
 - Failed: http://dcdn.example.com/triggers/failed

7.1. Creating Triggers

Examples of uCDN triggering activity in dCDN:

7.1.1. Preposition

An example of a preposition request, a POST to the "AllTriggers" collection.

Note that a preposition request must not include any "metadata.patterns" or "content.patterns":
REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: 318
```

```
{
  "triggers" : [{
    "type": "preposition",

    "metadata.urls" : [ "http://metadata.example.com/a/b/c" ],
    "content.urls" : [
      "http://www.example.com/a/b/c/1",
      "http://www.example.com/a/b/c/2",
      "http://www.example.com/a/b/c/3",
      "http://www.example.com/a/b/c/4"
    ]
  }]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sun, 15 Apr 2012 19:33:48 GMT
Content-Length: 533
Content-Type: application/vnd.cdni.control.trigger.status+json
```

Location: http://dcdn.example.com/triggers/0
Server: example-server/0.1

```
{
  "ctime": 1334518428,
  "etime": 1334518436,
  "mtime": 1334518428,
  "status": "pending",
  "triggers": [
    {
      "content.urls": [
        "http://www.example.com/a/b/c/1",
        "http://www.example.com/a/b/c/2",
        "http://www.example.com/a/b/c/3",
        "http://www.example.com/a/b/c/4"
      ],
      "metadata.urls": [
        "http://metadata.example.com/a/b/c"
      ],
      "type": "preposition"
    }
  ]
}
```

7.1.2. Invalidate

An example of an invalidate request, another POST to the "AllTriggers" collection. This instructs dCDN to re-validate the content at "http://www.example.com/a/index.html", as well as any metadata and content whose URLs are prefixed by "http://metadata.example.com/a/b/" and "http://www.example.com/a/b/" respectively, using case-insensitive matching.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: 386
```

```
{
  "triggers" : [{
    "type": "invalidate",

    "metadata.patterns" : [
      { "pattern.string" : "http://metadata.example.com/a/b/",
        "pattern.flags" : 3 }
    ]
  }]
}
```



```
    ],
    "content.urls" : [ "http://www.example.com/a/index.html" ],
    "content.patterns" : [
      { "pattern.string" : "http://www.example.com/a/b/",
        "pattern.flags" : 3 }
    ]
  }
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sun, 15 Apr 2012 19:33:48 GMT
Content-Length: 681
Content-Type: application/vnd.cdni.control.trigger.status+json
Location: http://dcdn.example.com/triggers/1
Server: example-server/0.1

{
  "ctime": 1334518428,
  "etime": 1334518436,
  "mtime": 1334518428,
  "status": "pending",
  "triggers": [
    {
      "content.patterns": [
        {
          "pattern.flags": 3,
          "pattern.string": "http://www.example.com/a/b/"
        }
      ],
      "content.urls": [
        "http://www.example.com/a/index.html"
      ],
      "metadata.patterns": [
        {
          "pattern.flags": 3,
          "pattern.string": "http://metadata.example.com/a/b/"
        }
      ],
      "type": "invalidate"
    }
  ]
}
```

7.2. Examining Trigger Status

Once triggers have been created, uCDN can check their status as shown in these examples.

7.2.1. Collection of All Triggers

The uCDN can fetch the set of all the triggers it has created and which have not yet been deleted or removed as expired. After creation of the "preposition" and "invalidate" triggers shown above, this collection might look as follows:

REQUEST:

```
GET /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 418
Expires: Sun, 15 Apr 2012 19:34:48 GMT
Server: example-server/0.1
ETag: "-1621644187315998047"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:48 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.2. Filtered Collections of Triggers

The filtered collections are also available to uCDN. Before dCDN starts processing the two triggers shown above, both will appear in the collection of Pending Triggers, for example:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 418
Expires: Sun, 15 Apr 2012 19:34:48 GMT
Server: example-server/0.1
ETag: "8328231265607503653"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:48 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

At this point, if no other triggers had been created, the other filtered views of the triggers would be empty. For example:

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 53
Expires: Sun, 15 Apr 2012 19:34:49 GMT
Server: example-server/0.1
ETag: "-654105208640281650"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:49 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [],
  "staleresourcetime": 86400
}
```

7.2.3. Trigger Status Resources

The Trigger Status Resources can also be examined for detail about individual triggers. For example, for the "preposition" and "invalidate" triggers from previous examples:

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 533
Expires: Sun, 15 Apr 2012 19:34:48 GMT
Server: example-server/0.1
ETag: "-6152251159861441046"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:48 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1334518428,
  "etime": 1334518436,
  "mtime": 1334518428,
  "status": "pending",
  "triggers": [
    {
      "content.urls": [
        "http://www.example.com/a/b/c/1",
        "http://www.example.com/a/b/c/2",
        "http://www.example.com/a/b/c/3",
        "http://www.example.com/a/b/c/4"
      ],
      "metadata.urls": [
        "http://metadata.example.com/a/b/c"
      ],
      "type": "preposition"
    }
  ]
}
```

REQUEST:

```
GET /triggers/1 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 681
Expires: Sun, 15 Apr 2012 19:34:49 GMT
Server: example-server/0.1
ETag: "3718394748848505642"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:49 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1334518428,
  "etime": 1334518436,
  "mtime": 1334518428,
  "status": "pending",
  "triggers": [
    {
      "content.patterns": [
        {
          "pattern.flags": 3,
          "pattern.string": "http://www.example.com/a/b/"
        }
      ],
      "content.urls": [
        "http://www.example.com/a/index.html"
      ],
      "metadata.patterns": [
        {
          "pattern.flags": 3,
          "pattern.string": "http://metadata.example.com/a/b/"
        }
      ],
      "type": "invalidate"
    }
  ]
}
```

7.2.4. Polling for Change

The uCDN may use the Entity Tags of collections or resources when polling for change in status, as shown in the following examples:
REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "8328231265607503653"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sun, 15 Apr 2012 19:34:48 GMT
Server: example-server/0.1
ETag: "8328231265607503653"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:48 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-6152251159861441046"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sun, 15 Apr 2012 19:34:48 GMT
Server: example-server/0.1
ETag: "-6152251159861441046"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:48 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

When the triggered activity is complete, the contents of the filtered collections will be updated, along with their Entity Tags. For

example, when the two example triggers are complete, the collections of pending and complete triggers may look like:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 53
Expires: Sun, 15 Apr 2012 19:34:53 GMT
Server: example-server/0.1
ETag: "-7056231826368088123"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:53 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [],
  "staleresourcetime": 86400
}
```


REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 418
Expires: Sun, 15 Apr 2012 19:34:59 GMT
Server: example-server/0.1
ETag: "1388228818267892536"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:33:59 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.5. Cancelling or Removing a Trigger

To request dCDN to cancel a Trigger, uCDN may delete the Trigger Resource. It may also delete completed and failed triggers to reduce the size of the collections. For example, to remove the "preposition" request from earlier examples:

REQUEST:

```
DELETE /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 204 No Content
Date: Sun, 15 Apr 2012 19:33:59 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: example-server/0.1
```

This would, for example, cause the collection of completed triggers shown in the example above to be updated to:

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 237
Expires: Sun, 15 Apr 2012 19:35:00 GMT
Server: example-server/0.1
ETag: "-8850203857096517156"
Cache-Control: max-age=60
Date: Sun, 15 Apr 2012 19:34:00 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

8. IANA Considerations

TBD.

9. Security Considerations

The dCDN must ensure that each uCDN only has access to its own Trigger Status Resources.

It is anticipated that a common authentication mechanism will be used by this and other CDNI Interconnect interfaces, the mechanism must exist but is not identified in this document.

The dCDN must ensure that activity triggered by uCDN only affects metadata or content originating from that uCDN.

10. Acknowledgements

TBD.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

11.2. Informative References

- [I-D.cjlmw-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., and K. Leung, "CDN Interconnect Metadata", draft-cjlmw-cdni-metadata-00 (work in progress), July 2012.
- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN

Interconnection", draft-ietf-cdni-framework-01 (work in progress), July 2012.

[I-D.ietf-cdni-problem-statement]

Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-ietf-cdni-problem-statement-08 (work in progress), June 2012.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[XML-BASE]

Marsh, J., Ed. and R. Tobin, Ed., "XML Base (Second Edition) - <http://www.w3.org/TR/xmlbase/>", January 2009.

Authors' Addresses

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: rmurray@velocix.com

Ben Niven-Jenkins
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: ben@velocix.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 5, 2013

R. Murray
B. Niven-Jenkins
Velocix (Alcatel-Lucent)
April 3, 2013

CDNI Control Interface / Triggers
draft-murray-cdni-triggers-03

Abstract

This document describes the part of the CDN Interconnect Control Interface that allows a CDN to trigger activity in an interconnected CDN that is configured to deliver content on its behalf. The upstream CDN can use this mechanism to request that the downstream CDN pre-positions metadata or content, or that it re-validate or purge metadata or content. The upstream CDN can monitor the status of activity that it has triggered in the downstream CDN.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 October 5, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents
(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Terminology	5
2. Model for CDNI Triggers	5
2.1. Timing of Triggered Activity	7
2.2. Trigger Results	7
3. Collections of Trigger Status Resources	7
4. CDNI Trigger interface	8
4.1. Creating Triggers	10
4.2. Checking Status	11
4.2.1. Polling Trigger Status Resource collections	11
4.2.2. Polling Trigger Status Resources	11
4.3. Deleting Triggers	11
4.4. Expiry of Trigger Status Resources	12
4.5. Error Handling	12
5. Properties of Triggers	13
5.1. Properties of Trigger Requests	13
5.1.1. Content URLs	14
5.2. Properties of Trigger Status Resources	14
5.3. Properties of ErrorDesc	15
5.4. Properties of Trigger Collections	16
5.5. Trigger Resource Simple Data Type Descriptions	16
5.5.1. TriggerType	16
5.5.2. TriggerStatus	16
5.5.3. URLs	17
5.5.4. AbsoluteTime	17
5.5.5. ErrorCode	17
6. JSON Encoding of Objects	17
6.1. JSON Encoding of Embedded Types	18
6.1.1. TriggerType	18
6.1.2. TriggerStatus	18
6.1.3. PatternMatch	19
6.1.4. ErrorDesc	19
6.1.5. ErrorCode	20
6.1.6. Relationship	20
6.2. MIME Media Types	21
7. Examples	21

7.1. Creating Triggers	21
7.1.1. Preposition	21
7.1.2. Invalidate	23
7.2. Examining Trigger Status	24
7.2.1. Collection of All Triggers	24
7.2.2. Filtered Collections of Triggers	25
7.2.3. Trigger Status Resources	27
7.2.4. Polling for Change	29
7.2.5. Cancelling or Removing a Trigger	32
7.2.6. Error Reporting	34
8. IANA Considerations	35
9. Security Considerations	35
10. Acknowledgements	35
11. References	35
11.1. Normative References	35
11.2. Informative References	35
Authors' Addresses	36

1. Introduction

[RFC6707] introduces the Problem scope for CDN Interconnection (CDNI) and lists the four categories of interfaces that may be used to compose a CDNI solution (Control, Metadata, Request Routing, Logging).

[I-D.ietf-cdni-framework] expands on the information provided in [RFC6707] and describes each of the interfaces and the relationships between them in more detail.

This draft concentrates on the "High" and "Medium" priority requirements for the CDNI Control Interface identified in section 4 of [I-D.ietf-cdni-requirements], reproduced here for convenience:

CNTL-1 [HIGH] The CDNI Control interface shall allow the Upstream CDN to request that the Downstream CDN (and, if cascaded CDNs are supported by the solution, that the potential cascaded Downstream CDNs) perform the following actions on an object or object set:

- * Mark an object or set of objects and/or its CDNI metadata as "stale" and revalidate them before they are delivered again
- * Delete an object or set of objects and/or its CDNI metadata from the CDN surrogates and any storage. Only the object(s) and CDNI metadata that pertain to the requesting Upstream CDN are allowed to be purged.

CNTL-2 [HIGH] The CDNI Control interface shall allow the Downstream CDN to report on the completion of these actions (by itself, and if cascaded CDNs are supported by the solution, by potential cascaded Downstream CDNs), in a manner appropriate for the action (e.g. synchronously or asynchronously). The confirmation receipt should include a success or failure indication. The failure indication is used if the Downstream CDN cannot delete the content in its storage.

CNTL-3 [HIGH] The CDNI Control interface shall support initiation and control by the Upstream CDN of pre-positioned CDNI metadata acquisition by the Downstream CDN.

CNTL-4 [MED] The CDNI Control interface should support initiation and control by the Upstream CDN of pre-positioned content acquisition by the Downstream CDN.

CNTL-12 [MED] The CDNI Control interface should allow for multiple content items identified by a Content Collection ID to be purged using a single Content Purge action.

This document describes the CI/T interface, Control Interface / Triggers. It does not consider those parts of the control interface that relate to configuration, bootstrapping or authentication of CDN Interconnect interfaces.

- o Section 2 outlines the model for the Trigger Interface at a high level.
- o Section 3 describes collections of Trigger Resources.
- o Section 4 defines the RESTful web service provided by dCDN.
- o Section 5 lists properties of Trigger Requests and Status Resources.
- o Section 6 defines a JSON encoding for Trigger Requests and Status Resources.
- o Section 7 contains example messages.

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

2. Model for CDNI Triggers

A trigger, sent from uCDN to dCDN, is a request for dCDN to do some work relating to data originating from uCDN.

The trigger may request action on either metadata or content, the following actions can be requested:

- o preposition - used to instruct dCDN to fetch metadata from uCDN, or content from any origin including uCDN.
- o invalidate - used to instruct dCDN to revalidate specific metadata or content before re-using it.
- o purge - used to instruct dCDN to delete specific metadata or content.

The CI/T interface is a RESTful web service offered by dCDN. It allows creation and deletion of triggers, and tracking of the triggered activity. When dCDN accepts a trigger it creates a resource describing status of the triggered activity, a Trigger Status Resource. The uCDN may poll Trigger Status Resources to monitor progress.

Requests to invalidate and purge metadata or content apply to all variants of that data with a given URI.

The dCDN maintains a collection of Trigger Status Resources for each uCDN, each uCDN only has access to its own collection and the location of that collection is shared when CDN interconnection is established.

To trigger activity in dCDN, uCDN will POST to the collection of Trigger Status Resources. If dCDN accepts the trigger, it creates a new Trigger Status Resource and returns its location to uCDN. To

monitor progress, uCDN may GET the Trigger Status Resource. To cancel a trigger, or remove a trigger from the collection once its activity has been completed, uCDN may DELETE the Trigger Status Resource.

In addition to the collection of all Trigger Status Resources for uCDN, uCDN shall have access to filtered views of that collection. These filtered views are defined in Section 3 and include collections of active and completed triggers. These collections provide a mechanism for polling the status of multiple jobs.

Figure 1 is an example showing the basic message flow used by the uCDN to trigger activity in dCDN, and for uCDN to discover the status of that activity. Only successful triggering is shown. Examples of the messages are given in Section 7.

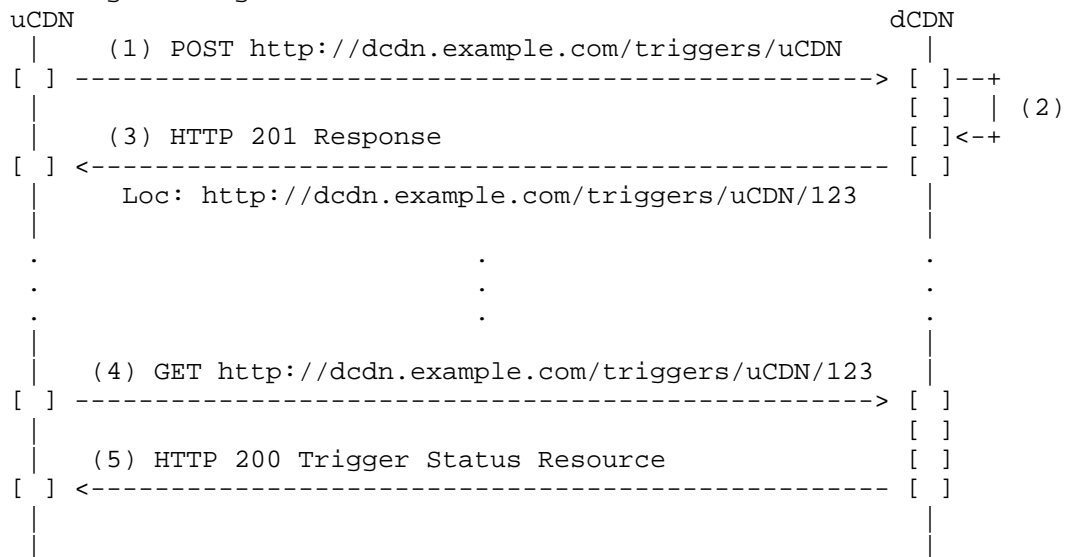


Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are:

1. uCDN triggers action in dCDN by posting to a collection of Trigger Status Resources, "http://dcdn.example.com/triggers/uCDN". The URL of this was given to uCDN when the trigger interface was established.
2. dCDN authenticates the request, validates the trigger and if it accepts the request, creates a new Trigger Status Resource.
3. dCDN responds to uCDN with an HTTP 201 response status, and the location of the Trigger Status Resource.

4. uCDN may repeatedly poll the Trigger Status Resource in dCDN.
5. dCDN responds with the Trigger Status Resource, describing progress or results of the triggered activity.

The remainder of this document describes the messages, Trigger Status Resources, and collections of Trigger Status Resources in more detail.

2.1. Timing of Triggered Activity

Timing of triggered activity is under dCDN control, including its start-time and pacing of the activity in the network.

Invalidate and purge triggers MUST be applied to all data acquired before the trigger was created in dCDN. The dCDN MAY apply the triggers to data acquired after trigger creation.

If uCDN wishes to invalidate or purge content, then immediately preposition replacement content at the same URLs, it must ensure the dCDN has completed the invalidate/purge before initiating the prepositioning. If it fails to do that and the requests overlap, and dCDN passes the triggers on to a further dCDN in a cascade, that CDN may preposition content that has not yet been invalidated/purged in its uCDN.

2.2. Trigger Results

Each Trigger Request may operate on multiple data items. The trigger shall be reported as "complete" only if all actions can be completed successfully, otherwise it shall be reported as "failed". The reasons for failure and URLs or Patterns affected shall be enumerated in the Trigger Status Resource. For more detail, see section Section 4.5.

If a dCDN is also acting as uCDN in a cascade, it MUST forward triggers to any downstream CDNs that may have data affected by the trigger. The trigger MUST NOT be reported as complete in a CDN until it is complete in all of its downstream CDNs. A trigger MAY be reported as failed as soon as it fails in a CDN or in any of its downstream CDNs.

3. Collections of Trigger Status Resources

As described in Section 2, Trigger Status Resources exist in dCDN to report the status of activity triggered by each uCDN.

A collection of Trigger Status Resources is a resource that contains

a reference to each Trigger Status Resource in that collection.

To trigger activity in dCDN, uCDN creates a new Trigger Status Resource by posting to dCDN's collection of uCDN's Trigger Status Resources. The URL of each Trigger Status Resource is generated by the dCDN when it accepts the trigger, and returned to uCDN. This immediately enables uCDN to check the status of that trigger.

The dCDN must present a different set of Trigger Status Resources to each interconnected uCDN, only Trigger Status Resources belonging to a uCDN shall be visible to it. The dCDN may, for example, achieve this by offering different collection URLs to uCDNs, or by filtering the response based on the client uCDN.

The dCDN resource representing the collection of all uCDN's Trigger Status Resources is accessible to uCDN. This collection lists all uCDN triggers that have been accepted by dCDN, and have not yet been deleted by uCDN or expired and removed by dCDN.

In order to allow uCDN to check status of multiple jobs in a single request, dCDN shall also maintain collections representing filtered views of the collection of all Trigger Status Resources. The filtered collections are:

- o Pending - Trigger Status Resources for triggers that have been accepted, but not yet acted upon.
- o Active - Trigger Status Resources for triggered activity that is currently being processed in dCDN.
- o Complete - Trigger Status Resources representing activity that completed successfully.
- o Failed - Trigger Status Resources representing activity that failed.

4. CDNI Trigger interface

This section describes an interface to enable an upstream CDN to trigger defined activities in a downstream CDN. The interface is intended to be independent of the set of activities defined now, or that may be defined in future.

The CI/T interface is built on the principles of RESTful web services. Requests are made over HTTP, and the HTTP Method defines the operation the request would like to perform. The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses from servers implementing the CI/T interface that contain a response body SHOULD include an ETag to enable validation of cached versions of returned

resources.

Servers implementing the CI/T interface MUST support the HTTP GET, HEAD, POST and DELETE methods. The only representation specified in this document is JSON.

Trigger Requests are POSTed to a URI in dCDN. If the trigger is accepted by dCDN, it creates a Trigger Status Resource and returns its URI to dCDN in an HTTP 201 response. The triggered activity can then be monitored by uCDN using that resource and the collections described in Section 3.

The URI that Trigger Requests should be POSTed to needs to be either discovered by or configured in the upstream CDN. Performing a GET on that URI retrieves a collection of the URIs of all Trigger Status Resources. The URI of each Trigger Status Resource is also returned to uCDN when it is created. This means all Trigger Status Resources can be discovered, so CI/T servers are free to assign whatever structure they desire to the URIs for CI/T resources. CI/T clients MUST NOT make any assumptions regarding the structure of CI/T URIs or the mapping between CI/T objects and their associated URIs. Therefore any URIs present in the examples below are purely illustrative and are not intended to impose a definitive structure on CI/T interface implementations.

The CI/T interface builds on top of HTTP, so CI/T servers may make use of any HTTP feature when implementing the CI/T interface. For example, a CI/T server may make use of HTTP's caching mechanisms to indicate that the returned response/representation has not been modified since it was last returned, reducing the processing needed to determine whether the status of triggered activity has changed.

This specification is neutral with regard to the transport below the HTTP layer.

[Editor's note: It is anticipated that decisions on use of HTTPS for other CDNI interfaces will be adopted for Triggers.]

Discovery of the CI/T Interface is outside the scope of this document. It is anticipated that a common mechanism for discovery of all CDNI interfaces will be defined.

The dCDN must ensure that activity triggered by uCDN only affects metadata or content originating from that uCDN. Since only one CDN can be authoritative for a given item of metadata or content, this requirement means there cannot be any "loops" in trigger requests between CDNs.

4.1. Creating Triggers

To create a new trigger, uCDN makes an HTTP POST to the unfiltered collection of its triggers. The request body of that POST is a Trigger Request.

dCDN validates and authenticates that request, if it is malformed or uCDN does not have sufficient access rights it MAY reject the request immediately. In this case, it SHALL respond with an appropriate 4xx HTTP error code and no resource shall be created on dCDN.

If the request is accepted, uCDN SHALL create a new Trigger Status Resource. The HTTP response to dCDN SHALL have status code 201 and the URI of the Trigger Status Resource in the Location header field. The HTTP response MAY include the content of the newly created Trigger Status Resource, this is recommended particularly in cases where the trigger has completed immediately.

Once a Trigger Status Resource has been created dCDN MUST NOT re-use its location, even after that resource has been removed through deletion or expiry.

The "request" property of the Trigger Status Resource SHALL contain the information posted in the body of the Trigger Request. Note that this need not be a byte-for-byte copy. For example, in the JSON representation the dCDN may re-serialise the information differently.

If the trigger is queued by dCDN for later action, the "status" property of the Trigger Status Resource SHALL be "pending". Once trigger processing has started the "status" SHALL be "active".

A trigger may result in no activity in dCDN if, for example, it is an invalidate or purge request for data the dCDN has not acquired, or a prepopulate request for data it has already acquired. In this case, the "status" of the Trigger Status Resource shall be "complete" and the Trigger Status Resource shall be added to the dCDN collection of Complete Triggers.

If dCDN is not able to track triggered activity, it MAY indicate that it has undertaken to complete the activity but will not report completion or any further errors. To do this, it must set the trigger status to "complete", with an estimated completion time in the future ("etime" greater than "mtime").

Once created, Trigger Status Resources may be deleted by uCDN but not modified. The dCDN MUST reject PUT and POST requests from uCDN to Trigger Status Resources using HTTP status code 403.

4.2. Checking Status

The uCDN has two ways to check progress of activity it has triggered in dCDN, described in the following sections.

Because the triggers protocol is based on HTTP, Entity Tags may be used by the uCDN as cache validators, as defined in section 3.11 of [RFC2616], to check for change in status of a resource or collection of resources without re-fetching the whole resource or collection.

The dCDN should use the cache control headers for responses to GETs for Trigger Status Resources and Collections to indicate the frequency at which it recommends uCDN should poll for change.

4.2.1. Polling Trigger Status Resource collections

uCDN can fetch the collection of its Trigger Status Resources, or filtered views of that collection.

This makes it possible to poll status of all triggered activity in a single request. If dCDN moves a Trigger Status Resource from the Active to the Completed collection, uCDN may chose to fetch the result of that activity.

When polling in this way, uCDN may choose to use HTTP Entity Tags to monitor for change, rather than repeatedly fetching the whole collection.

4.2.2. Polling Trigger Status Resources

uCDN has a reference (URI provided by the dCDN) for each Trigger Status Resource it has created, it may fetch that resource at any time.

This may be used to retrieve progress information, and to fetch the result of triggered activity.

4.3. Deleting Triggers

The uCDN MAY delete Trigger Status Resources at any time, using the HTTP DELETE method.

Once deleted, the references to a Trigger Status Resource MUST be removed from all Trigger Status Resource collections. Subsequent requests for the resource shall be handled as required by HTTP, and so will receive responses with status 404 or 410.

If a "pending" Trigger Status Resource is deleted, dCDN SHOULD NOT

start processing of that activity. Deleting a "pending" trigger does not however guarantee that it is not started because, once it has triggered activity, uCDN cannot control the timing of that activity. Processing may, for example, start after the DELETE is sent by uCDN and before the DELETE is processed by dCDN.

If an "active" Trigger Status Resource is deleted, dCDN MAY stop processing the triggered activity. However, as with deletion of a "pending" trigger, dCDN does not guarantee this.

Deletion of a "complete" or "failed" Trigger Status Resource requires no processing in dCDN other than deletion of the resource.

4.4. Expiry of Trigger Status Resources

The dCDN MAY choose to automatically delete Trigger Status Resources some time after they become completed or failed. In this case, dCDN will remove the resource and respond to subsequent requests for it with HTTP status 404 or 410.

If dCDN performs this housekeeping, it MUST have reported the length of time after which completed Trigger Status Resources become stale via a property of the collection of all Trigger Status Resources. It is recommended that Trigger Status Resources are automatically deleted 24 hours after they become completed or failed.

To ensure it has access to the status of its completed and failed triggers, it is recommended that uCDN's polling interval is half the time after which records for completed activity will be considered stale.

4.5. Error Handling

A CI/T server may reject a trigger request using HTTP status codes, for example 400 if the request is malformed or 401 if the client does not have permission to create triggers or it is trying to act on another CDN's data.

If any part of the trigger request fails the trigger shall be reported as "failed" once its activity is complete, or if no further errors will be reported. The "errors" property in the Trigger Status Resource will be used to enumerate which actions failed and the reasons for failure, and may be present while the trigger is still "pending" or "active" if the trigger is still running for some URLs or Patterns in the trigger request.

Once a request has been accepted, processing errors are reported in the Trigger Status Resource using a list of "ErrorDesc". Each

ErrorDesc is used to report errors against one or more of the URLs or Patterns in the trigger request.

If a surrogate affected by a trigger is offline in dCDN, or dCDN is unable to pass a trigger request on to any of its affected dCDNs; dCDN should report an error if the request is abandoned, otherwise it must keep the trigger in state "pending" or "active" until it's acted upon or uCDN chooses to cancel it. Or, if the request is queued and dCDN will not report further status, dCDN may report the trigger as "complete" with an "etime" in the future.

Note that an "invalidate" trigger may be reported as "complete" when surrogates that may have the data are offline, if those surrogates will not use the affected data without first revalidating it when they are back online. This does not apply to "preposition" or "purge" triggers.

5. Properties of Triggers

5.1. Properties of Trigger Requests

Properties of Trigger Requests are defined in the following subsections.

Property: type

Description: This property defines the type of the trigger:

Type: TriggerType

Mandatory: Yes

Property: metadata.urls

Description: The uCDN URL for the metadata the trigger applies to.

Type: URLs

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Property: content.urls

Description: URLs of content data the trigger applies to, see Section 5.1.1.

Type: URLs

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Property: content.ccid

Description: The Content Collection IDentifier of data the trigger applies to.

Type: List of strings

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Property: metadata.patterns

Description: The metadata the trigger applies to.

Type: List of PatternMatch

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty, and metadata.patterns MUST NOT be present if the TriggerType is Preposition.

Property: content.patterns

Description: The content data the trigger applies to.

Type: List of PatternMatch

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty, and content.patterns MUST NOT be present if the TriggerType is Preposition.

5.1.1. Content URLs

To refer to content in dCDN, uCDN must present URLs in the same form clients will use to access content in that dCDN, after transformation to remove any surrogate-specific parts of a 302-redirect URL form. By definition, it is always possible to locate content based on URLs in this form.

If content URLs are transformed by an intermediate CDN in a cascade, that intermediate CDN must transform URLs in trigger requests it passes to its dCDN.

[Editor's note: Design for CDNI Metadata transformation, including discussion of URL transformation, is being undertaken as part of the work on the metadata interface. The intention is to align with that document or make reference to it when it's complete.]

When processing trigger requests, CDNs may ignore the URL scheme (http or https) in comparing URLs. For example, for an invalidate or purge trigger, content may be invalidated or purged regardless of the protocol clients use to request it.

5.2. Properties of Trigger Status Resources

Property: trigger

Description: The properties of trigger request that created this record.

Type: TriggerRequest

Mandatory: Yes

Property: ctime

Description: Time at which the request was received by dCDN. Time is local to dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Type: AbsoluteTime

Mandatory: Yes

Property: mtime

Description: Time at which the resource was last modified.

Time is local to dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Type: AbsoluteTime

Mandatory: Yes

Property: etime

Description: Estimate of the time at which dCDN expects to complete the activity. Time is local to dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Type: AbsoluteTime

Mandatory: No

Property: status

Description: Current status of the triggered activity.

Type: TriggerStatus

Mandatory: Yes

Property: errors

Description: List of ErrorDesc.

Mandatory: No.

5.3. Properties of ErrorDesc

An ErrorDesc object is used to report failure for URLs and patterns in a trigger request.

Property: error

Type: ErrorCode.

Mandatory: Yes.

Description: List of metadata.urls, content.urls, metadata.patterns, content.patterns

Description: Metadata and content references copied from the trigger request. Only those URLs and patterns to which the error applies shall be included in each property, but those URLs and patterns shall be exactly as they appear in the request, dCDN must not generalise the URLs. (For example, if uCDN requests prepositioning of URLs

"http://ucdn.example.com/a" and "http://ucdn.example.com/b", dCDN may not generalise its error report to Pattern

"http://ucdn.example.com/*").

Mandatory: At least one of these properties is mandatory in each ErrorDesc.
Property: description
Description: A String containing a human-readable description of the error.
Mandatory: No.

5.4. Properties of Trigger Collections

Property: links
Description: References to Trigger Status Resources in the collection.
Type: List of Relationships.
Mandatory: Yes
Property: staleresourcetime
Description: The length of time for which dCDN guarantees to keep a completed Trigger Status Resource. After this time, dCDN MAY delete the resource and all references to it from collections.
Type: Integer, time in seconds.
Mandatory: Yes, in the collection of all Trigger Status Resources if dCDN deletes stale entries. If the property is present in the filtered collections, it MUST have the same value as in the collection of all Trigger Status Resources.

5.5. Trigger Resource Simple Data Type Descriptions

This section describes the simpler data types that are used for properties of Trigger Status resources.

5.5.1. TriggerType

This type defines the type of action being triggered, permitted actions are:

- o Preposition - a request for dCDN to acquire metadata or content.
- o Invalidate - a request for dCDN to invalidate metadata or content. After servicing this request the dCDN will not use the specified data without first re-validating it using, for example, an "If-None-Match" HTTP request. The dCDN need not erase the associated data.
- o Purge - a request for dCDN to erase metadata or content. After servicing the request, the specified data must not be held on dCDN.

5.5.2. TriggerStatus

This type describes the current status of a Trigger, possible values are:

- o Pending - the trigger has not yet been acted upon.
- o Active - the trigger is currently being acted upon.
- o Complete - the triggered activity completed successfully, or the trigger has been accepted and no further status update will be made.
- o Failed - the triggered activity could not be completed.

5.5.3. URLs

This type describes a set of references to metadata or content, it is simply a list of absolute URLs.

5.5.4. AbsoluteTime

Times are expressed in seconds since the UNIX epoch.

5.5.5. ErrorCode

This type is used by dCDN to report failures in trigger processing.

- o EMETA - dCDN was unable to acquire metadata required to fulfil the request.
- o ECONTENT - dCDN was unable to acquire content (preposition triggers only).
- o EPERM - uCDN does not have permission to trigger the requested activity (for example, the data is owned by another CDN).
- o EREJECT - dCDN is not willing to fulfil the request (for example, a preposition request for content at a time when dCDN would not accept Request Routing requests from uCDN).
- o ECDN - An internal error in dCDN or one of its downstream CDNs.

6. JSON Encoding of Objects

This encoding is based on that described in [I-D.ietf-cdni-metadata], but has been reproduced here while metadata work is in progress. Once that work is complete, the authors would look to align with the structure of the metadata draft and make reference to common definitions as appropriate.

The encoding for a CI/T object is a JSON object containing a dictionary of (key,value) pairs where the keys are the property names, and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource). Likewise, the values associated with each key are

dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource).

The "trigger" property of the top level JSON object lists the requested action.

Key: trigger

Description: An object specifying the trigger type and a set of data to act upon.

Type: A JSON object.

Mandatory: Yes.

Object keys in JSON are case sensitive and therefore any dictionary key defined by this document (for example the names of CI/T object properties) MUST always be represented in lowercase.

In addition to the properties of an object, the following additional keys may be present.

Key: base

Description: Provides a prefix for any relative URLs in the object. This is similar to the XML base tag [XML-BASE]. If absent, all URLs in the remainder of the document must be absolute URLs.

Type: URI

Mandatory: No

Key: links

Description: The relationships of this object to other addressable objects.

Type: Array of Relationships.

Mandatory: Yes

6.1. JSON Encoding of Embedded Types

6.1.1. TriggerType

Key: type

Description: One of "preposition", "invalidate" or "purge".

Type: string

6.1.2. TriggerStatus

Key: status

Description: One of "pending", "active", "failed", "complete"

Type: string

6.1.3. PatternMatch

A PatternMatch is encoded as a JSON Object containing a string to match and flags describing the type of match.

Key: pattern

Description: A pattern for string matching. The pattern may contain the wildcards * and ?, where * matches any sequence of characters (including the empty string) and ? matches exactly one character. The three literals \, * and ? should be escaped as \\, * and \?

Type: String

Mandatory: Yes

Key: case-sensitive

Description: Flag indicating whether or not case-sensitive matching should be used.

Type: Boolean

Mandatory: No, default is case-insensitive match.

Key: match-query-string

Description: Flag indicating whether or not the query string should be included in the pattern match.

Type: Boolean

Mandatory: No, default is not to include query.

Example of case-sensitive prefix match against

```
"http://www.example.com/trailers/":  
{  
  "pattern": "http://www.example.com/trailers/*",  
  "case-sensitive": true  
}
```

6.1.4. ErrorDesc

ErrorDesc shall be encoded as a JSON object with the following keys:

Key: error

Type: ErrorCode

Mandatory: Yes

Keys: metadata.urls, content.urls

Type: Array of strings

Mandatory: At least one of metadata.* or content.* must be present.

Keys: metadata.patterns, content.patterns

Type: Array of PatternMatch

Mandatory: At least one of metadata.* or content.* must be present.

Key: description

Type: String
Mandatory: No.

6.1.5. ErrorCode

One of the strings "EMETA", "ECONTENT", "EPERM", "EREJECT" or "ECDN".

6.1.6. Relationship

JSON: A dictionary with the following keys:

- o href - The URI of the of the addressable object being referenced.
- o rel - The Relationship between the referring object and the object it is referencing.
- o type - The MIME Media Type of the referenced object. See Section 6.2 for the MIME Media Types of objects specified in this document.
- o title - An optional title for the Relationship/link.

Note: The above structure follows the pattern of atom:link in [RFC4287].

Example Relationship to a CI/T Resource within a CI/T Collection:

```
{
  "href": "http://triggers.cdni.example.com/trigger/12345",
  "rel": "Trigger",
  "type": "application/vnd.cdni.control.trigger.status+json"
}
```

The format of relationship values is expected to align with other CDNI interfaces. For example, rather than use simple names (like "Trigger" in this case), there may be a namespace that allows well-known and proprietary values to co-exist.

6.2. MIME Media Types

Table 1 lists the MIME Media Type for each trigger object (resource) that is retrievable through the CI/T interface.

Note: A prefix of "vnd.cdni" is used, however it is expected that a more appropriate prefix will be used if the CDNI WG accepts this document.

Data Object	MIME Media Type
TriggerStatus	application/ vnd.cdni.control.trigger.status+json
TriggerCollection	application/ vnd.cdni.control.trigger.collection+json

Table 1: MIME Media Types for CDNI Trigger resources

7. Examples

The following sections provide examples of different CI/T objects encoded as JSON.

No authentication is shown in the following illustrative examples, it is anticipated that authentication mechanisms will be aligned with other CDNI Interfaces as and when those mechanisms are defined.

Discovery of the triggers interface is out of scope of this document. In an implementation, all URLs are under control of dCDN and the uCDN must not attempt to ascribe any meaning to individual elements of the path. In examples in this section, the following URLs are used as the location of the collections of triggers:

- o Collection of all Triggers belonging to one uCDN:
http://dcdn.example.com/triggers
- o Filtered collections:
 - Pending: http://dcdn.example.com/triggers/pending
 - Active: http://dcdn.example.com/triggers/active
 - Complete: http://dcdn.example.com/triggers/complete
 - Failed: http://dcdn.example.com/triggers/failed

7.1. Creating Triggers

Examples of uCDN triggering activity in dCDN:

7.1.1. Preposition

An example of a preposition request, a POST to the "AllTriggers" collection.

Note that "metadata.patterns" and "content.patterns" are not allowed in a preposition trigger request.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: 315
```

```
{
  "trigger" : {
    "type": "preposition",

    "metadata.urls" : [ "http://metadata.example.com/a/b/c" ],
    "content.urls" : [
      "http://www.example.com/a/b/c/1",
      "http://www.example.com/a/b/c/2",
      "http://www.example.com/a/b/c/3",
      "http://www.example.com/a/b/c/4"
    ]
  }
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sat, 23 Feb 2013 14:20:06 GMT
Content-Length: 472
Content-Type: application/vnd.cdni.control.trigger.status+json
Location: http://dcdn.example.com/triggers/0
Server: example-server/0.1
```

```
{
  "ctime": 1361629206,
  "etime": 1361629214,
  "mtime": 1361629206,
  "status": "pending",
  "trigger": {
    "content.urls": [
      "http://www.example.com/a/b/c/1",
      "http://www.example.com/a/b/c/2",
      "http://www.example.com/a/b/c/3",
      "http://www.example.com/a/b/c/4"
    ],
    "metadata.urls": [
      "http://metadata.example.com/a/b/c"
    ],
    "type": "preposition"
  }
}
```

7.1.2. Invalidate

An example of an invalidate request, another POST to the "AllTriggers" collection. This instructs dCDN to re-validate the content at "http://www.example.com/a/index.html", as well as any metadata and content whose URLs are prefixed by "http://metadata.example.com/a/b/" and "http://www.example.com/a/b/" respectively, using case-insensitive matching.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: 352

{
  "trigger" : {
    "type": "invalidate",

    "metadata.patterns" : [
      { "pattern" : "http://metadata.example.com/a/b/*" }
    ],

    "content.urls" : [ "http://www.example.com/a/index.html" ],
    "content.patterns" : [
      { "pattern" : "http://www.example.com/a/b/*",
        "case-sensitive" : true
      }
    ]
  }
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Length: 551
Content-Type: application/vnd.cdni.control.trigger.status+json
Location: http://dcdn.example.com/triggers/1
Server: example-server/0.1

{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
```

```
"trigger": {
  "content.patterns": [
    {
      "case-sensitive": true,
      "pattern": "http://www.example.com/a/b/*"
    }
  ],
  "content.urls": [
    "http://www.example.com/a/index.html"
  ],
  "metadata.patterns": [
    {
      "pattern": "http://metadata.example.com/a/b/*"
    }
  ],
  "type": "invalidate"
}
```

7.2. Examining Trigger Status

Once triggers have been created, uCDN can check their status as shown in these examples.

7.2.1. Collection of All Triggers

The uCDN can fetch the set of all the triggers it has created and which have not yet been deleted or removed as expired. After creation of the "preposition" and "invalidate" triggers shown above, this collection might look as follows:

REQUEST:

```
GET /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "1484827667515030767"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.2. Filtered Collections of Triggers

The filtered collections are also available to uCDN. Before dCDN starts processing the two triggers shown above, both will appear in the collection of Pending Triggers, for example:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-970375801048973013"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

At this point, if no other triggers had been created, the other filtered views of the triggers would be empty. For example:

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 53
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-654105208640281650"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [],
  "staleresourcetime": 86400
}
```

7.2.3. Trigger Status Resources

The Trigger Status Resources can also be examined for detail about individual triggers. For example, for the "preposition" and "invalidate" triggers from previous examples:

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 472
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "-7651038857765989381"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629206,
  "etime": 1361629214,
  "mtime": 1361629206,
  "status": "pending",
  "trigger": {
    "content.urls": [
      "http://www.example.com/a/b/c/1",
      "http://www.example.com/a/b/c/2",
      "http://www.example.com/a/b/c/3",
      "http://www.example.com/a/b/c/4"
    ],
    "metadata.urls": [
      "http://metadata.example.com/a/b/c"
    ],
    "type": "preposition"
  }
}
```

REQUEST:

```
GET /triggers/1 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 551
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-1103964172288811711"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
  "trigger": {
    "content.patterns": [
      {
        "case-sensitive": true,
        "pattern": "http://www.example.com/a/b/*"
      }
    ],
    "content.urls": [
      "http://www.example.com/a/index.html"
    ],
    "metadata.patterns": [
      {
        "pattern": "http://metadata.example.com/a/b/*"
      }
    ],
    "type": "invalidate"
  }
}
```

7.2.4. Polling for Change

The uCDN may use the Entity Tags of collections or resources when polling for change in status, as shown in the following examples:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-970375801048973013"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-970375801048973013"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-7651038857765989381"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "-7651038857765989381"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

When the triggered activity is complete, the contents of the filtered collections will be updated, along with their Entity Tags. For example, when the two example triggers are complete, the collections of pending and complete triggers may look like:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-970375801048973013"
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 53
Expires: Sat, 23 Feb 2013 14:21:13 GMT
Server: example-server/0.1
ETag: "-7056231826368088123"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:13 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [],
  "staleresourcetime": 86400
}
```

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:20 GMT
Server: example-server/0.1
ETag: "2013095476705515794"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.5. Cancelling or Removing a Trigger

To request dCDN to cancel a Trigger, uCDN may delete the Trigger Resource. It may also delete completed and failed triggers to reduce the size of the collections. For example, to remove the "preposition" request from earlier examples:

REQUEST:

```
DELETE /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 204 No Content
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: example-server/0.1
```

This would, for example, cause the collection of completed triggers shown in the example above to be updated to:

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 239
Expires: Sat, 23 Feb 2013 14:21:20 GMT
Server: example-server/0.1
ETag: "4257416552489354137"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.6. Error Reporting

In this example uCDN has requested prepositioning of "http://newsite.example.com/index.html", but dCDN was unable to locate metadata for that site:

REQUEST:

```
GET /triggers/2 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 505
Expires: Sat, 23 Feb 2013 14:21:28 GMT
Server: example-server/0.1
ETag: "2621489144226897896"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:28 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629220,
  "errors": [
    {
      "content.urls": [
        "http://newsite.example.com/index.html"
      ],
      "description":
        "No HostIndex entry found for newsite.example.com",
      "error": "EMETA"
    }
  ],
  "etime": 1361629228,
  "mtime": 1361629224,
  "status": "active",
  "trigger": {
    "content.urls": [
      "http://newsite.example.com/index.html"
    ],
    "type": "preposition"
  }
}
```

8. IANA Considerations

TBD.

9. Security Considerations

The dCDN must ensure that each uCDN only has access to its own Trigger Status Resources.

It is anticipated that a common authentication mechanism will be used by this and other CDNI Interconnect interfaces, the mechanism must exist but is not identified in this document.

The dCDN must ensure that activity triggered by uCDN only affects metadata or content originating from that uCDN.

10. Acknowledgements

TBD.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

11.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-03 (work in progress), February 2013.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata",

draft-ietf-cdni-metadata-01 (work in progress),
February 2013.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network
Interconnection (CDNI) Requirements",
draft-ietf-cdni-requirements-05 (work in progress),
February 2013.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom
Syndication Format", RFC 4287, December 2005.

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

[XML-BASE]

Marsh, J., Ed. and R. Tobin, Ed., "XML Base (Second
Edition) - <http://www.w3.org/TR/xmlbase/>", January 2009.

Authors' Addresses

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: rmurray@velocix.com

Ben Niven-Jenkins
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: ben@velocix.com

Content Delivery Networks
Interconnection
Internet-Draft
Intended status: Informational
Expires: January 17, 2013

J. Seedorf
NEC
July 16, 2012

CDNI Request Routing with ALTO
draft-seedorf-cdni-request-routing-alto-02

Abstract

Network Service Providers (NSPs) are currently considering to deploy Content Delivery Networks (CDNs) within their networks. As a consequence of this development, there is a need for interconnecting these local CDNs. The necessary interfaces for inter-connecting CDNs are currently being defined in the Content Delivery Networks Interconnection (CDNI) WG. This document focusses on the Request Routing Interface of CDNI, and more specifically on how the solutions currently being defined in the Application Layer Traffic Optimization (ALTO) WG can improve CDNI request routing. The overall intention behind this document is to foster discussions (in the CDNI as well as in the ALTO WG) regarding if, how, and under what conditions ALTO can be useful to optimize CDNI request routing. As basis for this discussion, this document provides concrete examples of how ALTO can be integrated within CDNI request routing and in particular in the process of selecting a downstream CDN. The examples in this document are based on the use cases and examples currently being discussed in the CDNI WG.

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 January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. ALTO within CDNI Request Routing	4
3. Assumptions and High-Level Design Considerations	6
4. Selection of a Downstream CDN with ALTO	8
4.1. Footprint Advertisement with ALTO Network Map	8
4.2. Using ALTO maps to convey Additional Information for Downstream CDN Selection	8
4.3. Example of Selecting a Downstream CDN based on ALTO Maps	10
4.4. Advantages of using ALTO	11
5. Useful ALTO extensions for CDNI Request Routing	13
6. Security Considerations	15
7. Summary and Outlook	16
8. Acknowledgements	17
9. Informative References	18
Author's Address	20

1. Introduction

Many Network Service Providers (NSPs) are currently considering or have already started to deploy Content Delivery Networks (CDNs) within their networks. As a consequence of this development, there is a need for interconnecting these local CDNs. Content Delivery Networks Interconnection (CDNI) has the goal of standardizing protocols to enable such interconnection of CDNs [I-D.ietf-cdni-problem-statement].

The CDNI problem statement envisions four interfaces to be standardized within the IETF for CDN interconnection [I-D.ietf-cdni-problem-statement]:

- o CDNI Request Routing Interface
- o CDNI Metadata Interface
- o CDNI Logging Interface
- o CDNI Control Interface

This document focusses solely on the CDNI Request Routing Interface. In particular, this document shows concrete examples of how ALTO [RFC5693] can be integrated in CDNI request routing. The goal of this document is to show in what cases ALTO can benefit CDNI request routing, giving concrete examples and explaining how ALTO improves CDNI request routing in each of these examples. The examples used in this document are based on the use cases and request routing proposals currently being discussed in the CDNI WG [I-D.ietf-cdni-use-cases] [I-D.peterson-CDNI-strawman] and in the ALTO WG [I-D.jenkins-alto-cdn-use-cases]. The overall rationale of this document is to foster discussions (in the CDNI as well as in the ALTO WG) regarding if, how, and under what conditions ALTO can be useful to optimize CDNI request routing. Most importantly, the document has the goal of finding consensus regarding which part of the CDNI request routing interface can use ALTO.

A previous version of this document [I-D.seedorf-alto-for-cdni] contained detailed examples of actual request routing and surrogate selection with ALTO. This version solely focuses on selection of a downstream CDN and how ALTO can support such downstream CDN selection.

Throughout this document, we use the terminology for CDNI defined in [I-D.ietf-cdni-problem-statement].

2. ALTO within CDNI Request Routing

The main purpose of the CDNI Request Routing Interface is described in [I-D.ietf-cdni-problem-statement] as follows: "The CDNI Request Routing interface enables a Request Routing function in an upstream CDN to query a Request Routing function in a downstream CDN to determine if the downstream CDN is able (and willing) to accept the delegated content request and to allow the downstream CDN to control what the upstream Request Routing function should return to the User Agent in the redirection message". On a high level, the scope of the CDNI Request Routing Interface therefore contains two main tasks:

- o A) Determining if the downstream CDN is willing to accept a delegated content request
- o B) Redirecting the content request coming from an upstream CDN to the proper entry point or entity in the downstream CDN

More precisely, in [I-D.ietf-cdni-framework] the request routing interface is broadly divided into two functionalities:

- o 1) the asynchronous advertisement of footprint and capabilities by a dCDN that allows a uCDN to decide whether to redirect particular user requests to that dCDN;
- o 2) the synchronous operation of actually redirecting a user request.

According to consensus found at the CDNI working group session at IETF-82, we refer to 1) as "Request Routing Interface - Footprint and Capabilities Advertisement" and 2) as "Request Routing Interface - Redirection" in this document. A previous version of this document [I-D.seedorf-alto-for-cdni] provided some concrete examples how ALTO could be used for the actual "redirection" part of the request routing interface. Based on feedback received from the CDNI working group (mostly at the IETF-82 meeting), this document solely focuses on the "Footprint and Capabilities Advertisement" part of the request routing interface. In particular, the scope of the current version of this document is to show how ALTO [RFC5693] can be used for selecting a downstream CDN. Thus, the scope of the current document is to provide examples and discuss how a downstream CDN can advertise its footprint and other information by means of ALTO.

Application Layer Traffic Optimization (ALTO) is an approach for guiding the resource provider selection process in distributed applications that can choose among several candidate resources providers to retrieve a given resource. By conveying network layer (topology) information, an ALTO server can provide important

information to "guide" the resource provider selection process in distributed applications. Usually, it is assumed that an ALTO server conveys information these applications cannot measure themselves [RFC5693].

Originally, ALTO was motivated by the huge amount of cross-ISP traffic generated by P2P applications [RFC5693]. Recently, however, ALTO is also being considered for improving the request routing in CDNs [I-D.jenkins-alto-cdn-use-cases]. In this context, it has also been proposed to use ALTO for selecting an entry-point in a downstream NSP's network (see section 3.4 "CDN delivering Over-The-Top of a NSP's network" in [I-D.jenkins-alto-cdn-use-cases]). Also, the CDNI problem statement explicitly mentions ALTO as a candidate protocol for "algorithms for selection of CDN or Surrogate by Request-Routing systems" [I-D.ietf-cdni-problem-statement]. Yet, there have not been concrete proposals so far on how to use ALTO in the context of CDN interconnection. This document tries to close this gap by giving some examples on how ALTO could be used within CDNI request routing.

3. Assumptions and High-Level Design Considerations

In this section we list some assumptions and design issues to be considered when using ALTO for CDNI "footprint and capabilities advertisement":

- o As explicitly being out-of-scope for CDNI [I-D.ietf-cdni-problem-statement], the examples used in this document assume that ingestion of content or acquiring content across CDNs is not part of request routing as considered within CDNI standardization work. The focus of using ALTO (as considered in this document) is hence on request routing only, assuming that the content (desired by the end user) is available in the downstream CDN (or can be acquired by the downstream CDN by some means).
- o Federation Model: "footprint and capabilities advertisement" and in general CDN request routing depends on the federation model among the CDN providers. Designing a suitable solution thus depends on whether a solution is needed for different settings, where CDNs consist of both NSP CDNs (serving individual ASes) and general, traditional CDNs (such as Akamai). We assume that CDNI is not designed for a setting where only NSP CDNs each serve a single AS only.
- o Many CDNs can claim that they can serve any host on the Internet due to Internet connectivity. For example, even if an NSP CDN A does not have surrogate servers inside network C, A can legitimately claim that it can serve customers inside network C. Hence, what a downstream CDN should inform an upstream CDN is performance and capability, and not (only) reachability or coverage. Although one may turn performance into (binary) reachability by defining a threshold, the metric can be content-dependent (image, video, files) or artificial. The requirement of conveying performance and capability is consistent with the context: after all, the foundation of the CDN business is to improve user QoE.
- o In this document, we assume that the upstream CDN (uCDN) makes the decision on selecting a downstream CDN, based on information that each downstream CDN has made available to the upstream CDN. Further, we assume that in principle more than one dCDN may be suitable for a given end-user request (i.e. different dCDNs may claim "overlapping" footprints). The uCDN hence potentially has to select among several candidate downstream CDNs for a given end user request.

- o The term "footprint" has not been precisely defined by the CDNI working group yet [I-D.spp-cdni-rr-foot-cap-semantics]. In this document we assume that some notion of IP prefix-range is suitable to define a footprint of a downstream CDN. Thus, a dCDN footprint may be expressed as an IP-prefix range that a given dCDN claims it can cover. However, in principle any host can reach any other host on the Internet. Thus, simple "coverage" of IP-prefix ranges seems insufficient for a uCDN to make a choice of a dCDN (see [I-D.spp-cdni-rr-foot-cap-semantics] for a discussion of this issue). The uCDN needs additional information that is "tagged along" (either implicitly or explicitly) with such a coverage footprint. Such additional information has the purpose of conveying to the uCDN more information about a footprint, so that the uCDN can judge/assess the delivery quality that is associated with a given dCDN footprint (or part of that footprint, in the likely case that the delivery quality is not the same for a whole dCDN footprint).
- o It is not clear what kind(s) of business, contract, and operational relationships two peering CDNs may form. For the Internet, we see provider-customer and peering as two main relations; providers may use different charging models (e.g., 95-percentile, total volume) and may provide different SLAs. Given such unknown characteristics of CDN peering business agreements, we should design the protocol to support as much diverse potential business and operational models as possible.

4. Selection of a Downstream CDN with ALTO

Under the considerations stated in Section 3, ALTO can help the upstream CDN provider to select a proper downstream CDN provider for a given end user request as follows: Each downstream CDN provider hosts an ALTO server which provides ALTO information (i.e. ALTO network maps and ALTO cost maps [I-D.ietf-alto-protocol]) to an ALTO client at the upstream CDN provider. A network map provided by each of several candidate downstream CDNs can provide information to the upstream CDN provider about each dCDN's "coverage" footprint, e.g. regarding geographical coverage, the (exact or rough) location of "surrogates", the IP-prefix ranges the dCDN claims it can "cover" with "good" delivery quality, or similar. Additional ALTO network maps or cost maps can provide an upstream CDN provider additional information about the footprint each individual dCDN offers, e.g. the "cost" or quality associated with delivering certain content via the downstream CDN which provided such a map. "Cost" in this context is a generic term; many types of costs are possible and can be useful in the context of CDNI request routing (see Section 4.2 for a detailed discussion), e.g. average link load, expected delay, or monetary costs.

4.1. Footprint Advertisement with ALTO Network Map

An ALTO network map contains a "set of Network Location groupings" [I-D.ietf-alto-protocol]. The groupings are defined in the form of so-called "PIDs". A PID is an identifier to group network location endpoints, e.g. IP-addresses in the form of prefixes (see section 4 in [I-D.ietf-alto-protocol] for details).

The concept of an ALTO network map (and the PIDs contained therein) is a natural and straightforward candidate for CDNI footprint advertisement: The downstream CDN provider groups the IP-addresses in its footprint into PIDs and makes these groupings available to an upstream CDN via an ALTO network map. With such a network map, the upstream CDN provider can easily match a given end user request with the footprint of the downstream CDN provider to see if a given downstream CDN can in principle provide "coverage" for the IP-address of the end user. Whenever the footprint changes, the downstream CDN creates an updated network map and makes it available via its ALTO server.

4.2. Using ALTO maps to convey Additional Information for Downstream CDN Selection

Additional information (so that the uCDN can judge/assess the delivery quality that is associated with a given dCDN footprint it received in a network map from the dCDN) can be conveyed to a uCDN with

additional ALTO network maps or ALTO cost maps that the dCDN will provide via its ALTO server. An ALTO cost map contains costs between defined groupings of a corresponding network map (i.e. costs between PIDs): "An ALTO Cost Map defines Path Costs pairwise amongst sets of source and destination Network Locations" [I-D.ietf-alto-protocol]. This concept enables the provider of a cost map to express (and quantify) preferences of a destination network location with respect to a given source network location.

In the context of CDNI, the ALTO cost map concept is an extensive tool to facilitate selection of the "best" downstream CDN because it enables the upstream CDN provider to assess a candidate downstream CDN based on other factors besides simply network coverage (coverage footprint). Most importantly, the cost map concept provides a means for a downstream CDN provider to convey a multitude of dynamically changing information which the upstream CDN provider cannot measure itself (or only roughly estimate) otherwise.

For instance, the following types of "delivery cost" can be conveyed by a downstream CDN provider via ALTO for each combination of source PID and destination PID:

- o Latency: the expected/average RTT
- o Bandwidth: the maximum bandwidth (e.g. due too bottlenecks)
- o Monetary Costs: The amount of actual monetary costs the downstream CDN provider would charge for the delivery of content to a given destination (see also [I-D.liu-cdni-cost])

Normally, an ALTO cost map defines "costs" pairwise among two PIDs [I-D.ietf-alto-protocol]. In the current scope of the CDNI working group, however, the destination of a request routing redirection will always be the request router of the selected downstream CDN (as direct redirection to dCDN surrogates is currently out of scope of the CDNI work). The destination PID in an ALTO cost map offered by a dCDN is thus always (i.e. in all entries) the same. Moreover, this destination PID semantically refers to the whole dCDN (or to its request router, as the decision where to route within the dCDN is outside the scope of the CDNI request routing interface). Therefore, a corresponding ALTO network map for footprint coverage offered by a dCDN should always contain a special PID that covers the whole footprint of the dCDN, i.e. the overall, whole prefix range the dCDN claims it can cover (obviously, it can additionally contain smaller prefix ranges in other PIDs, so that a cost map can have pairwise costs entries of these smaller-scale source PIDs to the overall dCDN coverage PID).

Note that such ALTO cost maps are always of the type N-to-1, i.e. "costs" are expressed for each of N end user source PIDs to 1 single dCDN request router PID. Semantically, the source PID in a CDNI ALTO cost map is thus the end user location, whereas the destination is the request router to which the uCDN redirects the end user request. Note that this perspective is driven by the CDNI request routing. An alternative way - seen from the perspective of content retrieval - would be to have a 1-to-N cost map where the source is always the dCDN and the destination is the end user (with the semantic "if the source dCDN would deliver content to an end user in the destination PID, the costs would be the following").

Alternatively to using cost maps for expressing delivery quality for a given coverage footprint, ALTO networks map could be used. In this case, an additional network map provided by the dCDN groups the dCDN's coverage footprint into several PIDs, where each PID name has a certain "quality" semantic. In other words, all IP-prefixes in a certain PID have the same "quality", and the meaning of this quality is expressed by the PID name.

4.3. Example of Selecting a Downstream CDN based on ALTO Maps

In the following, we will outline an example of dCDN selection by a uCDN based on ALTO maps provided by each dCDN. In the example, an ALTO network map "NM_cov" is used to express the overall "coverage" footprint of each dCDN. In addition (as outlined in Section 4.2), each dCDN provides one or more ALTO cost maps "CM_1", "CM_2", ..., "CM_n" to express the delivery "costs"/quality associated with each PID in the corresponding "NM_cov" coverage footprint network map.

Consider the following example: An upstream CDN (uCDN) has agreed on CDN interconnection with several downstream CDNs (dCDN-a, dCDN-b, and dCDN-c). Each of these downstream CDNs runs an ALTO server to provide information about what locations it can deliver content to (coverage footprint) by means of a network map "NM_cov" and at which "cost" (additional delivery quality information) by means of one or more cost maps "CM_1", "CM_2", ..., "CM_n". uCDN has downloaded from each candidate downstream CDN "NM_cov" and one or more ALTO cost maps (e.g. by using the "Filtered Cost Map" option and different "cost-types" as specified in 7.7.3.2. of [I-D.ietf-alto-protocol]). The ALTO network map provides "coverage" (footprint) for each downstream CDN as aggregated network locations in the form of ALTO PIDs. The cost maps provide the upstream CDN information regarding the delivery quality the selection of each individual downstream CDN would imply depending on the given location of an end user request.

Whenever the upstream CDN receives a request from an end user and has determined that this request is best served by an interconnected

dCDN, the uCDN uses ALTO maps to make a redirection decision. For a given request, assume that only the ALTO network maps provided by dCDN-a and dCDN-c, "NM_cov(dCDN-a)" and "NM_cov(dCDN-c)", indicate that these downstream CDNs can deliver content to the location of the request. In this case, the ALTO costs maps received from dCDN-a and dCDN-c provide useful additional information to the upstream CDN in order to make a selection decision regarding either dCDN-a or dCDN-c. For instance, if both downstream CDNs have provided two ALTO cost maps "CM_monetary" and "CM_latency" - one regarding monetary costs and one regarding expected latency for delivery - uCDN can make a downstream CDN selection based on its preferences: If one downstream CDN can deliver cheaper, but the other faster, ALTO cost maps provide such information in detail to the upstream CDN. This enables the upstream CDN to make a well-considered downstream CDN selection. In particular, the uCDN decision may take into account different delivery quality indicators or other factors (which can be weighted by the upstream CDN to make a decision).

4.4. Advantages of using ALTO

The following reasons make ALTO a suitable candidate protocol for downstream CDN selection as part of CDNI request routing:

- o CDN request routing is done at the application layer. ALTO is a protocol specifically designed to improve application layer traffic (and application layer connections among hosts on the Internet) by providing additional information to applications that these applications could not easily retrieve themselves. For CDNI, this is exactly the case: a uCDN wants to improve application layer CDN request routing by using dedicated information (provided by a dCDN) that the uCDN could not easily obtain otherwise.
- o The semantics of an ALTO network are an exact match for the needed information to convey a footprint by a downstream CDN, in particular if such a footprint is being expressed by IP-prefix ranges.
- o ALTO cost maps are suitable to express various types of delivery "cost" and can hence be used by an upstream to judge the delivery quality associated with a given dCDN for a given end user request. Further, an ALTO cost map can convey relevant network topology information other than simply routing hops or reachability. This facilitates advanced and more sophisticated selection of a downstream CDN based on various metrics by the upstream CDN and increases flexibility to cover different use cases and business models for CDN interconnection.

- o Flexible granularity: The concept of the PID and ALTO network/cost maps allows for different degrees of granularity. This enables a dCDN to differentiate the delivery quality for serving an end user request on a fine granularity depending on the end user location (and not only express delivery quality e.g. on an AS-level). It remains at the discretion of each dCDN how fine-granular the ALTO network and cost maps are that it publishes.
- o ALTO maps can be signed and hence provide inherent integrity protection (see Section 6)

5. Useful ALTO extensions for CDNI Request Routing

It is envisioned that yet-to-be-defined ALTO extensions will be standardized that make the ALTO protocol more suitable and useful for applications other than the originally considered P2P use case [I-D.marocco-alto-next]. Some of these extensions to the ALTO protocol would be useful for ALTO to be used as a protocol within CDNI request routing, and in particular within the "Footprint and Capabilities Advertisement" part of the CDNI request routing interface.

The following proposed extensions to ALTO would be beneficial to facilitate CDNI request routing with ALTO as outlined in Section 4:

- o Server-initiated Notifications and Incremental Updates: In case the footprint or the capabilities of a downstream CDN change abruptly (i.e. unexpectedly from the perspective of an upstream CDN), server initiated notifications would enable a dCDN to directly inform an upstream CDN about such changes. Consider the case where - due to failure - part of the footprint of the dCDN is not functioning, i.e. the CDN cannot serve content to such clients with reasonable QoS. Without server-initiated notifications, the uCDN might still use a very recent network and cost map from dCDN, and therefore redirect request to dCDN which it cannot serve. Similarly, the possibility for incremental updates would enable efficient conveyance of the aforementioned (or similar) status changes by the dCDN to the uCDN. A proposal for server-initiated ALTO updates can be found in [I-D.marocco-alto-ws]. A discussion of incremental ALTO updates can be found in [I-D.schwan-alto-incr-updates].
- o Content Availability on Hosts: A dCDN might want to express CDN capabilities in terms of certain content types (e.g. codecs/formats, or content from certain content providers). A new endpoint property for ALTO that would be able to express such "content availability" would enable a dCDN to make available such information to an upstream CDN. This would enable a uCDN to determine if a given dCDN actually has the capabilities for a given request with respect to the type of content requested.
- o Resource Availability on Hosts or Links: The capabilities on links (e.g. maximum bandwidth) or caches (e.g. average load) might be useful information for an upstream CDN for optimized downstream CDN selection. For instance, if a uCDN receives a streaming request for content with a certain bitrate, it needs to know if it is likely that a dCDN can fulfill such stringent application-level requirements (i.e. can be expected to have enough consistent bandwidth) before it redirects the request. In general, if ALTO

could convey such information via new endpoint properties, it would enable more sophisticated means for downstream CDN selection with ALTO.

6. Security Considerations

One important security consideration is the proper authentication of advertisement information provided by a downstream CDN. The ALTO protocol provides a specification for a signature of ALTO maps (see 8.2.2. of [I-D.ietf-alto-protocol]). ALTO thus provides a proper means for protecting the integrity of footprint advertisement information.

More Security Considerations will be discussed in a future version of this document.

7. Summary and Outlook

This document presented concrete examples of how ALTO can be used within the downstream CDN selection of CDNI Request Routing. Further, the document provides arguments why ALTO is a meaningful protocol in this context. Essentially, ALTO network and cost maps are a means to provide detailed and various types of information to an upstream CDN, in order to facilitate well-considered downstream CDN selection.

The intention of this document is to find consensus in the CDNI WG that ALTO is a useful protocol for CDNI request routing, and that ALTO has many benefits for proper selection of a downstream CDN. The overall objective is to form agreement on how ALTO should be used within the CDNI request routing protocol. It is the intention to capture the outcome of such continuing discussions in future versions of this document.

8. Acknowledgements

Jan Seedorf is partially supported by the COAST project (Content Aware Searching, retrieval and sTreaming, <http://www.coast-fp7.eu>), a research project supported by the European Commission under its 7th Framework Program (contract no. 248036). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the COAST project or the European Commission.

Thanks to Richard Yang for providing valuable comments, and for contributing some design considerations and assumptions.

9. Informative References

- [RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, October 2009.
- [I-D.peterson-CDNI-strawman]
Peterson, L. and J. Hartman, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-peterson-CDNI-strawman-01 (work in progress), May 2011.
- [I-D.ietf-cdni-problem-statement]
Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-ietf-cdni-problem-statement-08 (work in progress), June 2012.
- [I-D.marocco-alto-next]
Marocco, E. and V. Gurbani, "Extending the Application-Layer Traffic Optimization (ALTO) Protocol", draft-marocco-alto-next-00 (work in progress), January 2012.
- [I-D.ietf-alto-protocol]
Alimi, R., Penno, R., and Y. Yang, "ALTO Protocol", draft-ietf-alto-protocol-12 (work in progress), July 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.
- [I-D.ietf-cdni-use-cases]
Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-09 (work in progress), July 2012.
- [I-D.marocco-alto-ws]
Marocco, E. and J. Seedorf, "WebSocket-based server-to-client notifications for the Application-Layer Traffic Optimization (ALTO) Protocol", draft-marocco-alto-ws-01 (work in progress), July 2012.
- [I-D.schwan-alto-incr-updates]
Schwan, N. and B. Roome, "ALTO Incremental Updates",

draft-schwan-alto-incr-updates-02 (work in progress),
July 2012.

[I-D.jenkins-alto-cdn-use-cases]

Niven-Jenkins, B., Watson, G., Bitar, N., Medved, J., and
S. Previdi, "Use Cases for ALTO within CDNs",
draft-jenkins-alto-cdn-use-cases-03 (work in progress),
June 2012.

[I-D.seedorf-alto-for-cdni]

Seedorf, J., "ALTO for CDNI Request Routing",
draft-seedorf-alto-for-cdni-00 (work in progress),
October 2011.

[I-D.ietf-cdni-framework]

Peterson, L. and B. Davie, "Framework for CDN
Interconnection", draft-ietf-cdni-framework-00 (work in
progress), April 2012.

[I-D.liu-cdni-cost]

Liu, H., "A Cost Perspective on Using Multiple CDNs",
draft-liu-cdni-cost-00 (work in progress), October 2011.

[I-D.spp-cdni-rr-foot-cap-semantics]

Seedorf, J., Peterson, J., and S. Previdi, "CDNI Request
Routing: Footprint and Capabilities Semantics",
draft-spp-cdni-rr-foot-cap-semantics-00 (work in
progress), March 2012.

Author's Address

Jan Seedorf
NEC Laboratories Europe, NEC Europe Ltd.
Kurfuersten-Anlage 36
Heidelberg 69115
Germany

Phone: +49 (0) 6221 4342 221
Email: jan.seedorf@neclab.eu
URI: <http://www.neclab.eu>

CDNI	J. Seedorf
Internet-Draft	HFT Stuttgart - Univ. of Applied Sciences
Intended status: Standards Track	Y. Yang
Expires: January 3, 2018	Tongji/Yale
	K. Ma
	Ericsson
	J. Peterson
	Neustar
	July 2, 2017

Content Delivery Network Interconnection (CDNI) Request Routing: CDNI
Footprint and Capabilities Advertisement using ALTO
draft-seedorf-cdni-request-routing-alto-10

Abstract

The Content Delivery Networks Interconnection (CDNI) WG is defining a set of protocols to inter-connect CDNs, to achieve multiple goals such as extending the reach of a given CDN to areas that are not covered by that particular CDN. One componet that is needed to achieve the goal of CDNI is the CDNI Request Routing Footprint & Capabilities Advertisement interface (FCI) [RFC7336]. [RFC8008] has defined precisely the semantics of FCI and provided guidelines on the FCI protocol, but the exact protocol is explicitly outside the scope of that document. In this document, we define an FCI protocol using the Application Layer Traffic Optimization (ALTO) protocol.

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 January 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Background	4
2.1. Semantics of FCI Advertisement	4
2.2. ALTO Background and Benefits	5
3. CDNI FCI ALTO Service	7
3.1. Server Response Encoding	8
3.1.1. Media Type	8
3.1.2. Meta Information	8
3.1.3. Data Information	8
3.2. Protocol Errors	8
3.3. Examples	8
3.3.1. Basic Example	8
3.3.2. Incremental FCI Update Example	9
3.3.3. FCI Using ALTO Network Map Example	9
4. Security Considerations	9
5. Acknowledgements	10
6. References	10
6.1. Normative References	10
6.2. Informative References	11
Authors' Addresses	11

1. Introduction

Many Network Service Providers (NSPs) are currently considering or have already started to deploy Content Delivery Networks (CDNs) within their networks. As a consequence of this development, there is a need for interconnecting these local CDNs. Content Delivery Networks Interconnection (CDNI) has the goal of standardizing protocols to enable such interconnection of CDNs [RFC6707].

The CDNI problem statement [RFC6707] defines four interfaces to be standardized within the IETF for CDN interconnection:

- o CDNI Request Routing Interface
- o CDNI Metadata Interface
- o CDNI Logging Interface
- o CDNI Control Interface

The main purpose of the CDNI Request Routing Interface is described in [RFC6707] as follows: "The CDNI Request Routing interface enables a Request Routing function in an Upstream CDN to query a Request Routing function in a Downstream CDN to determine if the Downstream CDN is able (and willing) to accept the delegated Content Request. It also allows the Downstream CDN to control what should be returned to the User Agent in the redirection message by the upstream Request Routing function." On a high level, the scope of the CDNI Request Routing Interface therefore contains two main tasks:

- o determining if the downstream CDN is willing to accept a delegated content request;
- o redirecting the content request coming from an upstream CDN to the proper entry point or entity in the downstream CDN.

Correspondingly, the request routing interface is broadly divided into two functionalities:

- o CDNI FCI: the advertisement from a dCDN to a uCDN or a query from a uCDN to a dCDN for the uCDN to decide whether to redirect particular user requests to that dCDN;
- o CDNI RI: the synchronous operation of actually redirecting a user request.

This document focuses solely on CDNI FCI, with a goal to specify a new Application Layer Traffic Optimization (ALTO) [RFC7285] service called 'CDNI/FCI Service', to transport and update CDNI FCI JSON objects, which are defined in a separate document in [RFC8008].

Throughout this document, we use the terminology for CDNI defined in [RFC6707] and [RFC8008].

2. Background

The design of CDNI FCI transport using ALTO depends on understanding of both FCI semantics and ALTO. Hence, we start with a review of both.

2.1. Semantics of FCI Advertisement

The CDNI document on "Footprint and Capabilities Semantics" [RFC8008] defines the semantics for the CDNI FCI. It thus provides guidance on what Footprint and Capabilities mean in a CDNI context and how a protocol solution should in principle look like. The definitions in [RFC8008] depend on [RFC8006]. Here we briefly summarize key related points of [RFC8008] and [RFC8006]. For a detailed discussion, the reader is referred to the RFCs.

- o Footprint and capabilities are tied together and cannot be interpreted independently from each other. In such cases, i.e. where capabilities must be expressed on a per footprint basis, it may be beneficial to combine footprint and capabilities advertisement. [RFC8008] integrates footprint and capabilities with an approach of "capabilities with footprint restrictions".
- o Given that a large part of Footprint and Capabilities Advertisement will actually happen in contractual agreements, the semantics of CDNI Footprint and Capabilities advertisement refer to answering the following question: what exactly still needs to be advertised by the CDNI FCI? For instance, updates about temporal failures of part of a footprint can be useful information to convey via the CDNI request routing interface. Such information would provide updates on information previously agreed in contracts between the participating CDNs. In other words, the CDNI FCI is a means for a dCDN to provide changes/updates regarding a footprint and/or capabilities it has prior agreed to serve in a contract with a uCDN. Hence, server push and incremental encoding will be necessary techniques.
- o Multiple types of footprints are defined in [RFC8006]:
 - * List of ISO Country Codes
 - * List of AS numbers
 - * Set of IP-prefixes

A 'set of IP-prefixes' must be able to contain full IP addresses, i.e., a /32 for IPv4 and a /128 for IPv6, and also IP prefixes with an arbitrary prefix length. There must also be support for

multiple IP address versions, i.e., IPv4 and IPv6, in such a footprint.

- o For all of these mandatory-to-implement footprint types, footprints can be viewed as constraints for delegating requests to a dCDN: A dCDN footprint advertisement tells the uCDN the limitations for delegating a request to the dCDN. For IP prefixes or ASN(s), the footprint signals to the uCDN that it should consider the dCDN a candidate only if the IP address of the request routing source falls within the prefix set (or ASN, respectively). The CDNI specifications do not define how a given uCDN determines what address ranges are in a particular ASN. Similarly, for country codes a uCDN should only consider the dCDN a candidate if it covers the country of the request routing source. The CDNI specifications do not define how a given uCDN determines the country of the request routing source. Multiple footprint constraints are additive, i.e. the advertisement of different types of footprint narrows the dCDN candidacy cumulatively.
- o The following capabilities are defined as 'base' capabilities, i.e. ones that are needed in any case and therefore constitute mandatory capabilities to be supported by the CDNI FCI:
 - * Delivery Protocol (e.g., HTTP vs. RTMP)
 - * Acquisition Protocol (for acquiring content from a uCDN)
 - * Redirection Mode (e.g., DNS Redirection vs. HTTP Redirection as discussed in [RFC7336])
 - * Capabilities related to CDNI Logging (e.g., supported logging mechanisms)
 - * Capabilities related to CDNI Metadata (e.g., authorization algorithms or support for proprietary vendor metadata)

2.2. ALTO Background and Benefits

Application Layer Traffic Optimization (ALTO) [RFC7285] is an approach for guiding the resource provider selection process in distributed applications that can choose among several candidate resources providers to retrieve a given resource. By conveying network layer (topology) information, an ALTO server can provide important information to "guide" the resource provider selection process in distributed applications. Usually, it is assumed that an ALTO server conveys information these applications cannot measure themselves [RFC5693].

Originally, ALTO was motivated by the huge amount of cross-ISP traffic generated by P2P applications [RFC5693]. Recently, however, ALTO is also being considered for improving the request routing in CDNs [I-D.jenkins-alto-cdn-use-cases]. In this context, it has also been proposed to use ALTO for selecting an entry-point in a downstream NSP's network (see section 3.4 "CDN delivering Over-The-Top of a NSP's network" in [I-D.jenkins-alto-cdn-use-cases]). Also, the CDNI problem statement explicitly mentions ALTO as a candidate protocol for "algorithms for selection of CDN or Surrogate by Request-Routing systems" [RFC6707].

The following reasons make ALTO a suitable candidate protocol for downstream CDN selection as part of CDNI request routing and in particular for an FCI protocol:

- o CDN request routing is done at the application layer. ALTO is a protocol specifically designed to improve application layer traffic (and application layer connections among hosts on the Internet) by providing additional information to applications that these applications could not easily retrieve themselves. For CDNI, this is exactly the case: a uCDN wants to improve application layer CDN request routing by using dedicated information (provided by a dCDN) that the uCDN could not easily obtain otherwise.
- o The semantics of an ALTO network map are an exact match for the needed information to convey a footprint by a downstream CDN, in particular if such a footprint is being expressed by IP-prefix ranges.
- o Security: ALTO maps can be signed and hence provide inherent integrity protection (see Section 4)
- o RESTful-Design: The ALTO protocol has undergone extensive revisions in order to provide a RESTful design regarding the client-server interaction specified by the protocol. A CDNI FCI interface based on ALTO would inherit this RESTful design.
- o Error-handling: The ALTO protocol has undergone extensive revisions in order to provide sophisticated error-handling, in particular regarding unexpected cases. A CDNI FCI interface based on ALTO would inherit this thought-through and mature error-handling.
- o Filtered network map: The ALTO Map Filtering Service (see [RFC7285] for details) would allow a uCDN to query only for parts of an ALTO map.

- o Server-initiated Notifications and Incremental Updates: In case the footprint or the capabilities of a downstream CDN change abruptly (i.e. unexpectedly from the perspective of an upstream CDN), server initiated notifications would enable a dCDN to directly inform an upstream CDN about such changes. Consider the case where - due to failure - part of the footprint of the dCDN is not functioning, i.e. the CDN cannot serve content to such clients with reasonable QoS. Without server-initiated notifications, the uCDN might still use a very recent network and cost map from dCDN, and therefore redirect request to dCDN which it cannot serve. Similarly, the possibility for incremental updates would enable efficient conveyance of the aforementioned (or similar) status changes by the dCDN to the uCDN. The newest design of ALTO supports server pushed incremental updates [I-D.ietf-alto-incr-update-sse].
- o Content Availability on Hosts: A dCDN might want to express CDN capabilities in terms of certain content types (e.g. codecs/formats, or content from certain content providers). The new endpoint property for ALTO would enable a dCDN to make available such information to an upstream CDN. This would enable a uCDN to determine if a given dCDN actually has the capabilities for a given request with respect to the type of content requested.
- o Resource Availability on Hosts or Links: The capabilities on links (e.g. maximum bandwidth) or caches (e.g. average load) might be useful information for an upstream CDN for optimized downstream CDN selection. For instance, if a uCDN receives a streaming request for content with a certain bitrate, it needs to know if it is likely that a dCDN can fulfill such stringent application-level requirements (i.e. can be expected to have enough consistent bandwidth) before it redirects the request. In general, if ALTO could convey such information via new endpoint properties, it would enable more sophisticated means for downstream CDN selection with ALTO.

3. CDNI FCI ALTO Service

The ALTO protocol is based on an ALTO Information Service Framework which consists of several services, where all ALTO services are 'provided through a common transport protocol, messaging structure and encoding, and transaction model' [RFC7285]. The ALTO protocol specification [RFC7285] defines several such services, e.g. the ALTO map service.

This document defines a new ALTO Service called 'CDNI Footprint & Capabilities Advertisement Service' which conveys JSON objects of media type 'application/cdni'. This media type and JSON object

format is defined in [RFC8006] and [RFC8008]; this document specifies how to transport such JSON objects via the ALTO protocol with the ALTO 'CDNI Footprint & Capabilities Advertisement Service'.

3.1. Server Response Encoding

3.1.1. Media Type

The media type of the CDNI FCI Map is 'application/cdni'.

3.1.2. Meta Information

The 'meta' field of a FCI response MUST include 'vtag', which is an ALTO Version Tag of the retrieved FCIMapData according to [RFC7285] (Section 10.3.). It thus contains a 'resource-id' attribute, and a 'tag' is an identifier string.

3.1.3. Data Information

The data component of a CDNI FCI resource is named 'cdni-fcimap' which is a JSON object defined by [RFC8008]. This JSON object is derived from ResponseEntityBase as specified in the ALTO protocol [RFC7285] (Section 8.4.).

3.2. Protocol Errors

Protocol errors are handled as specified in the ALTO protocol [RFC7285] (Section 8.5.).

3.3. Examples

3.3.1. Basic Example

The following example shows an CDNI FCI response as in [RFC8008], however with meta-information as defined in Section 3.1.2 of this document.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/cdni,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 439
Content-Type: application/cdni
{
  "meta" : {
    "vtag": {
      "resource-id": "my-default-fcimap",
      "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
    }
  },
  "cdni-fcimap": {
    "capabilities": [
      {
        "capability-type": "FCI.DeliveryProtocol",
        "capability-value": {
          "delivery-protocols": [
            "http/1.1",
          ]
        },
        "footprints": [
          <Footprint objects>
        ]
      }
    ]
  }
}
```

3.3.2. Incremental FCI Update Example

3.3.3. FCI Using ALTO Network Map Example

4. Security Considerations

One important security consideration is the proper authentication of advertisement information provided by a downstream CDN. The ALTO protocol provides a specification for a signature of ALTO information (see 8.2.2. of [RFC7285]). ALTO thus provides a proper means for protecting the integrity of FCI information.

More Security Considerations will be discussed in a future version of this document.

5. Acknowledgements

The authors would like to thank Kevin Ma, Daryl Malas, and Matt Caulfield for their timely reviews and invaluable comments.

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

6. References

6.1. Normative References

- [RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, DOI 10.17487/RFC5693, October 2009, <<http://www.rfc-editor.org/info/rfc5693>>.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <<http://www.rfc-editor.org/info/rfc6707>>.
- [RFC6770] Bertrand, G., Ed., Stephan, E., Burbidge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, DOI 10.17487/RFC6770, November 2012, <<http://www.rfc-editor.org/info/rfc6770>>.
- [RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <<http://www.rfc-editor.org/info/rfc7285>>.
- [RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <<http://www.rfc-editor.org/info/rfc7336>>.

- [RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <<http://www.rfc-editor.org/info/rfc7337>>.
- [RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <<http://www.rfc-editor.org/info/rfc8006>>.
- [RFC8008] Seedorf, J., Peterson, J., Previdi, S., van Brandenburg, R., and K. Ma, "Content Delivery Network Interconnection (CDNI) Request Routing: Footprint and Capabilities Semantics", RFC 8008, DOI 10.17487/RFC8008, December 2016, <<http://www.rfc-editor.org/info/rfc8008>>.

6.2. Informative References

- [I-D.ietf-alto-incr-update-sse]
Roome, W. and Y. Yang, "ALTO Incremental Updates Using Server-Sent Events (SSE)", draft-ietf-alto-incr-update-sse-07 (work in progress), July 2017.
- [I-D.jenkins-alto-cdn-use-cases]
Niven-Jenkins, B., Watson, G., Bitar, N., Medved, J., and S. Previdi, "Use Cases for ALTO within CDNs", draft-jenkins-alto-cdn-use-cases-03 (work in progress), June 2012.
- [I-D.ma-cdni-capabilities]
Ma, K. and J. Seedorf, "CDNI Footprint & Capabilities Advertisement Interface", draft-ma-cdni-capabilities-09 (work in progress), April 2016.

Authors' Addresses

Jan Seedorf
HFT Stuttgart - Univ. of Applied Sciences
Schellingstrasse 24
Stuttgart 70174
Germany

Phone: +49-0711-8926-2801
Email: jan.seedorf@hft-stuttgart.de

Y.R. Yang
Tongji/Yale University
51 Prospect Street
New Haven, CT 06511
United States of America

Email: yry@cs.yale.edu
URI: <http://www.cs.yale.edu/~yry/>

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
United States of America

Phone: +1-978-844-5100
Email: kevin.j.ma@ericsson.com

Jon Peterson
NeuStar
1800 Sutter St Suite 570
Concord, CA 94520
United States of America

Email: jon.peterson@neustar.biz

CDNI
Internet-Draft
Intended status: Informational
Expires: April 25, 2013

J. Seedorf
NEC
J. Peterson
Neustar
S. Previdi
Cisco
October 22, 2012

CDNI Request Routing: Footprint and Capabilities Semantics
draft-spp-cdni-rr-foot-cap-semantics-02

Abstract

This document tries to capture the semantics of the "Footprint and Capabilities Advertisement" part of the CDNI Request Routing interface, i.e. the desired meaning and what "Footprint and Capabilities Advertisement" is expected to offer within CDNI. The discussion in this document has the goal to facilitate the choosing of one or more suitable protocols for "Footprint and Capabilities Advertisement" within CDNI Request Routing.

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 April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Apparent Understanding of CDNI Footprint and Capabilities Advertisement	4
2.1. Description of footprint and capabilities advertisement in existing CDNI documents	4
2.2. Summary of understanding of footprint and capabilities advertisement in existing CDNI documents	6
3. Design Decisions for Footprint and Capabilities	7
3.1. Advertising Limited Coverage	7
3.2. Capabilities and Dynamic Data	8
3.3. Advertisement versus Queries	9
3.4. Avoiding or Handling 'cheating' Downstream CDNs	9
3.5. Focus on Main Use Cases may Simplify Things	10
4. Towards Semantics for Footprint Advertisement	11
5. Towards Semantics for Capabilities Advertisement	13
6. Open Issues and Questions	16
7. Security Considerations	17
8. Conclusion	18
9. References	19
9.1. Normative References	19
9.2. Informative References	19
Appendix A. Acknowledgment	20
Authors' Addresses	21

1. Introduction

The CDNI working group is working on a set of protocols to enable the interconnection of multiple CDNs to a CDN federation. This CDN-federation should serve multiple purposes, as discussed in [I-D.ietf-cdni-use-cases], for instance, to extend the reach of a given CDN to areas in the network which are not covered by this particular CDN.

The goal of this document is to achieve a clear understanding in the CDNI WG about the semantics associated with the CDNI request routing interface, in particular regarding the "footprint and capabilities advertisement" of a downstream CDN. To narrow down undecided aspects of these semantics, this document first tries to capture the common understanding of what the "footprint and capabilities advertisement" should offer and accomplish, i.e. what seems to be agreed. Then, the document will discuss open questions. It is the goal of this document to capture the outcome of discussions and answers to open questions in future versions of this draft. In particular, this document summarizes the progress of the recently formed CDNI design team on "footprint and capabilities advertisement".

General assumptions in this document:

- o The CDNs participating in the CDN federation have already performed a boot strap process, i.e., they have connected to each other, either directly or indirectly, and can exchange information amongst each other.
- o The uCDN has received footprint and/or capability advertisements from a set of dCDNs. Footprint advertisement and capability advertisement need not use the same underlying protocol.
- o The upstream CDN (uCDN) receives the initial request-routing request from the endpoint requesting the resource.

This document is organized as follows. We first recap the descriptions regarding "footprint and capabilities advertisement" in existing documents and try to distill the apparent common understanding of the terms "footprint" and "capabilities" in the CDNI request routing context. We then separately discuss the semantics of the footprint advertisement mechanism, and the capability advertisement mechanism. Finally, we list open issues and questions to be discussed in the CDNI WG.

Comments and discussions about this memo should be directed to the CDNI WG: cdni@ietf.org.

2. Apparent Understanding of CDNI Footprint and Capabilities Advertisement

In the following, we will summarize the descriptions of the CDNI "footprint and capabilities advertisement" as part of the "request routing" interface in existing documents. We will then carve out the apparent common understanding of what this interface is intended to offer and accomplish.

2.1. Description of footprint and capabilities advertisement in existing CDNI documents

The CDNI problem statement draft [I-D.ietf-cdni-problem-statement] describes footprint and capabilities advertisement as: "enabling an upstream CDN to determine if a downstream CDN is able (and willing) to accept the delegated content request". In addition, the draft says "the CDNI Request Routing interface is also expected to enable a downstream CDN to provide to the upstream CDN (static or dynamic) information (e.g. resources, footprint, load) to facilitate selection of the downstream CDN by the upstream CDN request routing system when processing subsequent content requests from User Agents". It thus considers "resources" and "load" as capabilities to be advertised by the downstream CDN.

The CDNI use cases draft [I-D.ietf-cdni-use-cases] describes capabilities as "... supported range of devices and User Agents or the supported range of delivery technologies". Examples for such capabilities given are specific delivery protocols, technology migration, and meeting a certain QoS.

The CDNI requirements draft [I-D.ietf-cdni-requirements] lists several requirements relevant for the "footprint and capabilities advertisement" part of the CDNI request routing interface. In summary, the following requirements for the CDNI Request Routing Interface and general requirements are relevant for the understanding of the semantics of the "footprint and capabilities advertisement":

- o GEN-4 [HIGH], "The CDNI solution shall not require intra-CDN information to be exposed to other CDNs for effective and efficient delivery of the content. Examples of intra-CDN information include surrogate topology, surrogate status, cached content, etc."
- o GEN-9 [MED], "The CDNI solution should support cascaded CDN redirection (CDN1 redirects to CDN2 that redirects to CDN3) to an arbitrary number of levels beyond the first level."

- o GEN-10 [MED], "The CDNI solution should support an arbitrary topology of interconnected CDNs (i.e. the CDN topology cannot be restricted to a tree, a loop-free topology, etc.)."
- o GEN-11 [HIGH], "The CDNI solution shall prevent looping of any CDNI information exchange."
- o REQ-1 [HIGH], allowing the downstream CDN "to communicate to the Upstream CDN coarse information about the Downstream CDN ability and/or willingness to handle requests from the Upstream CDN. For example, this could potentially include a binary signal ("Downstream CDN ready/not-ready to take additional requests from Upstream CDN") to be used in case of excessive load or failure condition in the Downstream CDN."
- o REQ-2 [MED], allowing the downstream CDN to communicate capabilities such as supported content types and delivery protocols, a set of metrics/attributes (e.g. Streaming bandwidth, storage resources, distribution and delivery priority), a set of affinities (e.g. Preferences, indication of distribution/delivery fees), information to facilitate request redirection, as well as footprint information (e.g. "layer-3 coverage").
- o REQ-3 [MED], "In the case of cascaded redirection, the CDNI Request-Routing interface shall allow the Downstream CDN to also include in the information communicated to the Upstream CDN, information on the capabilities, resources and affinities of CDNs to which the Downstream CDN may (in turn) redirect requests received by the Upstream CDN. In that case, the CDNI Request-Routing interface shall prevent looping of such information exchange."
- o REQ-4 [LOW], allowing the downstream CDN to communicate "aggregate information on CDNI administrative limits and policy" (e.g. the maximum number of requests redirected by the Upstream CDN to be served simultaneously by the Downstream CDN or maximum aggregate volume of content (e.g. in Terabytes) to be delivered by the Downstream CDN over a time period).
- o REQ-11 [LOW], "The CDNI Request-Routing protocol may support a mechanism allowing an Upstream CDN to avoid redirecting a request to a Downstream CDN if that is likely to result in the total redirection time exceeding some limit."

Note that in REQ-2 [MED] "Layer-3 coverage" is given as an example of what "footprint" information might convey in the CDNI requirements draft [I-D.ietf-cdni-requirements]. Also, note that REQ-3 [MED] addresses cascaded (transitive) downstream CDNs. In such a case, a

downstream CDN needs to include (in its advertisement information that it conveys to an upstream CDN) also information the footprint and capabilities of any further transitive downstream CDN. Such information may be included implicitly (i.e. the cascaded dCDN is oblivious to the uCDN), or explicitly (i.e. the cascaded dCDN of the fact that there is a cascaded dCDN is visible to the uCDN). In any case, a logic is needed to process incoming footprint information from a cascaded dCDN and decide if/how it is to be re-advertised/aggregated when advertising footprint to an upstream CDN.

The CDNI framework draft [I-D.davie-cdni-framework] describes a "footprint" as in [I-D.previdi-cdni-footprint-advertisement], consisting of two parts: 1) "a class of end user requests (represented, for example, by a set of IP prefixes, or a geographic region) that the dCDN is willing and able to serve directly, without use of another dCDN", and 2) "the connectivity of the dCDN to other CDNs that may be able to serve content to users on behalf of dCDN". The term "connectivity" has recently been replaced with "reachability" in [I-D.previdi-cdni-footprint-advertisement]. Further, examples for capabilities are "the ability to handle certain types of content (e.g. specific streaming formats) or quality of service (QoS)."

2.2. Summary of understanding of footprint and capabilities advertisement in existing CDNI documents

In summary, neither the term "footprint" nor "capabilities" are clearly defined. Also, a very broad range of potential capabilities is listed in the existing documents.

3. Design Decisions for Footprint and Capabilities

A large part of the difficulty lies in understanding what we mean when try to define footprint to terms of "coverage" or "reachability." While the operators of CDNs pick strategic locations to situate caches, a cache with a public IPv4 address is reachable by any endpoint on the Internet unless some policy enforcement precludes the use of the cache.

Some CDNs aspire to cover the entire world, henceforth global CDNs; which we will henceforth call global CDNs. The footprint advertised by such a CDN in the CDNi environment would, from a coverage or reachability perspective, presumably cover all prefixes. More interesting for CDNi use cases, however, are CDNs that claim a more limited coverage, but seek to federate with other CDNs in order to create a single CDN fabric which shares resources fairly.

The key to understanding the semantics of footprint and capability advertisement lies in understand why a dCDN would advertise a limited coverage area, and how a uCDN would use such advertisements to decide among one of several dCDNs.

3.1. Advertising Limited Coverage

The basic use case that would motivate a dCDN to advertise a limited coverage is that the CDN was built to cover only a particular portion of the Internet. For example, an ISP could purpose-build a CDN to serve only their own customers by situating caches in close topological proximity to high concentrations of their subscribers. The ISP knows the prefixes it has allocated to end users and thus can easily construct a list of prefixes that its caches were positioned to serve.

When such a purpose-built CDN joined a federation, however, and advertises its footprint to a uCDN, the original intended coverage of the CDN might not represent its actual value to the federation of CDNs. Consider an ISP-A and ISP-B that both field their own CDNs, which they federate through CDNi. A given user E, who is customer of ISP-B, might happen to be topologically closest to a cache fielded by ISP-A, if E happens to live in a region where ISP-B has few customers and ISP-A has many. In this case, should ISP-A's CDN "cover" E? If ISP-B's CDN has a failure condition, should the uCDN understand that ISP-A's caches are potentially available back-ups - and if so, how does ISP-A advertise itself as a "standby" for E? What about the case where CDNs advertising to the same uCDN express overlapping coverage (for example, a federation mixing global and limited CDNs)?

The answers to these questions greatly depend on how much information

we want the uCDN to use to make a selection of a dCDN. If a uCDN has three dCDNs to choose from that "cover" the IP address of user E, obviously the uCDN might be interested to know how optimal the coverage is from each of the dCDNs - coverage need not be binary, either provided or not provided. dCDNs could advertise a coverage "score," for example, and provided that they all reported scores fairly on the same scale, uCDNs could use that to make their topological optimality decision. Alternatively, dCDNs could for their footprint advertise the IP addresses of their caches rather than prefix "coverage," and let the uCDN decide for itself (based on its own topological intelligence) which dCDN has better resources to serve a given user.

3.2. Capabilities and Dynamic Data

In cases where the apparent footprint of dCDNs overlaps, uCDNs might also want to rely on a host of other factors to evaluate the respective merits of dCDNs. These include facts related to the caches themselves, to the network where the cache is deployed, to the nature of the resource sought and to the administrative policies of the respective networks.

In the absence of network-layer impediments to reaching caches, the choice to limit coverage is necessarily an administrative policy. Much policy must be agreed upon before CDNs can merge into federations, including questions of membership, compensation, volumes and so on. A uCDN certainly will factor these sorts of considerations into its decision to select a dCDN, but there is probably little need for dCDNs to actually advertise them through an interface - they will be settled out of band as a precondition for federating.

Other facts about the dCDN would be expressed through the interface to the uCDN. Some capabilities of a dCDN are static, and some are highly dynamic. Expressing the total storage built into its caches, for example, changes relatively rarely, whereas the amount storage in use at any given moment is highly volatile. Network bandwidth similarly could be expressed as either total bandwidth available to a cache, or based on the current state of the network. A cache may at one moment lack a particular resource in storage, but have it the next.

The semantics of the capabilities interface will depend on how much of the dCDN state needs to be pushed to the uCDN in order for it to make a decision.

3.3. Advertisement versus Queries

In a federated CDN environment, each dCDN shares some of its state with the uCDN, which the uCDN uses to build a unified picture of all of the dCDNs available to it. In architectures that share detailed capability information, the uCDN could basically perform the entire request-routing intelligence down to selecting a particular cache before sending the request to the dCDN (note that within the current CDNI WG scope, such direct selection of specific caches by the uCDN is out of scope). However, when the uCDN must deal with many potential dCDNs, this approach does not scale. Especially as CDNs scale up from dozens or hundreds of caches to thousands or tens of thousands, the volume of updates to footprint and capability may become onerous.

Were the volume of updates to exceed the volumes of requests to the uCDN, it might make more sense for the uCDN to query dCDNs upon receiving requests, instead of receiving advertisements and tracking the state of dCDNs itself. The advantage of querying dCDNs would be that much of the dynamic data that dCDNs cannot share with the uCDN would now be factored into the uCDN's decision. dCDNs need not replicate any state to the uCDN - uCDNs could effectively operate in a stateless mode.

The semantics of both footprint and capability advertisement depend on the service model here: are there cases where a synchronous query/response model would work better for the uCDN decision than a state replication model?

3.4. Avoiding or Handling 'cheating' Downstream CDNs

In a situation where more than one dCDN is willing to serve a given end user request, it might be attractive for a dCDN to "cheat" in the sense that the dCDN provides inaccurate information to the uCDN in order to convince the uCDN to select it opposed to "competing" other dCDNs. It is therefore desirable to take away the incentive for dCDNs to cheat (in information advertised) as much as possible. One option here is to make the information the dCDN advertises somehow verifiable for the uCDN. On the other hand, a cheating dCDN might be avoided or handled by the fact that there will be strong contractual agreements between a uCDN and a dCDN, so that a dCDN would risk severe penalties or legal consequences when caught cheating.

Overall, it seems that information a dCDN advertises should (in the long run) be somehow verifiable by the uCDN. However, it is probably an overly strict requirement to demand that such verification must be possible "immediately", i.e. during the request routing process

itself. If the uCDN can detect a cheating dCDN at a later stage, it should suffice for the dCDN to "de-incentivize" cheating because it would negatively affect long-term business relationships with uCDNs.

3.5. Focus on Main Use Cases may Simplify Things

To narrow down semantics for "footprint" and "capabilities" in the CDNI context, it can be useful to initially focus on key use cases to be addressed by the CDNI WG that are to be envisioned the main deployments in the foreseeable future. In this regard, a main realistic use case is the existence of ISP-owned CDNs, which essentially cover a certain operator's network. At the same time, however, the possibility of overlapping footprints should not be excluded, i.e. the scenario where more than one dCDN claims it can serve a given end user request. Further, it seems reasonable to assume that in most use cases it is the uCDN that makes the decision on selecting a certain dCDN for request routing based on information the uCDN has received from this particular dCDN.

4. Towards Semantics for Footprint Advertisement

Based on the characterizations in existing documents (see Section 2), we can distill some "rough" candidates for definitions of a footprint:

- o Footprint could be defined by "layer-3 coverage", where coverage refers to a set of prefixes, a geographic region, or similar boundary.
- o Footprint could alternatively be defined as "a class of end user requests a dCDN is willing to serve".

Independent of the exact definition of footprint, a footprint likely needs to be able to include the connectivity of a given dCDN to other CDNs that may be able to serve content to users on behalf of that dCDN, to cover cases where there is a transitive CDN interconnection. Further, the downstream CDN must be able to express its footprint to an interested upstream CDN (uCDN) in a comprehensive form, e.g., as a complete data set containing the complete footprint. Making incremental updates, however, to express dynamic changes in state is also desirable.

Different concrete candidates for a footprint are imaginable (set of prefixes, a geographic region, ...). Among these, "set of IP-prefixes" may potentially be a useful footprint in the CDDNI context. Such footprint information must be able to contain full IP addresses (i.e., a /32 for IPv4 and a /128 for IPv6) and also IP prefixes with an arbitrary prefix length. There must be support for multiple IP address version, i.e., IPv4 and IPv6 in the footprint.

Roughly speaking, footprint can be defined as "willingness to serve" by a downstream CDN. However, in addition to simply "willingness to serve", the uCDN needs to have some additional information to make a dCDN selection decision. The uCDN needs additional information so that it can assess "how well" a given dCDN can actually serve a given end user request; otherwise, any dCDN can claim it can deliver content to the whole world. One can imagine that such additional information is implicitly associated with a given footprint, e.g. due to (from the request routing interface's perspective out-of-band) contractual agreements (e.g. SLAs), business relationships, or perceived dCDN quality in the past. As an alternative, such additional information could also be explicitly be tagged along with the footprint.

It is reasonable to assume that a significant part of the actual footprint advertisement will happen in contractual agreements between participating CDNs, i.e. prior to the advertisement phase of the CDNI

request routing protocol. The reason for this assumption is that any contractual agreement is likely to contain specifics about the dCDN coverage (i.e. the dCDN footprint) the contractual agreement applies to. In particular, additional information to judge the delivery quality associated with a given dCDN footprint might be defined in contractual agreements (i.e. outside of the CDNI RR interface). Further, one can assume that dCDN contractual agreements about the delivery quality associated with a given footprint will probably be based on high-level aggregated statistics (i.e. not too detailed).

Given that a large part of footprint advertisement will actually happen in contractual agreements, the semantics of CDNI footprint advertisement refer to answering the following question: what exactly still needs to be advertised by the CDNI request routing interface? For instance, updates about temporal failures of part of a footprint can be useful information to convey via the CDNI request routing interface. Such information would provide updates on information previously agreed in contracts between the participating CDNs. In other words, the CDNI request routing footprint advertisement is a means for a dCDN to provide changes/updates regarding a footprint it has prior agreed to serve in a contract with a uCDN.

5. Towards Semantics for Capabilities Advertisement

The dCDN must be able to express its general capabilities to the uCDN. These general capabilities could express if the dCDN supports a given service, for instance, WWW delivery, Video on Demand (VoD) delivery based on flash or apple technologies, or live streaming based on RTSP.

The dCDN must be able to express particular capabilities for the delivery in a particular footprint area. For example, the dCDN might in general offer VoD but not in some areas, either for maintenance reasons or because this particular area cannot deliver this type of service. Hence, in certain cases footprint and capabilities are tied together and cannot be interpreted independently from each other. In such cases, i.e. where capabilities must be expressed on a per footprint basis, it may be beneficial to combine footprint and capabilities advertisement.

High-level and very rough semantics for (and characteristics of) capabilities are thus:

- o Capabilities are types of information that allow a uCDN to determine if a downstream CDN is able (and willing) to accept the delegated content request.
- o Capabilities are types of information that possibly may change over time based on the state of the network or caches.

Candidates for capabilities seem to fall into several broad categories. Some are capabilities associated with a resource itself, like the codecs or streaming technologies in which a particular resource is available. Some capabilities are associated with the cache: these include the load state, available storage resources, and so on. Some capabilities are associated with the network where the cache is deployed, including available bandwidth for streaming, availability of QoS mechanisms, and so on.

Some capabilities reflect administrative restrictions or policies that may affect the decisions made up a uCDN. It seems unlikely these factors will be shared through the interface, however.

Potential Cache capabilities are:

- o aggregate information (i.e. not for individual caches, but still helpful for dCDN selection) about load, or "excessive load"
- o aggregate information (i.e. not for individual caches, but still helpful for dCDN selection) about available resources, storage

resources

- o aggregate information (i.e. not for individual caches, but still helpful for dCDN selection) regarding failure conditions

Potential Resource Capabilities are:

- o supported range of playback devices
- o supported range of delivery technologies
- o specific delivery protocols
- o supported content types (MIME)

Potential Network Capabilities are:

- o meeting a certain QoS
- o distribution and delivery priority
- o streaming bandwidth

Outside the scope of capability advertisements are administrative capabilities, such as:

- o policy (membership in the federation, etc)
- o administrative limits, e.g. maximum aggregate volume of content delivered monthly
- o indication of distribution/delivery fees

At a first glance, the above list of candidates contains useful capabilities to convey via an advertisement interface (and indeed the candidate capabilities listed above have been suggested in CDNI drafts, see Section 2). However, advertising capabilities that change highly dynamically (e.g. real-time delivery performance metrics, CDN resource load, or other highly dynamically changing QoS information) is probably not a good idea. First, out of the multitude of possible metrics and capabilities, it is hard to agree on a subset and the precise metrics to be used. Second, and perhaps more importantly, it seems not feasible to specify such highly dynamically changing capabilities and the corresponding metrics within the CDNI charter time-frame.

Useful capabilities to convey are resource capabilities: Such information does not change highly dynamically and in many cases such

information is absolutely necessary to decide on a dCDN for a given end user request. For instance, if an end user request concerns the delivery of a movie with a certain protocol (e.g. RTMP), the uCDN needs to know if a given dCDN has the capability of supporting this delivery protocol.

Similar to footprint advertisement, it is reasonable to assume that a significant part of the actual (resource) capabilities advertisement will happen in contractual agreements between participating CDNs, i.e. prior to the advertisement phase of the CDNI request routing protocol. The role of capabilities advertisement is hence rather to enable the dCDN to update a uCDN on changes since a contract has been set up (e.g. in case a new delivery protocol is suddenly being added to the list of supported delivery protocols of a given dCDN, or in case a certain delivery protocol is suddenly not being supported anymore due to failures).

Under the above considerations, the semantics of capabilities advertisement are roughly the following:

- o The main category of useful capabilities to convey is resource capabilities.
- o Advertisement refers to conveying information to a uCDN about changes/updates of certain capabilities with respect to a given contract.

Given these semantics, it needs to be decided what capabilities exactly are useful and how these can be expressed. Since the details of CDNI contracts are not known at the time of this writing (i.e. at this point in time there are only very preliminary trials ongoing and no real deployments with actual contracts between CDNs yet), it remains to be seen what capabilities will be used to define agreements between CDNs in practice. One implication for standardization may be to initially only specify very few mandatory capabilities for advertisement and have on top of that a flexible protocol that allows to exchange additional capabilities when needed. Still, agreement needs to be found on what core capabilities are really needed to convey among CDNs. As discussed in Section 3.5, finding the concrete answers to these questions can benefit from focusing on a small number of key use cases that are highly relevant and contain enough complexity to help in understanding what concrete capabilities are needed to facilitate CDN Interconnection.

6. Open Issues and Questions

The following open issues deserve further discussion in the CDNI WG:

- o What is the service model of this interface: Does the uCDN always query the dCDNs? Or does the dCDN always push information to the uCDNs?
- o What exactly is a footprint based on? (e.g. prefix, geographic area, ASN, or location of surrogates/resources)
- o Does a footprint need to include the "reachability" of the dCDN to other CDNs that may be able to serve content to users on behalf of dCDN?
- o What is the assumed business relationship between the uCDN and the dCDN? Is the uCDN always the "authoritative" CDN provider which transitively has itself contracted several downstream CDN providers?
- o How exactly can a given dCDN derive its footprint?
- o To what extent should capability advertisement include or factor in dynamic attributes?

7. Security Considerations

Security considerations will be discussed in a future version of this document.

8. Conclusion

This document tries to capture the semantics of the "Footprint and Capabilities Advertisement" part of the CDNI Request Routing interface, i.e. the desired meaning and what "Footprint and Capabilities Advertisement" is expected to offer within CDNI, also reflecting ongoing discussions in the CDNI design team on "Footprint and Capabilities Advertisement". Several key design decisions and open questions have been discussed.

The discussion in this document has the objective to facilitate the choosing of one or more suitable protocols for "Footprint and Capabilities Advertisement" within CDNI Request Routing. It is the goal of this document to capture the outcome of further progress of the CDNI WG and the design team on "Footprint and Capabilities Advertisement" including answers to currently open questions in future versions of this draft.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

- [I-D.davie-cdni-framework]
Davie, B. and L. Peterson, "Framework for CDN Interconnection", draft-davie-cdni-framework-01 (work in progress), October 2011.
- [I-D.ietf-cdni-problem-statement]
Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-ietf-cdni-problem-statement-08 (work in progress), June 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.
- [I-D.ietf-cdni-use-cases]
Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-10 (work in progress), August 2012.
- [I-D.peterson-cdni-strawman]
Peterson, L. and J. Hartman, "A Simple Approach to CDN Interconnection", draft-peterson-cdni-strawman-01 (work in progress), May 2011.
- [I-D.previdi-cdni-footprint-advertisement]
Previdi, S., Faucheur, F., Faucheur, F., Medved, J., and L. Faucheur, "CDNI Footprint Advertisement", draft-previdi-cdni-footprint-advertisement-02 (work in progress), September 2012.
- [I-D.xiaoyan-cdni-requestrouting]
He, X., Li, J., Dawkins, S., and G. Chen, "Request Routing for CDN Interconnection", draft-xiaoyan-cdni-requestrouting-01 (work in progress), June 2011.

Appendix A. Acknowledgment

Jan Seedorf is partially supported by the CHANGE project (CHANGE: Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions, <http://www.change-project.eu/>), a research project supported by the European Commission under its 7th Framework Program (contract no. 257422). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the CHANGE project or the European Commission.

Jan Seedorf has been partially supported by the COAST project (Content Aware Searching, retrieval and sTreaming, <http://www.coast-fp7.eu/>), a research project supported by the European Commission under its 7th Framework Program (contract no. 248036). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the COAST project or the European Commission.

Martin Stiernerling provided initial input to this document and valuable comments to the ongoing discussions among the authors of this document.

Authors' Addresses

Jan Seedorf
NEC
Kurfuerstenanlage 36
Heidelberg 69115
Germany

Phone: +49 6221 4342 221
Fax: +49 6221 4342 155
Email: seedorf@neclab.eu

Jon Peterson
NeuStar
1800 Sutter St Suite 570
Concord CA 94520
USA

Phone:
Fax:
Email: jon.peterson@neustar.biz

Stefano Previdi
Cisco Systems
Via Del Serafico 200
Rome 0144
Italy

Phone:
Fax:
Email: sprevidi@cisco.com

CDNI
Internet-Draft
Intended status: Informational
Expires: August 29, 2013

J. Seedorf
NEC
J. Peterson
Neustar
S. Previdi
Cisco
R. van Brandenburg
TNO
K. Ma
Azuki Systems, Inc.
February 25, 2013

CDNI Request Routing: Footprint and Capabilities Semantics
draft-spp-cdni-rr-foot-cap-semantics-04

Abstract

This document tries to capture the semantics of the "Footprint and Capabilities Advertisement" part of the CDNI Request Routing interface, i.e. the desired meaning and what "Footprint and Capabilities Advertisement" is expected to offer within CDNI. The discussion in this document has the goal to facilitate the choosing of one or more suitable protocols for "Footprint and Capabilities Advertisement" within CDNI Request Routing.

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 29, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction and scope	3
2. CDNI FCI in existing CDNI Documents	4
3. Design Decisions for Footprint and Capabilities	7
3.1. Advertising Limited Coverage	7
3.2. Capabilities and Dynamic Data	8
3.3. Advertisement versus Queries	9
3.4. Avoiding or Handling 'cheating' dCDNs	9
3.5. Focus on Main Use Cases may Simplify Things	10
4. Main Use Case to foster the Clarification of Semantics	11
5. Towards Semantics for Footprint Advertisement	12
6. Towards Semantics for Capabilities Advertisement	14
7. Open Issues and Questions	17
8. Security Considerations	18
9. References	19
9.1. Normative References	19
9.2. Informative References	19
Appendix A. Acknowledgment	20
Authors' Addresses	21

1. Introduction and scope

The CDNI working group is working on a set of protocols to enable the interconnection of multiple CDNs to a CDN federation. This CDN-federation should serve multiple purposes, as discussed in [RFC6770], for instance, to extend the reach of a given CDN to areas in the network which are not covered by this particular CDN.

The goal of this document is to achieve a clear understanding in the CDNI WG about the semantics associated with the CDNI Request Routing Footprint & Capabilities Advertisement Interface (from now on referred to as FCI), in particular the type of information a downstream CDN 'advertises' regarding its footprint and capabilities. To narrow down undecided aspects of these semantics, this document tries to establish a common understanding of what the FCI should offer and accomplish in the context of CDN Interconnection.

It is explicitly outside the scope of this document to decide on specific protocols to use for the FCI.

General assumptions in this document:

- o The CDNs participating in the CDN federation have already performed a boot strap process, i.e., they have connected to each other, either directly or indirectly, and can exchange information amongst each other.
- o The uCDN has received footprint and/or capability advertisements from a set of dCDNs. Footprint advertisement and capability advertisement need not use the same underlying protocol.
- o The upstream CDN (uCDN) receives the initial request-routing request from the endpoint requesting the resource.

This document is organized as follows. First, a recap of the definition of "footprint and capabilities advertisement" in existing documents is given, attempting to distill the apparent common understanding of what the terms 'footprint' and 'capabilities' mean in the context of CDNI. Then, the detailed semantics of the footprint advertisement mechanism and the capability advertisement mechanism will be discussed. Finally, open issues and questions to be discussed in the CDNI WG will be listed.

2. CDNI FCI in existing CDNI Documents

In the following section, the existing descriptions of the CDNI FCI interface in existing documents will be examined. After this, the apparent common understanding of what this interface is intended to offer and accomplish will be carved out.

The CDNI Problem Statement [RFC6707] describes footprint and capabilities advertisement as: "[enabling] a Request Routing function in an Upstream CDN to query a Request Routing function in a Downstream CDN to determine if the Downstream CDN is able (and willing) to accept the delegated Content Request". In addition, the draft says "the CDNI Request Routing interface is also expected to enable a downstream CDN to provide to the upstream CDN (static or dynamic) information (e.g. resources, footprint, load) to facilitate selection of the downstream CDN by the upstream CDN request routing system when processing subsequent content requests from User Agents". It thus considers "resources" and "load" as capabilities to be advertised by the downstream CDN.

The CDNI Use Cases document [RFC6770] describes capabilities as "... supported range of devices and User Agents or the supported range of delivery technologies". Examples for such capabilities given are specific delivery protocols, technology migration, and meeting a certain QoS.

The CDNI requirements draft [I-D.ietf-cdni-requirements] lists several requirements relevant for the "footprint and capabilities advertisement" part of the CDNI request routing interface. In summary, the following requirements for the CDNI Request Routing Interface and general requirements are relevant for the understanding of the semantics of "footprint and capabilities advertisement":

- o GEN-4 [HIGH], "The CDNI solution shall not require intra-CDN information to be exposed to other CDNs for effective and efficient delivery of the content. Examples of intra-CDN information include surrogate topology, surrogate status, cached content, etc."
- o GEN-9 [MED], "The CDNI solution should support cascaded CDN redirection (CDN1 redirects to CDN2 that redirects to CDN3) to an arbitrary number of levels beyond the first level."
- o GEN-10 [MED], "The CDNI solution should support an arbitrary topology of interconnected CDNs (i.e. the CDN topology cannot be restricted to a tree, a loop-free topology, etc.)."

- o GEN-11 [HIGH], "The CDNI solution shall prevent looping of any CDNI information exchange."
- o REQ-1 [HIGH], allowing the downstream CDN "to communicate to the Upstream CDN coarse information about the Downstream CDN ability and/or willingness to handle requests from the Upstream CDN. For example, this could potentially include a binary signal ("Downstream CDN ready/not-ready to take additional requests from Upstream CDN") to be used in case of excessive load or failure condition in the Downstream CDN."
- o REQ-2 [MED], allowing the downstream CDN to communicate capabilities such as supported content types and delivery protocols, a set of metrics/attributes (e.g. Streaming bandwidth, storage resources, distribution and delivery priority), a set of affinities (e.g. Preferences, indication of distribution/delivery fees), information to facilitate request redirection, as well as footprint information (e.g. "layer-3 coverage").
- o REQ-3 [MED], "In the case of cascaded redirection, the CDNI Request-Routing interface shall allow the Downstream CDN to also include in the information communicated to the Upstream CDN, information on the capabilities, resources and affinities of CDNs to which the Downstream CDN may (in turn) redirect requests received by the Upstream CDN. In that case, the CDNI Request-Routing interface shall prevent looping of such information exchange."
- o REQ-4 [LOW], allowing the downstream CDN to communicate "aggregate information on CDNI administrative limits and policy" (e.g. the maximum number of requests redirected by the Upstream CDN to be served simultaneously by the Downstream CDN or maximum aggregate volume of content (e.g. in Terabytes) to be delivered by the Downstream CDN over a time period).
- o REQ-11 [LOW], "The CDNI Request-Routing protocol may support a mechanism allowing an Upstream CDN to avoid redirecting a request to a Downstream CDN if that is likely to result in the total redirection time exceeding some limit."

Note that in REQ-2 [MED] "Layer-3 coverage" is given as an example of what "footprint" information might convey in the CDNI requirements draft [I-D.ietf-cdni-requirements]. Also, note that REQ-3 [MED] addresses cascaded (transitive) downstream CDNs. In such a case, a downstream CDN needs to include (in its advertisement information that it conveys to an upstream CDN) aggregate footprint and capabilities information for any further transitive downstream CDNs. Such information may be included implicitly (i.e. the cascaded dCDN

is oblivious to the uCDN), or explicitly (i.e. the cascaded dCDN of the fact that there is a cascaded dCDN is visible to the uCDN). In either case, logic is needed to process incoming footprint information from a cascaded dCDN and decide if/how it is to be re-advertised/aggregated when advertising footprint to an upstream CDN.

The CDNI framework draft [I-D.ietf-cdni-framework] describes a "footprint" as in [I-D.previdi-cdni-footprint-advertisement], consisting of two parts: 1) "a class of end user requests (represented, for example, by a set of IP prefixes, or a geographic region) that the dCDN is willing and able to serve directly, without use of another dCDN", and 2) "the connectivity of the dCDN to other CDNs that may be able to serve content to users on behalf of dCDN". The term "connectivity" has recently been replaced with "reachability" in [I-D.previdi-cdni-footprint-advertisement], and as discussed above, "without use of another dCDN" may include aggregated transitive dCDNs. Further examples for capabilities are "the ability to handle certain types of content (e.g. specific streaming formats) or quality of service (QoS)." Content handling capabilities discussed in [I-D.ma-cdni-capabilities] include delivery and acquisition protocols, redirection modes, and metadata related capabilities (e.g., authorization algorithm).

From reading the various draft listed above, it is safe to conclude that neither the term 'footprint' nor 'capabilities' has been clearly and unambiguously defined in these documents and a very broad range of potential capabilities is listed.

3. Design Decisions for Footprint and Capabilities

A large part of the difficulty in discussing the FCI lies in understanding what exactly is meant when trying to define footprint in terms of "coverage" or "reachability." While the operators of CDNs pick strategic locations to situate caches, a cache with a public IPv4 address is reachable by any endpoint on the Internet unless some policy enforcement precludes the use of the cache.

Some CDNs aspire to cover the entire world, which we will henceforth call global CDNs. The footprint advertised by such a CDN in the CDNI environment would, from a coverage or reachability perspective, presumably cover all prefixes. Potentially more interesting for CDNI use cases, however, are CDNs that claim a more limited coverage, but seek to federate with other CDNs in order to create a single CDN fabric which shares resources.

Futhermore, not all capabilities need be footprint restricted. Depending upon the use case, the optimal semantics of "footprints with capability attributes" vs. "capabilities with footprint restrictions" are not clear.

The key to understanding the semantics of footprint and capability advertisement lies in understand why a dCDN would advertise a limited coverage area, and how a uCDN would use such advertisements to decide among one of several dCDNs. The following section will discuss some of the trade-offs and design decisions that need to be decided upon for the CDNI FCI.

3.1. Advertising Limited Coverage

The basic use case that would motivate a dCDN to advertise a limited coverage is that the CDN was built to cover only a particular portion of the Internet. For example, an ISP could purpose-build a CDN to serve only their own customers by situating caches in close topological proximity to high concentrations of their subscribers. The ISP knows the prefixes it has allocated to end users and thus can easily construct a list of prefixes that its caches were positioned to serve.

When such a purpose-built CDN joins a federation, however, and advertises its footprint to a uCDN, the original intended coverage of the CDN might not represent its actual value to the federation of CDNs. Consider an ISP-A and ISP-B that both field their own CDNs, which they federate through CDNI. A given user E, who is customer of ISP-B, might happen to be topologically closest to a cache fielded by ISP-A, if E happens to live in a region where ISP-B has few customers and ISP-A has many. In this case, should ISP-A's CDN "cover" E? If

ISP-B's CDN has a failure condition, should the uCDN understand that ISP-A's caches are potentially available back-ups - and if so, how does ISP-A advertise itself as a "standby" for E? What about the case where CDNs advertising to the same uCDN express overlapping coverage (for example, a federation mixing global and limited CDNs)?

The answers to these questions greatly depend on how much information we want the uCDN to use to make a selection of a dCDN. If a uCDN has three dCDNs to choose from that "cover" the IP address of user E, obviously the uCDN might be interested to know how optimal the coverage is from each of the dCDNs - coverage need not be binary, either provided or not provided. dCDNs could advertise a coverage "score," for example, and provided that they all reported scores fairly on the same scale, uCDNs could use that to make their topological optimality decision. Alternatively, dCDNs could for their footprint advertise the IP addresses of their caches rather than prefix "coverage," and let the uCDN decide for itself (based on its own topological intelligence) which dCDN has better resources to serve a given user.

In summary, the semantics of advertising footprint depend on whether such qualitative metrics for expressing footprint (such as the coverage 'score' mentioned above) should be part of the CDNI FCI, or if it should focus just on 'binary' footprint.

3.2. Capabilities and Dynamic Data

In cases where the apparent footprint of dCDNs overlaps, uCDNs might also want to rely on a host of other factors to evaluate the respective merits of dCDNs. These include facts related to the caches themselves, to the network where the cache is deployed, to the nature of the resource sought and to the administrative policies of the respective networks.

In the absence of network-layer impediments to reaching caches, the choice to limit coverage is necessarily an administrative policy. Much policy must be agreed upon before CDNs can merge into federations, including questions of membership, compensation, volumes and so on. A uCDN certainly will factor these sorts of considerations into its decision to select a dCDN, but there is probably little need for dCDNs to actually advertise them through an interface - they will be settled out of band as a precondition for federating.

Other facts about the dCDN would be expressed through the interface to the uCDN. Some capabilities of a dCDN are static, and some are highly dynamic. Expressing the total storage built into its caches, for example, changes relatively rarely, whereas the amount storage in

use at any given moment is highly volatile. Network bandwidth similarly could be expressed as either total bandwidth available to a cache, or based on the current state of the network. A cache may at one moment lack a particular resource in storage, but have it the next.

The semantics of the capabilities interface will depend on how much of the dCDN state needs to be pushed to the uCDN and qualitatively how often that information should be updated.

3.3. Advertisement versus Queries

In a federated CDN environment, each dCDN shares some of its state with the uCDN, which the uCDN uses to build a unified picture of all of the dCDNs available to it. In architectures that share detailed capability information, the uCDN could basically perform the entire request-routing intelligence down to selecting a particular cache before sending the request to the dCDN (note that within the current CDNI WG scope, such direct selection of specific caches by the uCDN is out of scope). However, when the uCDN must deal with many potential dCDNs, this approach does not scale. Especially as CDNs scale up from dozens or hundreds of caches to thousands or tens of thousands, the volume of updates to footprint and capability may become onerous.

Were the volume of updates to exceed the volumes of requests to the uCDN, it might make more sense for the uCDN to query dCDNs upon receiving requests (as is the case in the recursive redirection mode described in [I-D.ietf-cdni-framework]), instead of receiving advertisements and tracking the state of dCDNs itself. The advantage of querying dCDNs would be that much of the dynamic data that dCDNs cannot share with the uCDN would now be factored into the uCDN's decision. dCDNs need not replicate any state to the uCDN - uCDNs could effectively operate in a stateless mode.

The semantics of both footprint and capability advertisement depend on the service model here: are there cases where a synchronous query/response model would work better for the uCDN decision than a state replication model?

3.4. Avoiding or Handling 'cheating' dCDNs

In a situation where more than one dCDN is willing to serve a given end user request, it might be attractive for a dCDN to 'cheat' in the sense that the dCDN provides inaccurate information to the uCDN in order to convince the uCDN to select it opposed to 'competing' dCDNs. It could therefore be desirable to take away the incentive for dCDNs to cheat (in information advertised) as much as possible. One option

here is to make the information the dCDN advertises somehow verifiable for the uCDN. On the other hand, a cheating dCDN might be avoided or handled by the fact that there will be strong contractual agreements between a uCDN and a dCDN, so that a dCDN would risk severe penalties or legal consequences when caught cheating.

Overall, it seems that information a dCDN advertises should (in the long run) be somehow qualitatively verifiable by the uCDN, though possibly through non-real-time out-of-band audits. It is probably an overly strict requirement to mandate that such verification be possible "immediately", i.e. during the request routing process itself. If the uCDN can detect a cheating dCDN at a later stage, it should suffice for the uCDN to "de-incentivize" cheating because it would negatively affect the long-term business relationship with a particular dCDN.

3.5. Focus on Main Use Cases may Simplify Things

To narrow down semantics for "footprint" and "capabilities" in the CDNI context, it can be useful to initially focus on key use cases to be addressed by the CDNI WG that are to be envisioned the main deployments in the foreseeable future. In this regard, a main realistic use case is the existence of ISP-owned CDNs, which essentially cover a certain operator's network. At the same time, however, the possibility of overlapping footprints should not be excluded, i.e. the scenario where more than one dCDN claims it can serve a given end user request. The ISPs may also choose to federate with a fallback global CDN.

It seems reasonable to assume that in most use cases it is the uCDN that makes the decision on selecting a certain dCDN for request routing based on information the uCDN has received from this particular dCDN. It may be assumed that 'cheating' CDNs will be dealt with via means outside the scope of CDNI and that the information advertised between CDNs is accurate. In addition, excluding the use of qualitative information (e.g., cache proximity, delivery latency, cache load) to predict the quality of delivery would further simplify the use case allowing it to better focus on the basic functionality of the FCI.

4. Main Use Case to foster the Clarification of Semantics

Focusing on a main use case that contains a simple (yet somewhat challenging), realistic, and generally imaginable scenario can help in narrowing down the requirements for the CDNI FCI. To this end, the following (simplified) use case can help in clarifying the semantics of footprint and capabilities for CDNI. In particular, the intention of the use case is to clarify what information needs to be exchanged on the CDNI FCI, what types of information need to be supported in a mandatory fashion (and which should be considered optional), and what types of information need to be updated with respect to a priori established CDNI contracts.

In short, one can imagine the following use case: A given uCDN has several dCDNs. It selects one dCDN for delivery protocol A and footprint 1 and another dCDN for delivery protocol B and footprint 1. The dCDN that serves delivery protocol B has a further, transitive (level-2) dCDN, that serves delivery protocol B in a subset of footprint 1 where the first-level dCDN cannot serve delivery protocol B itself. What happens if capabilities change in the transitive level-2 dCDN that might affect how the uCDN selects a level-1 dCDN (e.g. in case the level-2 dCDN cannot serve delivery protocol B anymore)? How will these changes be conveyed to the uCDN? In particular, what information does the uCDN need to be able to select a new first-level dCDN, either for all of footprint 1 or only for the subset of footprint 1 that the transitive level-2 dCDN served on behalf of the first-level dCDN?

5. Towards Semantics for Footprint Advertisement

Roughly speaking, "footprint" can be defined as "ability and willingness to serve" by a downstream CDN. However, in addition to simple "ability and willingness to serve", the uCDN may wish to have additional information to make a dCDN selection decision, e.g., "how well" a given dCDN can actually serve a given end user request. The "ability and willingness" to serve should be distinguished from the subjective qualitative measurement of "how well" it was served. One can imagine that such additional information is implicitly associated with a given footprint, e.g. due to contractual agreements (e.g. SLAs), business relationships, or perceived dCDN quality in the past. As an alternative, such additional information could also be explicitly tagged along with the footprint.

It is reasonable to assume that a significant part of the actual footprint advertisement will happen in contractual agreements between participating CDNs, i.e. prior to the advertisement phase using the CDNI FCI. The reason for this assumption is that any contractual agreement is likely to contain specifics about the dCDN coverage (i.e. the dCDN footprint) the contractual agreement applies to. In particular, additional information to judge the delivery quality associated with a given dCDN footprint might be defined in contractual agreements (i.e. outside of the CDNI FCI). Further, one can assume that dCDN contractual agreements about the delivery quality associated with a given footprint will probably be based on high-level aggregated statistics (i.e. not too detailed).

Given that a large part of footprint advertisement will actually happen in contractual agreements, the semantics of CDNI footprint advertisement refer to answering the following question: what exactly still needs to be advertised by the CDNI FCI? For instance, updates about temporal failures of part of a footprint can be useful information to convey via the CDNI request routing interface. Such information would provide updates on information previously agreed in contracts between the participating CDNs. In other words, the CDNI FCI is a means for a dCDN to provide changes/updates regarding a footprint it has prior agreed to serve in a contract with a uCDN.

Generally speaking, one can imagine two categories of footprint to be advertised by a dCDN:

- o Footprint could be defined based on (layer-3) "coverage/reachability", where coverage/reachability refers to a set of prefixes, a geographic region, or similar boundary. The dCDN claims that it can cover/reach 'end user requests coming from this footprint'.

- o Footprint could be defined based on "resources", where resources refers to surrogates/caches a dCDN claims to have (e.g., the location of surrogates/resources). The dCDN claims that 'from this footprint' it can serve incoming end user requests.

For each of these footprint types, there are capabilities associated with a given footprint, i.e. the capabilities (e.g., delivery protocol, redirection mode, metadata) supported in the coverage area for a "coverage/reachability" defined footprint, or the capabilities of resources (e.g., delivery protocol, redirection mode, metadata support) for a "resources" defined footprint. It seems clear that "coverage/reachability" types of footprint must be supported within CDNI. It needs to be decided, whether - in addition to "coverage/reachability" types - also "resource" types of footprint should be supported within CDNI.

Different concrete candidates for a "coverage/reachability" footprint are imaginable. In particular, the following concrete types of footprint seem useful in the CDNI context: 'set of IP-prefixes', 'AS number list', and 'geographic region'. Among these, 'set of IP-prefixes' must be able to contain full IP addresses (i.e., a /32 for IPv4 and a /128 for IPv6) and also IP prefixes with an arbitrary prefix length. There must also be support for multiple IP address versions, i.e., IPv4 and IPv6, in the footprint.

Independent of the exact type of a footprint, a footprint might also include the connectivity of a given dCDN to other CDNs that may be able to serve content to users on behalf of that dCDN, to cover cases where there is a transitive CDN interconnection. Further, the downstream CDN must be able to express its footprint to an interested upstream CDN (uCDN) in a comprehensive form, e.g., as a complete data set containing the complete footprint. Making incremental updates, however, to express dynamic changes in state is also desirable.

6. Towards Semantics for Capabilities Advertisement

In general, the dCDN must be able to express its general capabilities to the uCDN. These general capabilities could express if the dCDN supports a given service, for instance, HTTP delivery, RTP/RTSP delivery or RTMP. Furthermore, the dCDN must be able to express particular capabilities for the delivery in a particular footprint area. For example, the dCDN might in general offer RTMP but not in some specific areas, either for maintenance reasons or because the caches covering this particular area cannot deliver this type of service. Hence, in certain cases footprint and capabilities are tied together and cannot be interpreted independently from each other. In such cases, i.e. where capabilities must be expressed on a per footprint basis, it may be beneficial to combine footprint and capabilities advertisement.

A high-level and very rough semantic for capabilities is thus the following: Capabilities are types of information that allow a uCDN to determine if a downstream CDN is able (and willing) to accept (and properly handle) a delegated content request. In addition, Capabilities are characterized by the fact that this information may possibly change over time based on the state of the network or caches.

At a first glance, several broad categories of capabilities seem useful to convey via an advertisement interface (and indeed many such candidate capabilities have been suggested in CDNI drafts, see Section 2). However, advertising capabilities that change highly dynamically (e.g. real-time delivery performance metrics, CDN resource load, or other highly dynamically changing QoS information) should probably not be in scope for the CDNI FCI. First, out of the multitude of possible metrics and capabilities, it is hard to agree on a subset and the precise metrics to be used. Second, and perhaps more importantly, it seems not feasible to specify such highly dynamically changing capabilities and the corresponding metrics within the CDNI charter time-frame.

Useful capabilities refer to information that does not change highly dynamically and which in many cases is absolutely necessary to decide on a particular dCDN for a given end user request. For instance, if an end user request concerns the delivery of a video file with a certain protocol (e.g. RTMP), the uCDN needs to know if a given dCDN has the capability of supporting this delivery protocol.

Similar to footprint advertisement, it is reasonable to assume that a significant part of the actual (resource) capabilities advertisement will happen in contractual agreements between participating CDNs, i.e. prior to the advertisement phase using the CDNI FCI. The role

of capability advertisement is hence rather to enable the dCDN to update a uCDN on changes since a contract has been set up (e.g. in case a new delivery protocol is suddenly being added to the list of supported delivery protocols of a given dCDN, or in case a certain delivery protocol is suddenly not being supported anymore due to failures). Capabilities advertisement thus refers to conveying information to a uCDN about changes/updates of certain capabilities with respect to a given contract.

Given these semantics, it needs to be decided what exact capabilities are useful and how these can be expressed. Since the details of CDNI contracts are not known at the time of this writing (and the CDNI interface should probably be agnostic to these contracts anyway), it remains to be seen what capabilities will be used to define agreements between CDNs in practice. One implication for standardization may be to initially only specify a very limited set of mandatory capabilities for advertisement and have on top of that a flexible data model that allows exchanging additional capabilities when needed. Still, agreement needs to be found on which capabilities (if any) should be mandatory among CDNs. As discussed in Section 3.5, finding the concrete answers to these questions can benefit from focusing on a small number of key use cases that are highly relevant and contain enough complexity to help in understanding what concrete capabilities are needed to facilitate CDN Interconnection.

Under the above considerations, the following capabilities seem useful as 'base' capabilities, i.e. ones that are needed in any case and therefore constitute mandatory capabilities to be supported by the CDNI FCI:

- o Delivery Protocol (e.g., HTTP vs. RTMP)
- o Acquisition Protocol (for acquiring content from a uCDN)
- o Redirection Mode (e.g., DNS Redirection vs. HTTP Redirection as discussed in [I-D.ietf-cdni-framework])
- o Capabilities related to CDNI Logging (e.g., supported logging mechanisms)
- o Capabilities related to CDNI Metadata (e.g., authorization algorithms or support for proprietary vendor metadata)

It is not feasible to enumerate all the possible options for the mandatory capabilities listed above (e.g., all the potential delivery protocols or metadata options) or anticipate all the future needs for additional capabilities. It would be unreasonable to burden the CDNI

FCI specification with defining each supported capability. The CDNI FCI specification should define a generic protocol for conveying any capability information. It would be reasonable to define a registry which initially contains the mandatory capabilities listed above, but may be extended as needs dictate. The CDNI FCI specification SHOULD define the registry (and the rules for adding new entries to the registry) for the different capability types. Each capability type MAY further have a list of valid values. The individual CDNI interface specifications which define a given capability SHOULD define any necessary registries (and the rules for adding new entries to the registry) for the values advertised for a given capability type.

The mandatory capabilities listed above generally relate to information that is configured on a content asset or group of assets basis via CDNI metadata. The capability requirements for acquisition and delivery protocol, redirection mode, and other mandatory metadata capabilities (e.g. authorization algorithms) are defined in [I-D.ietf-cdni-metadata].

Note: CDNI interface support for logging configuration (i.e., control interface vs. metadata interface) has not yet been decided. Once it has been decided, the corresponding CDNI interface specification should define the associated capability requirements.

7. Open Issues and Questions

The following open issues deserve further discussion in the CDNI WG:

- o What is the service model of this interface: Does the uCDN always query the dCDNs? Or does the dCDN always push information to the uCDNs?
- o In addition to "reachability" types of footprint, should also "resource" types of footprint be considered by CDNI (e.g., the location of surrogates/resources a dCDN has)?
- o Does a footprint need to explicitly include the "transitive reachability" of a dCDN to further dCDNs that may be able to serve content to users on behalf of dCDN?
- o What is the assumed business relationship between the uCDN and the dCDN? Is the uCDN always the "authoritative" CDN provider which transitively has itself contracted several downstream CDN providers?
- o How exactly can a given dCDN derive its footprint?
- o Should the footprint/capabilities advertisement interface only signal the delta with respect to a given contract (between a uCDN and a dCDN) or send the whole dCDN state each time?

8. Security Considerations

Security considerations will be discussed in a future version of this document.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-03 (work in progress), February 2013.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata", draft-ietf-cdni-metadata-00 (work in progress), October 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-05 (work in progress), February 2013.
- [I-D.ma-cdni-capabilities]
Ma, K., "Content Distribution Network Interconnection (CDNI) Capabilities Interface", draft-ma-cdni-capabilities-01 (work in progress), February 2013.
- [I-D.previdi-cdni-footprint-advertisement]
Previdi, S., Faucheur, F., Faucheur, F., Medved, J., and L. Faucheur, "CDNI Footprint Advertisement", draft-previdi-cdni-footprint-advertisement-02 (work in progress), September 2012.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.
- [RFC6770] Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, November 2012.

Appendix A. Acknowledgment

Jan Seedorf is partially supported by the CHANGE project (CHANGE: Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions, <http://www.change-project.eu/>), a research project supported by the European Commission under its 7th Framework Program (contract no. 257422). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the CHANGE project or the European Commission.

Jan Seedorf has been partially supported by the COAST project (Content Aware Searching, retrieval and sTreaming, <http://www.coast-fp7.eu/>), a research project supported by the European Commission under its 7th Framework Program (contract no. 248036). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the COAST project or the European Commission.

Martin Stiernerling provided initial input to this document and valuable comments to the ongoing discussions among the authors of this document.

Authors' Addresses

Jan Seedorf
NEC
Kurfuerstenanlage 36
Heidelberg 69115
Germany

Phone: +49 6221 4342 221
Fax: +49 6221 4342 155
Email: seedorf@neclab.eu

Jon Peterson
NeuStar
1800 Sutter St Suite 570
Concord CA 94520
USA

Phone:
Fax:
Email: jon.peterson@neustar.biz

Stefano Previdi
Cisco Systems
Via Del Serafico 200
Rome 0144
Italy

Phone:
Fax:
Email: sprevidi@cisco.com

Ray van Brandenburg
TNO
Brassersplein 2
Delft 2612CT
The Netherlands

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

Kevin J. Ma
Azuki Systems, Inc.
43 Nagog Park
Acton MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.ma@azukisystems.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 10, 2013

E. Stephan
G. Bertrand
A. Marrec
Y. Le Louedec
France Telecom - Orange
November 06, 2012

CDNI Control Operations
draft-stephan-cdni-control-operations-00

Abstract

This memo discusses the role of the CDNI control interface and proposes a framework clarifying the control operations for CDNI.

It does not make any assumption about the protocol implementation, which may be either manual or automated. Furthermore, both uCDN and dCDN could initiate Control operations.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 May 10, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	4
1.1. Terminology	4
2. The role of the CDNI Control Interface	5
2.1. Pre-position, revalidate and purge metadata and content . .	6
2.2. Bootstrapping of the other CDNI interfaces	7
3. CDNI Control Framework	8
3.1. CDNI Control Operations	9
3.2. Control Parameters	9
3.2.1. Service Agreement level	10
3.2.2. Delivery Service Level	13
3.2.3. class-of-requests Level	14
4. IANA Considerations	15
5. Security Considerations	15
6. References	15
6.1. Normative References	15
6.2. Informative References	16
Authors' Addresses	16

1. Introduction

The CDNI control Interface is introduced by [RFC6707] along with the CDNI Metadata, CDNI Request Routing, CDNI Acquisition and CDNI Logging interfaces. Then [I-D.ietf-cdni-framework] specifies the high-level functions performed by each of the CDNI interfaces and the relationships between them. It defines the role of the CDNI control interface as encompassing "the operations to discover, initialize, and parameterize the other CDNI interfaces, as well as operations to pre-position, revalidate, and purge both metadata and content."

Nevertheless [RFC6707] indicates that the "exact inter-CDN control functionality required to be supported by the CDNI Control interface is less well defined than the other three CDNI interfaces", and [I-D.ietf-cdni-framework] reports that "There is some ambiguity as to the line between the set of trigger-based operations in the CDNI Control interface and the Metadata interface". The present memo proposes to fill this gap. It discusses the role of the CDNI control interface and proposes a framework for working out the specification of the control operations for CDNI.

1.1. Terminology

The reader must be familiar with the terminology given by [RFC6707] and by the drafts [I-D.ietf-cdni-requirements] and [I-D.ietf-cdni-framework].

Following are terms and abbreviations introduced in the document:

Control Parameter: A parameter whose value may be exchanged for controlling the configuration of the CDNI interfaces;

Control Operation: The exchange of messages for discovering, initializing, and parameterizing a CDNI interface.

Bootstrapping: A control operation allowing the exchange of the Parameters required to establish the connectivity between servers which enforce the CDNI interfaces, including security aspects like server authentication, encryption, etc.

Class-of-requests: A Class-of-requests identifies a set of content Requests, related to a specific CSP, received from clients in a given footprint and sharing common properties. These properties include:

- * Any header, URL parameter, query parameter of an applicative content request (e.g. of an HTTP content request)

- * Any header, or sub-domain of the FQDN of a DNS lookup request

Examples:

- * Class-of-Requests = all the requests that include the HTTP header "User-Agent: Mozilla/5.0" related to CSP "http://*.cdn.example.com" from AS3215
- * Class-of-Requests = all the DNS requests from anywhere and related to CSP "cdn*.example.com"

Delivery Service: A Delivery Service is defined by a set of Class-of-Requests, related to a given CSP, and a list of parameters that apply to all these Class-of-Requests (logging format, delivery quality/capabilities requirements...)

Service Agreement: A service agreement is defined by a uCDN identifier, a dCDN identifier, a set of Delivery Services and a list of parameters that apply to the Service Agreement.

Once a Service Agreement is agreed between the administrative entities managing the CDNs to be interconnected, the upstream CDN and the downstream CDN of the CDN interconnection must be configured according to this agreed Service Agreement. For instance, a given uCDN (uCDN1) may request a given dCDN (dCDN1) to configure one Delivery Service for handling content requests for HTTP Adaptive streaming videos delegated by uCDN1 and related to a specific CSP (CSP1) and another one for handling content requests for static pictures delegated by uCDN1 and related to CSP1. These Delivery services would belong to the Service Agreement between uCDN1 and dCDN1 for CSP1. In this simple example, uCDN1 may request dCDN1 to include Delivery Service information in its CDNI Logging, to help uCDN1 to provide relevant reports to CSP1.

CDNI Entity: we call CDNI Entity the end-point of a CDNI interface.

2. The role of the CDNI Control Interface

This section discusses the role of the CDNI Control interface.

The [I-D.ietf-cdni-framework] defines the role of the CDNI control interface as:

- o "the operations to discover, initialize, and parameterize the other CDNI interfaces" which are: CDNI request Routing, metadata and Logging.

- * The CDNI Control interface has the responsibility of bootstrapping the other CDNI interfaces, and of updating or deleting instances of the other CDNI interfaces.
- o "as well as operations to pre-position, revalidate, and purge both metadata and content".
- * This is the role of the "trigger" interface which specifications are still work in progress in the draft document[I-D.murray-cdni-triggers].

2.1. Pre-position, revalidate and purge metadata and content

Operations to pre-position, revalidate, and purge both metadata and content are defined in the draft document [I-D.murray-cdni-triggers]. That draft document [I-D.murray-cdni-triggers] defines a "message flow for trigger" based on a RESTfull web services, and specific metadata related activities an upstream CDN can trigger.

The "message flow for trigger" allows:

- o The upstream CDN to trigger activity in downstream CDN ;
- o The upstream CDN to discover the status of the activity.

The activities an upstream CDN can trigger in the downstream CDN and defined in [I-D.murray-cdni-triggers] are all dedicated to operations on content objects and content metadata:

- o Fetch metadata from uCDN or content from any origin server including uCDN (preposition) ;
- o Revalidate specific metadata or content before re-using it (invalidate) ;
- o Delete specific metadata or content purge.

The interface specified in the draft "CDN Interconnect triggers" [I-D.murray-cdni-triggers] should not be part of the CDNI control interface. The main raison is that the trigger interface itself need to be bootstrapped (which is the role of the CDNI control interface).

There are also other reasons, including the following ones:

- o Parameters needed to bootstrap a CDNI interface are static, while trigger-based operations are dynamic ;

- o All trigger activities defined in [I-D.murray-cdni-triggers] are specific to metadata ;
- o The CDNI entities for (i.e. the "extremities" of) the CDNI control interface and for the CDNI trigger interface are not necessary the same.

2.2. Bootstrapping of the other CDNI interfaces

When an upstream CDN operator wants to delegate to a downstream CDN operator the handling of the content requests belonging to a given set of class-of-requests, the uCDN and dCDN establish a Service Agreement. The Service Agreement includes all information required to set up the CDN interconnection. This Service Agreement can be specified with a hierarchical datamodel, as depicted in Figure 1:

- o A service agreement includes an identifier of uCDN, an identifier of dCDN, one or several delivery services, and a set of control parameters that apply to all these delivery services. For example, let us consider a service agreement that includes two delivery services: a former one corresponding to a Vod application with "premium" delivery requirements, and a latter one corresponding to a Vod application with "standard" delivery requirements ;
- o Each Delivery service associates one or several Class-of-requests from the same CSP with a set of control parameters that apply to all these Class-of-requests. To continue with the example introduced just above, let us consider that the delivery service corresponding to the Vod application with "premium" delivery requirements include two different class-of-requests: a former one corresponding to all the requests from end-users based in France for `http://*.cdn.example.com`, and a latter one corresponding to all the requests from end-users in France for `rtmp://*.cdn.example.com` ;
- o Each Class-of-requests identifies uniquely a set of requests with a given footprint and specific properties. In the example introduced just above, one of the considered class-of-request corresponds to all the requests from end-users based in France for `http://*.cdn.example.com`. And another one corresponds to all the requests from end-users in France for `rtmp://*.cdn.example.com` ;

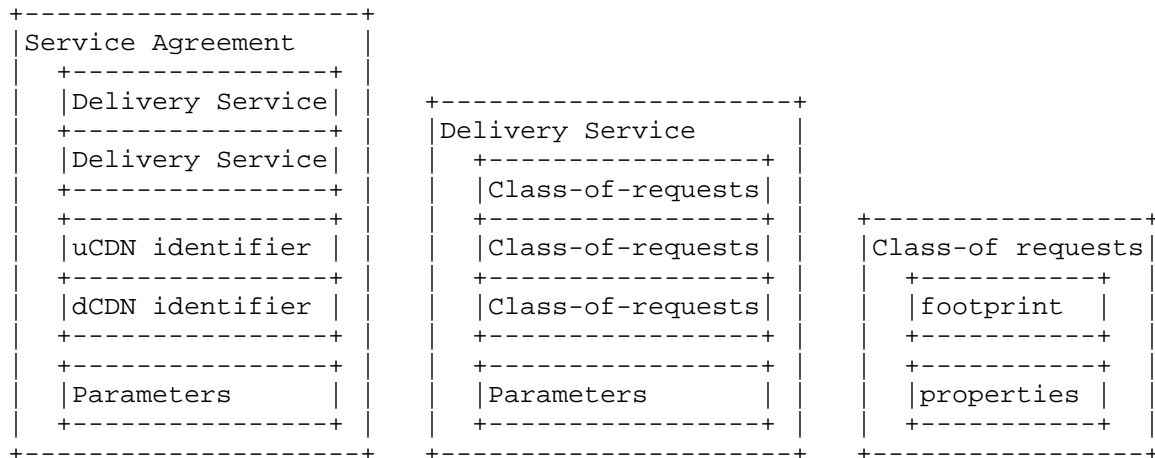


Figure 1

Once a Service Agreement is agreed between the operators of the CDNs to be interconnected, the upstream CDN and the downstream CDN must be configured for this CDNI interconnection according to the agreed Service Agreement. Namely, CDNI Request Routing interface, CDNI metadata interface, CDNI logging interface and CDNI "trigger" interface must be bootstrapped according to the Service Agreement.

3. CDNI Control Framework

The CDNI control operations can possibly be performed either manually or via some automated network protocols; it means the CDNI control interface can possibly be implemented manually or via some automated network protocols. Moreover, only a part of these operations are handled by the CDNI control interface. The other operations are handled directly by the other CDNI interfaces (namely CDNI request routing, CDNI metadata, CDNI logging, CDNI acquisition and CDNI trigger interfaces).

As described in Figure 1, the scope of a Control Operation according to a Control Parameter value may cover either all the Delivery Services of a Service agreement or a single Delivery Service. As an example, the configuration of the format of Logging records may apply to a single Delivery Service. Furthermore, nothing precludes Control Parameters associated to a set of Class-of-requests to change. For instance, an uCDN could require the dCDN to use two Logging formats for a given Delivery Service: one for the normal mode of operations and one temporarily for debugging the CDN interconnection.

3.1. CDNI Control Operations

The enforcement of a Service Agreement requires the configuration of each CDNI functional interface (i.e., including CDNI Metadata, CDNI Logging and CDNI Request Routing).

The value of the Control Parameters is expected to change rarely for a given Delivery Service. The core Control Operations are the following:

1. The configuration of the CDNI interfaces ;
2. The removal of this configuration when it is not required anymore (e.g., end of the scheduling of the considered Delivery Service).

We illustrate Control Operations on Figure 2. The two CDNs exchange the control parameters previously agreed by the operators of the CDNs. Then, each CDN uses these control parameters to configure all the required CDNI interfaces appropriately.

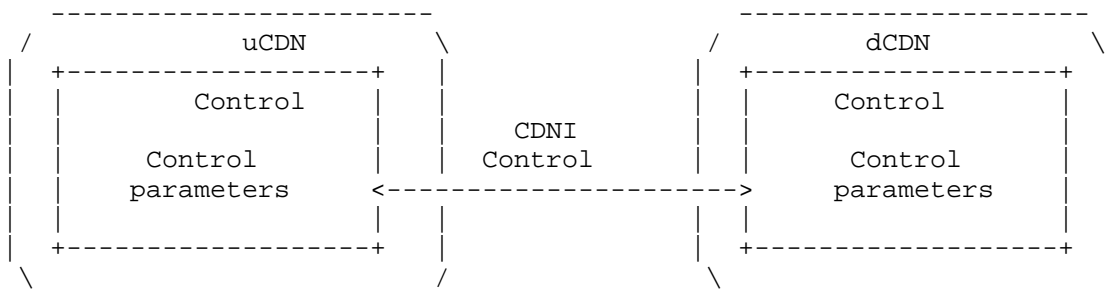


Figure 2

Figure 1: Bootstrapping of CDNI Interfaces

3.2. Control Parameters

It is in the scope of the CDNI Control interface specification to define the minimal set of Control Parameters required for initializing each CDNI interface for a given Service Agreement.

Subsequent sections introduce the information to exchange through the CDNI Control interface to initialize each CDNI interface for a given Service Agreement, a given Delivery Service and a given Class-of-request.

3.2.1. Service Agreement level

This section introduces the information to exchange through the CDNI Control interface to initialize each CDNI interface for a given Service Agreement.

- o Service Agreement identifier;
- o uCDN identifier;
- o dCDN identifier;
- o List of Delivery Service identifier;

Section Section 3.2.2 introduces information to exchange through the CDNI Control interface to initialize each CDNI interface for a given Delivery Service

- o Connectivity and security control parameters;

Section Section 3.2.1.1 lists information to exchange through the CDNI Control interface to establish a secure connectivity among CDNI entities.

- o Redirection control parameters

Section Section 3.2.1.2 lists information to exchange through the CDNI Control interface to redirect Class-of-requests of a given Service Agreement.

3.2.1.1. Connectivity and security control parameters

A CDNI Entity acts as the end point for a CDNI interface. A minimal set of information must be exchanged via the control interface to enable the connectivity and the security among CDNI entities.

As an example securing a CDNI logging interface using HTTPS requires the exchange of certificates between the interconnected CDNs:

1. Certificate identifying dCDN's CDNI logging entity;
2. Certificate identifying uCDN's CDNI logging entity;
3. Certificate of the authority of certification;

A CDN having an CDNi entity (e.g. CDNI Logging data server, CDNI acquisition server) protected by a firewall with strict access rules must add the IP addresses of the entities of the other CDNs which are

allowed to connect to this entity. As examples:

- o When a uCDN has a dedicated CDNI acquisition server protected by a firewall the dCDN must provide the uCDN with the addresses of each dCDN content servers that could acquire content from this CDNI acquisition server;
- o When the CDNI Logging Server of a dCDN is protected by a firewall the uCDN must provide the uCDN with the addresses of the uCDN entities which might access to CDNI logging information on the dCDN Logging server;

Subsequent sections give an overview of the security Control Parameters related to each CDNI interface (CDNI Logging, CDNI Request Routing, CDNI Metadata and CDNI Triggers)

3.2.1.1.1. Triggers

We list below the pieces of information required for the bootstrapping of the triggers interface for a given Service Agreement.

The CDNI trigger interface is a RESTful web service offered by dCDN, according to the draft [I-D.murray-cdni-triggers]. The CDNI Control interface must enable the communication of the following parameters from dCDN to uCDN:

- o the uri of the CDNI trigger entity within dCDN (i.e. of the server within the dCDN dedicated to the CDNI trigger interface):
URI_dCDN_trigger_server;
- o Certificate identifying dCDN;
- o Certificate of the authority of certification;

According to the draft [I-D.murray-cdni-triggers], "The dCDN must ensure that each uCDN only has access to its own Trigger Status Resources and the dCDN must ensure that activity triggered by uCDN only affects metadata or content originating from that uCDN". Consequently, the CDNI Control interface must enable the dCDN to authenticate the uCDN. This means the CDNI Control interface must enable the communication of the following parameter from uCDN to dCDN:

- o Certificate identifying uCDN;
- o Certificate of the authority of certification;

3.2.1.1.2. CDNI Metadata

We list below the pieces of information required for the bootstrapping of the CDNI Metadata interface for a given Service Agreement.

According to the draft [I-D.ietf-cdni-metadata], 'If the URI for the HostIndex object is not manually configured in the downstream CDN then the HostIndex URI could be discovered via the CDNI Control interface. An upstream CDN would provide the URI of the HostIndex object to the downstream CDN via the CDNI Control Interface.'. In this second case, the CDNI Control interface must enable the communication of the following Control Parameters from uCDN to dCDN:

- o the uri of the CDNI metadata entity within dCDN (i.e. of the server within the dCDN dedicated to the CDNI metadata interface):
URI_uCDN_metadata_server;
- o Certificate identifying uCDN metadata server;
- o Certificate of the authority of certification;

According to the draft [I-D.ietf-cdni-metadata], 'If a malicious metadata server is contacted by a downstream CDN, the malicious server may provide metadata to the downstream CDN which denies service for any piece of content to any user agent. The malicious server may also provide metadata which directs a downstream CDN to a malicious origin server instead of the actual origin server. A malicious metadata client could request metadata for a piece of content from an upstream CDN. The metadata information may then be used to glean information regarding the uCDN or to contact an upstream origin server. The uCDN is expected to authenticate client requests to prevent this situation.' This means the CDNI Control interface must enable the communication of the following parameter from uCDN to dCDN:

- o Certificate identifying dCDN metadata server;
- o Certificate of the authority of certification;

3.2.1.1.3. CDNI Logging

We list below the pieces of information required for the bootstrapping of the CDNI Logging interface for a given Service Agreement.

- o URI and protocol to use to request CDNI logging data;

- * URI_dCDN_logging_server;
 - * the protocol to use to exchange CDNI Logging data;
 - * etc.
- o parameters securing the CDNI logging interface

3.2.1.1.4. CDNI Request Routing

We list below the pieces of information required for the bootstrapping of the CDNI Request Routing interface for a given Service Agreement.

- o Control parameters securing the Request Routing CDNI interface
 - * Certificate identifying dCDN Request Routing/redirection CDNI entity;
 - * Certificate identifying uCDN Request Routing/redirection CDNI entity;
 - * Certificate identifying dCDN Request Routing/footprint CDNI entity;
 - * Certificate identifying uCDN Request Routing/footprint CDNI entity;
 - * Certificate of the authority of certification;

[Ed. note: this section will be updated once there will be an agreed specification for the CDNI Request Routing interface within the CDNI WG.]

3.2.1.2. Redirection parameters

This section lists the piece of information to exchange through the CDNI Control interface to redirect Class-of-requests of the Service Agreement.

[Ed. note: this section will be updated once there will be an agreed specification for the CDNI Request Routing interface within the CDNI WG.]

3.2.2. Delivery Service Level

This section introduces the information to exchange through the CDNI Control interface to initialize each CDNI interface for a given

delivery Service.

- o Delivery Service identifier;
- o List of identifiers of class-of-requests ;

Section Section 3.2.3 introduces information to exchange through the CDNI Control interface to initialize each CDNI interface for a given Class-of-requests

- o CDNI Logging parameters;

Section Section 3.2.2.1 lists information to exchange through the CDNI Control interface to describe the CDNI logging data for a given Delivey Service.

- o CDNI Delivery requirement parameters

Section Section 3.2.2.2 lists information to exchange through the CDNI Control interface to describe requirements for handling class-of-requests of a given Delivey Service.

3.2.2.1. CDNI Logging parameters

This section lists a first set of information to exchange through the CDNI Control interface to describe CDNI logging data for a given Delivery Service:

- o CDNI Logging data format,
- o etc.

3.2.2.2. CDNI Delivery Requirements parameters

This section lists information to exchange through the CDNI Control interface to describe requirements for handling class-of-requests of a given Delivery Service.

[Ed. note: this section will be updated once there will be an agreed specification for the CDNI Request Routing interface within the CDNI WG.]

3.2.3. class-of-requests Level

This section introduces the information to exchange through the CDNI Control interface to initialize each CDNI interface for a given Class-of-request.

- o class-of-requests identifier;

- o footprint;

- [Ed. note: this section will be updated once there will be an agreed specification for the CDNI Request Routing interface within the CDNI WG.]

- o Content request commun properties;

- [Ed. note: this section will be updated once there will be an agreed specification for the CDNI Request Routing interface within the CDNI WG.]

4. IANA Considerations

This document makes no request of IANA.

5. Security Considerations

The bootstrapping and the configuration of the CDNI interfaces are very sensible operations.

Any weakness may harm the CDN that is directly attacked, as well as the CDNs interconnected with it. In addition it may endanger the security of the eyeballs and of the distributed content.

6. References

6.1. Normative References

[I-D.ietf-cdni-framework]

Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-01 (work in progress), July 2012.

[I-D.ietf-cdni-metadata]

Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata", draft-ietf-cdni-metadata-00 (work in progress), October 2012.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements",

draft-ietf-cdni-requirements-03 (work in progress),
June 2012.

[I-D.ietf-cdni-use-cases]

Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-10 (work in progress), August 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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

[I-D.bertrand-cdni-logging]

Bertrand, G., Emile, S., Peterkofsky, R., Faucheur, F., and P. Grochocki, "CDNI Logging Interface", draft-bertrand-cdni-logging-02 (work in progress), October 2012.

[I-D.murray-cdni-triggers]

Murray, R. and B. Niven-Jenkins, "CDN Interconnect Triggers", draft-murray-cdni-triggers-01 (work in progress), August 2012.

Authors' Addresses

Emile Stephan
France Telecom - Orange
2 avenue Pierre Marzin
Lannion F-22307
France

Email: emile.stephan@orange.com

Gilles Bertrand
France Telecom - Orange
38-40 rue du General Leclerc
Issy les Moulineaux, 92130
France

Phone: +33 1 45 29 89 46
Email: gilles.bertrand@orange.com

Anne Marrec
France Telecom - Orange
2 avenue Pierre Marzin
Lannion, 22307
France

Phone: +33 2 96 05 18 71
Email: anne.marrec@orange.com

Yannick Le Louedec
France Telecom - Orange
2 avenue Pierre Marzin
Lannion, 22307
France

Phone: +33 2 96 05 17 64
Email: yannick.lelouedec@orange.com

