

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

M. Caulfield
Cisco Systems
October 27, 2014

CDNI Rate Pacing
draft-caulfield-cdni-rate-pacing-02

Abstract

Rate pacing is a class of network traffic shaping which limits the transmission rate of data over a network. This document defines CDNI extensions for downstream CDNs to support rate pacing on behalf of upstream CDNs.

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 30, 2015.

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	2
2. CDNI Interfaces Impact	3
2.1. Footprint & Capabilities Interface	3
2.2. Metadata Interface	3
2.2.1. RatePacing Metadata	4
2.3. Request Routing Redirection Interface	4
2.4. Logging Interface	5
2.5. Control Interface	5
3. Token Bucket Rate Pacing Algorithm and Parameters	5
3.1. TokenBucketParams Object	6
3.2. Token Bucket Metadata Example	6
4. IANA Considerations	7
4.1. CDNI Rate Pacing Algorithms Registry	7
5. Security Considerations	7
6. Acknowledgements	8
7. Normative References	8
Author's Address	8

1. Introduction

Rate pacing is a class of network traffic shaping which limits the transmission rate of data over a network. In the context of a Content Delivery Network (CDN), rate pacing provides an important business advantage to a Content Service Provider (CSP) by ensuring that a CDN which is delivering content on behalf of that CSP does not deliver significantly more data than necessary to an end client.

For example, suppose an end client is watching some Constant Bit Rate (CBR) video encoded at 1500 kbps. In the absence of rate pacing, the CDN delivering this content may send it to the client at 3000 kbps. If the client chooses to terminate the session before watching the entire video, up to half the transmitted data is wasted. This waste leads to unnecessary cost for the CSP and diminished useful capacity for the CDN.

Rate pacing requires configuration on a per-content basis. In order to enable rate pacing in a CDNI environment, the CDNI interfaces need to be extended to optionally support this feature.

This document describes:

1. CDNI interface extensions required for supporting rate pacing
 2. a token bucket rate pacing algorithm for CDNs
2. CDNI Interfaces Impact
- 2.1. Footprint & Capabilities Interface

[I-D.ietf-cdni-footprint-capabilities-semantics] defines the CDNI Footprint and Capabilities semantics. But at the time of writing, no FCI syntax specification has been accepted as a working group document.

[I-D.ietf-cdni-footprint-capabilities-semantics] states that:

"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."

This document defines a new capability type: "RatePacing" to be added to the FCI capability types registry. The value of this capability contains one or more rate pacing algorithm names from the Rate Pacing algorithms registry (Section 4.1). For example, the value may be "token-bucket/v1" to indicate that the advertising CDN supports the token bucket algorithm described later in this document.

A CDN MAY advertise the "RatePacing" capability in the FCI if it implements this specification. A CDN MUST advertise the "token-bucket/v1" as a value in the list of algorithms if it advertises "RatePacing" as a capability.

2.2. Metadata Interface

A new RatePacing metadata object is defined to represent the configuration for rate pacing. The RatePacing object has MIME type "application/cdni.RatePacing.v1". RatingPacing MAY appear within the metadata list of either HostMetadata or PathMetadata (i.e. may have either host-level scope or a path-level scope). The following section defines the properties of the RatingPacing object.

2.2.1. RatePacing Metadata

The presence of the RatePacing Metadata indicates that a dCDN MUST comply with this specification in order to deliver a piece of content. The metadata indicates the rate pacing algorithm name required for delivering the content and the relevant parameters for that algorithm.

Property: algo

Description: Rate pacing algorithm name from the Rate Pacing Algorithms registry. Dictates the structure of the "params" value. For example, "token-bucket/v1".

Type: String

Mandatory-to-Specify: Yes.

Property: params

Description: An object containing algorithm-specific properties and values which are relevant to the rate pacing algorithm specified by the "algo" property. Each algorithm dictates its own parameters.

Type: Object

Mandatory-to-Specify: Yes.

2.3. Request Routing Redirection Interface

The RRI is not impacted by rate pacing. However, if the metadata for a piece of content indicates that rate pacing is required by the uCDN, then a request router should only redirect requests for that content to CDNs which advertise "RatePacing" as a capability. The request router should also limit its choice of dCDNs to those which advertise the same rate pacing algorithm as is specified by the rate pacing metadata. Note that this behavior is not specific to rate pacing and is true of any CDNI feature.

For example, if the metadata for a piece of content includes a GenericMetadata object of type "application/cdni.RatePacing.v1" and the "algo" property in the value of that GenericMetadata is "token-bucket/v1", then the request router of the uCDN should only redirect requests for that piece of content to dCDNs which advertise a capability type of "RatePacing" and a capability value of "token-bucket/v1".

2.4. Logging Interface

The rate at which a piece of content was delivered MAY be indicated via the LI. The "sc-rate" field indicates the rate in bytes per second as a decimal number. The bytes measured should correspond to the sc-entity-bytes field.

sc-rate:

format: DEC

field value: the average rate in bytes per second at which a response was delivered from Surrogate to client.

occurrence: there MUST be zero or exactly one instances of this field.

Note that existing fields defined by [I-D.ietf-cdni-logging] include the bytes delivered and the time taken to service a request, which could be used to estimate the delivery rate. However, the time taken includes the acquisition latency which is not relevant to rate pacing.

2.5. Control Interface

The CI is not impacted by rate pacing.

3. Token Bucket Rate Pacing Algorithm and Parameters

Token bucket is one example of a rate pacing algorithm. Token bucket is described generically by [RFC1363].

The token bucket algorithm is characterized by two parameters:

1. Rate - the number of tokens added to the bucket per second
2. Size - the maximum number of tokens in the bucket

This document specifies the CDNI Token Bucket Rate Pacing algorithm. It is based on the generic token bucket algorithm described above, but applied to a CDNI context.

For the purpose of this document, each token represents one byte transmitted as part of the body of an HTTP response from a Surrogate in a dCDN. Tokens do not represent bytes which are part of HTTP headers, the HTTP status line, TCP signaling, or any lower layer protocol.

The algorithm name "token-bucket/v1" is registered as a Rate Pacing algorithm. This algorithm name MUST appear as the value of the "RatePacing" capability. This name may also appear as the value of the "algo" property in the "RatePacing" metadata object.

If a RatePacing metadata object's "algo" value is "token-bucket/v1" then the metadata object's "params" MUST be an object of type TokenBucketParams, described below.

3.1. TokenBucketParams Object

Property: rate

Description: Rate of tokens per second to be added to the bucket as described by the token bucket algorithm. This value MUST be a positive integer. Each token represents one byte.

Type: Integer

Mandatory-to-Specify: Yes.

Property: size

Description: Maximum number of tokens per bucket as described by the token bucket algorithm. This value MUST be a positive integer.

Type: Integer

Mandatory-to-Specify: Yes.

3.2. Token Bucket Metadata Example

```
{
  "metadata": [
    {
      "generic-metadata-type": "application/cdni.RatePacing.v1",
      "generic-metadata-value": {
        "algo": "token-bucket/v1",
        "params": {
          "rate": 100000,
          "size": 25000
        }
      }
    }
  ]
}
```

4. IANA Considerations

This document requests the following of IANA:

Addition of RatePacing in the CDNI Capability Registry defined in TBD.

Addition of "RatePacing" to the standard partition of the CDNI GenericMetadata Type Registry defined in [I-D.ietf-cdni-metadata]:

Type name	Specificati on	Version	MTE	STR
application/cdni.RatePacing .v1	RFCthis	1	true	true

Addition of "sc-rate" in the CDNI Logging Field Names Registry defined in [I-D.ietf-cdni-logging].

4.1. CDNI Rate Pacing Algorithms Registry

IANA is requested to create a new registry, CDNI Rate Pacing Algorithms. The following table defines the initial values of the registry:

Algorithm Name	Specification
token-bucket/v1	RFCthis

New rate pacing algorithm registrations MUST specify RatePacing parameter objects as shown in Section 3.1 and MUST describe the algorithm for rate pacing.

5. Security Considerations

A malicious CSP might attempt to use rate pacing to instruct a dCDN to delivery some content at a very low rate thereby in order to exhaust the resources of a dCDN by forcing connection state to be maintained for longer than usual. The decision to enforce a rate is left to the discretion of a dCDN. An implementation of rate pacing should implement reasonable lower (and upper) bounds to avoid such cases.

6. Acknowledgements

The author would like to thank Francois Le Faucheur for his contributions and feedback.

7. 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-03 (work in progress), July 2014.
- [I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Opreescu, I., and R.
Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-
logging-14 (work in progress), October 2014.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., Leung, K.,
and K. Ma, "CDN Interconnection Metadata", draft-ietf-
cdni-metadata-07 (work in progress), July 2014.
- [RFC1363] Partridge, C., "A Proposed Flow Specification", RFC 1363,
September 1992.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

Author's Address

Matt Caulfield
Cisco Systems
1414 Massachusetts Ave
Boxborough, MA 01719
USA

Phone: +1 978 936 9307
Email: mcaulfie@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 7, 2015

R. Murray
B. Niven-Jenkins
Velocix (Alcatel-Lucent)
September 3, 2014

CDNI Control Interface / Triggers
draft-ietf-cdni-control-triggers-04

Abstract

This document describes the part of the CDN Interconnection 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 invalidates or purges 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 7, 2015.

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	4
2. Model for CDNI Triggers	4
2.1. Timing of Triggered Activity	6
2.2. Trigger Results	6
3. Collections of Trigger Status Resources	6
4. CDNI Trigger Interface	7
4.1. Creating Triggers	8
4.2. Checking Status	10
4.2.1. Polling Trigger Status Resource collections	10
4.2.2. Polling Trigger Status Resources	10
4.3. Cancelling Triggers	10
4.4. Deleting Triggers	11
4.5. Expiry of Trigger Status Resources	12
4.6. Loop Detection and Prevention	12
4.7. Error Handling	13
4.8. Content URLs	13
5. CI/T Object Properties and Encoding	14
5.1. CI/T Objects	14
5.1.1. CI/T Commands	14
5.1.2. Trigger Status Resource	15
5.1.3. Trigger Collection	16
5.2. Properties of CI/T Objects	17
5.2.1. Trigger Specification	18
5.2.2. Trigger Type	19
5.2.3. Trigger Status	19
5.2.4. PatternMatch	20
5.2.5. Absolute Time	21
5.2.6. Error Description	21
5.2.7. Error Code	22
6. Examples	22
6.1. Creating Triggers	23
6.1.1. Preposition	23
6.1.2. Invalidate	24
6.2. Examining Trigger Status	25
6.2.1. Collection of All Triggers	25

6.2.2.	Filtered Collections of Triggers	26
6.2.3.	Trigger Status Resources	28
6.2.4.	Polling for Change	30
6.2.5.	Removing a Trigger	33
6.2.6.	Error Reporting	35
7.	IANA Considerations	36
7.1.	Media type registrations	36
7.1.1.	CI/T Commands	36
7.1.2.	CI/T Trigger Status Resource	37
7.1.3.	CI/T Trigger Collection	38
8.	Security Considerations	39
8.1.	Authentication, Confidentiality, Integrity Protection	39
8.2.	Denial of Service	39
9.	Acknowledgements	40
10.	References	40
10.1.	Normative References	40
10.2.	Informative References	40
	Authors' Addresses	41

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).

[RFC7336] expands on the information provided in [RFC6707] and describes each of the interfaces and the relationships between them in more detail.

This document describes the "CI/T" interface, "CDNI Control interface / Triggers". It does not consider those parts of the control interface that relate to configuration, bootstrapping or authentication of CDN Interconnect interfaces. Section 4 of [RFC7337] identifies the requirements specific to the CI interface, requirements applicable to the CI/T interface are CI-1 to CI-6.

- o Section 2 outlines the model for the CI/T Interface at a high level.
- o Section 3 describes collections of Trigger Status Resources.
- o Section 4 defines the web service provided by the dCDN.
- o Section 5 lists properties of CI/T Commands and Status Resources.
- o Section 6 contains example messages.

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

2. Model for CDNI Triggers

A trigger, sent from the uCDN to the dCDN, is a request for the dCDN to do some work relating to data originating from the uCDN.

The trigger can request action on either metadata or content, the following actions can be requested:

- o preposition - used to instruct the dCDN to fetch metadata from the uCDN, or content from any origin including the uCDN.
- o invalidate - used to instruct the dCDN to revalidate specific metadata or content before re-using it.
- o purge - used to instruct the dCDN to delete specific metadata or content.

Multiple representations of an HTTP resource may share the same URL. Requests to invalidate and purge metadata or content apply to all resource representations with matching URLs.

The CI/T interface is a web service offered by the dCDN. It allows creation and deletion of triggers, and tracking of the triggered activity. When the dCDN accepts a trigger it creates a resource describing status of the triggered activity, a Trigger Status Resource. The uCDN can poll Trigger Status Resources to monitor progress.

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 the dCDN, the uCDN POSTs a CI/T Command to the collection of Trigger Status Resources. If the dCDN accepts the trigger, it creates a new Trigger Status Resource and returns its location to the uCDN. To monitor progress, the uCDN can GET the Trigger Status Resource. To request cancellation of a trigger the uCDN can POST to the collection of Trigger Status Resources, or simply DELETE the Trigger Status Resource.

In addition to the collection of all Trigger Status Resources for the uCDN, the dCDN can maintain 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 the dCDN, and for the uCDN to discover the status of that activity. Only successful triggering is shown. Examples of the messages are given in Section 6.

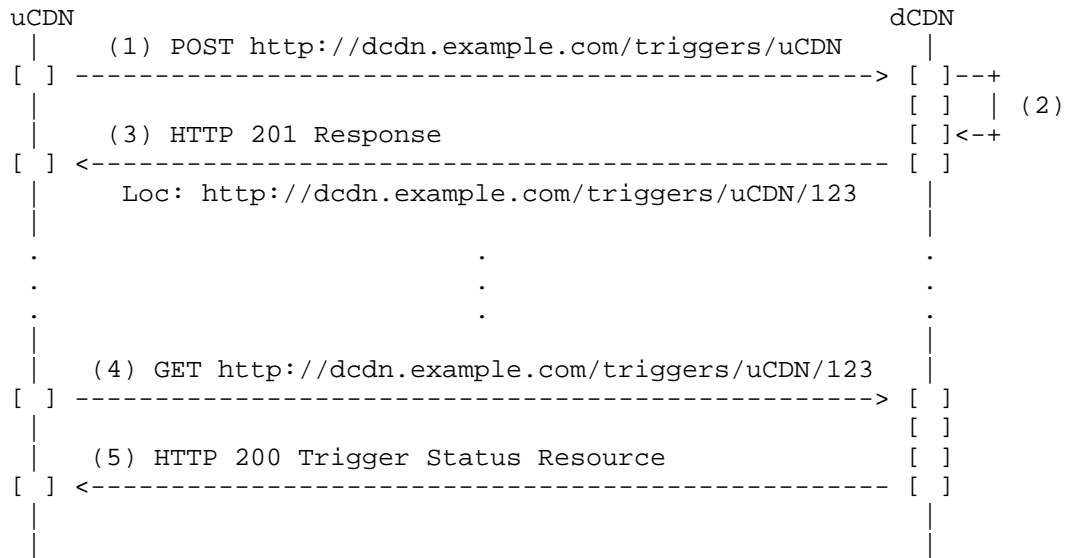


Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are:

1. The uCDN triggers action in the dCDN by posting a CI/T Command to a collection of Trigger Status Resources, "http://dcdn.example.com/triggers/uCDN". The URL of this was given to the uCDN when the trigger interface was established.
2. The dCDN authenticates the request, validates the trigger and if it accepts the request, creates a new Trigger Status Resource.
3. The dCDN responds to the uCDN with an HTTP 201 response status, and the location of the Trigger Status Resource.
4. The uCDN can repeatedly poll the Trigger Status Resource in the dCDN.
5. The 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 the execution of triggered activity is under the dCDN's 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 the dCDN. The dCDN MAY apply the triggers to data acquired after trigger creation.

If the uCDN wishes to invalidate or purge content, then immediately pre-position replacement content at the same URLs, it SHOULD ensure the dCDN has completed the invalidate/purge before initiating the prepositioning. Otherwise, there is a risk that the dCDN pre-positions the new content, then immediately invalidates or purges it (as a result of the two uCDN requests running in parallel).

2.2. Trigger Results

Each trigger can operate on multiple metadata and content URLs. The trigger MUST NOT be reported as "complete" until all actions have been completed successfully. The reasons for failure, and URLs or Patterns affected, SHOULD be enumerated in the Trigger Status Resource. For more detail, see section Section 4.7.

If a dCDN is also acting as a 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. If a trigger is reported as 'processed' in any dCDN, intermediate CDNs MUST NOT report 'complete', instead they must also report 'processed'. A trigger MAY be reported as 'failed' as soon as it fails in a CDN or in any of its downstream CDNs. A cancelled trigger MUST be reported as 'cancelling' until it has been reported as 'cancelled', 'complete', or 'failed' by all dCDNs in a cascade.

3. Collections of Trigger Status Resources

As described in Section 2, Trigger Status Resources exist in the 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.

The dCDN MUST make a collection of a uCDN's Trigger Status Resources available to that uCDN. This collection includes all of the uCDN triggers that have been accepted by the dCDN, and have not yet been deleted by the uCDN, or expired and removed by the dCDN (as described in section Section 4.4). Trigger Status Resources belonging to a uCDN MUST NOT be visible to any other CDN. The dCDN could, for example, achieve this by offering different collection URLs to each uCDN, or by filtering the response based on the client uCDN.

To trigger activity in a dCDN, or to cancel triggered activity, the uCDN POSTs a CI/T Command to the dCDN's collection of the uCDN's Trigger Status Resources.

In order to allow the uCDN to check the status of multiple jobs in a single request, the dCDN SHOULD also maintain collections representing filtered views of the collection of all Trigger Status Resources. If it implements these filtered collections, the dCDN MUST include links to them in the collection of all triggers. 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 the dCDN.
- o Complete - Trigger Status Resources representing activity that completed successfully and 'processed' triggers for which no further status updates will be made by the dCDN.
- o Failed - Trigger Status Resources representing activity that failed or was cancelled by the uCDN.

4. CDNI Trigger Interface

This section describes an interface to enable an upstream CDN to trigger activity in a downstream CDN.

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 dCDNs implementing CI/T that contain a response body SHOULD include an ETag to enable validation of cached versions of returned resources.

All dCDNs implementing CI/T MUST support the HTTP GET, HEAD, POST and DELETE methods as defined in [RFC7231]. The only representation specified in this document is JSON, [RFC7159].

The URL of the dCDN's collection of all Trigger Status Resources needs to be either discovered by, or configured in, the uCDN. The mechanism for discovery of that URL is outside the scope of this document.

CI/T Commands are POSTed to the dCDN's collection of all Trigger Status Resources. If a command to create a new trigger is accepted by the dCDN, it creates a new Trigger Status Resource and returns its URI to the dCDN in an HTTP 201 response. The triggered activity can then be monitored by the uCDN using that resource and the collections described in Section 3.

The URI of each Trigger Status Resource is returned to the uCDN when it is created. This means all Trigger Status Resources can be discovered, so dCDNs are free to assign whatever structure they desire to the URIs for CI/T resources. Therefore uCDNs MUST NOT make any assumptions regarding the structure of CI/T URIs or the mapping between CI/T objects and their associated URIs. URIs present in the examples in this document 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 dCDNs may make use of any HTTP feature when implementing the CI/T interface. For example, a dCDN SHOULD make use of HTTP's caching mechanisms to indicate that a requested response/representation has not been modified, reducing the uCDN's processing needed to determine whether the status of triggered activity has changed.

The dCDN MUST ensure that activity triggered by the uCDN only affects metadata or content originating from that uCDN.

4.1. Creating Triggers

To create a new trigger, the uCDN makes an HTTP POST to the dCDN's collection of all of the uCDN's Trigger Status Resources. The request body of that POST is a CI/T Command with a "trigger", as described in Section 5.1.1.

The dCDN validates and authenticates that request, if it is malformed or the uCDN does not have sufficient access rights it MUST either respond with an appropriate 4xx HTTP error code and a resource MUST NOT be created on the dCDN, or create a 'failed' Trigger Status Resource containing an appropriate error description.

If the request is accepted, the uCDN MUST create a new Trigger Status Resource. The HTTP response to the dCDN MUST have status code 201 and the URI of the Trigger Status Resource in the Location header field. The HTTP response SHOULD 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 the dCDN MUST NOT re-use its location, even after that resource has been removed.

The "trigger" property of the Trigger Status Resource contains the Trigger Specification posted in the body of the CI/T Command. 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 dCDN is not able to track the execution of triggered activity, it MUST indicate that it has accepted the request but will not be providing further status updates. To do this, it sets the "status" of the Trigger Status Resource to "processed". In this case, CI/T processing should continue as for a "complete" request, so the Trigger Status Resource MUST be added to the dCDN's collection of Complete Triggers. The dCDN SHOULD also provide an estimated completion time for the request, by using the "etime" property of the Trigger Status Resource. This will allow the uCDN to schedule repositioning after an earlier delete of the same URLs is expected to have finished.

If the dCDN is able to track the execution of triggered activity, the trigger is queued by the dCDN for later action, the "status" property of the Trigger Status Resource MUST be "pending". Once trigger processing has started the "status" MUST be "active". Finally, once the triggered activity is complete, the trigger status MUST be set to "complete" or "failed".

A trigger may result in no activity in the dCDN if, for example, it is an invalidate or purge request for data the dCDN has not yet acquired, or a prepopulate request for data it has already acquired and which is still valid. In this case, the "status" of the Trigger Status Resource MUST be "processed" or "complete", and the Trigger Status Resource MUST be added to the dCDN's collection of Complete Triggers.

Once created, Trigger Status Resources can be cancelled or deleted by the uCDN, but not modified. The dCDN MUST reject PUT and POST requests from the uCDN to Trigger Status Resources by responding with an appropriate HTTP status code.

4.2. Checking Status

The uCDN has two ways to check progress of activity it has triggered in the dCDN, described in sections Section 4.2.1 and Section 4.2.2.

To check for change in status of a resource or collection of resources without re-fetching the whole resource or collection, Entity Tags SHOULD be included by the dCDN for the uCDN to use as cache validators, as defined in [RFC7232].

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 the uCDN should poll for change.

4.2.1. Polling Trigger Status Resource collections

The 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 the dCDN moves a Trigger Status Resource from the Active to the Completed collection, the uCDN can fetch the result of that activity.

When polling in this way, the uCDN SHOULD use HTTP Entity Tags to monitor for change, rather than repeatedly fetching the whole collection.

4.2.2. Polling Trigger Status Resources

The uCDN has a URI provided by the dCDN for each Trigger Status Resource it has created, it may fetch that resource at any time.

This can be used to retrieve progress information, and to fetch the result of triggered activity.

When polling in this way, the uCDN SHOULD use HTTP Entity Tags to monitor for change, rather than repeatedly fetching the Trigger Status Resource.

4.3. Cancelling Triggers

The uCDN can request cancellation of a trigger by POSTing a Trigger "cancel" Command to the collection of all triggers.

The uCDN should respond to that command appropriately, for example with HTTP status code 200 "OK" if the cancellation has been processed and the trigger is inactive, 202 "Accepted" if the command has been

accepted but the trigger remains active, or 403 "Forbidden" if cancellation is not supported by the dCDN.

If cancellation of a "pending" Trigger Status Resource is accepted by the dCDN, the dCDN SHOULD NOT start processing of that activity. Cancelling a "pending" trigger does not however guarantee that not activity is started, because the uCDN cannot control the timing of that activity. Processing could, for example, start after the POST is sent by the uCDN but before that request is processed by the dCDN.

If cancellation of an "active" or "processed" Trigger Status Resource is accepted by the dCDN, the dCDN SHOULD stop processing the triggered activity. However, as with cancellation of a "pending" trigger, the dCDN does not guarantee this.

If the triggered activity cannot be stopped immediately, the trigger's status MUST be set to "cancelling" and the Trigger Status Resource remains in the collection of active triggers. If processing is stopped before normal completion, the trigger's status MUST be set to "cancelled" and included in the collection of failed triggers.

Cancellation of a "complete" or "failed" Trigger Status Resource requires no processing in the dCDN, its status MUST NOT be changed to "cancelled".

4.4. Deleting Triggers

The uCDN can delete Trigger Status Resources at any time, using the HTTP DELETE method. The effect is similar to cancellation, but no Trigger Status Resource remains afterwards.

Once deleted, the references to a Trigger Status Resource MUST be removed from all Trigger Status Resource collections. Subsequent requests to GET the deleted resource SHOULD fail.

If a "pending" Trigger Status Resource is deleted, the dCDN SHOULD NOT start processing of that activity. Deleting a "pending" trigger does not however guarantee that it has not started because the uCDN cannot control the timing of that activity. Processing may, for example, start after the DELETE is sent by the uCDN but before that request is processed by the dCDN.

If an "active" or "processed" Trigger Status Resource is deleted, the dCDN MAY stop processing the triggered activity. However, as with deletion of a "pending" trigger, the dCDN does not guarantee this.

Deletion of a "complete" or "failed" Trigger Status Resource requires no processing in the dCDN other than deletion of the resource.

4.5. Expiry of Trigger Status Resources

The dCDN can choose to automatically delete Trigger Status Resources some time after they become "complete", "processed", "failed" or "cancelled". In this case, the dCDN will remove the resource and respond to subsequent requests for it with an HTTP error.

If the dCDN performs this housekeeping, it MUST have reported the length of time after which completed Trigger Status Resources will be deleted via a property of the collection of all Trigger Status Resources. It is recommended that Trigger Status Resources are not automatically deleted for at least 24 hours after they become "complete", "processed", "failed" or "cancelled".

To ensure it is able to get the status of its completed and failed triggers, it is recommended that the uCDN's polling interval is less than the time after which records for completed activity will be deleted.

4.6. Loop Detection and Prevention

Given three CDNs, A, B and C. If CDNs B and C delegate delivery of CDN A's content to each other, CDN A's triggers could be passed between CDNs B and C in a loop. More complex networks of CDNs could contain similar loops involving more hops.

In order to prevent and detect such CI/T loops, each CDN uses a CDN Provider ID to uniquely identify itself. Each CDN MUST insert its CDN Provider ID into the cdn-path key of every CI/T Command it originates or cascades. When receiving CI/T commands a dCDN MUST check the cdn-path and reject any CI/T Command which already contains its own CDN Provider ID in the cdn-path. Transit CDNs MUST check the cdn-path and not cascade the CI/T Command to dCDNs that are already listed in cdn-path.

The CDN Provider Id consists of 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 "AS64496:0".

If the RI interface described in [I-D.ietf-cdni-redirection] is implemented by the dCDN, the CI/T and RI interfaces SHOULD use the same CDN Provider Id.

4.7. Error Handling

A dCDN can reject a CI/T Command using HTTP status codes. For example, 400 if the request is malformed, or 401 if the uCDN does not have permission to create triggers or it is trying to act on another CDN's data.

If any part of the trigger fails, the trigger SHOULD 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 can be present while the trigger is still "pending" or "active", if the trigger is still running for some URLs or Patterns in the Trigger Specification.

Once a request has been accepted, processing errors are reported in the Trigger Status Resource using a list of Error Descriptions. Each Error Description is used to report errors against one or more of the URLs or Patterns in the Trigger Specification.

If a surrogate affected by a trigger is offline in the dCDN, or the dCDN is unable to pass a CI/T Command on to any of its cascaded dCDNs:

- o If the request is abandoned by the dCDN, the dCDN SHOULD report an error.
- o An "invalidate" trigger may be reported as "complete" when surrogates that may have the data are offline. In this case, surrogates MUST NOT use the affected data without first revalidating it when they are back online.
- o "preposition" and "purge" triggers can be reported as "processed" if affected caches are offline and the activity will complete when they return to service.
- o Otherwise, the dCDN SHOULD keep the trigger in state "pending" or "active" until the trigger is acted upon, or the uCDN chooses to cancel it.

4.8. Content URLs

To refer to content in the dCDN, the uCDN MUST present URLs in the same form as in the metadata it supplied to the dCDN. By definition, it is always possible for the dCDN to locate content based on URLs in this form.

Therefore, if content URLs are transformed by an intermediate CDN in a cascade, that intermediate CDN MUST transform URLs in CI/T commands it passes to its dCDN.

When processing Trigger Specifications, CDNs MUST ignore the URL scheme (http or https) in comparing URLs. For example, for an invalidate or purge trigger, content MUST be invalidated or purged regardless of the protocol clients use to request it.

5. CI/T Object Properties and Encoding

CI/T Commands, Trigger Status Resources and Trigger Collections and their properties are encoded using JSON, as defined in sections Section 5.1.1, Section 5.2.1, and Section 5.1.2.

Names in JSON are case sensitive and therefore the names and literal values specified here MUST always use lower-case.

Unrecognised name/value pairs in JSON objects SHOULD NOT be treated as an error by either the uCDN or dCDN.

5.1. CI/T Objects

The top-level objects defined by the CI/T interface are described in this section. Each has an associated MIME Media Type. The encoding of values used by these objects is described in Section 5.2.

5.1.1. CI/T Commands

CI/T Commands SHOULD use a MIME Media Type of application/cdni.ci.TriggerCommand+json.

A CI/T Command is encoded as a JSON object containing the following name/value pairs.

Name: trigger

Description: A specification of the trigger type, and a set of data to act upon.

Value: A Trigger Specification, as defined in Section 5.2.1.

Mandatory: No, but exactly one of "trigger" or "cancel" MUST be present in a CI/T Command.

Name: cancel

Description: The URLs of Trigger Status Resources for triggers that the uCDN wants to cancel.

Value: A JSON array of URLs represented as JSON strings.

Mandatory: No, but exactly one of "trigger" or "cancel" MUST be present in a CI/T Command.

Name: cdn-path

Description: The CDN Provider Identifiers of CDNs that have already accepted the CI/T Command.

Value: A JSON array of JSON strings, where each string is a CDN Provider Identifier as defined in Section 4.6.

Mandatory: Yes.

5.1.2. Trigger Status Resource

Trigger Status Resources SHOULD use a MIME Media Type of application/cdni.ci.TriggerStatus+json.

A Trigger Status Resource is encoded as a JSON object containing the following name/value pairs.

Name: trigger

Description: The Trigger Specification that was used to create this Trigger Status Resource.

Value: A Trigger Specification, as defined in Section 5.2.1.

Mandatory: Yes

Name: ctime

Description: Time at which the CI/T Command was received by the dCDN. Time is determined by the dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: Yes

Name: mtime

Description: Time at which the Trigger Status Resource was last modified. Time is determined by the dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: Yes

Name: etime

Description: Estimate of the time at which the dCDN expects to complete the activity. Time is determined by the dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: No

Name: status

Description: Current status of the triggered activity.

Value: Trigger Status, as defined in Section 5.2.3.

Mandatory: Yes

Name: errors

Description: Descriptions of errors that have occurred while processing a Trigger Command.

Value: A list of Error Descriptions, as defined in Section 5.2.6.

Mandatory: No.

5.1.3. Trigger Collection

Trigger Collections SHOULD use a MIME Media Type of application/cdni.ci.TriggerCollection+json.

A Trigger Collection is encoded as a JSON object containing the following name/value pairs.

Name: triggers

Description: Links to Trigger Status Resources in the collection.

Value: A JSON array of URLs represented as JSON strings.

Mandatory: Yes

Name: staleresourcetime

Description: The length of time for which the dCDN guarantees to keep a completed Trigger Status Resource. After this time, the dCDN SHOULD delete the resource and all references to it from collections.

Value: A JSON number, integer time in seconds.

Mandatory: Yes, in the collection of all Trigger Status Resources if the 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.

Names: coll-all, coll-pending, coll-active, coll-complete, coll-failed

Description: Link to a Trigger Collection.

Value: A URL represented as a JSON string.

Mandatory: Links to filtered collections are mandatory in the collection of all Trigger Status Resources, if the dCDN implements the filtered collections. Otherwise, optional.

Name: cdn-id

Description: The CDN Provider Identifier of the dCDN.

Value: A JSON string, the dCDN's CDN Provider Identifier, as defined in Section 4.6.

Mandatory: Only in the collection of all Trigger Status Resources, if the dCDN implements the filtered collections. Optional in the filtered collections.

5.2. Properties of CI/T Objects

This section defines the values that can appear in the top level objects described in Section 5.1, and their encodings.

5.2.1. Trigger Specification

A Trigger Collection is encoded as a JSON object containing the following name/value pairs.

An unrecognised name/value pair in the Trigger Specification object contained in a CI/T Command SHOULD be preserved in the Trigger Specification of any Trigger Status Resource it creates.

Name: type

Description: This property defines the type of the trigger.

Value: Trigger Type, as defined in Section 5.2.2.

Mandatory: Yes

Name: metadata.urls

Description: The uCDN URLs of the metadata the trigger applies to.

Value: A JSON array of URLs represented as JSON strings.

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Name: content.urls

Description: URLs of content the trigger applies to, see Section 4.8.

Value: A JSON array of URLs represented as JSON strings.

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Name: content.ccid

Description: The Content Collection Identifier of content the trigger applies to. The 'ccid' is a grouping of content, as defined by [I-D.ietf-cdni-metadata].

Value: A JSON array of strings, where each string is a Content Collection Identifier.

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Name: metadata.patterns

Description: The metadata the trigger applies to.

Value: A JSON array of Pattern Match, as defined in Section 5.2.4.

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.

Name: content.patterns

Description: The content data the trigger applies to.

Value: A JSON array of Pattern Match, as defined in Section 5.2.4.

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.

5.2.2. Trigger Type

Trigger Type is used in a Trigger Specification to describe trigger action. It MUST be one of the JSON strings in the following table:

JSON String	Description
preposition	A request for the dCDN to acquire metadata or content.
invalidate	A request for the 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.
purge	A request for the dCDN to erase metadata or content. After servicing the request, the specified data MUST NOT be held on the dCDN.

5.2.3. Trigger Status

This describes the current status of a Trigger. It MUST be one of the JSON strings in the following table:

JSON String	Description
pending	The trigger has not yet been acted upon.
active	The trigger is currently being acted upon.
complete	The triggered activity completed successfully.
processed	The trigger has been accepted and no further status update will be made (can be used in cases where completion cannot be confirmed).
failed	The triggered activity could not be completed.
cancelling	The triggered activity is still in progress, but the trigger has been cancelled by the uCDN.
cancelled	The triggered activity was cancelled by the uCDN.

5.2.4. PatternMatch

A Pattern Match consists of a string pattern to match, and flags describing the type of match.

It is encoded as a JSON object with the following name/value pairs:

Name: pattern

Description: A pattern for string matching.

Value: A JSON string representing the pattern. 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 "?" MUST be escaped as "\\\"", "*" and "\\?".

Mandatory: Yes.

Name: case-sensitive

Description: Flag indicating whether or not case-sensitive matching should be used.

Value: One of the JSON values 'true' or 'false'.

Mandatory: No, default is case-insensitive match.

Name: match-query-string

Description: Flag indicating whether or not the query string should be included in the pattern match.

Value: One of the JSON values 'true' or 'false'.

Mandatory: No, default is not to include the query string in the pattern match.

Example of case-sensitive prefix match against
"http://www.example.com/trailers/":

```
{  
  "pattern": "http://www.example.com/trailers/*",  
  "case-sensitive": true  
}
```

5.2.5. Absolute Time

A JSON number, seconds since the UNIX epoch.

5.2.6. Error Description

An Error Description is used to report failure of a Trigger Command, or in the activity it triggered.

Name: error

Value: Error Code, as defined in Section 5.2.7.

Mandatory: Yes.

Names: metadata.urls, content.urls, metadata.patterns,
content.patterns

Description: Metadata and content references copied from the Trigger Specification. Only those URLs and patterns to which the error applies are included in each property, but those URLs and patterns MUST be exactly as they appear in the request, the dCDN MUST NOT generalise the URLs. (For example, if the uCDN requests prepositioning of URLs "http://content.example.com/a" and "http://content.example.com/b", the dCDN must not generalise its error report to Pattern "http://content.example.com/*".)

Value: A JSON array of JSON strings, where each string is copied from a 'content.*' or 'metadata.*' value in the corresponding Trigger Specification.

Mandatory: At least one of these name/value pairs is mandatory in each Error Description object.

Name: description

Description: A human-readable description of the error.

Value: A JSON string, the human-readable description.

Mandatory: No.

5.2.7. Error Code

This type is used by the dCDN to report failures in trigger processing.

Error Code	Description
emeta	The dCDN was unable to acquire metadata required to fulfil the request.
econtent	The dCDN was unable to acquire content (preposition triggers only).
eperm	The uCDN does not have permission to trigger the requested activity (for example, the data is owned by another CDN).
ereject	The dCDN is not willing to fulfil the request (for example, a preposition request for content at a time when the dCDN would not accept Request Routing requests from the uCDN).
ecdN	An internal error in the dCDN or one of its downstream CDNs.
ecancelled	The uCDN cancelled the request.

6. Examples

The following sections provide examples of different CI/T objects encoded as JSON.

Discovery of the triggers interface is out of scope of this document. In an implementation, all CI/T URLs are under the control of the dCDN. The uCDN MUST NOT attempt to ascribe any meaning to individual elements of the path.

In examples in this section, the URL 'http://dcdn.example.com/triggers' is used as the location of the collection of all triggers, and the CDN Provider Id of uCDN is "AS64496:1".

6.1. Creating Triggers

Examples of the uCDN triggering activity in the dCDN:

6.1.1. 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 Specification.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/cdni.ci.TriggerCommand+json
Content-Length: 347

{
  "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"
    ]
  },
  "cdn-path" : [ "AS64496:1" ]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sun, 31 Aug 2014 09:53:18 GMT
Content-Length: 472
Content-Type: application/cdni.ci.TriggerStatus+json
Location: http://dcdn.example.com/triggers/0
Server: example-server/0.1

{
  "ctime": 1409478798,
  "etime": 1409478806,
```

```
"mtime": 1409478798,
"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"
}
```

6.1.1.2. Invalidate

An example of an invalidate request, another POST to the "AllTriggers" collection. This instructs the 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/cdni.ci.TriggerCommand+json
Content-Length: 384

{
  "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
      }
    ]
  }
}
```



```
    },  
    "cdn-path" : [ "AS64496:1" ]  
  }  
}
```

RESPONSE:

```
HTTP/1.1 201 Created  
Date: Sun, 31 Aug 2014 09:53:19 GMT  
Content-Length: 551  
Content-Type: application/cdni.ci.TriggerStatus+json  
Location: http://dcdn.example.com/triggers/1  
Server: example-server/0.1  
  
{  
  "ctime": 1409478799,  
  "etime": 1409478807,  
  "mtime": 1409478799,  
  "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"  
  }  
}
```

6.2. Examining Trigger Status

Once triggers have been created, the uCDN can check their status as shown in these examples.

6.2.1. 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: 347
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "-6516741166528256414"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni.ci.TriggerCollection+json
```

```
{
  "cdn-id": "AS64496:0",
  "coll-active": "/triggers/active",
  "coll-complete": "/triggers/complete",
  "coll-failed": "/triggers/failed",
  "coll-pending": "/triggers/pending",
  "staleresourcetime": 86400,
  "triggers": [
    "http://dcdn.example.com/triggers/0",
    "http://dcdn.example.com/triggers/1"
  ]
}
```

6.2.2. Filtered Collections of Triggers

The filtered collections are also available to the uCDN. Before the 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: 153
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "5012053611544832286"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni.ci.TriggerCollection+json
```

```
{
  "staleresourcetime": 86400,
  "triggers": [
    "http://dcdn.example.com/triggers/0",
    "http://dcdn.example.com/triggers/1"
  ]
}
```

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: 56
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "2986340333785000363"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni.ci.TriggerCollection+json
```

```
{
  "staleresourcetime": 86400,
  "triggers": []
}
```

6.2.3. 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: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "-4765587034697674779"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni.ci.TriggerStatus+json
```

```
{
  "ctime": 1409478798,
  "etime": 1409478806,
  "mtime": 1409478798,
  "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: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "-7657333837290433420"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni.ci.TriggerStatus+json
```

```
{
  "ctime": 1409478799,
  "etime": 1409478807,
  "mtime": 1409478799,
  "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"
  }
}
```

6.2.4. Polling for Change

The uCDN SHOULD 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: "5012053611544832286"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "5012053611544832286"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni.ci.TriggerCollection+json
```

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-4765587034697674779"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "-4765587034697674779"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni.ci.TriggerStatus+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 might 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: 56
Expires: Sun, 31 Aug 2014 09:54:29 GMT
Server: example-server/0.1
Etag: "-4471185573414616962"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:29 GMT
Content-Type: application/cdni.ci.TriggerCollection+json

{
  "staleresourcetime": 86400,
  "triggers": []
}
```


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: 153
Expires: Sun, 31 Aug 2014 09:54:30 GMT
Server: example-server/0.1
Etag: "-1508172875796647067"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:30 GMT
Content-Type: application/cdni.ci.TriggerCollection+json
```

```
{
  "staleresourcetime": 86400,
  "triggers": [
    "http://dcdn.example.com/triggers/0",
    "http://dcdn.example.com/triggers/1"
  ]
}
```

6.2.5. Removing a Trigger

To request the dCDN to cancel a Trigger, the uCDN can delete the Trigger Resource. It can 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, 31 Aug 2014 09:53:30 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: 106
Expires: Sun, 31 Aug 2014 09:54:30 GMT
Server: example-server/0.1
Etag: "-1842390246836476263"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:30 GMT
Content-Type: application/cdni.ci.TriggerCollection+json
```

```
{
  "staleresourcetime": 86400,
  "triggers": [
    "http://dcdn.example.com/triggers/1"
  ]
}
```

6.2.6. Error Reporting

In this example the uCDN has requested prepositioning of "http://newsite.example.com/index.html", but the 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: Sun, 31 Aug 2014 09:54:38 GMT
Server: example-server/0.1
Etag: "-3893590191073700822"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:38 GMT
Content-Type: application/cdni.ci.TriggerStatus+json
```

```
{
  "ctime": 1409478810,
  "errors": [
    {
      "content.urls": [
        "http://newsite.example.com/index.html"
      ],
      "description":
        "No HostIndex entry found for newsite.example.com",
      "error": "emeta"
    }
  ],
  "etime": 1409478818,
  "mtime": 1409478814,
  "status": "active",
  "trigger": {
    "content.urls": [
      "http://newsite.example.com/index.html"
    ],
    "type": "preposition"
  }
}
```

7. IANA Considerations

7.1. Media type registrations

7.1.1. CI/T Commands

The MIME media type for CI/T Commands is application/cdni.ci.TriggerCommand+json.

Type Name: application

Subtype name: cdni.ci.TriggerCommand+json

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security Considerations: See [RFCthis], Section 8

Interoperability Considerations: Described in [RFCthis]

Published Specification: [RFCthis]

Applications that use this media type: No known applications currently use this media type.

Additional Information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File Extensions: N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information: IESG <iesg@ietf.org>

Intended Usage: COMMON

Restrictions on usage: None

Author: Rob Murray <rob.murray@alcatel-lucent.com>

Change controller: IESG <iesg@ietf.org>

Note: No "charset" parameter is defined for this registration because a charset parameter is not defined for application/json [RFC7159].

7.1.2. CI/T Trigger Status Resource

The MIME media type for CI/T Trigger Status Resources is application/cdni.ci.TriggerStatus+json.

Type Name: application

Subtype name: cdni.ci.TriggerStatus+json

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security Considerations: See [RFCthis], Section 8

Interoperability Considerations: Described in [RFCthis]

Published Specification: [RFCthis]

Applications that use this media type: No known applications currently use this media type.

Additional Information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File Extensions: N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information: IESG
<iesg@ietf.org>

Intended Usage: COMMON

Restrictions on usage: None

Author: Rob Murray <rob.murray@alcatel-lucent.com>

Change controller: IESG <iesg@ietf.org>

Note: No "charset" parameter is defined for this registration because a charset parameter is not defined for application/json [RFC7159].

7.1.3. CI/T Trigger Collection

The MIME media type for CI/T Trigger Collections is application/cdni.ci.TriggerCollection+json.

Type Name: application

Subtype name: cdni.ci.TriggerCollection+json

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security Considerations: See [RFCthis], Section 8

Interoperability Considerations: Described in [RFCthis]

Published Specification: [RFCthis]

Applications that use this media type: No known applications currently use this media type.

Additional Information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File Extensions: N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information: IESG <iesg@ietf.org>

Intended Usage: COMMON

Restrictions on usage: None

Author: Rob Murray <rob.murray@alcatel-lucent.com>

Change controller: IESG <iesg@ietf.org>

Note: No "charset" parameter is defined for this registration because a charset parameter is not defined for application/json [RFC7159].

8. Security Considerations

8.1. Authentication, Confidentiality, Integrity Protection

A CI/T dCDN server implementation MUST support TLS transport for HTTP (https) as per [RFC2818]. The use of TLS for transport of the CI/T interface allows the dCDN and the uCDN to authenticate each other (to ensure they are receiving CI/T Commands from, or reporting status to, an authenticated CDN).

In an environment where any such protection is required, TLS SHOULD be used for transport of the CI/T requests and responses, unless alternate methods are used for ensuring that only authorised clients are able to access their own data (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). Both parties of the transaction (the uCDN and the dCDN) SHOULD use mutual authentication.

A TLS implementation of CI/T MUST support the TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 cipher suite ([RFC5288]). An implementation of the CI/T Interface SHOULD prefer cipher suites which support perfect forward secrecy over cipher suites that don't.

HTTP requests that attempt to access or operate on CI/T data belonging to another CDN MUST be rejected using, for example, HTTP "403 Forbidden" or "404 Not Found".

Note that in a "diamond" configuration, where one uCDN's content can be acquired via more than one directly-connected uCDN, it may not be possible for the dCDN to determine from which uCDN it acquired content. In this case, the dCDN MUST allow each uCDN from which the content could have been acquired to act upon that content using CI/T Commands.

8.2. Denial of Service

This document does not define a specific mechanism to protect against Denial of Service (DoS) attacks on the CI/T. However, CI/T endpoints can be protected against DoS attacks through the use of TLS transport and/or via mechanisms outside the scope of the CI/T interface, such as firewalling or use of Virtual Private Networks (VPNs).

Depending on the implementation, triggered activity may consume significant processing and bandwidth in the dCDN. A malicious or

faulty uCDN could use this to generate unnecessary load in the dCDN. The dCDN should consider mechanisms to avoid overload, for example by rate-limiting acceptance or processing of CI/T Commands, or batching up its processing.

9. Acknowledgements

The authors thank Kevin Ma for his input.

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.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.
- [RFC7232] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, June 2014.

10.2. Informative References

- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-07 (work in progress), July 2014.
- [I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection Interface for CDN Interconnection", draft-ietf-cdni-redirection-03 (work in progress), August 2014.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, August 2008.

- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.
- [RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, August 2014.
- [RFC7337] Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, August 2014.

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: November 20, 2016

R. Murray
B. Niven-Jenkins
Nokia
May 19, 2016

CDNI Control Interface / Triggers
draft-ietf-cdni-control-triggers-15

Abstract

This document describes the part of the CDN Interconnection 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 invalidates or purges 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 November 20, 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. Model for CDNI Triggers	4
2.1. Timing of Triggered Activity	6
2.2. Scope of Triggered Activity	6
2.2.1. Multiple Interconnected CDNs	7
2.3. Trigger Results	8
3. Collections of Trigger Status Resources	8
4. CDNI Trigger Interface	9
4.1. Creating Triggers	10
4.2. Checking Status	11
4.2.1. Polling Trigger Status Resource collections	12
4.2.2. Polling Trigger Status Resources	12
4.3. Cancelling Triggers	12
4.4. Deleting Triggers	13
4.5. Expiry of Trigger Status Resources	14
4.6. Loop Detection and Prevention	14
4.7. Error Handling	15
4.8. Content URLs	16
5. CI/T Object Properties and Encoding	16
5.1. CI/T Objects	16
5.1.1. CI/T Commands	16
5.1.2. Trigger Status Resource	17
5.1.3. Trigger Collection	19
5.2. Properties of CI/T Objects	20
5.2.1. Trigger Specification	20
5.2.2. Trigger Type	21
5.2.3. Trigger Status	22
5.2.4. PatternMatch	23
5.2.5. Absolute Time	24
5.2.6. Error Description	24
5.2.7. Error Code	25
6. Examples	26
6.1. Creating Triggers	26
6.1.1. Preposition	26
6.1.2. Invalidate	28

6.2.	Examining Trigger Status	29
6.2.1.	Collection of All Triggers	29
6.2.2.	Filtered Collections of Trigger Status Resources	30
6.2.3.	Individual Trigger Status Resources	32
6.2.4.	Polling for Change	34
6.2.5.	Deleting Trigger Status Resources	37
6.2.6.	Error Reporting	38
7.	IANA Considerations	39
7.1.	CDNI Payload Type Parameter Registrations	40
7.2.	CDNI CI/T Trigger Types Registry	40
7.3.	CDNI CI/T Error Codes Registry	40
8.	Security Considerations	41
8.1.	Authentication, Authorization, Confidentiality, Integrity Protection	41
8.2.	Denial of Service	42
8.3.	Privacy	43
9.	Acknowledgements	43
10.	References	43
10.1.	Normative References	43
10.2.	Informative References	44
Appendix A.	Formalization of the JSON Data	45
Authors' Addresses	46

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).

[RFC7336] expands on the information provided in [RFC6707] and describes each of the interfaces and the relationships between them in more detail.

This document describes the "CI/T" interface, "CDNI Control interface / Triggers". It does not consider those parts of the control interface that relate to configuration, bootstrapping or authentication of CDN Interconnect interfaces. Section 4 of [RFC7337] identifies the requirements specific to the CI interface, requirements applicable to the CI/T interface are CI-1 to CI-6.

- o Section 2 outlines the model for the CI/T Interface at a high level.
- o Section 3 describes collections of Trigger Status Resources.
- o Section 4 defines the web service provided by the dCDN.

- o Section 5 lists properties of CI/T Commands and Status Resources.
- o Section 6 contains example messages.

1.1. Terminology

This document reuses the terminology defined in [RFC6707] and uses "uCDN" and "dCDN" as shorthand for "Upstream CDN" and "Downstream CDN", respectively.

2. Model for CDNI Triggers

A CI/T Command, sent from the uCDN to the dCDN, is a request for the dCDN to do some work relating to data associated with content requests originating from the uCDN.

There are two types of CI/T Command: CI/T Trigger Commands, and CI/T Cancel Commands. The CI/T Cancel Command can be used to request cancellation of an earlier CI/T Trigger Command. A CI/T Trigger Command is of one of the following types:

- o preposition - used to instruct the dCDN to fetch metadata from the uCDN, or content from any origin including the uCDN.
- o invalidate - used to instruct the dCDN to revalidate specific metadata or content before re-using it.
- o purge - used to instruct the dCDN to delete specific metadata or content.

The CI/T interface is a web service offered by the dCDN. It allows CI/T commands to be issued, and triggered activity to be tracked. The CI/T interface builds on top of HTTP/1.1 [RFC7230]. References to URL in this document relate to http/https URIs, as defined in [RFC7230] section 2.7.

When the dCDN accepts a CI/T Command it creates a resource describing status of the triggered activity, a Trigger Status Resource. The uCDN can poll Trigger Status Resources to monitor progress.

The dCDN maintains at least one collection of Trigger Status Resources for each uCDN. Each uCDN only has access to its own collections, the locations of which are shared when CDN interconnection is established.

To trigger activity in the dCDN, the uCDN POSTs a CI/T Command to the collection of Trigger Status Resources. If the dCDN accepts the CI/T Command, it creates a new Trigger Status Resource and returns its

location to the uCDN. To monitor progress, the uCDN can GET the Trigger Status Resource. To request cancellation of a CI/T Trigger Command the uCDN can POST to the collection of Trigger Status Resources, or simply DELETE the Trigger Status Resource.

In addition to the collection of all Trigger Status Resources for the uCDN, the dCDN can maintain filtered views of that collection. These filtered views are defined in Section 3 and include collections of Trigger Status Resources corresponding to active and completed CI/T Trigger Commands. 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 the dCDN, and for the uCDN to discover the status of that activity. Only successful triggering is shown. Examples of the messages are given in Section 6.

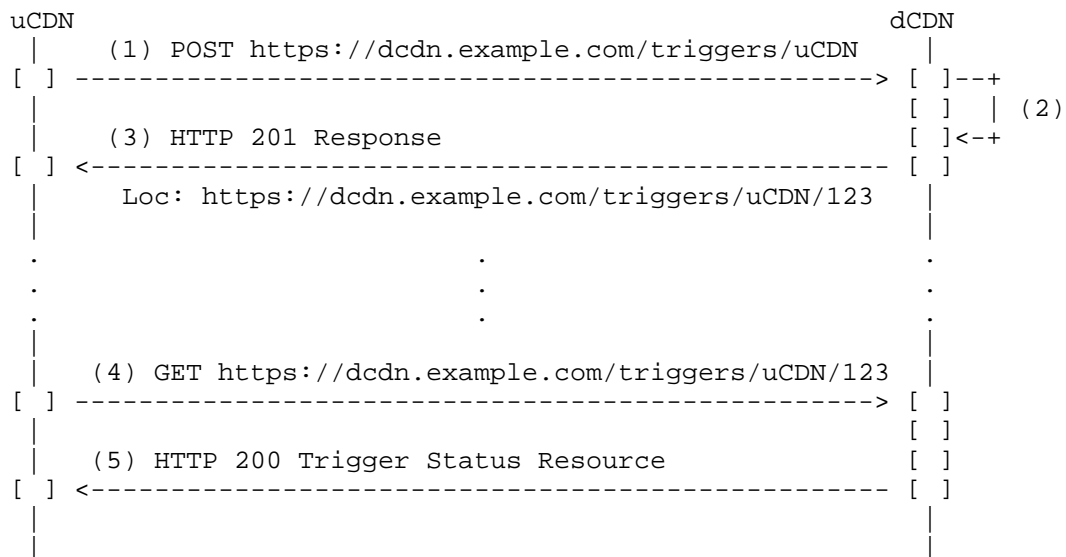


Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are:

1. The uCDN triggers action in the dCDN by posting a CI/T Command to a collection of Trigger Status Resources, "https://dcdn.example.com/triggers/uCDN". The URL of this was given to the uCDN when the CI/T interface was established.

2. The dCDN authenticates the request, validates the CI/T Command and, if it accepts the request, creates a new Trigger Status Resource.
3. The dCDN responds to the uCDN with an HTTP 201 response status, and the location of the Trigger Status Resource.
4. The uCDN can poll, possibly repeatedly, the Trigger Status Resource in the dCDN.
5. The dCDN responds with the Trigger Status Resource, describing progress or results of the CI/T Trigger Command.

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 the execution of CI/T Commands is under the dCDN's control, including its start-time and pacing of the activity in the network.

CI/T invalidate and purge commands MUST be applied to all data acquired before the command was accepted by the dCDN. The dCDN SHOULD NOT apply CI/T invalidate and purge commands to data acquired after the CI/T Command was accepted, but this may not always be achievable so the uCDN cannot count on that.

If the uCDN wishes to invalidate or purge content then immediately pre-position replacement content at the same URLs, it SHOULD ensure the dCDN has completed the invalidate/purge before initiating the prepositioning. Otherwise, there is a risk that the dCDN pre-positions the new content, then immediately invalidates or purges it (as a result of the two uCDN requests running in parallel).

Because the CI/T Command timing is under the dCDN's control, the dCDN implementation can choose whether to apply CI/T invalidate and purge commands to content acquisition that has already started when the command is received.

2.2. Scope of Triggered Activity

Each CI/T Command can operate on multiple metadata and content URLs.

Multiple representations of an HTTP resource may share the same URL. CI/T Trigger Commands that invalidate or purge metadata or content apply to all resource representations with matching URLs.

2.2.1. Multiple Interconnected CDNs

In a network of interconnected CDNs a single uCDN will originate a given item of metadata and associated content, it may distribute that metadata and content to more than one dCDN, which may in-turn distribute that metadata and content to further-downstream CDNs.

An intermediate CDN is a dCDN that passes on CDNI metadata and content to further-downstream dCDNs.

A diamond configuration is one where a dCDN can acquire metadata and content originated in one uCDN from that uCDN itself and an intermediate CDN, or via more than one intermediate CDN.

CI/T commands originating in the single source uCDN affect metadata and content in all dCDNs but, in a diamond configuration, it may not be possible for the dCDN to determine which uCDN it acquired content from. In this case a dCDN MUST allow each uCDN from which it may have acquired the content to act upon that content using CI/T Commands.

In all other cases, a dCDN MUST reject CI/T Commands from a uCDN that act on another uCDN's data using, for example, HTTP "403 Forbidden".

Security considerations are discussed further in Section 8.

The diamond configuration may lead to inefficient interactions, but the interactions are otherwise harmless. For example:

- o When the uCDN issues an invalidate CI/T command, a dCDN will receive that command from multiple directly connected uCDNs. The dCDN may schedule multiple those commands separately, and the last may affect content already revalidated following execution of the invalidate command scheduled first.
- o If one of a dCDN's directly-connected uCDNs loses its rights to distribute content, it may issue a CI/T purge command. That purge may affect content the dCDN could retain because it's distributed by another directly-connected uCDN. But, that content can be re-acquired by the dCDN from the remaining uCDN.
- o When the uCDN originating an item of content issues a CI/T purge followed by a preposition - two directly connected uCDNs will pass those commands to a dCDN. That dCDN implementation need not merge those operations, or notice the repetition. In which case the purge issued by one uCDN will complete before the other. The first uCDN to finish its purge may then forward the preposition trigger, and content pre-positioned as a result might be affected

by the still-running purge issued by the other uCDN. However, the dCDN will re-acquire that content as needed, or when it's asked to pre-position the content by the second uCDN. A dCDN implementation could avoid this interaction by knowing which uCDN it acquired the content from, or it could minimize the consequences by recording the time at which the invalidate/purge command was received and not applying it to content acquired after that time.

2.3. Trigger Results

Possible states for a Trigger Status Resource are defined in section Section 5.2.3.

The CI/T Trigger Command MUST NOT be reported as 'complete' until all actions have been completed successfully. The reasons for failure, and URLs or Patterns affected, SHOULD be enumerated in the Trigger Status Resource. For more detail, see section Section 4.7.

If a dCDN is also acting as a uCDN in a cascade, it MUST forward CI/T Commands to any downstream CDNs that may be affected. The CI/T Trigger Command MUST NOT be reported as 'complete' in a CDN until it is 'complete' in all of its downstream CDNs. If a CI/T Trigger Command is reported as 'processed' in any dCDN, intermediate CDNs MUST NOT report 'complete', instead they MUST also report 'processed'. A CI/T Command MAY be reported as 'failed' as soon as it fails in a CDN or in any of its downstream CDNs. A cancelled CI/T Trigger Command MUST be reported as 'cancelling' until it has been reported as 'cancelled', 'complete', or 'failed' by all dCDNs in a cascade.

3. Collections of Trigger Status Resources

As described in Section 2, Trigger Status Resources exist in the 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.

The dCDN MUST make a collection of a uCDN's Trigger Status Resources available to that uCDN. This collection includes all of the Trigger Status Resources created for CI/T Commands from the uCDN that have been accepted by the dCDN, and have not yet been deleted by the uCDN, or expired and removed by the dCDN (as described in section Section 4.4). Trigger Status Resources belonging to a uCDN MUST NOT be visible to any other CDN. The dCDN could, for example, achieve this by offering different collection URLs to each uCDN, and by

filtering the response based on the uCDN with which the HTTP client is associated.

To trigger activity in a dCDN, or to cancel triggered activity, the uCDN POSTs a CI/T Command to the dCDN's collection of the uCDN's Trigger Status Resources.

In order to allow the uCDN to check the status of multiple jobs in a single request, the dCDN MAY also maintain collections representing filtered views of the collection of all Trigger Status Resources. These filtered collections are optional-to-implement but, if implemented, the dCDN MUST include links to them in the collection of all Trigger Status Resources. The filtered collections are:

- o Pending - Trigger Status Resources for CI/T Trigger Commands that have been accepted, but not yet acted upon.
- o Active - Trigger Status Resources for CI/T Trigger Commands that are currently being processed in the dCDN.
- o Complete - Trigger Status Resources representing activity that completed successfully, and 'processed' CI/T Trigger Commands for which no further status updates will be made by the dCDN.
- o Failed - Trigger Status Resources representing CI/T Commands that failed or were cancelled by the uCDN.

4. CDNI Trigger Interface

This section describes an interface to enable an upstream CDN to trigger activity in a downstream CDN.

The CI/T interface builds on top of HTTP, so dCDNs may make use of any HTTP feature when implementing the CI/T interface. For example, a dCDN SHOULD make use of HTTP's caching mechanisms to indicate that a requested response/representation has not been modified, reducing the uCDN's processing needed to determine whether the status of triggered activity has changed.

All dCDNs implementing CI/T MUST support the HTTP GET, HEAD, POST and DELETE methods as defined in [RFC7231].

The only representation specified in this document is JSON, [RFC7159]. It MUST be supported by the uCDN and by the dCDN.

The URL of the dCDN's collection of all Trigger Status Resources needs to be either discovered by, or configured in, the uCDN. The

mechanism for discovery of that URL is outside the scope of this document.

CI/T Commands are POSTed to the dCDN's collection of all Trigger Status Resources. If a CI/T Trigger Command is accepted by the dCDN, the dCDN creates a new Trigger Status Resource and returns its URI to the uCDN in an HTTP 201 response. The triggered activity can then be monitored by the uCDN using that resource and the collections described in Section 3.

The URI of each Trigger Status Resource is returned to the uCDN when it is created, and URIs of all Trigger Status Resources are listed in the dCDN's collection of all Trigger Status Resources. This means all Trigger Status Resources can be discovered by the uCDN, so dCDNs are free to assign whatever structure they desire to the URIs for CI/T resources. Therefore uCDNs MUST NOT make any assumptions regarding the structure of CI/T URIs or the mapping between CI/T objects and their associated URIs. URIs present in the examples in this document are purely illustrative and are not intended to impose a definitive structure on CI/T interface implementations.

4.1. Creating Triggers

To issue a CI/T Command, the uCDN makes an HTTP POST to the dCDN's collection of all of the uCDN's Trigger Status Resources. The request body of that POST is a CI/T Command, as described in Section 5.1.1.

The dCDN validates the CI/T Command. If the command is malformed or the uCDN does not have sufficient access rights, the dCDN MUST either respond with an appropriate 4xx HTTP error code and not create a Trigger Status Resource, or create a 'failed' Trigger Status Resource containing an appropriate error description.

When a CI/T Trigger Command is accepted, the uCDN MUST create a new Trigger Status Resource which will convey a specification of the CI/T Command and its current status. The HTTP response to the dCDN MUST have status code 201 and MUST convey the URI of the Trigger Status Resource in the Location header field. The HTTP response SHOULD include the content of the newly created Trigger Status Resource. This is particularly important in cases where the CI/T Trigger Command has completed immediately.

Once a Trigger Status Resource has been created the dCDN MUST NOT re-use its URI, even after that Trigger Status Resource has been removed.

The dCDN SHOULD track and report on progress of CI/T Trigger Commands using a Trigger Status Resource, Section 5.1.2. If the dCDN is not able to do that, it MUST indicate that it has accepted the request but will not be providing further status updates. To do this, it sets the status of the Trigger Status Resource to "processed". In this case, CI/T processing should continue as for a "complete" request, so the Trigger Status Resource MUST be added to the dCDN's collection of Complete Trigger Status Resources. The dCDN SHOULD also provide an estimated completion time for the request, by using the "etime" property of the Trigger Status Resource. This will allow the uCDN to schedule prepositioning after an earlier delete of the same URLs is expected to have finished.

If the dCDN is able to track the execution of CI/T Commands and a CI/T Command is queued by the dCDN for later action, the status property of the Trigger Status Resource MUST be "pending". Once processing has started the "status" MUST be "active". Finally, once the CI/T Command is complete, the status MUST be set to "complete" or "failed".

A CI/T Trigger Command may result in no activity in the dCDN if, for example, it is an invalidate or purge request for data the dCDN has not yet acquired, or a pre-position request for data it has already acquired and which is still valid. In this case, the "status" of the Trigger Status Resource MUST be "processed" or "complete", and the Trigger Status Resource MUST be added to the dCDN's collection of Complete Trigger Status Resources.

Once created, Trigger Status Resources can be cancelled or deleted by the uCDN, but not modified. The dCDN MUST reject PUT and POST requests from the uCDN to Trigger Status Resources by responding with an appropriate HTTP status code, for example 405 "Method Not Allowed".

4.2. Checking Status

The uCDN has two ways to check progress of CI/T Commands it has issued to the dCDN, described in sections Section 4.2.1 and Section 4.2.2.

To allow the uCDN to check for change in status of a Trigger Status Resource or collection of Trigger Status Resources without re-fetching the whole Resource or Collection, the dCDN SHOULD include Entity Tags for the uCDN to use as cache validators, as defined in [RFC7232].

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 the uCDN should poll for change.

4.2.1. Polling Trigger Status Resource collections

The 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 CI/T Trigger Commands in a single request. If the dCDN moves a Trigger Status Resource from the Active to the Completed collection, the uCDN can fetch the result of that activity.

When polling in this way, the uCDN SHOULD use HTTP Entity Tags to monitor for change, rather than repeatedly fetching the whole collection. An example of this is given in section Section 6.2.4.

4.2.2. Polling Trigger Status Resources

The uCDN has a URI provided by the dCDN for each Trigger Status Resource it has created, it may fetch that Trigger Status Resource at any time.

This can be used to retrieve progress information, and to fetch the result of the CI/T Command.

When polling in this way, the uCDN SHOULD use HTTP Entity Tags to monitor for change, rather than repeatedly fetching the Trigger Status Resource.

4.3. Cancelling Triggers

The uCDN can request cancellation of a CI/T Trigger Command by POSTing a CI/T Cancel Command to the collection of all Trigger Status Resources.

The dCDN is required to accept and respond to the CI/T Cancel Command, but the actual cancellation of a CI/T Trigger Command is optional-to-implement.

The dCDN MUST respond to the CI/T Cancel Command appropriately, for example with HTTP status code 200 "OK" if the cancellation has been processed and the CI/T Command is inactive, 202 "Accepted" if the command has been accepted but the CI/T Command remains active, or 501 "Not Implemented" if cancellation is not supported by the dCDN.

If cancellation of a "pending" Trigger Status Resource is accepted by the dCDN, the dCDN SHOULD NOT start processing of that activity. Issuing a CT/T Cancel Command for a "pending" Trigger Status Resource does not however guarantee that the corresponding activity will not be started, because the uCDN cannot control the timing of that activity. Processing could, for example, start after the POST is sent by the uCDN but before that request is processed by the dCDN.

If cancellation of an "active" or "processed" Trigger Status Resource is accepted by the dCDN, the dCDN SHOULD stop processing the CI/T Command. However, as with cancellation of a "pending" CI/T Command, the dCDN does not guarantee this.

If the CI/T Command cannot be stopped immediately, the status in the corresponding Trigger Status Resource MUST be set to "cancelling", and the Trigger Status Resource MUST remain in the collection of Trigger Status Resources for active CI/T Commands. If processing is stopped before normal completion, the status value in the Trigger Status Resource MUST be set to "cancelled", and the Trigger Status Resource MUST be included in the collection of failed CT/T Trigger Commands.

Cancellation of a "complete" or "failed" Trigger Status Resource requires no processing in the dCDN. Its status MUST NOT be changed to "cancelled".

4.4. Deleting Triggers

The uCDN can delete Trigger Status Resources at any time, using the HTTP DELETE method. The effect is similar to cancellation, but no Trigger Status Resource remains afterwards.

Once deleted, the references to a Trigger Status Resource MUST be removed from all Trigger Status Resource collections. Subsequent requests to GET the deleted Trigger Status Resource SHOULD be rejected by the dCDN with an HTTP error.

If a "pending" Trigger Status Resource is deleted, the dCDN SHOULD NOT start processing of that activity. Deleting a "pending" Trigger Status Resource does not however guarantee that it has not started because the uCDN cannot control the timing of that activity. Processing may, for example, start after the DELETE is sent by the uCDN but before that request is processed by the dCDN.

If an "active" or "processed" Trigger Status Resource is deleted, the dCDN SHOULD stop processing the CI/T Command. However, as with deletion of a "pending" Trigger Status Resource, the dCDN does not guarantee this.

Deletion of a "complete" or "failed" Trigger Status Resource requires no processing in the dCDN other than deletion of the Trigger Status Resource.

4.5. Expiry of Trigger Status Resources

The dCDN can choose to automatically delete Trigger Status Resources some time after they become "complete", "processed", "failed" or "cancelled". In this case, the dCDN will remove the Trigger Status Resource and respond to subsequent requests for it with an HTTP error.

If the dCDN does remove Trigger Status Resources automatically, it MUST report the length of time after which it will do so, using a property of the collection of all Trigger Status Resources. It is RECOMMENDED that Trigger Status Resources are not automatically deleted by the dCDN for at least 24 hours after they become "complete", "processed", "failed" or "cancelled".

To ensure it is able to get the status of its Trigger Status Resources for completed and failed CI/T Commands, it is RECOMMENDED that the uCDN polling interval is less than the time after which records for completed activity will be deleted.

4.6. Loop Detection and Prevention

Given three CDNs, A, B and C, if CDNs B and C delegate delivery of CDN A's content to each other, CDN A's CI/T Commands could be passed between CDNs B and C in a loop. More complex networks of CDNs could contain similar loops involving more hops.

In order to prevent and detect such CI/T loops, each CDN uses a CDN Provider ID to uniquely identify itself. In every CI/T Command it originates or cascades, each CDN MUST append an array element containing its CDN Provider ID to a JSON array under an entry named "cdn-path". When receiving CI/T Commands a dCDN MUST check the cdn-path and reject any CI/T Command which already contains its own CDN Provider ID in the cdn-path. Transit CDNs MUST check the cdn-path and not cascade the CI/T Command to dCDNs that are already listed in cdn-path.

The CDN Provider Id consists of the two characters "AS" followed by the CDN Provider's Autonomous System number [RFC1930], 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 "AS64496:0".

If the CDN provider has multiple Autonomous Systems, the same AS number SHOULD be used in all messages from that CDN provider, unless there are multiple distinct CDNs.

If the RI interface described in [I-D.ietf-cdni-redirection] is implemented by the dCDN, the CI/T and RI interfaces SHOULD use the same CDN Provider Id.

4.7. Error Handling

A dCDN can signal rejection of a CI/T Command using HTTP status codes. For example, 400 if the request is malformed, or 403 or 404 if the uCDN does not have permission to issue CI/T Commands or it is trying to act on another CDN's data.

If any part of the CI/T Trigger Command fails, the trigger SHOULD 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 can be present while the Trigger Status Resource is still "pending" or "active", if the CI/T Trigger Command is still running for some URLs or Patterns in the Trigger Specification.

Once a request has been accepted, processing errors are reported in the Trigger Status Resource using a list of Error Descriptions. Each Error Description is used to report errors against one or more of the URLs or Patterns in the Trigger Specification.

If a surrogate affected by a CI/T Trigger Command is offline in the dCDN, or the dCDN is unable to pass a CI/T Command on to any of its cascaded dCDNs:

- o If the CI/T Command is abandoned by the dCDN, the dCDN SHOULD report an error.
- o A CI/T "invalidate" command may be reported as "complete" when surrogates that may have the data are offline. In this case, surrogates MUST NOT use the affected data without first revalidating it when they are back online.
- o CI/T "preposition" and "purge" commands can be reported as "processed" if affected caches are offline and the activity will complete when they return to service.
- o Otherwise, the dCDN SHOULD keep the Trigger Status Resource in state "pending" or "active" until the CI/T Command is acted upon, or the uCDN chooses to cancel it.

4.8. Content URLs

If content URLs are transformed by an intermediate CDN in a cascade, that intermediate CDN MUST similarly transform URLs in CI/T Commands it passes to its dCDN.

When processing Trigger Specifications, CDNs MUST ignore the URL scheme (http or https) in comparing URLs. For example, for a CI/T invalidate or purge command, content MUST be invalidated or purged regardless of the protocol clients use to request it.

5. CI/T Object Properties and Encoding

The CI/T Commands, Trigger Status Resources and Trigger Collections and their properties are encoded using JSON, as defined in sections Section 5.1.1, Section 5.2.1, and Section 5.1.2. They MUST use the MIME Media Type 'application/cdni', with parameter 'ptype' values as defined below and in Section 7.1.

Names in JSON are case sensitive. The names and literal values specified in the present document MUST always use lower-case.

JSON types, including 'object', 'array', 'number' and 'string' are defined in [RFC7159].

Unrecognised name/value pairs in JSON objects SHOULD NOT be treated as an error by either the uCDN or dCDN. They SHOULD be ignored in the processing, and passed on by dCDN to any further dCDNs in a cascade.

5.1. CI/T Objects

The top-level objects defined by the CI/T interface are described in this section.

The encoding of values used by these objects is described in Section 5.2.

5.1.1. CI/T Commands

CI/T Commands MUST use a MIME Media Type of 'application/cdni; ptype=ci-trigger-command'.

A CI/T Command is encoded as a JSON object containing the following name/value pairs.

Name: trigger

Description: A specification of the trigger type, and a set of data to act upon.

Value: A Trigger Specification, as defined in Section 5.2.1.

Mandatory: No, but exactly one of "trigger" or "cancel" MUST be present in a CI/T Command.

Name: cancel

Description: The URLs of Trigger Status Resources for CI/T Trigger Commands that the uCDN wants to cancel.

Value: A non-empty JSON array of URLs represented as JSON strings.

Mandatory: No, but exactly one of "trigger" or "cancel" MUST be present in a CI/T Command.

Name: cdn-path

Description: The CDN Provider Identifiers of CDNs that have already issued the CI/T Command to their dCDNs.

Value: A non-empty JSON array of JSON strings, where each string is a CDN Provider Identifier as defined in Section 4.6.

Mandatory: Yes.

5.1.1.2. Trigger Status Resource

Trigger Status Resources MUST use a MIME Media Type of 'application/cdni; ptype=ci-trigger-status'.

A Trigger Status Resource is encoded as a JSON object containing the following name/value pairs.

Name: trigger

Description: The Trigger Specification posted in the body of the CI/T Command. 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.

Value: A Trigger Specification, as defined in Section 5.2.1.

Mandatory: Yes

Name: ctime

Description: Time at which the CI/T Command was received by the dCDN. Time is determined by the dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: Yes

Name: mtime

Description: Time at which the Trigger Status Resource was last modified. Time is determined by the dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: Yes

Name: etime

Description: Estimate of the time at which the dCDN expects to complete the activity. Time is determined by the dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: No

Name: status

Description: Current status of the triggered activity.

Value: Trigger Status, as defined in Section 5.2.3.

Mandatory: Yes

Name: errors

Description: Descriptions of errors that have occurred while processing a Trigger Command.

Value: An array of Error Description, as defined in Section 5.2.6. An empty array is allowed, and equivalent to omitting "errors" from the object.

Mandatory: No.

5.1.3. Trigger Collection

Trigger Collections MUST use a MIME Media Type of 'application/cdni; ptype=ci-trigger-collection'.

A Trigger Collection is encoded as a JSON object containing the following name/value pairs.

Name: triggers

Description: Links to Trigger Status Resources in the collection.

Value: A JSON array of zero or more URLs, represented as JSON strings.

Mandatory: Yes

Name: staleresourcetime

Description: The length of time for which the dCDN guarantees to keep a completed Trigger Status Resource. After this time, the dCDN SHOULD delete the Trigger Status Resource and all references to it from collections.

Value: A JSON number, which must be a positive integer, representing time in seconds.

Mandatory: Yes, in the collection of all Trigger Status Resources if the 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.

Names: coll-all, coll-pending, coll-active, coll-complete, coll-failed

Description: Link to a Trigger Collection.

Value: A URL represented as a JSON string.

Mandatory: Links to all of the filtered collections are mandatory in the collection of all Trigger Status Resources, if the dCDN implements the filtered collections. Otherwise, optional.

Name: cdn-id

Description: The CDN Provider Identifier of the dCDN.

Value: A JSON string, the dCDN's CDN Provider Identifier, as defined in Section 4.6.

Mandatory: Only in the collection of all Trigger Status Resources, if the dCDN implements the filtered collections. Optional in the filtered collections (the uCDN can always find the dCDN's cdn-id in the collection of all Trigger Status Resources, but the dCDN can choose to repeat that information in its implementation of filtered collections).

5.2. Properties of CI/T Objects

This section defines the values that can appear in the top level objects described in Section 5.1, and their encodings.

5.2.1. Trigger Specification

A Trigger Collection is encoded as a JSON object containing the following name/value pairs.

An unrecognised name/value pair in the Trigger Specification object contained in a CI/T Command SHOULD be preserved in the Trigger Specification of any Trigger Status Resource it creates.

Name: type

Description: This property defines the type of the CI/T Trigger Command.

Value: Trigger Type, as defined in Section 5.2.2.

Mandatory: Yes

Name: metadata.urls

Description: The uCDN URLs of the metadata the CI/T Trigger Command applies to.

Value: A JSON array of URLs represented as JSON strings.

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Name: content.urls

Description: URLs of content the CI/T Trigger Command applies to, see Section 4.8.

Value: A JSON array of URLs represented as JSON strings.

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Name: content.ccid

Description: The Content Collection Identifier of content the trigger applies to. The 'ccid' is a grouping of content, as defined by [I-D.ietf-cdni-metadata].

Value: A JSON array of strings, where each string is a Content Collection Identifier.

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Name: metadata.patterns

Description: The metadata the trigger applies to.

Value: A JSON array of Pattern Match, as defined in Section 5.2.4.

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.

Name: content.patterns

Description: The content data the trigger applies to.

Value: A JSON array of Pattern Match, as defined in Section 5.2.4.

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.

5.2.2. Trigger Type

Trigger Type is used in a Trigger Specification to describe trigger action.

All trigger types MUST be registered in the IANA CI/T Trigger Types registry (see Section 7.2).

A dCDN receiving a request containing a trigger type it does not recognise or does not support MUST reject the request by creating a Trigger Status Resource with status to "failed" and the "errors" array containing an Error Description with error "eunsupported".

The following trigger types are defined by this document:

JSON String	Description
preposition	A request for the dCDN to acquire metadata or content.
invalidate	A request for the 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.
purge	A request for the dCDN to erase metadata or content. After servicing the request, the specified data MUST NOT be held on the dCDN (the dCDN should re-acquire the metadata or content from uCDN if it needs it).

5.2.3. Trigger Status

This describes the current status of a Trigger. It MUST be one of the JSON strings in the following table:

JSON String	Description
pending	The CI/T Trigger Command has not yet been acted upon.
active	The CI/T Trigger Command is currently being acted upon.
complete processed	The CI/T Trigger Command completed successfully. The CI/T Trigger Command has been accepted and no further status update will be made (can be used in cases where completion cannot be confirmed).
failed	The CI/T Trigger Command could not be completed.
cancelling	Processing of the CI/T Trigger Command is still in progress, but the CI/T Trigger Command has been cancelled by the uCDN.
cancelled	The CI/T Trigger Command was cancelled by the uCDN.

5.2.4. PatternMatch

A Pattern Match consists of a string pattern to match against a URI, and flags describing the type of match.

It is encoded as a JSON object with the following name/value pairs:

Name: pattern

Description: A pattern for URI matching.

Value: A JSON string representing the pattern. 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 "?" MUST be escaped as "\\ ", "*" and "\\?".

Mandatory: Yes.

Name: case-sensitive

Description: Flag indicating whether or not case-sensitive matching should be used.

Value: One of the JSON values 'true' (the matching is case-sensitive) or 'false' (the matching is case-insensitive).

Mandatory: No, default is case-insensitive match.

Name: match-query-string

Description: Flag indicating whether or not the query part of the URI should be included in the pattern match.

Value: One of the JSON values 'true' (the full URI including the query part should be compared against the given pattern), or 'false' (the query part of the URI should be dropped before comparison with the given pattern).

Mandatory: No, default is 'false', the query part of the URI should be dropped before comparison with the given pattern.

Example of case-sensitive prefix match against
"https://www.example.com/trailers/":

```
{  
  "pattern": "https://www.example.com/trailers/*",  
  "case-sensitive": true  
}
```

5.2.5. Absolute Time

A JSON number, seconds since the UNIX epoch, 00:00:00 UTC on 1 January 1970.

5.2.6. Error Description

An Error Description is used to report failure of a CI/T Command, or in the activity it triggered. It is encoded as a JSON object with the following name/value pairs:

Name: error

Value: Error Code, as defined in Section 5.2.7.

Mandatory: Yes.

Names: metadata.urls, content.urls, metadata.patterns,
content.patterns

Description: Metadata and content references copied from the Trigger Specification. Only those URLs and patterns to which the error applies are included in each property, but those URLs and patterns MUST be exactly as they appear in the request, the dCDN MUST NOT generalise the URLs. (For example, if the uCDN requests prepositioning of URLs "https://content.example.com/a" and "https://content.example.com/b", the dCDN must not

generalise its error report to Pattern
"https://content.example.com/*".)

Value: A JSON array of JSON strings, where each string is copied from a 'content.*' or 'metadata.*' value in the corresponding Trigger Specification.

Mandatory: At least one of these name/value pairs is mandatory in each Error Description object.

Name: description

Description: A human-readable description of the error.

Value: A JSON string, the human-readable description.

Mandatory: No.

5.2.7. Error Code

This type is used by the dCDN to report failures in trigger processing. All error codes MUST be registered in the IANA CI/T Error Codes registry (see Section 7.3). Unknown error codes MUST be treated as fatal errors, and the request MUST NOT be automatically retried without modification.

The following error codes are defined by this document, and MUST be supported by an implementation of the CI/T interface.

Error Code	Description
emeta	The dCDN was unable to acquire metadata required to fulfil the request.
econtent	The dCDN was unable to acquire content (CT/T preposition commands only).
eperm	The uCDN does not have permission to issue the CI/T Command (for example, the data is owned by another CDN).
ereject	The dCDN is not willing to fulfil the CI/T Command (for example, a preposition request for content at a time when the dCDN would not accept Request Routing requests from the uCDN).
ecdn	An internal error in the dCDN or one of its downstream CDNs.
ecancelled	The uCDN cancelled the request.
eunsupported	The Trigger Specification contained a "type" that is not supported by the dCDN. No action was taken by the dCDN other than to create a Trigger Status Resource in state "failed".

6. Examples

The following sections provide examples of different CI/T objects encoded as JSON.

Discovery of the triggers interface is out of scope of this document. In an implementation, all CI/T URLs are under the control of the dCDN. The uCDN MUST NOT attempt to ascribe any meaning to individual elements of the path.

In examples in this section, the URL 'https://dcdn.example.com/triggers' is used as the location of the collection of all Trigger Status Resources, and the CDN Provider Id of uCDN is "AS64496:1".

6.1. Creating Triggers

Examples of the uCDN triggering activity in the dCDN:

6.1.1. Preposition

An example of a CI/T preposition command, a POST to the collection of all Trigger Status Resources.

Note that "metadata.patterns" and "content.patterns" are not allowed in a preposition Trigger Specification.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger-command
Content-Length: 352

{
  "trigger" : {
    "type": "preposition",

    "metadata.urls" : [ "https://metadata.example.com/a/b/c" ],
    "content.urls" : [
      "https://www.example.com/a/b/c/1",
      "https://www.example.com/a/b/c/2",
      "https://www.example.com/a/b/c/3",
      "https://www.example.com/a/b/c/4"
    ]
  },
  "cdn-path" : [ "AS64496:1" ]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Wed, 04 May 2016 08:48:10 GMT
Content-Length: 467
Content-Type: application/cdni; ptype=ci-trigger-status
Location: https://dcdn.example.com/triggers/0
Server: example-server/0.1

{
  "ctime": 1462351690,
  "etime": 1462351698,
  "mtime": 1462351690,
  "status": "pending",
  "trigger": {
    "content.urls": [
      "https://www.example.com/a/b/c/1",
      "https://www.example.com/a/b/c/2",
      "https://www.example.com/a/b/c/3",
      "https://www.example.com/a/b/c/4"
    ],
    "metadata.urls": [
      "https://metadata.example.com/a/b/c"
    ],
  },
}
```

```
        "type": "preposition"
    }
}
```

6.1.2. Invalidate

An example of a CI/T invalidate command, another POST to the collection of all Trigger Status Resources. This instructs the dCDN to re-validate the content at "https://www.example.com/a/index.html", as well as any metadata and content whose URLs are prefixed by "https://metadata.example.com/a/b/" using case-insensitive matching, and "https://www.example.com/a/b/" respectively, using case-sensitive matching.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger-command
Content-Length: 387

{
  "trigger" : {
    "type": "invalidate",

    "metadata.patterns" : [
      { "pattern" : "https://metadata.example.com/a/b/*" }
    ],

    "content.urls" : [ "https://www.example.com/a/index.html" ],
    "content.patterns" : [
      { "pattern" : "https://www.example.com/a/b/*",
        "case-sensitive" : true
      }
    ]
  },
  "cdn-path" : [ "AS64496:1" ]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Length: 545
Content-Type: application/cdni; ptype=ci-trigger-status
Location: https://dcdn.example.com/triggers/1
```

Server: example-server/0.1

```
{
  "ctime": 1462351691,
  "etime": 1462351699,
  "mtime": 1462351691,
  "status": "pending",
  "trigger": {
    "content.patterns": [
      {
        "case-sensitive": true,
        "pattern": "https://www.example.com/a/b/*"
      }
    ],
    "content.urls": [
      "https://www.example.com/a/index.html"
    ],
    "metadata.patterns": [
      {
        "pattern": "https://metadata.example.com/a/b/*"
      }
    ],
    "type": "invalidate"
  }
}
```

6.2. Examining Trigger Status

Once Trigger Status Resources have been created, the uCDN can check their status as shown in these examples.

6.2.1. Collection of All Triggers

The uCDN can fetch the collection of all Trigger Status Resources it has created that 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: 341
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "-936094426920308378"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```

```
{
  "cdn-id": "AS64496:0",
  "coll-active": "/triggers/active",
  "coll-complete": "/triggers/complete",
  "coll-failed": "/triggers/failed",
  "coll-pending": "/triggers/pending",
  "staleresourcetime": 86400,
  "triggers": [
    "https://dcdn.example.com/triggers/0",
    "https://dcdn.example.com/triggers/1"
  ]
}
```

6.2.2. Filtered Collections of Trigger Status Resources

The filtered collections are also available to the uCDN. Before the dCDN starts processing the two CI/T Trigger Commands 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: 152
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "4331492443626270781"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```

```
{
  "staleresourcetime": 86400,
  "triggers": [
    "https://dcdn.example.com/triggers/0",
    "https://dcdn.example.com/triggers/1"
  ]
}
```

At this point, if no other Trigger Status Resources had been created, the other filtered views 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: 54
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "7958041393922269003"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

{
  "staleresourcetime": 86400,
  "triggers": []
}
```

6.2.3. Individual Trigger Status Resources

The Trigger Status Resources can also be examined for detail about individual CI/T Trigger Commands. For example, for the CI/T "preposition" and "invalidate" commands 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: 467
Expires: Wed, 04 May 2016 08:49:10 GMT
Server: example-server/0.1
ETag: "6990548174277557683"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:10 GMT
Content-Type: application/cdni; ptype=ci-trigger-status
```

```
{
  "ctime": 1462351690,
  "etime": 1462351698,
  "mtime": 1462351690,
  "status": "pending",
  "trigger": {
    "content.urls": [
      "https://www.example.com/a/b/c/1",
      "https://www.example.com/a/b/c/2",
      "https://www.example.com/a/b/c/3",
      "https://www.example.com/a/b/c/4"
    ],
    "metadata.urls": [
      "https://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: 545
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "-554385204989405469"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-status
```

```
{
  "ctime": 1462351691,
  "etime": 1462351699,
  "mtime": 1462351691,
  "status": "pending",
  "trigger": {
    "content.patterns": [
      {
        "case-sensitive": true,
        "pattern": "https://www.example.com/a/b/*"
      }
    ],
    "content.urls": [
      "https://www.example.com/a/index.html"
    ],
    "metadata.patterns": [
      {
        "pattern": "https://metadata.example.com/a/b/*"
      }
    ],
    "type": "invalidate"
  }
}
```

6.2.4. Polling for Change

The uCDN SHOULD use the Entity Tags of collections or Trigger Status 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: "4331492443626270781"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "4331492443626270781"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "6990548174277557683"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Wed, 04 May 2016 08:49:10 GMT
Server: example-server/0.1
ETag: "6990548174277557683"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:10 GMT
Content-Type: application/cdni; ptype=ci-trigger-status
```

When the CI/T Trigger Command is complete, the contents of the filtered collections will be updated along with their Entity Tags. For example, when the two example CI/T Trigger Commands are complete, the collections of pending and complete Trigger Status Resources might 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: 54
Expires: Wed, 04 May 2016 08:49:15 GMT
Server: example-server/0.1
ETag: "1337503181677633762"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:15 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

{
  "staleresourcetime": 86400,
  "triggers": []
}
```

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: 152
Expires: Wed, 04 May 2016 08:49:22 GMT
Server: example-server/0.1
ETag: "4481489539378529796"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:22 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```

```
{
  "staleresourcetime": 86400,
  "triggers": [
    "https://dcdn.example.com/triggers/0",
    "https://dcdn.example.com/triggers/1"
  ]
}
```

6.2.5. Deleting Trigger Status Resources

The uCDN can delete completed and failed Trigger Status Resources to reduce the size of the collections, as described in Section 4.4. For example, to delete 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: Wed, 04 May 2016 08:48:22 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 Trigger Status Resources 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: 105
Expires: Wed, 04 May 2016 08:49:22 GMT
Server: example-server/0.1
ETag: "-6938620031669085677"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:22 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```

```
{
  "staleresourcetime": 86400,
  "triggers": [
    "https://dcdn.example.com/triggers/1"
  ]
}
```

6.2.6. Error Reporting

In this example the uCDN has requested prepositioning of "https://newsite.example.com/index.html", but the 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: 486
Expires: Wed, 04 May 2016 08:49:26 GMT
Server: example-server/0.1
ETag: "5182824839919043757"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:26 GMT
Content-Type: application/cdni; ptype=ci-trigger-status
```

```
{
  "ctime": 1462351702,
  "errors": [
    {
      "content.urls": [
        "https://newsite.example.com/index.html"
      ],
      "description": "newsite.example.com not in HostIndex",
      "error": "emeta"
    }
  ],
  "etime": 1462351710,
  "mtime": 1462351706,
  "status": "active",
  "trigger": {
    "content.urls": [
      "https://newsite.example.com/index.html"
    ],
    "type": "preposition"
  }
}
```

7. IANA Considerations

[RFC Editor Note: Please replace references to [RFCthis] in this section with this document's RFC number before publication.]

7.1. CDNI Payload Type Parameter Registrations

The IANA is requested to register the following new Payload Types in the CDNI Payload Type Parameter registry defined by [RFC7736], for use with the 'application/cdni' MIME media type.

Payload Type	Specification
ci-trigger-command	[RFCthis]
ci-trigger-status	[RFCthis]
ci-trigger-collection	[RFCthis]

7.2. CDNI CI/T Trigger Types Registry

The IANA is requested to create a new "CDNI CI/T Trigger Types" subregistry under the "Content Delivery Networks Interconnection (CDNI) Parameters" registry.

Additions to the CDNI CI/T Error Code Registry will be made via "RFC Required" as defined in [RFC5226].

The initial contents of the CDNI CI/T Trigger Types Registry comprise the names and descriptions listed in section Section 5.2.2 of this document, with this document acting as the specification.

7.3. CDNI CI/T Error Codes Registry

The IANA is requested to create a new "CDNI CI/T Error Codes" subregistry under the "Content Delivery Networks Interconnection (CDNI) Parameters" registry.

Additions to the CDNI CI/T Error Codes Registry will be made via "Specification Required" as defined in [RFC5226]. The Designated Expert will verify that new error code registrations do not duplicate existing error code definitions (in name or functionality), prevent gratuitous additions to the namespace, and prevent any additions to the namespace that would impair the interoperability of CDNI implementations.

The initial contents of the CDNI CI/T Error Codes Registry comprise the names and descriptions of the Error Codes listed in Section 5.2.7 of this document, with this document acting as the specification.

8. Security Considerations

The CI/T interface provides a mechanism to allow a uCDN to generate requests into the dCDN and to inspect its own CI/T requests and their current state. The CI/T interface does not allow access to or modification of the uCDN or dCDN metadata relating to content delivery, or to the content itself. It can only control the presence of that metadata in the dCDN, and the processing work and network utilisation involved in ensuring that presence.

By examining pre-positioning requests to a dCDN, and correctly interpreting content and metadata URLs, an attacker could learn the uCDN or content owner's predictions for future content popularity. By examining invalidate or purge requests, an attacker could learn about changes in the content owner's catalogue.

By injecting CI/T commands an attacker, or a misbehaving uCDN, would generate work in the dCDN and uCDN as they process those requests. And so would a man in the middle attacker modifying valid CI/T commands generated by the uCDN. In both cases, that would decrease the dCDN caching efficiency by causing it to unnecessarily acquire or re-acquire content metadata and/or content.

A dCDN implementation of CI/T MUST restrict the actions of a uCDN to the data corresponding to that uCDN. Failure to do so would allow uCDNs to detrimentally affect each other's efficiency by generating unnecessary acquisition or re-acquisition load.

An origin that chooses to delegate its delivery to a CDN is trusting that CDN to deliver content on its behalf, CDN-interconnection is an extension of that trust to downstream CDNs. That trust relationship is a commercial arrangement, outside the scope of the CDNi protocols. So, while a malicious CDN could deliberately generate load on a dCDN using the CI/T, the protocol does not otherwise attempt to address malicious behaviour between interconnected CDNs.

8.1. Authentication, Authorization, Confidentiality, Integrity Protection

A CI/T implementation MUST support TLS transport for HTTP (https) as per [RFC2818] and [RFC7230].

TLS MUST be used by the server-side (dCDN) and the client-side (uCDN) of the CI/T interface, including authentication of the remote end, unless alternate methods are used for ensuring the security of the information in the CI/T 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 CI/T interface allows:

- o The dCDN and the uCDN to authenticate each other using TLS client auth and TLS server auth.

And, once they have mutually authenticated each other, it allows:

- o The dCDN and the uCDN to authorize each other (to ensure they are receiving CI/T Commands from, or reporting status to, an authorized CDN).
- o CDNI commands and responses to be transmitted with confidentiality.
- o Protection of the integrity of CDNI commands and responses.

When TLS is used, the general TLS usage guidance in [RFC7525] MUST be followed.

The mechanisms for access control are dCDN-specific, not standardised as part of this CI/T specification.

HTTP requests that attempt to access or operate on CI/T data belonging to another CDN MUST be rejected using, for example, HTTP "403 Forbidden" or "404 Not Found". This is intended to prevent unauthorised users from generating unnecessary load in dCDN or uCDN due to revalidation, reacquisition, or unnecessary acquisition.

When deploying a network of interconnected CDNs, the possible inefficiencies related to the "diamond" configuration discussed in Section 2.2.1 should be considered.

8.2. Denial of Service

This document does not define a specific mechanism to protect against Denial of Service (DoS) attacks on the CI/T. However, CI/T endpoints can be protected against DoS attacks through the use of TLS transport and/or via mechanisms outside the scope of the CI/T interface, such as firewalling or use of Virtual Private Networks (VPNs).

Depending on the implementation, triggered activity may consume significant processing and bandwidth in the dCDN. A malicious or faulty uCDN could use this to generate unnecessary load in the dCDN. The dCDN should consider mechanisms to avoid overload, for example by

rate-limiting acceptance or processing of CI/T Commands, or batching up its processing.

8.3. Privacy

The CI/T protocol does not carry any information about individual End Users of a CDN, there are no privacy concerns for End Users.

The CI/T protocol does carry information which could be considered commercially sensitive by CDN operators and content owners. The use of mutually authenticated TLS to establish a secure session for the transport of CI/T data, as discussed in Section 8.1, provides confidentiality while the CI/T data is in transit, and prevents parties other than the authorised dCDN from gaining access to that data. The dCDN MUST ensure that it only exposes CI/T data related to a uCDN to clients it has authenticated as belonging to that uCDN.

9. Acknowledgements

The authors thank Kevin Ma for his input, and Carsten Bormann for his review and formalization of the JSON data.

10. References

10.1. Normative References

- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma,
"CDN Interconnection Metadata", draft-ietf-cdni-
metadata-16 (work in progress), April 2016.
- [RFC1930] Hawkinson, J. and T. Bates, "Guidelines for creation,
selection, and registration of an Autonomous System (AS)",
BCP 6, RFC 1930, DOI 10.17487/RFC1930, March 1996,
<<http://www.rfc-editor.org/info/rfc1930>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [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>>.

- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.
- [RFC7232] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, June 2014.
- [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, May 2015.

10.2. Informative References

- [I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-08 (work in progress), March 2016.
- [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.
- [RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, August 2014.
- [RFC7337] Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, August 2014.
- [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. Formalization of the JSON Data

This appendix is non-normative.

The JSON data described in this document has been formalised using CDDL [I-D.greevenbosch-appsawg-cbor-cddl] as follows:

CIT-object = CIT-command / Trigger-Status-Resource / Trigger-Collection

CIT-command ; use media type application/cdni; ptype=ci-trigger-command
= {
 ? trigger: Triggerspec
 ? cancel: [* URI]
 cdn-path: [* Cdn-PID]
}

Trigger-Status-Resource ; application/cdni; ptype=ci-trigger-status
= {
 trigger: Triggerspec
 ctime: Absolute-Time
 mtime: Absolute-Time
 ? etime: Absolute-Time
 status: Trigger-Status
 ? errors: [* Error-Description]
}

Trigger-Collection ; application/cdni; ptype=ci-trigger-collection
= {
 triggers: [* URI]
 ? staleresourcetime: int ; time in seconds
 ? coll-all: URI
 ? coll-pending: URI
 ? coll-active: URI
 ? coll-complete: URI
 ? coll-failed: URI
 ? cdn-id: Cdn-PID
}

Triggerspec = { ; 5.2.1
 type: Trigger-Type
 ? metadata.urls: [* URI]
 ? content.urls: [* URI]
 ? content.ccid: [* Ccid]
 ? metadata.patterns: [* Pattern-Match]
 ? content.patterns: [* Pattern-Match]
}

Trigger-Type = "preposition" / "invalidate" / "purge" ; 5.2.2

Trigger-Status = "pending" / "active" / "complete" / "processed"
/ "failed" / "cancelling" / "cancelled" ; 5.2.3

Pattern-Match = { ; 5.2.4
 pattern: tstr
 ? case-sensitive: bool
 ? match-query-string: bool
}

Absolute-Time = number ; seconds since UNIX epoch, 5.2.5

Error-Description = { ; 5.2.6
 error: Error-Code
 ? metadata.urls: [* URI]
 ? content.urls: [* URI]
 ? metadata.patterns: [* Pattern-Match]
 ? content.patterns: [* Pattern-Match]
 ? description: tstr
}

Error-Code = "emeta" / "econtent" / "eperm" / "ereject"
/ "ecdn" / "ecancelled" ; 5.2.7

Ccid = tstr ; see I-D.ietf-cdni-metadata

Cdn-PID = tstr .regexp "AS[0-9]+:[0-9]+"

URI = tstr

Authors' Addresses

Rob Murray
Nokia
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: rob.murray@nokia.com

Ben Niven-Jenkins
Nokia
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: ben.niven-jenkins@nokia.com

CDNI
Internet-Draft
Intended status: Informational
Expires: April 30, 2015

J. Seedorf
NEC
J. Peterson
Neustar
S. Previdi
Cisco
R. van Brandenburg
TNO
K. Ma
Ericsson
October 27, 2014

CDNI Request Routing: Footprint and Capabilities Semantics
draft-ietf-cdni-footprint-capabilities-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.

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 30, 2015.

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 and Scope	3
2. Design Decisions for Footprint and Capabilities	4
2.1. Advertising Limited Coverage	4
2.2. Capabilities and Dynamic Data	5
2.3. Advertisement versus Queries	6
2.4. Avoiding or Handling 'cheating' dCDNs	7
2.5. Focusing on Main Use Cases	7
3. Main Use Case to Consider	8
4. Semantics for Footprint Advertisement	8
5. Semantics for Capabilities Advertisement	11
6. Negotiation of Support for Optional Types of Footprint/Capabilities	14
7. Capability Advertisement Object	14
7.1. Base Advertisement Object	14
7.2. Redirection Mode Advertisement Object	15
7.3. Advertisement Object Serialization	15
8. IANA Considerations	16
8.1. Capabilities Registry	16
8.2. Footprint Sub-Registry	16
8.3. Protocol Sub-Registry	17
8.4. Redirection Mode Sub-Registry	17
8.5. Logging Record Type Sub-Registry	17
8.6. Logging Field Name Sub-Registry	17
8.7. Metadata Type Sub-Registry	17
9. Security Considerations	18
10. References	18
10.1. Normative References	18
10.2. Informative References	19
Appendix A. Acknowledgment	19
Authors' Addresses	20

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.

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 RFC 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 range of different footprint definitions and possible capabilities is very broad. Attempting to define a comprehensive advertisement solution quickly becomes intractable. The CDNI requirements draft [I-D.ietf-cdni-requirements] lists the specific requirements for the CDNI Footprint & Capabilities Advertisement Interface in order to disambiguate footprints and capabilities with respect to CDNI. This document attempts to distill the apparent common understanding of what the terms 'footprint' and 'capabilities' mean in the context of CDNI, and detail the semantics of the footprint advertisement mechanism and the capability advertisement mechanism.

2. 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.

2.1. 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.

2.2. 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 of 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.

2.3. 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?

2.4. 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.

2.5. Focusing on Main Use Cases

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.

3. Main Use Case to Consider

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?

4. 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) to which the contractual agreement applies. 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 "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. The following such types of footprint are mandatory and must be supported by the CDNI FCI:

- o List of ISO Country Codes
- o List of AS numbers
- o 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.

"Resource" types of footprints are more specific than "coverage/reachability" types of footprints, where the actual coverage/reachability are extrapolated from the resource location (e.g., netmask applied to resource IP address to derive IP-prefix). The specific methods for extrapolating coverage/reachability from resource location are beyond the scope of this document. In the degenerate case, the resource address could be specified as a coverage/reachability type of footprint, in which case no extrapolation is necessary. Resource types of footprints may expose the internal structure of a CDN network which may be undesirable. As such, the resource types of footprints are not considered mandatory to support for CDNI.

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.

In addition to these mandatory "coverage/reachability" types of footprint, other optional "coverage/reachability" types of footprint or "resource" types of footprint may be defined by future specifications. To facilitate this, a clear process for specifying optional footprint types in a IANA registry must be specified. This includes the specification of the level of oversight necessary (e.g., WG decision or expert review) for adding new optional footprints to a IANA registry as well as the specification of a template regarding

design choices that must be captured by new optional types of footprints.

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 data set containing the complete footprint. Making incremental updates, however, to express dynamic changes in state is also desirable.

5. 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, 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 2.5, 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 CDNI Logging (i.e., supported logging fields)
- o CDNI Metadata (i.e., supported GenericMetadata types)

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. Instead, the CDNI FCI specification should define a generic protocol for conveying any capability information. In this respect, it seems reasonable to define a registry which initially contains the mandatory capabilities listed above, but may be extended as needs dictate. This document defines the registry (and the rules for adding new entries to the registry) for the different capability types (see Section 8). Each capability type MAY 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 "CDNI Logging Fields Names" registry defines all supported logging fields, including mandatory-to-implement logging fields. Advertisement of support for mandatory-to-implement logging fields SHOULD be supported but would be redundant. CDNs SHOULD NOT advertise support for mandatory-to-implement logging fields. The following logging fields are defined as optional in the CDNI Logging Interface document [I-D.ietf-cdni-logging]:

- o c-ip-anonimizing
- o s-ccid
- o s-sid

The "CDNI GenericMetadata Types" registry defines all supported GenericMetadata types, including mandatory-to-implement GenericMetadata types. Advertisement of support for mandatory-to-implement GenericMetadata types SHOULD be supported but would be redundant. CDNs SHOULD NOT advertise support for mandatory-to-implement GenericMetadata types. The CDNI Metadata Interface document [I-D.ietf-cdni-metadata] does not define any optional GenericMetadata types.

6. Negotiation of Support for Optional Types of Footprint/Capabilities

The notion of optional types of footprint and capabilities implies that certain implementations may not support all kinds of footprint and capabilities. Therefore, any FCI solution protocol must define how the support for optional types of footprint/capabilities will be negotiated between a uCDN and a dCDN that use the particular FCI protocol. In particular, any FCI solution protocol needs to specify how to handle failure cases or non-supported types of footprint/capabilities.

In general, a uCDN may ignore capabilities or types of footprint it does not understand; in this case it only selects a suitable downstream CDN based on the types of capabilities and footprint it understands. Similarly, if a dCDN does not use an optional capability or footprint which is, however, supported by a uCDN, this causes no problem for the FCI functionality because the uCDN decides on the remaining capabilities/footprint information that is being conveyed by the dCDN.

7. Capability Advertisement Object

To support extensibility, the FCI defines a generic base object (similar to the CDNI Metadata interface GenericMetadata object) [I-D.ietf-cdni-metadata] to facilitate a uniform set of mandatory parsing requirements for all future FCI objects.

[Ed. Note: MI/LI object definitions will build off the base object defined here, but will be specified in a separate draft.]

7.1. Base Advertisement Object

The FCIBase object is an abstraction for managing individual CDNI capabilities (see Section 8.1) in an opaque manner.

Property: capability-type

Description: CDNI Capability object type.

Type: MIME Type String (from Section 8)

Mandatory-to-Specify: Yes.

Property: capability-value

Description: CDNI Capability object.

Type: Format/Type is defined by the value of capability-type property above.

Mandatory-to-Specify: Yes.

7.2. Redirection Mode Advertisement Object

The Redirection Mode advertisement object is used to indicate support for one or more of the modes listed in the CDNI Capabilities Redirection Modes registry (see Section 8.4).

Property: redirection-modes

Description: List of supported CDNI Redirection Modes.

Type: List of Redirection Modes (from Section 8.4)

Mandatory-to-Specify: Yes.

[Ed. Note: Redirection Mode is not in MI/LI; it is FCI specific. We define it here as an example for the MI/LI FCI drafts.]

7.3. Advertisement Object Serialization

```
{
  "capabilities": [
    {
      "capability-type": "application/cdni.FCI.RedirectionMode.v1"
      "capability-value": {
        "redirection-modes": [
          "DNS-I",
          "HTTP-I"
        ]
      }
    },
    {
      "capability-type": "application/cdni.FCI.LI.s-ccid.v1"
      "capability-value": {
        "s-ccid-support": true
      }
    }
  ]
}
```

[Ed. Note: Need to add JSON serialization discussion.]

8. IANA Considerations

This document requests the registration of the following MIME Media Type under the IANA MIME Media Type registry (<http://www.iana.org/assignments/media-types/index.html>).

application/cdni.FCI.RedirectionMode.v1

8.1. Capabilities Registry

IANA registries are to be used for mandatory and optional types of footprint and capabilities. Therefore, the mandatory types of capabilities listed in this document (see Section 5) are to be registered with IANA. In order to prevent namespace collisions for capabilities a new IANA registry is requested for the "CDNI Capabilities" namespace. The namespace shall be split into two partitions: standard and optional.

The "standard" namespace partition is intended to contain mandatory to implement capabilities and conforms to the "IETF Review" policy as defined in [RFC5226]. The registry contains the name of the standard capability, the RFC number of the specification defining the capability (including the capability advertisement object format and serialization).

The following table defines the initial capabilities for the standard partition:

Capability	RFC
application/cdni.FCI.RedirectionMode.v1	RFCthis

The "optional" namespace partition conforms to the "Expert Review" policy as defined in [RFC5226]. The expert review is intended to prevent namespace hoarding and to prevent the definition of redundant capabilities. Vendors defining new capabilities which conflict with existing capabilities follow the guidelines for the "Specification Required" policy as defined in [RFC5226].

8.2. Footprint Sub-Registry

The "CDNI Metadata Footprint Types" namespace defined in the CDNI Metadata Interface document [I-D.ietf-cdni-metadata] contains the supported footprint formats for use in footprint advertisement. No further IANA action is required here.

8.3. Protocol Sub-Registry

The "CDNI Metadata Protocols" namespace defined in the CDNI Metadata Interface document [I-D.ietf-cdni-metadata] contains the supported protocol values for the Delivery Protocol and Acquisition Protocol capabilities. No further IANA action is required here.

8.4. Redirection Mode Sub-Registry

The "CDNI Capabilities Redirection Modes" namespace defines the valid redirection modes that may be advertised as supported by a CDN. Additions to the Redirection Mode namespace conform to the "IETF Review" policy as defined in [RFC5226].

The following table defines the initial Redirection Modes:

Redirection Mode	Description	RFC
DNS-I	Iterative DNS-based Redirection	RFCthis
DNS-R	Recursive DNS-based Redirection	RFCthis
HTTP-I	Iterative HTTP-based Redirection	RFCthis
HTTP-R	Recursive HTTP-based Redirection	RFCthis

8.5. Logging Record Type Sub-Registry

The "CDNI Logging Record-Types" namespace defined in the CDNI Logging Interface document [I-D.ietf-cdni-logging] contains the types of all supported logging record-types. No further IANA action is required here.

8.6. Logging Field Name Sub-Registry

The "CDNI Logging Fields Names" namespace defined in the CDNI Logging Interface document [I-D.ietf-cdni-logging] contains the names of all supported logging fields. No further IANA action is required here.

8.7. Metadata Type Sub-Registry

The "CDNI GenericMetadata Types" namespace defined in the CDNI Metadata Interface document [I-D.ietf-cdni-metadata] contains the names of the supported GenericMetadata objects. No further IANA action is required here.

9. Security Considerations

This specification describes the semantics for capabilities and footprint advertisement objects in content distribution networks. It does not, however, specify a concrete protocol for transporting those objects, or even a specific object syntax. Specific security mechanisms can only be selected for concrete protocols that instantiate these semantics. This document does, however, place some high-level security constraints on such protocols.

All protocols that implement these semantics are REQUIRED to provide integrity and authentication services. Without authentication and integrity, an attacker could trivially deny service by forging a footprint advertisement from a dCDN which claims the network has no footprint or capability. This would prevent the uCDN from delegating any requests to the dCDN. Since a pre-existing relationship between all dCDNs and uCDNs is assumed by CDNi, the exchange of any necessary credentials could be conducted before the FCI interface is brought online. The authorization decision to accept advertisements would also follow this pre-existing relationship and any contractual obligations that it stipulates.

It is not believed that there are any serious privacy risks in sharing footprint or capability information: it will represent highly aggregated data about networks and at best policy-related information about media, rather than any personally identifying information. However, particular dCDNs may wish to share information about their footprint with a uCDN but not with other, competing dCDNs. For example, if a dCDN incurs an outage that reduces footprint coverage temporarily, that may be information the dCDN would want to share confidentially with the uCDN. Protocols implementing these semantics SHOULD provide confidentiality services.

As specified in this document, the security requirements of the FCI could be met by hop-by-hop transport-layer security mechanisms coupled with domain certificates as credentials. There is no apparent need for further object-level security in this framework, as the trust relationships it defines are bilateral relationships between uCDNs and dCDNs rather than transitive relationships.

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.

- [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.
- [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.

10.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L., Davie, B., and R. Brandenburg, "Framework for CDN Interconnection", draft-ietf-cdni-framework-14 (work in progress), June 2014.
- [I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-14 (work in progress), October 2014.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-07 (work in progress), July 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.

Appendix A. Acknowledgment

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.

Martin Stiemerling provided initial input to this document and valuable comments to the ongoing discussions among the authors of this document. Thanks to Francois Le Faucheur and Scott Wainner for providing valuable comments and suggestions to the text.

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

Email: jon.peterson@neustar.biz

Stefano Previdi
Cisco Systems
Via Del Serafico 200
Rome 0144
Italy

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
Ericsson
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.j.ma@ericsson.com

CDNI
Internet-Draft
Intended status: Standards Track
Expires: November 21, 2016

J. Seedorf
NEC
J. Peterson
Neustar
S. Previdi
Cisco
R. van Brandenburg
TNO
K. Ma
Ericsson
May 20, 2016

CDNI Request Routing: Footprint and Capabilities Semantics
draft-ietf-cdni-footprint-capabilities-semantics-20

Abstract

This document captures the semantics of the "Footprint and Capabilities Advertisement" part of the CDNI Request Routing interface, i.e., the desired meaning of "Footprint" and "Capabilities" in the CDNI context, and what the "Footprint and Capabilities Advertisement Interface (FCI)" offers within CDNI. The document also provides guidelines for the CDNI FCI protocol. It further defines a Base Advertisement Object, the necessary registries for capabilities and footprints, and guidelines on how these registries can be extended in the future.

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 November 21, 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 and Scope	3
1.1. Terminology	4
2. Design Decisions for Footprint and Capabilities	5
2.1. Advertising Limited Coverage	5
2.2. Capabilities and Dynamic Data	6
2.3. Advertisement versus Queries	7
2.4. Avoiding or Handling 'cheating' dCDNs	7
3. Focusing on Capabilities with Footprint Restrictions	8
4. Footprint and Capabilities Extension	8
5. Capability Advertisement Object	10
5.1. Base Advertisement Object	10
5.2. Encoding	11
5.3. Delivery Protocol Capability Object	11
5.3.1. Delivery Protocol Capability Object Serialization	12
5.4. Acquisition Protocol Capability Object	12
5.4.1. Acquisition Protocol Capability Object Serialization	13
5.5. Redirection Mode Capability Object	13
5.5.1. Redirection Mode Capability Object Serialization	13
5.6. CDNI Logging Capability Object	14
5.6.1. CDNI Logging Capability Object Serialization	15
5.7. CDNI Metadata Capability Object	15
5.7.1. CDNI Metadata Capability Object Serialization	16
6. IANA Considerations	17
6.1. CDNI Payload Types	17
6.1.1. CDNI FCI DeliveryProtocol Payload Type	17
6.1.2. CDNI FCI AcquisitionProtocol Payload Type	18
6.1.3. CDNI FCI RedirectionMode Payload Type	18
6.1.4. CDNI FCI Logging Payload Type	18
6.1.5. CDNI FCI Metadata Payload Type	18

6.2. Redirection Mode Registry	18
7. Security Considerations	19
8. References	20
8.1. Normative References	20
8.2. Informative References	20
Appendix A. Main Use Case to Consider	21
Appendix B. Semantics for Footprint Advertisement	22
Appendix C. Semantics for Capabilities Advertisement	24
Appendix D. Acknowledgment	25
Authors' Addresses	26

1. Introduction and Scope

The CDNI working group is working on a set of protocols to enable the interconnection of multiple CDNs. This CDN interconnection (CDNI) can 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 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 (dCDN) '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 needs to offer and accomplish in the context of CDNI.

It is explicitly outside the scope of this document to decide on specific protocols to use for the FCI. However, guidelines for such FCI protocols are provided.

General assumptions in this document:

- o The CDNs participating in the interconnected CDN 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 upstream CDN (uCDN) receives footprint and/or capability advertisements from a set of dCDNs. Footprint advertisement and capability advertisement need not use the same underlying protocol.
- o The uCDN receives the initial request-routing request from the endpoint requesting the resource.

The CDNI Problem Statement [RFC6707] describes the Request Routing Interface as: "[enabling] a Request Routing function in a uCDN to query a Request Routing function in a dCDN to determine if the dCDN is able (and willing) to accept the delegated Content Request". In addition, RFC6707 says "the CDNI Request Routing interface is also expected to enable a dCDN to provide to the uCDN (static or dynamic) information (e.g., resources, footprint, load) to facilitate selection of the dCDN by the uCDN request routing system when processing subsequent content requests from User Agents". It thus considers "resources" and "load" as capabilities to be advertised by the dCDN.

The range of different footprint definitions and possible capabilities is very broad. Attempting to define a comprehensive advertisement solution quickly becomes intractable. The CDNI requirements draft [RFC7337] lists the specific requirements for the CDNI Footprint & Capabilities Advertisement Interface in order to disambiguate footprints and capabilities with respect to CDNI. This document defines a common understanding of what the terms 'footprint' and 'capabilities' mean in the context of CDNI, and details the semantics of the footprint advertisement mechanism and the capability advertisement mechanism.

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 Footprint: a description of a CDN's coverage area, i.e., the area from which client requests may originate for, and to which the CDN is willing to deliver, content. Note: There are many ways to describe a footprint, for example, by address range (e.g., IPv4/IPv6 CIDR), by network ID (e.g., ASN), by nation boundaries (e.g., country code), by GPS coordinates, etc. This document does not define or endorse the quality or suitability of any particular footprint description method; this document only defines a method for transporting known footprint descriptions in Footprint and Capabilities Advertisement messages.
- o Capability: a feature of a dCDN, upon which a uCDN relies on the dCDN supporting, when making delegation decisions. Support for a given feature can change over time and can be restricted to a limited portion of a dCDN's footprint. Note: There are many possible dCDN features that could be of interest to a uCDN. This document does not presume to define them all; this document describes a scheme for defining new capabilities and how to

transport them in Footprint and Capabilities Advertisement messages.

2. 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 surrogates, a surrogate with a public IPv4 address is reachable by any endpoint on the Internet unless some policy enforcement precludes the use of the surrogate.

Some CDNs aspire to cover the entire world; we refer to these as 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 interconnect with other CDNs in order to create a single CDN fabric which shares resources.

Furthermore, not all capabilities need to 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 understanding 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.

2.1. 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 surrogates 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 surrogates were positioned to serve.

When such a purpose-built CDN interconnects with other CDNs and advertises its footprint to a uCDN, however, the original intended coverage of the CDN might not represent its actual value to the interconnection of CDNs. Consider an ISP-A and ISP-B that both field their own CDNs, which they interconnect via CDNI. A given user E,

who is a customer of ISP-B, might happen to be topologically closer to a surrogate 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, is it ISP-A's CDN that "covers" E? If ISP-B's CDN has a failure condition, is it up to the uCDN to understand that ISP-A's surrogates are potentially available as 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, mixing global and limited CDNs)?

The answers to these questions greatly depend on how much information the uCDN wants 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. Alternately, dCDNs could advertise the IP addresses of their surrogates 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) are included as part of the CDNI FCI, or if the focus is just on 'binary' footprint.

2.2. Capabilities and Dynamic Data

In cases where the apparent footprints of dCDNs overlap, uCDNs might also want to rely on other factors to evaluate the respective merits of dCDNs. These include facts related to the surrogates themselves, to the network where the surrogate 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 surrogates, the choice to limit coverage is necessarily an administrative policy. Much policy needs to be agreed upon before CDNs can interconnect, 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 interconnection.

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 surrogates, for example, changes relatively rarely, whereas the amount of storage in use at any given moment is highly volatile. Network bandwidth similarly could be expressed as either total bandwidth available to a surrogate, or based on the current state of the network. A surrogate can 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 needs to be updated.

2.3. Advertisement versus Queries

In a CDNI environment, each dCDN shares some of its state with the uCDN. The uCDN uses this information to build a unified picture of all of the dCDNs available to it. In architectures that share detailed capability information, the uCDN could perform the entire request-routing operation down to selecting a particular surrogate in the dCDN. However, when the uCDN needs to deal with many potential dCDNs, this approach does not scale, especially for dCDNs with thousands or tens of thousands of surrogates; the volume of updates to footprint and capability becomes onerous.

Were the volume of FCI updates from dCDNs to exceed the volume 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 [RFC7336]), instead of receiving advertisements and tracking the state of dCDNs. 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?

2.4. 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 over '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 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, the information a dCDN advertises (in the long run) needs to 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 might suffice for the uCDN to "de-incentivize" cheating because it would negatively affect the long-term business relationship with a particular dCDN.

3. Focusing on Capabilities with Footprint Restrictions

Given the design considerations listed in the previous section, it seems reasonable to assume that in most 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 can 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., surrogate proximity, delivery latency, surrogate 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.

Further understanding that in most cases contractual agreements will define the basic coverage used in delegation decisions, the primary focus of FCI is on providing updates to the basic capabilities and coverage by the dCDNs. As such, FCI has chosen the semantics of "capabilities with footprint restrictions".

4. Footprint and Capabilities Extension

Other optional "coverage/reachability" types of footprint or "resource" types of footprint may be defined by future specifications. To facilitate this, a clear process for specifying optional footprint types in an IANA registry is specified in the CDNI Metadata Footprint Types registry (defined in the CDNI Metadata Interface document [I-D.ietf-cdni-metadata]).

This document also registers CDNI Payload Types [RFC7736] for the initial capability types (see Section 6):

- o Delivery Protocol (for delivering content to the end user)

- o Acquisition Protocol (for acquiring content from the uCDN or origin server)
- o Redirection Mode (e.g., DNS Redirection vs. HTTP Redirection as discussed in [RFC7336])
- o CDNI Logging (i.e., supported logging fields)
- o CDNI Metadata (i.e., supported Generic Metadata types)

Each payload type is prefaced with "FCI.". Updates to capability objects MUST indicate the version of the capability object in a newly registered payload type, e.g., by appending ".v2". Each capability type MAY have a list of valid values. Future specifications which define a given capability MUST define any necessary registries (and the rules for adding new entries to the registry) for the values advertised for a given capability type.

The "CDNI Logging record-types" registry [I-D.ietf-cdni-logging] defines all known record types, including mandatory-to-implement record-types Advertising support for mandatory-to-implement record-types would be redundant. CDNs SHOULD NOT advertise support for mandatory-to-implement record-types.

The "CDNI Logging Fields Names" registry [I-D.ietf-cdni-logging] defines all known logging fields. Logging fields may be reused by different record-types and be mandatory-to-implement in some record-types, but optional in other record-types. CDNs MUST advertise support for optional logging fields within the context of a specific record-type. CDNs SHOULD NOT advertise support for mandatory-to-implement logging fields, for a given record-type. The following logging fields are defined as optional for the "cdni_http_request_v1" record-type in the CDNI Logging Interface document [I-D.ietf-cdni-logging]:

- o s-ccid
- o s-sid

The CDNI Metadata Interface document [I-D.ietf-cdni-metadata] requires that CDNs be able to parse all the defined metadata objects, but does not require dCDNs to support enforcement of non-structural GenericMetadata objects. Advertising support for mandatory-to-enforce GenericMetadata types MUST be supported. Advertising support for non-mandatory-to-enforce GenericMetadata types SHOULD be supported. Advertisement of non-mandatory-to-enforce GenericMetadata MAY be necessary, e.g., to signal temporary outages and subsequent

recovery. It is expected that structural metadata will be supported at all times.

The notion of optional types of footprint and capabilities implies that certain implementations might not support all kinds of footprint and capabilities. Therefore, any FCI solution protocol MUST define how the support for optional types of footprint/capabilities will be negotiated between a uCDN and a dCDN that use the particular FCI protocol. In particular, any FCI solution protocol MUST specify how to handle failure cases or non-supported types of footprint/capabilities.

In general, a uCDN MAY ignore capabilities or types of footprints it does not understand; in this case it only selects a suitable dCDN based on the types of capabilities and footprint it understands. Similarly, if a dCDN does not use an optional capability or footprint which is, however, supported by a uCDN, this causes no problem for the FCI functionality because the uCDN decides on the remaining capabilities/footprint information that is being conveyed by the dCDN.

5. Capability Advertisement Object

To support extensibility, the FCI defines a generic base object (similar to the CDNI Metadata interface GenericMetadata object) [I-D.ietf-cdni-metadata] to facilitate a uniform set of mandatory parsing requirements for all future FCI objects.

Future object definitions (e.g. regarding CDNI Metadata or Logging) will build off the base object defined here, but will be specified in separate documents.

Note: In the following sections, the term "mandatory-to-specify" is used to convey which properties MUST be included when serializing a given capability object. When mandatory-to-specify is defined as "Yes" for an individual property, it means that if the object containing that property is included in an FCI message, then the mandatory-to-specify property MUST also be included.

5.1. Base Advertisement Object

The FCIBase object is an abstraction for managing individual CDNI capabilities in an opaque manner.

Property: capability-type

Description: CDNI Capability object type.

Type: FCI specific CDNI Payload type (from the CDNI Payload Types registry [RFC7736])

Mandatory-to-Specify: Yes.

Property: capability-value

Description: CDNI Capability object.

Type: Format/Type is defined by the value of capability-type property above.

Mandatory-to-Specify: Yes.

Property: footprints

Description: CDNI Capability Footprint.

Type: List of CDNI Footprint objects (as defined in [I-D.ietf-cdni-metadata]).

Mandatory-to-Specify: No.

5.2. Encoding

CDNI FCI objects MUST be encoded using JSON [RFC7159] and MUST also follow the recommendations of I-JSON [RFC7493]. FCI objects are composed of 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 capability or the CDNI Metadata Footprint Type of the footprint). 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 capability or the CDNI Metadata Footprint Type of the footprint).

Dictionary keys (properties) in JSON are case sensitive. By convention, any dictionary key (property) defined by this document MUST be lowercase.

5.3. Delivery Protocol Capability Object

The Delivery Protocol capability object is used to indicate support for one or more of the protocols listed in the CDNI Metadata Protocol

Types registry (defined in the CDNI Metadata Interface document [I-D.ietf-cdni-metadata]).

Property: delivery-protocols

Description: List of supported CDNI Delivery Protocols.

Type: List of Protocol Types (from the CDNI Metadata Protocol Types registry [I-D.ietf-cdni-metadata])

Mandatory-to-Specify: Yes.

5.3.1. Delivery Protocol Capability Object Serialization

The following shows an example of Delivery Protocol Capability Object Serialization, for a CDN that supports only HTTP/1.1 without TLS for content delivery.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.DeliveryProtocol",
      "capability-value": {
        "delivery-protocols": [
          "http/1.1",
        ]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

5.4. Acquisition Protocol Capability Object

The Acquisition Protocol capability object is used to indicate support for one or more of the protocols listed in the CDNI Metadata Protocol Types registry (defined in the CDNI Metadata Interface document [I-D.ietf-cdni-metadata]).

Property: acquisition-protocols

Description: List of supported CDNI Acquisition Protocols.

Type: List of Protocol Types (from the CDNI Metadata Protocol Types registry [I-D.ietf-cdni-metadata])

Mandatory-to-Specify: Yes.

5.4.1. Acquisition Protocol Capability Object Serialization

The following shows an example of Acquisition Protocol Capability Object Serialization, for a CDN that supports HTTP/1.1 with or without TLS for content acquisition.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.AcquisitionProtocol",
      "capability-value": {
        "acquisition-protocols": [
          "http/1.1",
          "https/1.1"
        ]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

5.5. Redirection Mode Capability Object

The Redirection Mode capability object is used to indicate support for one or more of the modes listed in the CDNI Capabilities Redirection Modes registry (see Section 6.2).

Property: redirection-modes

Description: List of supported CDNI Redirection Modes.

Type: List of Redirection Modes (from Section 6.2)

Mandatory-to-Specify: Yes.

5.5.1. Redirection Mode Capability Object Serialization

The following shows an example of Redirection Mode Capability Object Serialization, for a CDN that supports only iterative (but not recursive) redirection with HTTP and DNS.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.RedirectionMode",
      "capability-value": {
        "redirection-modes": [
          "DNS-I",
          "HTTP-I"
        ]
      }
    },
    {
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

5.6. CDNI Logging Capability Object

The CDNI Logging capability object is used to indicate support for CDNI Logging record-types, as well as CDNI Logging fields which are marked as optional for the specified record-types [I-D.ietf-cdni-logging].

Property: record-type

Description: Supported CDNI Logging record-type.

Type: String corresponding to an entry from the CDNI Logging record-types registry [I-D.ietf-cdni-logging])

Mandatory-to-Specify: Yes.

Property: fields

Description: List of supported CDNI Logging fields that are optional for the specified record-type.

Type: List of Strings corresponding to entries from the CDNI Logging Field Names registry [I-D.ietf-cdni-logging].

Mandatory-to-Specify: No. Default is that all optional fields are supported. Omission of this field MUST be interpreted as "all optional fields are supported". An empty list MUST be interpreted as "no optional fields are supported. Otherwise, if a list of fields is provided, the fields in that list MUST be interpreted as "the only optional fields that are supported".

5.6.1. 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>
      ]
    }
  ]
}
```

5.7. CDNI Metadata Capability Object

The CDNI Metadata capability object is used to indicate support for CDNI GenericMetadata types [I-D.ietf-cdni-metadata].

Property: metadata

Description: List of supported CDNI GenericMetadata types.

Type: List of Strings corresponding to entries from the CDNI Payload Type registry [RFC7736]) that correspond to CDNI GenericMetadata objects.

Mandatory-to-Specify: Yes. An empty list MUST be interpreted as "no GenericMetadata types are supported", i.e., "only structural metadata and simple types are supported"; otherwise, the list must be interpreted as containing "the only GenericMetadata types that are supported" (in addition to structural metadata and simple types) [I-D.ietf-cdni-metadata].

5.7.1. 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>
      ]
    }
  ]
}

```

6. IANA Considerations

6.1. CDNI Payload Types

This document requests the registration of the following CDNI Payload Types under the IANA CDNI Payload Type registry:

Payload Type	Specification
FCI.DeliveryProtocol	RFCthis
FCI.AcquisitionProtocol	RFCthis
FCI.RedirectionMode	RFCthis
FCI.Logging	RFCthis
FCI.Metadata	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.1.1. CDNI FCI DeliveryProtocol Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for supported delivery protocols

Interface: FCI

Encoding: see Section 5.3

6.1.2. CDNI FCI AcquisitionProtocol Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for supported acquisition protocols

Interface: FCI

Encoding: see Section 5.4

6.1.3. CDNI FCI RedirectionMode Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for supported redirection modes

Interface: FCI

Encoding: see Section 5.5

6.1.4. CDNI FCI Logging Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for supported CDNI Logging record-types and optional CDNI Logging Field Names.

Interface: FCI

Encoding: see Section 5.6

6.1.5. CDNI FCI Metadata Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for supported CDNI GenericMetadata types.

Interface: FCI

Encoding: see Section 5.7

6.2. Redirection Mode Registry

The IANA is requested to create a new "CDNI Capabilities Redirection Modes" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" category. The "CDNI Capabilities Redirection Modes" namespace defines the valid redirection modes that can be advertised as supported by a CDN. Additions to the Redirection Mode namespace conform to the "IETF Review" policy as defined in [RFC5226].

The following table defines the initial Redirection Modes:

Redirection Mode	Description	RFC
DNS-I	Iterative DNS-based Redirection	RFCthis
DNS-R	Recursive DNS-based Redirection	RFCthis
HTTP-I	Iterative HTTP-based Redirection	RFCthis
HTTP-R	Recursive HTTP-based Redirection	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

7. Security Considerations

This specification describes the semantics for capabilities and footprint advertisement objects across interconnected CDNs. It does not, however, specify a concrete protocol for transporting those objects. Specific security mechanisms can only be selected for concrete protocols that instantiate these semantics. This document does, however, place some high-level security constraints on such protocols.

All protocols that implement these semantics are REQUIRED to provide integrity and authentication services. Without authentication and integrity, an attacker could trivially deny service by forging a footprint advertisement from a dCDN which claims the network has no footprint or capability. This would prevent the uCDN from delegating any requests to the dCDN. Since a pre-existing relationship between all dCDNs and uCDNs is assumed by CDNI, the exchange of any necessary credentials could be conducted before the FCI interface is brought online. The authorization decision to accept advertisements would also follow this pre-existing relationship and any contractual obligations that it stipulates.

All protocols that implement these semantics are REQUIRED to provide confidentiality services. Some dCDNs are willing to share information about their footprint or capabilities with a uCDN but not with other, competing dCDNs. For example, if a dCDN incurs an outage that reduces footprint coverage temporarily, that could be information the dCDN would want to share confidentially with the uCDN.

As specified in this document, the security requirements of the FCI could be met by transport-layer security mechanisms coupled with domain certificates as credentials (e.g., TLS transport for HTTP as

per [RFC2818] and [RFC7230], with usage guidance from [RFC7525]) between CDNs. There is no apparent need for further object-level security in this framework, as the trust relationships it defines are bilateral relationships between uCDNs and dCDNs rather than transitive relationships.

8. References

8.1. Normative References

- [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-16 (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>>.
- [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>>.
- [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>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<http://www.rfc-editor.org/info/rfc7493>>.

8.2. Informative References

- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.

- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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. Main Use Case to Consider

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 can be considered optional), and what types of information need to be updated with respect to a priori established CDNI contracts.

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?

Appendix B. Semantics for Footprint Advertisement

Roughly speaking, "footprint" can be defined as "ability and willingness to serve" by a dCDN. However, in addition to simple "ability and willingness to serve", the uCDN could want 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, due to contractual agreements, SLAs, business relationships, or past perceptions of dCDN quality. 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, 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 (footprint) to which the contractual agreement applies. In particular, additional information to judge the delivery quality associated with a given dCDN footprint might be defined in contractual agreements, 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 and 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 "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 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:

- o capabilities such as delivery protocol, redirection mode, and metadata, which are supported in the coverage area for a "coverage/reachability" defined footprint, or
- o capabilities of resources, such as delivery protocol, redirection mode, and metadata, which apply to a "resource" defined footprint.

"Resource" types of footprints are more specific than "coverage/reachability" types of footprints, where the actual coverage/reachability are extrapolated from the resource location (e.g., netmask applied to resource IP address to derive IP-prefix). The specific methods for extrapolating coverage/reachability from resource location are beyond the scope of this document. In the degenerate case, the resource address could be specified as a coverage/reachability type of footprint, in which case no extrapolation is necessary. Resource types of footprints could expose the internal structure of a CDN network which could be undesirable. As such, the resource types of footprints are not considered mandatory to support for CDNI.

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: the

advertisement of different types of footprint narrows the dCDN candidacy cumulatively.

Independent of the exact type of a footprint, a footprint might also include the connectivity of a given dCDN to other CDNs that are able to serve content to users on behalf of that dCDN, to cover cases with cascaded CDNs. Further, the dCDN needs to be able to express its footprint to an interested uCDN in a comprehensive form, e.g., as a data set containing the complete footprint. Making incremental updates, however, to express dynamic changes in state is also desirable.

Appendix C. Semantics for Capabilities Advertisement

In general, the dCDN needs to 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 vs HTTPS delivery. Furthermore, the dCDN needs to be able to express particular capabilities for the delivery in a particular footprint area. For example, the dCDN might in general offer HTTPS but not in some specific areas, either for maintenance reasons or because the surrogates 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 need to be expressed on a per footprint basis, it could 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 dCDN is able (and willing) to accept (and properly handle) a delegated content request. In addition, Capabilities are characterized by the fact that this information can change over time based on the state of the network or surrogates.

At a first glance, several broad categories of capabilities seem useful to convey via an advertisement interface, 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 beyond the scope for 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, it seems infeasible to specify such highly dynamically changing capabilities and the corresponding metrics within a reasonable 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, 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 are better off being 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 could 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) will be mandatory among CDNs.

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. Instead, the CDNI FCI specification should define a generic protocol for conveying any capability information (e.g. with common encoding, error handling, and security mechanism; further requirements for the CDNI FCI Advertisement Interface are listed in [RFC7337]).

Appendix D. Acknowledgment

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.

Martin Stiernerling provided initial input to this document and valuable comments to the ongoing discussions among the authors of this document. Thanks to Francois Le Faucheur and Scott Wainner for providing valuable comments and suggestions to the text.

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

Email: jon.peterson@neustar.biz

Stefano Previdi
Cisco Systems
Via Del Serafico 200
Rome 0144
Italy

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
Ericsson
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.j.ma@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

B. Niven-Jenkins
R. Murray
Velocix (Alcatel-Lucent)
M. Caulfield
Cisco Systems
K. Ma
Ericsson
October 27, 2014

CDN Interconnection Metadata
draft-ietf-cdni-metadata-08

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 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 April 30, 2015.

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	4
1.2. Supported Metadata Capabilities	4
2. Design Principles	5
3. CDNI Metadata Data Model	6
3.1. HostIndex, HostMetadata and PathMetadata objects	7
3.2. Generic CDNI Metadata Object Properties	11
3.3. Metadata Inheritance and Override	13
4. Encoding-Independent CDNI Metadata Object Descriptions	14
4.1. Descriptions of the CDNI Structural Metadata Objects	15
4.1.1. HostIndex	15
4.1.2. HostMatch	15
4.1.3. HostMetadata	16
4.1.4. PathMatch	16
4.1.5. PathMetadata	17
4.1.6. PatternMatch	17
4.1.7. GenericMetadata	18
4.2. Description of the CDNI Generic Metadata Objects	19
4.2.1. Source Metadata	19
4.2.1.1. Source	20
4.2.2. LocationACL Metadata	21
4.2.2.1. LocationRule	21
4.2.2.2. Footprint	22
4.2.3. TimeWindowACL Metadata	22
4.2.3.1. TimeWindowRule	23
4.2.3.2. TimeWindow	23
4.2.4. ProtocolACL Metadata	24
4.2.4.1. ProtocolRule	24
4.2.5. DeliveryAuthorization Metadata	25
4.2.6. Cache	25
4.2.7. Grouping	26

4.3.	CDNI Metadata Simple Data Type Descriptions	26
4.3.1.	Link	26
4.3.2.	Protocol	27
4.3.3.	Endpoint	27
4.3.4.	URI	27
4.3.5.	Time	27
4.4.	Auth	27
4.4.1.	CredentialAuth Type	28
5.	CDNI Metadata Capabilities	28
6.	CDNI Metadata interface	29
6.1.	Transport	30
6.2.	Retrieval of CDNI Metadata resources	30
6.3.	Bootstrapping	31
6.4.	Encoding	32
6.4.1.	MIME Media Types	32
6.4.2.	JSON Encoding of Objects	32
6.4.2.1.	Encoded CDNI Metadata Example	33
6.5.	Extensibility	37
6.5.1.	Metadata Enforcement	37
6.5.2.	Metadata Conflicts	38
6.6.	Versioning	38
7.	IANA Considerations	39
7.1.	GenericMetadata Type Registry	40
7.1.1.	GenericMetadata Sub-Registries	41
7.1.1.1.	Footprint Sub-Registry	42
7.1.1.2.	Protocol Sub-Registry	42
7.1.1.3.	Authentication Type Sub-Registry	43
8.	Security Considerations	44
8.1.	Authentication	44
8.2.	Confidentiality	44
8.3.	Integrity	44
8.4.	Privacy	45
9.	Acknowledgements	45
10.	Contributing Authors	45
11.	References	45
11.1.	Normative References	46
11.2.	Informative References	46
	Authors' Addresses	47

1. Introduction

Content Delivery Networks Interconnection (CDNI) ([RFC6707]) enables a downstream CDN to service content requests on behalf of an upstream CDN.

The CDNI Metadata interface is discussed in [I-D.ietf-cdni-framework] along with four other interfaces that may be used to compose a CDNI solution (CDNI Control interface, CDNI Request Routing Redirection

interface, CDNI Footprint & Capabilities Advertisement interface and CDNI Logging interface). [I-D.ietf-cdni-framework] describes each interface, and the relationships between them. The requirements for the CDNI metadata interface are specified in [I-D.ietf-cdni-requirements].

The CDNI metadata associated with a piece of content (or with a set of content) 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.

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 Redirection interface.
- 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 RESTful 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 an object.

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 [I-D.ietf-cdni-requirements].

The following high-level functionality is configured via the metadata described 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.

This document supports HTTPv1.1 for delivery and both HTTPv1.1 and HTTPv1.1. over TLS for acquisition. All metadata is described in a protocol-agnostic manner.

Supporting unencrypted HTTPv2.0 for delivery (or unencrypted HTTPv2.0 or HTTPv2.0 over TLS for acquisition) only requires the registration of these protocol names in the CDNI Metadata Protocol Sub-Registry.

Supporting HTTPv1.1 over TLS or HTTPv2.0 over TLS for delivery requires specifying additional metadata objects to carry the properties required to establish a TLS session, for example metadata to describe the certificate to use as part of the TLS handshake.

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.
5. Leveraging of existing protocols.

Cacheability improves the latency of acquiring metadata while maintaining its freshness, and therefore improves 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 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., JSON) and data transport (e.g., HTTP).

3. CDNI Metadata Data Model

The CDNI Metadata Model describes a data structure for mapping redirection requests and content requests to metadata properties. Metadata properties describe how to acquire content from an upstream CDN, authorize access to content, 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 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 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 and stored by a dCDN once, even if it is referenced by the CDNI Metadata of multiple Hostnames or of multiple URI paths.

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 and PathMetadata objects

A HostIndex object contains (or references) a list of Hostnames (and/or IP addresses) for which content requests may be delegated to the downstream CDN. The HostIndex is the starting point for accessing the uCDN CDNI Metadata data store. It enables 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) against the HostMatch entries in the HostIndex, from there it can find HostMetadata which describes properties for a host and PathMetadata which may override those properties for given URI paths within the host.

HostMetadata and PathMetadata objects may also contain PathMatch objects which in turn contain PathMetadata objects. PathMetadata 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.

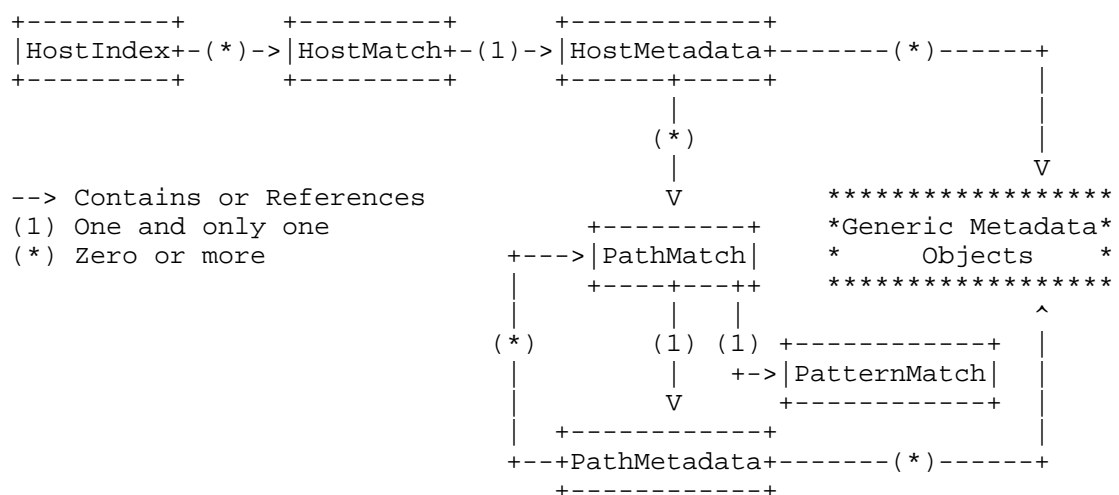


Figure 1: Relationships between CDNI Metadata Objects (Diagram Representation)

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)

The table below describes the HostIndex, HostMetadata and PathMetadata objects in more detail.

Data Object	Description
HostIndex	A HostIndex object lists HostMatch objects
HostMatch	A HostMatch object defines a hostname (or IP address) to match against a requested host, and contains (or references) a HostMetadata object which contains (or references) CDNI Metadata objects to be applied when a request matches against the hostname. For example, if "example.com" is a content provider, a HostMatch object may include an entry for "example.com" with the URI of the associated HostMetadata object.
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 (inside a

	<p>PatternMatch object which PathMatch object contains or references) to match against the requested URI 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. For example, a PathMatch object may include a PatternMatch object containing a pattern for the path "/movies/*" and may reference a PathMetadata object which contains (or references) the CDNI Metadata for content with that path.</p>
PatternMatch	<p>A PatternMatch object contains the pattern string and flags that describe the URI path that a PathMatch applies to.</p>
PathMetadata	<p>A PathMetadata object contains (or references) the CDNI GenericMetadata objects for content served with the associated URI path (defined in a PathMatch object). A PathMetadata object may also contain (or reference) 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.</p>
GenericMetadata	<p>A GenericMetadata object contains (or references) individual CDNI Metadata objects which define the specific policies and attributes needed to properly deliver the associated content. For example, a GenericMetadata object may describe the source from which a CDN may acquire a piece of content. The GenericMetadata object is an atomic unit that may be referenced by HostMetadata and/or PathMetadata objects, but SHOULD NOT contain references to other CDNI Metadata objects. The member objects of a specific CDNI Metadata object MAY be referenced by the GenericMetadata object.</p>

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 Metadata distribution, from the specifics of any given property (e.g., property semantics, enforcement options, 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 and the property type is not "mandatory-to-enforce", then that GenericMetadata object may be safely ignored and the dCDN MUST process the content request 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 may be "safe-to-redistribute" that metadata to another CDN without modification. For example, in the cascaded CDN case, a transit CDN may pass through "mandatory-to-enforce" metadata to a downstream CDN. 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 natively understood or supported by the transit CDN. However, for Metadata which requires translation, transparent redistribution of the uCDN Metadata values may not be appropriate. Certain Metadata may be safely, though possibly not optimally, redistributed unmodified. For example, source acquisition address may not be optimal if transparently redistributed, but may still work.

Redistribution safety MUST be specified for each GenericMetadata. If a CDN does not understand or support a given GenericMetadata property type and the property type is not "safe-to-redistribute", before redistributing the metadata, the CDN MUST set the "incomprehensible" flag for the GenericMetadata property that it did not understand and was marked as not "safe-to-redistribute". 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 may set the value of "incomprehensible" to true, a transit CDN MUST NOT change the value of "incomprehensible" from true to false.

The following table describes the action to be taken by a transit CDN (tCDN) for the different "mandatory-to-enforce" (MtE) and "safe-to-redistribute" (StR) cases, when the tCDN either does or does not understand the metadata in question:

MtE	StR	Metadata Understood	Actions
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 either by transforming not StR metadata (if the CDN know 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 and redistribute.
True	False	False	MUST NOT serve. MUST set "incomprehensible" to True when redistributing.

The following table describes the action to be taken by a dCDN for the different "mandatory-to-enforce" (MtE) and "incomprehensible" (Incomp) cases, when the dCDN either does or does not understand the metadata in question:

MtE	Metadata Understood	Incomp	Actions
False	True	False	Can serve.
False	True	True	Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.
False	False	False	Can serve.
False	False	True	Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.
True	True	False	Can serve.
True	True	True	Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.
True	False	False	MUST NOT serve.
True	False	True	MUST NOT serve.

3.3. Metadata Inheritance and Override

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. 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".
- o A single HostMetadata or PathMetadata object SHOULD NOT contain multiple GenericMetadata objects of the same type. If a list of

GenericMetadata contains objects of duplicate types, the receiver MUST ignore all but the first object of each type.

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 identify which specific properties apply to a given User Agent content request.

Section 4.2 provides the definitions for a base set of core metadata objects which may be contained within a GenericMetadata object. These objects are described as property objects, as they define the structure, semantics, and enforcement options for specific properties of the metadata being described. Property objects govern how User Agent requests for content are handled. Property objects may be composed of, or contain references to, other property sub-objects (i.e., property objects contained within or referenced by the property object that refers to that property sub-object). In those cases the value of the property sub-objects can be either a complete serialized representation of the sub-object, or a Link object that contains a URI and relationship 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 metadata objects specified in this document with additional standards based or vendor specific property objects that may be defined in the future in separate documents.

Downstream CDNs MUST support parsing of all CDNI metadata objects specified in this document. A dCDN does not have to implement the underlying functionality represented by the metadata object, though that may restrict the content that a given dCDN can serve. uCDNs as generators of CDNI Metadata only need to support generating the CDNI metadata that they need in order to express the policies and treatment required by the content they are describing.

Note: In the following sections, the term "mandatory-to-specify" is used to convey which property sub-objects MUST be specified for a given structural or property object. When mandatory-to-specify is set to true on a sub-object, it implies that if the property object containing that property sub-object is specified, then the mandatory-to-specify property sub-object MUST also be specified, e.g., a HostMatch property object without a host to match against does not make sense, therefore, the host is mandatory-to-specify inside a HostMatch property object.

4.1. Descriptions of the CDNI Structural Metadata Objects

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. It contains (or references) a list 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: List of HostMatch objects, in priority order.

Type: List of HostMatch objects

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 also contains or references a HostMetadata object 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

A HostMetadata object contains (or references) the CDNI Metadata properties for content served for a particular host (defined in the HostMatch object) and possibly child PathMatch objects.

Property: metadata

Description: List of host related metadata.

Type: List 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 PathMatch object that matches the content request being process is applied.

Type: List of PathMatch objects

Mandatory-to-Specify: No.

4.1.4. PathMatch

The PathMatch object contains (or references) an expression given as a PatternMatch object to match against a resource URI path and contains or references a PathMetadata object 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 path.

Type: PathMetadata

Mandatory-to-Specify: Yes.

4.1.5. PathMetadata

A PathMetadata object contains (or references) the CDNI Metadata properties for content served with the associated URI path (defined in a PathMatch object) and possibly child PathMatch objects.

Note that if DNS-based redirection is employed, then any metadata at the PathMetadata level or below will be inaccessible at request routing time because only the content request hostname is available at request routing time.

Property: metadata

Description: List of path related metadata.

Type: List of GenericMetadata objects

Mandatory-to-Specify: Yes.

Property: paths

Description: Path specific rules. First match applies.

Type: List of PathMatch objects

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 \?. 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.

Type: Boolean

Mandatory-to-Specify: No. Default is case-insensitive match.

Property: ignore-query-string

Description: List of query parameters which should be ignored when searching for a pattern match. If all query parameters should be ignored then the list MUST be empty.

Type: List of String

Mandatory-to-Specify: No. Default is 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: generic-metadata-type

Description: CDNI Metadata property object type.

Type: MIME Type String (from Section 7.1)

Mandatory-to-Specify: Yes.

Property: generic-metadata-value

Description: CDNI Metadata property object.

Type: Format/Type is defined by the value of generic-metadata-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: No. Default is to treat metadata as mandatory to enforce (i.e., true).

Property: safe-to-redistribute

Description: Flag identifying whether or not the property Metadata may be safely redistributed without modification.

Type: Boolean

Mandatory-to-Specify: No. Default is allow transparent redistribution (i.e., 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 the Metadata.

Type: Boolean

Mandatory-to-Specify: No. Default is comprehensible (i.e., false).

4.2. Description of the CDNI Generic Metadata Objects

The property 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 Media Type as defined in Section 7.1.

4.2.1. Source Metadata

Source Metadata provides the dCDN 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.

Endpoints within a source should be treated as equivalent/equal so one can specify a list of sources in preference order and for each source/preference rank one can specify a list of endpoints that are equivalent, e.g., a pool of servers that are not behind a load balancer.

Property: sources

Description: Sources from which the dCDN can acquire content, listed in order of preference.

Type: List of Source objects

Mandatory-to-Specify: No. Default is to use static configuration, out-of-band from the metadata interface.

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: acquisition-auth

Description: Authentication method to use when requesting content from this source.

Type: Auth

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, for example a pool of servers behind a load balancer.

Type: List of EndPoint objects

Mandatory-to-Specify: Yes.

Property: protocol

Description: Network retrieval protocol to use when requesting content from this source.

Type: Protocol

Mandatory-to-Specify: Yes.

4.2.2. LocationACL Metadata

LocationACL Metadata defines location-based restrictions.

A LocationACL which does not include a locations property results in an action of allow, meaning that delivery can be performed regardless of the User Agent's location. The action from the first footprint to match against the User Agent's location is the action a CDN MUST take. 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, and ProtocolACL are independent GenericMetadata objects, they may 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. Thus, in the example given, the request should be denied.

Property: locations

Description: Access control list which allows or blocks delivery based on client location.

Type: List of LocationRule objects

Mandatory-to-Specify: No. Default is allow all locations.

4.2.2.1. LocationRule

A LocationRule contains or references a list of Footprint objects and the corresponding action.

Property: footprints

Description: List of footprints to which the rule applies.

Type: List of Footprint objects

Mandatory-to-Specify: Yes.

Property: action

Description: Defines whether the rule specifies locations to allow or deny.

Type: Enumeration [allow|deny]

Mandatory-to-Specify: No. Default is deny.

4.2.2.2. Footprint

A Footprint object describes the footprint to which a LocationRule may be applied by, e.g., an IPv4 address range or a geographic location.

Property: footprint-type

Description: Registered footprint type (see Section 7.1.1.1).

Type: String

Mandatory-to-Specify: Yes.

Property: footprint-value

Description: Footprint object conforming to the specification associated with the registered footprint type.

Type: String

Mandatory-to-Specify: Yes.

4.2.3. TimeWindowACL Metadata

TimeWindowACL Metadata defines time-based restrictions.

A TimeWindowACL which does not include a times property results in an action of allow, meaning that delivery can be performed regardless of the time of the User Agent's request. The action from the first window to match against the current time is the action a CDN MUST take. 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, TimeWindowACL, and ProtocolACL are independent GenericMetadata objects, they may 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. Thus, in the example given, the request should be denied.

Property: times

Description: Description: Access control list which allows or blocks delivery based on request time.

Type: List of TimeWindowRule objects

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 and the corresponding action.

Property: windows

Description: List of time windows to which the rule applies.

Type: List of TimeWindow objects

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: No. Default is deny.

4.2.3.2. TimeWindow

A TimeWindow object describes a time range which may be applied by an TimeWindowACL, e.g., start 946717200 (i.e., 09:00AM 01/01/2000 UTC), end: 946746000 (i.e., 17:00AM 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.

A ProtocolACL which does not include a protocol-acl property results in an action of allow, meaning that delivery can be performed regardless of the protocol of the User Agent's request. The action from the first protocol to match against the request protocol is the action a CDN MUST take. 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 may 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. Thus, in the example given, the request should be denied.

Property: protocol-acl

Description: Description: Access control list which allows or blocks delivery based on delivery protocol.

Type: List of ProtocolRule objects

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 acquisition or delivery.

Property: protocols

Description: List of protocols to which the rule applies.

Type: List of protocol objects

Mandatory-to-Specify: Yes.

Property: action

Description: Defines whether the rule specifies protocols to allow or deny.

Type: Enumeration [allow|deny]

Mandatory-to-Specify: No. Default is deny.

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 method in the list is satisfied for that request.

Type: List of Auth objects

Mandatory-to-Specify: No. Default is no authorization required.

4.2.6. Cache

A Cache object describes the cache control parameters to be applied to the content by intermediate caches.

Property: ignore-query-string

Description: Allows a cache to ignore URI query string parameters while comparing URIs for equivalence. Each query parameter to ignore is specified in the list. If all query parameters should be ignored, then the list MUST be specified and MUST be empty.

Type: List of String

Mandatory-to-Specify: No. Default is to consider query string parameters when comparing URIs.

4.2.7. Grouping

A Grouping object identifies a large group of content to which a given asset belongs.

Property: ccid

Description: Content Collection identifier for an application-specific purpose such as logging.

Type: String

Mandatory-to-Specify: No. Default is an empty string.

Property: sid

Description: Session identifier for an application-specific purpose such as logging.

Type: String

Mandatory-to-Specify: No. Default is an empty string.

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 may be used in place of any of the objects or properties 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 and its type property is set to the type of the object it is replacing.

Property: href

Description: The URI of the addressable object being referenced.

Type: URI

Mandatory-to-Specify: Yes

Property: type

Description: The type of the object being referenced.

Type: String

Mandatory-to-Specify: No

4.3.2. Protocol

Protocol objects are used to specify registered protocols for content acquisition or delivery (see Section 7.1.1.2).

Type: String

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].

Type: String

4.3.4. URI

A URI as specified in [RFC3986].

Type: String

4.3.5. Time

A time value expressed in seconds since Unix epoch in the UTC timezone.

Type: Integer

4.4. Auth

An Auth object defines authentication and authorization methods to be used during content acquisition and content delivery, respectively.

Property: auth-type

Description: Registered Auth type (see Section 7.1.1.3).

Type: String

Mandatory-to-Specify: Yes.

Property: auth-value

Description: An object conforming to the specification associated with the Registered Auth type.

Type: String

Mandatory-to-Specify: Yes.

4.4.1. CredentialAuth Type

CredentialAuth is a Registered Auth type defining an object for encapsulating user credentials (i.e., username and password) (see Section 7.1.1.3). The CredentialAuth object contains the following properties:

Property: username

Description: Identification of user.

Type: String

Mandatory-to-Specify: Yes.

Property: password

Description: Password for user identified by username property.

Type: String

Mandatory-to-Specify: Yes.

5. CDNI Metadata Capabilities

CDNI Metadata is used to convey information pertaining to content delivery from uCDN to dCDN. For optional metadata, it may be useful for the uCDN to know if the dCDN supports the metadata, prior to delegating any content requests to the dCDN. If optional-to-implement metadata is "mandatory-to-enforce", and the dCDN does not support it, any delegated requests for that content will fail. The uCDN will likely want to avoid delegating those requests to that dCDN. Likewise, for any metadata which may be assigned optional

values, it may 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 may fail, so again the uCDN is likely to want to avoid delegating those requests to that dCDN.

The CDNI Footprint and Capabilities Interface (FCI) [I-D.ietf-cdni-framework] provides a means of advertising capabilities from dCDN to uCDN. Support for optional metadata and support for optional metadata values may be advertised using the FCI.

6. 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

- o Dynamically as required by the Downstream CDN to process received requests. For example in response to a query from an Upstream CDN 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.

The CDNI Metadata interface is built on the principles of RESTful 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 Data Model (as described in Section 3 through 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 a list of Hostnames for which the upstream CDN may delegate content delivery to the downstream CDN. 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 below 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.

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 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. A server implementation of the CDNI Metadata interface SHOULD reject all methods other than GET and HEAD.

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 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 for which the upstream CDN may delegate content delivery 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 HostMatch). 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. If any PathMetadata match the request (and are linked rather than embedded), 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 needs to determine 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). 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 always sufficient information carried in the DNS request from User Agents to determine the upstream CDN that redirected a particular request (e.g., when content from a given host is redirected to a given downstream CDN by more than one upstream CDN) and therefore downstream CDNs may have to apply local policy when deciding which upstream CDN's metadata to apply.

6.3. Bootstrapping

The URI for the HostIndex object of a given upstream CDN needs to be either configured in, or discovered by, 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. A mechanism allowing the downstream CDN to discover the URI of the HostIndex is outside the scope of this document.

6.4. Encoding

Objects 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 within a property of an addressable object.

The descriptions of objects use the phrase "X contains Y" to mean that Y is either directly embedded in X or 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 made separately addressable.

6.4.1. MIME Media Types

All MIME types for CDNI Metadata objects are prefixed with "application/cdni.". The MIME type for each object then contains the object name of that object as defined by this document. The object type name is followed by ".v" and the version number of the object type (e.g., ".v1"). Finally, the encoding type "+json" is appended. Table 3 lists a few examples of the MIME Media Type for some object (resource) that are retrievable through the CDNI Metadata interface.

Data Object	MIME Media Type
HostIndex	application/cdni.HostIndex.v1+json
HostMatch	application/cdni.HostMatch.v1+json
HostMetadata	application/cdni.HostMetadata.v1+json
PathMatch	application/cdni.PathMatch.v1+json
PathMetadata	application/cdni.PathMetadata.v1+json
Source	application/cdni.Source.v1+json
LocationACL	application/cdni.LocationACL.v1+json
LocationRule	application/cdni.LocationRule.v1+json

Table 3: Example MIME Media Types for CDNI Metadata objects

6.4.2. JSON Encoding of Objects

A CDNI Metadata object is encoded as 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. By convention any dictionary key defined by this document (for example the names of CDNI Metadata object properties) MUST be represented in lowercase.

In addition to the properties specified for 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 response MUST be absolute URLs.

Type: URI

Mandatory: No

Key: `_links`

Description: The links from this object to other addressable objects. Any property whose value is an object may be replaced by a link to an object with the same type as the property it replaces. The keys of the `_links` dictionary are the names of the properties being replaced. The values of the dictionary are Link objects with href set to the URI of the object and type set to the MIME type of the object being replaced.

Type: Dictionary object of Link objects

Mandatory: Yes

6.4.2.1. Encoded CDNI Metadata Example

A downstream CDN may request the HostIndex and receive the following object of type "application/cdni.HostIndex.v1+json":


```
{
  "hosts": [
    {
      "host": "video.example.com",
      "_links": {
        "host-metadata" : {
          "type": "application/cdni.HostMetadata.v1+json",
          "href": "http://metadata.ucdn.example/host1234"
        }
      }
    },
    {
      "host": "images.example.com",
      "_links": {
        "host-metadata" : {
          "type": "application/cdni.HostMetadata.v1+json",
          "href": "http://metadata.ucdn.example/host5678"
        }
      }
    }
  ]
}
```

If the incoming request has a Host header with "video.example.com" then the downstream CDN would fetch the next metadata object from "http://metadata.ucdn.example/host1234" expecting a MIME type of "application/cdni.HostMetadata.v1+json":

```
{
  "metadata": [
    {
      "generic-metadata-type": "application/cdni.SourceMetadata.v1+json",
      "generic-metadata-value": {
        "sources": [
          {
            "_links": {
              "acquisition-auth": {
                "auth-type": "application/cdni.Auth.v1+json",
                "href": "http://metadata.ucdn.example/auth1234"
              }
            },
            "endpoint": "acq1.ucdn.example",
            "protocol": "ftp"
          },
          {
            "_links": {
              "acquisition-auth": {
                "auth-type": "application/cdni.Auth.v1+json",

```

```

        "href": "http://metadata.ucdn.example/auth1234"
      }
    },
    "endpoint": "acq2.ucdn.example",
    "protocol": "http"
  }
]
},
{
  "generic-metadata-type": "application/cdni.LocationACL.v1+json",
  "generic-metadata-value": {
    "locations": [
      {
        "footprints": [
          {
            "footprint-type": "IPv4CIDR",
            "footprint-value": "192.168.0.0/16"
          }
        ],
        "action": "deny"
      }
    ]
  }
},
{
  "generic-metadata-type": "application/cdni.ProtocolACL.v1+json",
  "generic-metadata-value": {
    "protocol-acl": [
      {
        "protocols": [
          "ftp"
        ],
        "action": "deny"
      }
    ]
  }
},
{
  "paths": [
    {
      "path-pattern": {
        "pattern": "/video/trailers/*"
      },
      "_links": {
        "path-metadata": {
          "type": "application/cdni.PathMetadata.v1+json",
          "href": "http://metadata.ucdn.example/host1234/pathABC"
        }
      }
    }
  ]
}

```

```

    }
  },
  {
    "path-pattern": {
      "pattern": "/video/movies/*"
    },
    "_links": {
      "path-metadata": {
        "type": "application/cdni.PathMetadata.v1+json",
        "href": "http://metadata.ucdn.example/host1234/pathDCE"
      }
    }
  }
]
}

```

Suppose the path of the requested resource matches the `"/video/movies/*"` pattern, the next metadata requested would be for `"http://metadata.ucdn.example/host1234/pathDCE"` with an expected type of `"application/cdni.PathMetadata.v1+json"`:

```

{
  "metadata": [],
  "paths": [
    {
      "path-pattern": {
        "pattern": "/videos/movies/hd/*"
      },
      "_links": {
        "path-metadata": {
          "type": "application/cdni.PathMetadata.v1+json",
          "href":
            "http://metadata.ucdn.example/host1234/pathABC/path123"
        }
      }
    }
  ]
}

```

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/host1234/movies/hd"` with MIME type `"application/cdni.PathMetadata.v1+json"`:

```
{
  "metadata": [
    {
      "generic-metadata-type": "application/cdni.TimeWindowACL.v1+json",
      "generic-metadata-value": {
        "times": [
          "windows": [
            {
              "start": "1213948800",
              "end": "1327393200"
            }
          ],
          "action": "allow"
        }
      }
    ]
  }
}
```

6.5. Extensibility

The set of property Metadata may be extended with additional (standards based or vendor specific) 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. It is expected that additional property Metadata will be defined in the future and that the documents defining those property Metadata will be registered in the CDNI GenericMetadata Types registry Section 7.1.

Note: Identification, within the type name defined for a property Metadata object, of the organization that defined the extension property Metadata decreases the possibility of property Metadata type collisions.

6.5.1. Metadata Enforcement

At any given time, the set of GenericMetadata types supported by the uCDN may not match the set of GenericMetadata types supported by the dCDN.

In the 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 which 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 CDNI capabilities interface or through out-of-band negotiation between CDN operators) Metadata support may fluctuate or be inconsistent (e.g., due to mis-communication, mis-configuration, or temporary outage). Thus, the dCDN MUST always evaluate all Metadata associated with content requests and reject any requests where "mandatory-to-enforce" Metadata associated with the content cannot be enforced.

6.5.2. Metadata Conflicts

It is possible that new Metadata definitions may obsolete or conflict with 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.Auth.v2, vendor1.Auth, and vendor2.Auth) all conflict with 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 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 content 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.

6.6. Versioning

The version of CDNI Metadata Structural objects is conveyed inside the MIME-Type that is included in the HTTP Content-Type header. Upon responding to a request for an object, a metadata server MUST include a Content-Type header with the MIME-type containing the version number of the object. HTTP requests sent to a metadata server SHOULD include an Accept header with the MIME-type (which includes the version) of the expected object. Metadata clients can specify multiple MIME-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 MIME-types) it may decide to include both in the Accept header. The version of each object defined by this document is version 1. For example: "Content-Type: application/cdni.HostIndex.v1+json".

GenericMetadata objects include a "type" property which specifies the MIME-type of the GenericMetadata value. This MIME-type should also include a version. Any document which defines a new type of GenericMetadata MUST specify the version number which it describes. For example: "application/cdni.Location.v1+json".

7. IANA Considerations

This document requests the registration of the following MIME Media Type under the IANA MIME Media Type registry (<http://www.iana.org/assignments/media-types/index.html>).

application/cdni.HostIndex.v1
application/cdni.HostMetadata.v1
application/cdni.PathMatch.v1
application/cdni.PathMetadata.v1
application/cdni.PatternMatch.v1
application/cdni.GenericMetadata.v1
application/cdni.SourceMetadata.v1
application/cdni.Source.v1
application/cdni.LocationACL.v1
application/cdni.LocationRule.v1
application/cdni.Footprint.v1
application/cdni.TimeWindowACL.v1
application/cdni.TimeWindowRule.v1
application/cdni.TimeWindow.v1
application/cdni.ProtocolACL.v1
application/cdni.ProtocolRule.v1

```

application/cdni.Authorization.v1

application/cdni.Auth.v1

application/cdni.CredentialsAuth.v1

application/cdni.Cache.v1

application/cdni.Grouping.v1

```

7.1. GenericMetadata Type Registry

CDNI Metadata is distributed as a list of GenericMetadata objects which specify a type field and a type-specific value field, as described in Section 4.1.7. In order to prevent namespace collisions for GenericMetadata object types a new IANA registry is requested for the "CDNI GenericMetadata Types" namespace. The namespace shall be split into two partitions: standard and optional.

The standard namespace partition is intended to contain mandatory-to-implement capabilities and conforms to the "IETF Review" policy as defined in [RFC5226]. The registry contains the generic metadata type name, the RFC number of the specification defining the metadata type, the version number of the GenericMetadata set to which the standard capability applies, and boolean values indicating whether or not the new type is considered "mandatory-to-enforce" or "safe-to-redistribute" (as defined in Section 4.1.7).

The following table defines the initial values for the standard partition:

Type name	Specificati on	Versio n	MTE	STR
application/cdni.SourceMetadata.v1	RFCthis	1	true	true
application/cdni.LocationACL.v1	RFCthis	1	true	true
application/cdni.TimeWindowACL.v1	RFCthis	1	true	true
application/cdni.ProtocolACL.v1	RFCthis	1	true	true
application/cdni.Auth.v1	RFCthis	1	true	true
application/cdni.Cache.v1	RFCthis	1	true	true
application/cdni.Grouping.v1	RFCthis	1	true	true

The initial MI version number is set to 1. All of the initial GenericMetadata types are considered mandatory-to-implement for version 1. The version field should be incremented when new GenericMetadata type sets are added to the registry.

The "optional" namespace partition conforms to the "Expert Review" policy as defined in [RFC5226]. The expert review is intended to prevent namespace hoarding and to prevent the definition of redundant GenericMetadata types. Vendors defining new GenericMetadata types which conflict with existing GenericMetadata types follow the guidelines for the "Specification Required" policy as defined in [RFC5226]. The Version field in the registry is set to "-1" (negative one) for non-standard GenericMetadata types.

As with the initial GenericMetadata types defined in Section 4.2, future GenericMetadata type registrations will specify the information necessary for constructing and decoding the GenericMetadata object. This information includes the list of properties contained within the GenericMetadata object, and for each property, the specification should include a description, a type, and whether or not the given property is mandatory-to-specify.

Any document which defines a new GenericMetadata has to:

1. Allocate a new type in the GenericMetadata Type Registry (Section 7). Generic Metadata types should be descriptive and may be hierarchical to support aggregating groups of properties for the purpose of readability and for avoiding conflicts between vendor defined extensions. A dotted alpha-numeric notation is suggested for human readability.
2. Define the set of properties associated with the new type.
3. For each property, define a name, description, type, and whether or not the property is mandatory-to-specify.
4. Specify whether or not the new type is "mandatory-to-enforce" (vs optional-to-enforce).
5. Describe the semantics of the new type including its purpose and example of a use case to which it applies.

7.1.1. GenericMetadata Sub-Registries

Some of the initial standard GenericMetadata objects contain enumerated types which require registration (i.e., LocationACL footprint types, ProtocolACL protocols, and Auth protocols). The

following sections define the initial values for these GenericMetadata type sub-registries.

7.1.1.1. Footprint Sub-Registry

The IANA is requested to create a new "CDNI Metadata Footprint Types" sub-registry under the "CDNI GenericMetadata Types" 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 "Expert Review" policy as defined in [RFC5226]. The expert review should verify that new type definitions do not duplicate existing type definitions and prevent gratuitous additions to the namespace.

The following table defines the initial Footprint type values:

Type name	Description	Specification
IPv4CIDR	IPv4 address block using slash prefix length notation (e.g., 192.168.0.16/28). Single IP addresses can be expressed as /32.	RFCthis
IPv6CIDR	IPv6 address block using slash prefix length notation (e.g., fc80::0010/124). Single IP addresses can be expressed as /128.	RFCthis
ASN	Autonomous System (AS) Number	RFCthis
CountryCode	ISO 3166-1 alpha-2 code	RFCthis

7.1.1.2. Protocol Sub-Registry

The IANA is requested to create a new "CDNI Metadata Protocols" sub-registry under the "CDNI GenericMetadata Types" registry. The "CDNI Metadata Protocols" 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 "Expert Review" policy as defined in [RFC5226]. The expert review should 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:

Protocol	Description	Specification
HTTP	Hypertext Transfer Protocol -- HTTP/1.1	RFC2616
HTTPS	HTTP Over TLS	RFC2818
RTSP	Real Time Streaming Protocol	RFC2326
RTMP	Real-Time Messaging Protocol	http://www.adobe.com/devnet/rtmp.html
FTP	FILE TRANSFER PROTOCOL	RFC959
SFTP	SSH File Transfer Protocol	N/A
SCP	Secure Copy	N/A
fasp	Aspera fast, adaptive, secure protocol	N/A

7.1.1.3. Authentication Type Sub-Registry

The IANA is requested to create a new "CDNI Metadata Auth Types" sub-registry under the "CDNI GenericMetadata Types" registry. The "CDNI Metadata Auth Type" namespace defines the valid Auth object types used by the Auth object in Section 4.4. Additions to the Auth Type namespace conform to the "Expert Review" policy as defined in [RFC5226]. The expert review should verify that new type definitions do not duplicate existing type definitions and prevent gratuitous additions to the namespace.

The following table defines the initial Auth type values:

Type	Description	Specification
CredentialAuth	Simple username and password authentication as defined by Section 4.4.1.	RFCthis

8. Security Considerations

An implementation of the CDNI Metadata interface MUST support TLS transport as per [RFC2818] for message confidentiality and mutual authentication. An implementation of the CDNI Metadata interface MUST support the TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 cipher suite ([RFC2818]). An implementation of the CDNI Metadata interface SHOULD prefer cipher suites which support perfect forward secrecy over cipher suites that do not.

8.1. Authentication

Unauthorized access to metadata could result in denial of service. A malicious metadata server could provide metadata to a dCDN that denies service for one or more pieces of content to one or more user agents. A malicious metadata server could also provide metadata directing dCDNs to malicious origin servers instead of the actual origin servers. A malicious client could continuously issue large metadata requests to overload the uCDN metadata server.

Unauthorized access to metadata could 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 SHOULD use mutual authentication to prevent unauthorized access to metadata.

8.2. Confidentiality

Unauthorized viewing of metadata could result in leakage of private information. A third party could intercept metadata transactions in order to gain access to origin servers, as well as information pertaining to content restrictions.

An implementation of the CDNI Metadata interface SHOULD use strong encryption to prevent unauthorized viewing of metadata.

8.3. Integrity

Unauthorized modification of metadata could result in denial of service. A malicious proxy server could modify metadata destined to a dCDN in order to deny service for one or more pieces of content to one or more user agents. A malicious proxy server could also modify metadata directing dCDNs to malicious origin servers instead of the actual origin servers.

An implementation of the CDNI Metadata interface SHOULD use strong encryption and mutual authentication to prevent unauthorized modification of metadata.

8.4. Privacy

Content provider origin and policy information is conveyed through the CDNI Metadata interface. The distribution of this information to another CDN introduces potential content provider privacy protection concerns.

The use of TLS for transport of the CDNI Metadata as discussed above protects the confidentiality of content metadata by preventing any party other than the authorized dCDN from gaining access to content metadata.

9. Acknowledgements

The authors would like to thank David Ferguson and Francois Le Faucheur 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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

11.2. Informative References

- [I-D.ietf-cdni-framework] Peterson, L., Davie, B., and R. Brandenburg, "Framework for CDN Interconnection", draft-ietf-cdni-framework-14 (work in progress), June 2014.
- [I-D.ietf-cdni-redirection] Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection Interface for CDN Interconnection", draft-ietf-cdni-redirection-04 (work in progress), October 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.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 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

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

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 a 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.

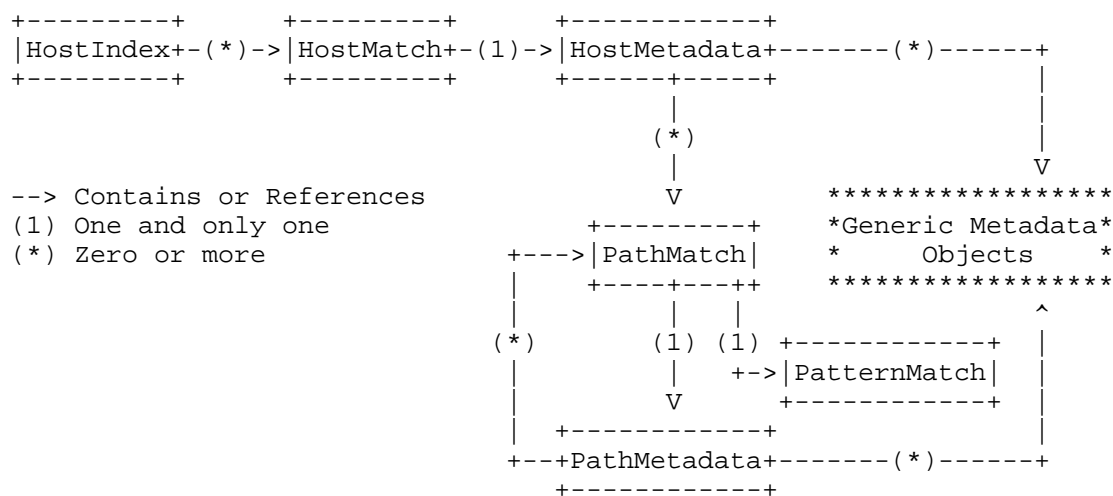


Figure 1: Relationships between CDNI Metadata Objects (Diagram Representation)

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

7.1.8. 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

7.1.9. 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

7.1.10. 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

7.1.11. 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

7.1.12. 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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2015

B. Niven-Jenkins, Ed.
Velocix (Alcatel-Lucent)
R. van Brandenburg, Ed.
TNO
October 25, 2014

Request Routing Redirection Interface for CDN Interconnection
draft-ietf-cdni-redirection-04

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 28, 2015.

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	2
2. Terminology	3
3. Interface function and operation overview	4
4. HTTP based interface for the Redirection Interface	5
4.1. Information passed in RI requests & responses	7
4.2. JSON encoding of RI requests & responses	8
4.3. MIME Media Types used by the RI interface	10
4.4. DNS redirection	10
4.4.1. DNS Redirection requests	10
4.4.2. DNS Redirection responses	12
4.5. HTTP Redirection	13
4.5.1. HTTP Redirection requests	13
4.5.2. HTTP Redirection responses	15
4.6. Cacheability and scope of responses	17
4.7. Error responses	19
4.8. Loop detection & prevention	23
5. Security Considerations	24
6. IANA Considerations	25
6.1. Media type registrations	25
6.1.1. CDNI RI requests	25
6.1.2. CDNI RI responses	26
6.2. RI Error response registry	27
7. Contributors	28
8. Acknowledgements	28
9. References	28
9.1. Normative References	28
9.2. Informative References	29
Authors' Addresses	29

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 [RFC7336] comprises of:

1. The asynchronous advertisement of footprint and capabilities by a downstream CDN that allows an 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 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

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 [RFC7337].

The User Agent will make a request to a request router in the uCDN using one of either DNS or HTTP. 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 that returned the RI response to the uCDN, or another CDN (if the 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 the 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 are required to support both of the request routing mechanisms illustrated in section 2.1 of [RFC7336], 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 document. 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 interface for the Redirection Interface

This document defines a simple interface for the Redirection Interface based on HTTP 1.1 [RFC7230], 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 examples of RI requests and responses below do not contain a complete set of HTTP headers for brevity, only the pertinent HTTP headers are shown.

The same HTTP interface is used for both DNS and HTTP redirection of User Agent 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 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:

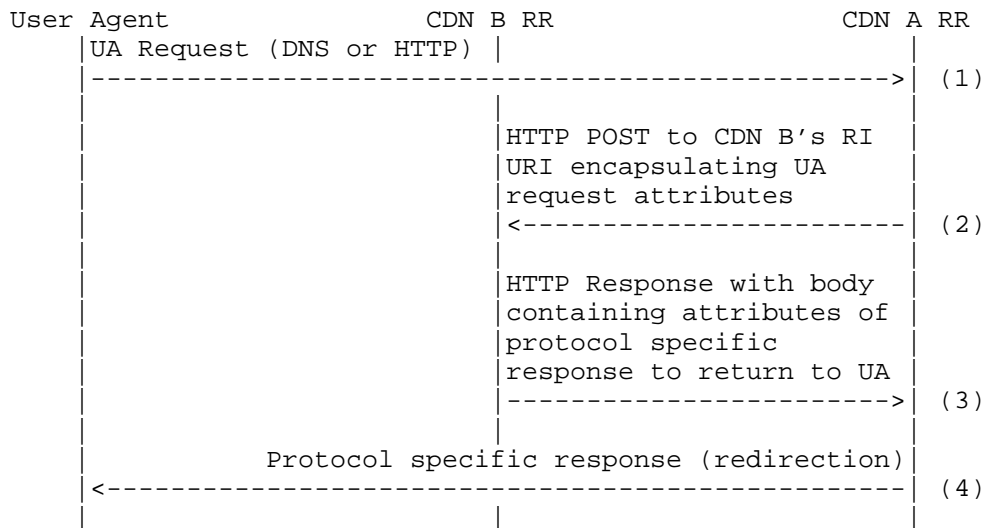


Figure 1: Generic Recursive Request Redirection message flow

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 ought to convey as much information as possible to the Downstream CDN, for example the URI of the requested content and the User Agent's IP address or subnet, 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 required 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 198.51.100.0 - 198.51.100.255".

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 [RFC7159] containing a dictionary of key:value pairs.

The following additional rules apply to all keys in RI requests and responses (whether in the top level object or in sub-objects):

- o Keys MUST always be encoded in lowercase.
- o Requests and responses MUST NOT contain duplicate keys.
- o Unknown keys MUST be ignored but the request or response MUST NOT be considered invalid unless the syntax of the request or response is invalid (i.e. a RI request or response MUST NOT be considered invalid on the basis that it contains unknown keys).
- o Requests or responses containing duplicate keys or containing keys that are not all lowercase are considered syntactically invalid.

The following top level keys are defined along with whether they are applicable to RI requests, RI responses or to both RI requests and responses:

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.
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 and responses MAY contain a cdn-path key. If the max-hops key is not present then there is no limit on the number of CDN hops that the RI request can be propagated along. If the first uCDN does not wish the RI request to be propagated beyond the dCDN it is making the request to then the uCDN MUST set max-hops to 1.

When cascading a RI request, a 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 MUST check the cdn-path and MUST NOT cascade the RI request to downstream CDNs that are already listed in cdn-path. The cdn-path MAY be reflected back in RI responses, although doing so could expose information to the uCDN that a dCDN may not wish to expose (for example, the existence of business relationships between a dCDN and other CDNs).

The presence of an error key within a response that also contains either a dns or http key does not automatically indicate that the RI request was unsuccessful as the error key MAY be used for communicating additional (e.g. debugging) information. When a response contains an error key as well as either a dns or http key, the error-code SHOULD be lxx (e.g. 100). See Section 4.7 for more details of encoding error information in RI responses.

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. MIME Media Types used by the RI interface

RI requests SHOULD use a MIME Media Type of application/cdni.redirectionrequest+json.

RI responses SHOULD use a MIME Media Type of application/cdni.redirectionresponse+json.

4.4. DNS redirection

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

4.4.1. 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 (usually either A or AAAA).
- o The class of DNS query made (usually 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).

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 (or prefix) 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.

A RI request for DNS-based redirection MUST include a dns dictionary. This dns dictionary MUST contain the following keys: resolver-ip, qtype, qclass, qname and the value of each MUST be the value of the appropriate part of the User Agent's DNS query/request.

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

```
POST /dcdn/ri HTTP/1.1
Host: rr1.dcdn.example.net
Content-Type: application/cdni.redirectionrequest+json
Accept: application/cdni.redirectionresponse+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": ["AS64496:0"],
  "max-hops": 3
}
```

4.4.2. DNS Redirection responses

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

- o The IP address(es) of (or the CNAME of) the RT (if the dCDN is performing DNS based redirection directly to a surrogate); or
- o The IP address(es) of (or the CNAME of) a RT which is a Request Router (if the dCDN is performing HTTP based redirection). A dCDN MUST NOT return a RT which is a Request Router if the dns-only key is set to True in the RI request.

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).
ttl	Integer	No	TTL of DNS response. Default is 0.

A successful RI response for DNS-based redirection MUST include a dns dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for DNS-based redirection MUST include an error dictionary. If a dns dictionary is included in the RI response, it MUST include at least one of the following keys: a, aaaa, cname. The dns dictionary MAY include both 'a' and 'aaaa' keys. If the dns dictionary contains a cname key it MUST NOT contain either an a or aaaa key.

An example of a successful RI response (dCDN->uCDN) for DNS based redirection with both a and aaaa keys is listed below :

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

```
{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "a" : ["203.0.113.200", "203.0.113.201"],
    "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
    "ttl" : 60
  }
}
```

A further example of a successful RI response (dCDN->uCDN) for DNS based redirection is listed below, in this case with a cname key containing the FQDN of the RT.

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

```
{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "cname" : ["rr1.dcdn.example"],
    "ttl" : 20
  }
}
```

4.5. HTTP Redirection

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

4.5.1. HTTP Redirection requests

For HTTP-based redirection the uCDN needs to 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.
- o The HTTP method requested by the User Agent

- o The HTTP version number requested by the User Agent.

The uCDN may also decide to pass the presence and value of particular HTTP headers included in the User Agent request to the dCDN.

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.
cs-uri	String	Yes	The URI requested by the UA.
cs-method	String	Yes	The method part of the request-line as defined in Section 3.1.1 of [RFC7230].
cs-version	String	Yes	The HTTP-version part of the request-line as defined in Section 3.1.1 of [RFC7230].
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.

A RI request for HTTP-based redirection MUST include an http dictionary. This http dictionary MUST contain the following keys: c-ip, cs-method, cs-version and cs-uri and the value of each MUST be the value of the appropriate part of the User Agent's DNS query/request.

The http dictionary of a RI request MUST contain a maximum of one cs(<headername>) key for each unique headername (HTTP header field). <headername> MUST be identical to the equivalent HTTP header field-name encoded in all lowercase. In the case where the User Agent request includes multiple HTTP header fields with the same field-name, it is up to the uCDN to determine how to handle this. One option would be to only send the contents of the first occurrence of that HTTP Header in the User Agent request. Another would be to combine the different HTTP Headers into a single value according to Section 3.2.2 of [RFC7230].

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

```
POST /dcdn/rrri HTTP/1.1
Host: rrl.dcdn.example.net
Content-Type: application/cdni.redirectionrequest+json
Accept: application/cdni.redirectionresponse+json
```

```
{
  "http": {
    "c-ip": "198.51.100.1",
    "cs-uri": "http://www.example.com",
    "cs-version": "HTTP/1.1",
    "cs-method": "GET"
  },
  "cdn-path": ["AS64496:0"],
  "max-hops": 3
}
```

4.5.2. HTTP Redirection responses

For a successful 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 performing HTTP based redirection 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 part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA (usually set to 302).
sc-version	String	Yes	The HTTP-version part of the status-Line as defined in Section 3.1.2 of [RFC7230] to return to the UA.
sc-reason	String	Yes	The reason-phrase part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.
cs-uri	String	Yes	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.
sc(<headername>)	String	No	The contents of the HTTP header named <HeaderName> to return to the UA. For example, sc(expires) would contain the content of the HTTP Expires header.

A successful RI response for HTTP-based redirection MUST include an http dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for HTTP-based redirection MUST include an error dictionary. If an http dictionary is included in the RI response, it MUST include at least the following keys: sc-status, sc-version, sc-reason, cs-uri, sc(location).

The http dictionary of a RI response MUST contain a maximum of one sc(<headername>) key for each unique headername (HTTP header field).

<headername> MUST be identical to the equivalent HTTP header field-name encoded in all lowercase.

The uCDN MAY decide to not return, override or alter some or all of the HTTP headers defined by sc(<headername>) keys before sending the HTTP response to the UA. It should be noted that in some cases, sending the HTTP Headers indicated by the dCDN transparently on to the UA might result in, for the uCDN, undesired behaviour. As an example, the dCDN might include sc(last-modified) and sc(expires) keys in the http dictionary, through which the dCDN may try to influence the cacheability of the response by the UA. If the uCDN would pass these HTTP headers on to the UA, this could mean that further requests from the uCDN would go directly to the dCDN, bypassing the uCDN and any logging it may perform on incoming requests. The uCDN is therefore recommended to carefully consider which HTTP headers to pass on, and which to either override or not pass on at all.

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/cdni.redirectionresponse+json
```

```
{
  "http": {
    "sc-status": 302,
    "sc-version": "HTTP/1.1",
    "sc-reason": "Found",
    "cs-uri": "http://www.example.com"
    "sc(location)":
      "http://surl.dcdn.example/ucdn/example.com",
    "sc(cache-control)" : "public, max-age=30"
  }
}
```

4.6. Cacheability and scope of responses

RI responses may be cacheable and may be reused by the uCDN in response to User Agent requests without the uCDN issuing another RI request to the dCDN if the RI response is considered cacheable & not stale according to the standard HTTP Cache-Control, etc mechanisms.

An RI response MUST NOT be reused unless the request from the User Agent would generate an identical RI request to the dCDN as the one that resulted in the cached RI response (except for the c-ip field

provided the User Agent's c-ip is covered by the scope in the original RI response).

Additionally, although RI requests only encode a single User Agent request to be redirected there may be cases where a dCDN wishes to indicate to the uCDN that the RI response can be reused for other User Agent requests without the uCDN having to make another request via the RI. For example a dCDN may know that it will always select the same Surrogates for a given set of User Agent IP addresses and in order to reduce request volume across the RI or to remove the additional latency associated with an RI request, the dCDN may wish to indicate that set of User Agent IP addresses to the uCDN in the initial RI response. This is achieved by including an optional scope dictionary in the RI response.

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 and a UA request comes in for which the User Agent's IP matches with the IP subnets in multiple of these cached responses, the uCDN SHOULD use the most recent cached response when determining the appropriate RI response to use.

The following is an example of a DNS redirection response from Section 4.4.2 that is cacheable by the uCDN for 30 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/24.


```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni.redirectionresponse+json
Cache-Control: public, max-age=30
```

```
{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "a" : ["203.0.113.200", "203.0.113.201"],
    "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
    "ttl" : 60
  }
  "scope" : {
    "iprange" : ["198.51.100.0/24"]
  }
}
```

Example of HTTP redirection response from Section 4.5.2 that is cacheable by the uCDN for 60 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/24.

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni.redirectionresponse+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" : ["198.51.100.0/24"]
  }
}
```

4.7. Error responses

From a uCDN perspective, there are two types of errors that can be the result of the transmission of an RI request to a dCDN: An HTTP protocol error signaled via an HTTP status code, indicating a problem with the reception or parsing of the RI request or the generation of the RI response by the dCDN, and a RI-level error specified in an RI

response message. This section deals with the latter type. The former type is outside the scope of this document.

There are numerous reasons for a dCDN not being able to return an affirmative RI response to a uCDN. Reasons may include both dCDN internal issues such as capacity problems, as well as reasons outside the influence of the dCDN, such as a malformed RI request. To aid with diagnosing the cause of errors, RI responses may include an error dictionary to provide additional information to the uCDN as to the reason/cause of the error. The intention behind the error dictionary is to aid with either manual or automatic diagnostics of issues. The resolution of such issues is outside the scope of this document and this document therefore does not specify the consequent actions a uCDN should take upon receiving a particular error code.

Error information (if present) is encoded as a set of key:value pairs within a JSON-encoded error dictionary as follows:

Key	Value	Mandatory	Description
error-code	Integer	No	A three-digit numeric code defined by the server to indicate the error(s) that occurred.
reason	String	No	A string providing further information related to the error.

The first digit of the error-code defines the class of error. There are 5 classes of error distinguished by the first digit of the error-code:

1xx: Informational (no error): The response should not be considered an error by the uCDN, which may proceed by redirecting the UA according to the values in the RI response. The error code and accompanying description may be used for informational purposes, e.g. for logging.

2xx: Reserved.

3xx: Reserved.

4xx: uCDN error: The dCDN can not or will not process the request due to something that is perceived to be a uCDN error, for example the RI request could not be parsed successfully by the dCDN. The

last two-digits may be used to more specifically indicate the source of the problem.

5xx: dCDN error: Indicates that the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason, for example the dCDN was able to parse the RI request but encountered an error for some reason. Examples include the dCDN not being able to retrieve the associated metadata or the dCDN being out of capacity.

The following error codes are defined and maintained by IANA (see Section 6):

Code	Reason	Description
100	<reason> (see Description)	Generic informational error-code meant for carrying a human-readable string
400	<reason> (see Description)	Generic error-code for uCDN errors where the dCDN can not or will not process the request due to something that is perceived to be a uCDN error. The reason field may be used to provide more details about the source of the error.
500	<reason> (see Description)	Generic error-code for dCDN errors where the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason. The reason field may be used to provide more details about the source of the error.
501	Unable to retrieve metadata	The dCDN is unable to retrieve the metadata associated with the content requested by the UA. This may indicate a configuration error or the content requested by the UA not existing.
502	Loop detected	The dCDN detected a redirection loop (see Section 4.8).
503	Maximum hops exceeded	The dCDN detected the maximum number of redirection hops exceeding max-hops (see Section 4.8).
504	Out of capacity	The dCDN does not currently have sufficient capacity to handle the UA request.
505	Delivery protocol not supported	The dCDN does not support the (set of) delivery protocols indicated in the CDNI Metadata of the content requested content by the UA.

Table 1

The following is an example of an unsuccessful RI response (dCDN->uCDN) for a DNS based User Agent request:

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

```
{
  "error" : {
    "code" : 504,
    "description" : "Out of capacity"
  }
}
```

The following is an example of a successful RI response (dCDN->uCDN) for a HTTP based User Agent request containing an error dictionary for informational purposes:

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni.redirectionresponse+json
Cache-Control: private, no-cache
```

```
{
  "http": {
    "sc-status": 302,
    "sc-version": "HTTP/1.1",
    "sc-reason": "Found",
    "cs-uri": "http://www.example.com"
    "sc(location)":
      "http://surl.dcdn.example/ucdn/example.com",
    "sc(cache-control)" : "public, max-age=30"
  },
  "error" : {
    "code" : 100,
    "description" :
      "This is a human-readable message meant for debugging purposes"
  }
}
```

4.8. 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 MUST check the cdn-path and reject any RI requests which already contain the downstream CDN's Provider ID in the cdn-path. Transit CDNs MUST check the cdn-path and not cascade the RI request to downstream CDNs that are already listed in cdn-path. Transit CDNs MUST NOT propagate

to any downstream CDNs if the number of CDN Provider IDs in `cdn-path` 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 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 "AS64496:0".

If a downstream CDN receives a RI request whose `cdn-path` already contains that downstream CDN's Provider ID the downstream CDN SHOULD send a RI response with an error code of 502.

If a downstream CDN receives a RI request where the number of CDN Provider IDs in `cdn-path` is greater than `max-hops`, the downstream CDN SHOULD send a RI response with an error code of 503.

It should be noted that the loop detection & prevention mechanisms described above only cover preventing and detecting loops within the RI itself. As well as loops with the RI itself, there is also the possibility of loops in the data plane, for example if the IP address(es) or URI(s) returned in RI responses do not resolve directly to a surrogate in the final dCDN there is the possibility that a User Agent may be continuously redirected through a loop of CDNs. The specification of solutions to address data plane request redirection loops between CDNs is outside of the scope of this document.

5. Security Considerations

Information passed over the RI could be considered personal or sensitive, for example RI requests contain parts of a User Agent's original request and RI responses reveal information about the dCDN's policy for which surrogates should serve which content/user locations.

The RI interface also provides a mechanism whereby a uCDN could probe a dCDN and infer the dCDN's edge topology by making repeated RI requests for different content and/or UA IP addresses and correlating the responses from the dCDN. Additionally the ability for a dCDN to indicate that a RI response applies more widely than the original request (via the scope dictionary) may significantly reduce the number of RI requests required to probe and infer the dCDN's edge topology.

The same information could be obtained in the absence of the RI interface, but it could be more difficult to gather as it would

require a distributed set of machines with a range of different IP addresses each making requests directly to the dCDN. However, the RI facilitates easier collection of such information as it enables a single client to query the dCDN for a redirection/surrogate selection on behalf of any UA IP address.

In order to prevent passive interception of RI messages the RI communications channel should be suitably secured (e.g. use of TLS).

In order to reduce the risk of information leakage to unauthorized parties, RI clients and servers SHOULD use suitable authentication prior to trusting the contents of RI messages.

6. IANA Considerations

6.1. Media type registrations

6.1.1. CDNI RI requests

The MIME media type for CDNI RI requests is application/cdni.redirectionrequest+json.

Type Name: application

Subtype name: cdni.redirectionrequest+json

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security Considerations: See [RFCthis], Section 5

Interoperability Considerations: Described in [RFCthis]

Published Specification: [RFCthis]

Applications that use this media type: No known applications currently use this media type.

Additional Information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File Extensions: N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information: IESG
<iesg@ietf.org>

Intended Useage: COMMON

Restrictions on usage: None

Author: Ben Niven-Jenkins <ben.niven-jenkins@alcatel-lucent.com>

Change controller: IESG <iesg@ietf.org>

Note: No "charset" parameter is defined for this registration because a charset parameter is not defined for application/json [RFC7159].

6.1.2. CDNI RI responses

The MIME media type for CDNI RI requests is application/cdni.redirectionresponse+json.

Type Name: application

Subtype name: cdni.redirectionresponse+json

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security Considerations: See [RFCthis], Section 5

Interoperability Considerations: Described in [RFCthis]

Published Specification: [RFCthis]

Applications that use this media type: No known applications currently use this media type.

Additional Information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File Extensions: N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information: IESG
<iesg@ietf.org>

Intended Useage: COMMON

Restrictions on usage: None

Author: Ben Niven-Jenkins <ben.niven-jenkins@alcatel-lucent.com>

Change controller: IESG <iesg@ietf.org>

Note: No "charset" parameter is defined for this registration because a charset parameter is not defined for application/json [RFC7159].

6.2. RI Error response registry

This document establishes a new IANA registry for CDNI RI Error response codes.

An expert reviewer is advised to examine new registrations for possible duplication with existing error codes and to ensure that the new code is in accordance with the error classes defined in section Section 4.7 of this document.

New registrations are required to provide the following information:

Code: A three-digit numeric error-code, in accordance with the error classes defined in section Section 4.7 of this document.

Reason: A string that provides further information related to the error that will be included in the JSON error dictionary with the 'reason'-key. Depending on the error-code semantics, the value of this field may be determined dynamically. In that case, the registration should set this value to '<reason>' and define its semantics in the description field.

Description: A brief description of the error code semantics.

Specification: An optional reference to a specification that defines in the error code in more detail.

The entries in table Table 1 are registered by this document.

7. Contributors

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

The following persons have participated as co-authors to this document:

Wang Danhua, Huawei, Email: wangdanhua@huawei.com

He Xiaoyan, Huawei, Email: hexiaoyan@huawei.com

Ge Chen, China Telecom, Email: cheng@gsta.com

Ni Wei, China Mobile, Email: niwei@chinamobile.com

Yunfei Zhang, Email: hishigh@gmail.com

Spencer Dawkins, Huawei, Email: spencer@wonderhamster.org

8. Acknowledgements

The authors would like to thank Taesang Choi, Francois le Faucheur and Scott Wainner 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.
- [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.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

9.2. Informative References

- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.
- [RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, August 2014.
- [RFC7337] Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, August 2014.

Authors' Addresses

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

Email: ben.niven-jenkins@alcatel-lucent.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: February 16, 2017

B. Niven-Jenkins, Ed.
Nokia
R. van Brandenburg, Ed.
TNO
August 15, 2016

Request Routing Redirection interface for CDN Interconnection
draft-ietf-cdni-redirection-20

Abstract

The Request Routing Interface comprises (1) the asynchronous advertisement of footprint and capabilities by a downstream Content Delivery Network (CDN) that allows an 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 February 16, 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	2
2. Terminology	3
3. Interface function and operation overview	4
4. HTTP based interface for the Redirection Interface	5
4.1. Information passed in RI requests & responses	7
4.2. JSON encoding of RI requests & responses	8
4.3. MIME Media Types used by the RI interface	10
4.4. DNS redirection	10
4.4.1. DNS Redirection requests	10
4.4.2. DNS Redirection responses	12
4.5. HTTP Redirection	14
4.5.1. HTTP Redirection requests	14
4.5.2. HTTP Redirection responses	16
4.6. Cacheability and scope of responses	18
4.7. Error responses	20
4.8. Loop detection & prevention	24
5. Security Considerations	25
5.1. Authentication, Authorization, Confidentiality, Integrity Protection	26
5.2. Privacy	26
6. IANA Considerations	27
6.1. CDNI Payload Type Parameter registrations	27
6.1.1. CDNI RI Redirection Request Payload Type	27
6.1.2. CDNI RI Redirection Response Payload Type	28
6.2. RI Error response registry	28
7. Contributors	29
8. Acknowledgements	29
9. References	29
9.1. Normative References	29
9.2. Informative References	31
Authors' Addresses	31

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 [RFC7336] comprises of:

1. The asynchronous advertisement of footprint and capabilities by a downstream CDN (dCDN) that allows an upstream CDN (uCDN) to decide whether to redirect particular user requests to that dCDN.
2. The synchronous operation of a uCDN requesting whether a dCDN is prepared to accept a user request and of a dCDN 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 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 [RTMP].

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 dCDN or a surrogate in a dCDN, etc.

3. Interface function and operation overview

The main function of the CDNI Redirection interface (RI) 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 [RFC7337].

The User Agent will make a request to a request router in the uCDN using one of either DNS or HTTP. 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. 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 that returned the RI response to the uCDN, or another CDN (if the dCDN delegates the delivery to another CDN); or
- o A request router (in the dCDN or another CDN), which may use a different redirection protocol (DNS or HTTP) than the one included in the RI request.

The Redirection interface operates between the request routing systems of a pair of interconnected CDNs. To enable communication over the Redirection interface, the uCDN needs to know the URI (end point) in the dCDN to send CDNI request routing queries.

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.

The Redirection interface is only relevant in the case of Recursive Request Redirection, as Iterative Request Redirection does not invoke any interaction over the Redirection interface between interconnected CDNs. Therefore, the scope of this document is limited to Recursive Request Redirection.

In the case of Recursive Request Redirection, in order to perform redirection of a request received from a User Agent, the uCDN queries the dCDN so that the dCDN can select and provide a Redirection Target. In cases where a uCDN has a choice of dCDNs it is up 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 rejects 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 uCDN->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 document. However, the Redirection interface is designed to be extensible and could be extended to support additional application level redirection protocols.

For both DNS & HTTP redirection, either HTTP or HTTPS could be used to connect to the Redirection Target. When HTTPS is used to connect to the uCDN, if the uCDN uses DNS redirection to identify the RT to the User Agent, then the new target domain name may not match the domain in the URL dereferenced to reach the uCDN; without operational precautions, and in the absence of DNSSEC, this can make a legitimate redirection look like a DNS-based attack to a User Agent and trigger security warnings. When DNS-based redirection with HTTPS is used, this specification assumes that any RT can complete the necessary TLS handshake with the User Agent. Any operational mechanisms this requires, e.g., private key distribution to surrogates and request routers in dCDNs, are outside the scope of this document.

This document also defines an RI loop prevention and detection mechanism as part of the Redirection interface.

4. HTTP based interface for the Redirection Interface

This document defines a simple interface for the Redirection interface based on HTTP [RFC7230], where the attributes of a User Agent's requests are encapsulated along with any other data that can aid the dCDN in processing the requests. The RI response encapsulates the attributes of the RT(s) that the uCDN should return to the User Agent (if it decides to utilize the dCDN for delivery) along with the policy for how the response can be reused. The examples of RI requests and responses below do not contain a complete set of HTTP headers for brevity; only the pertinent HTTP headers are shown.

The RI between the uCDN and dCDN uses the same HTTP interface to encapsulate the attributes of both DNS and HTTP requests received from User Agents, 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 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 dCDN to make a request routing decision, avoids having to try to 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) by defining additional protocol specific keys for RI requests and responses along with a specification how to process them.

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

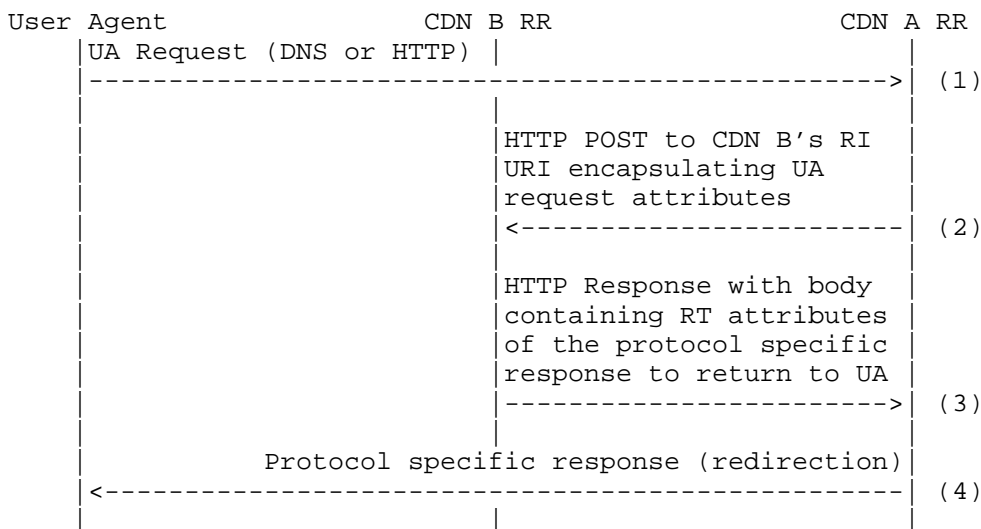


Figure 1: Generic Recursive Request Redirection message flow

1. The User Agent sends its (DNS or HTTP) request to CDN A. The Request Routing System of CDN A processes the request and, through local policy, recognizes that the request is best served by another CDN, specifically CDN B (or that CDN B may be 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 RI request and assuming the request is well formed, 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 uCDN.
2. Properties/parameters that the uCDN can use to control the dCDN's response or that can help the dCDN make its decision.

Generally, dCDNs can provide better routing decisions given additional information about the content request, e.g., the URI of the requested content or the User Agent's IP address or subnet. The set of information required to base a routing decision on can be highly dependent on the type of content delivered. A uCDN SHOULD only include information that is absolutely necessary for delivering that type of content. Cookies in particular are particularly sensitive from a security/privacy point of view and in general SHOULD NOT be conveyed in the RI Requests to the dCDN. The set of information necessary to be conveyed for a particular type of request is expected to be conveyed out of band between the uCDN and dCDN. See Section 5.2 for more detail on the privacy aspects of using RI Requests to convey information about UA requests.

In order for the dCDN 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 dCDN to be able to retrieve the required 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 dCDN may wish to return additional policy information to the uCDN to it with future RI requests. For example, the dCDN may wish to return a policy that expresses "this response can be reused without requiring an RI request for 60 seconds provided the User Agent's IP address is in the range 198.51.100.0 - 198.51.100.255".

These additional policies split into two basic categories:

- o Cacheability information signaled via the HTTP response headers of the RI response (to reduce the number of subsequent RI requests the uCDN needs to make).
- o The scope of a cacheable response signaled in the HTTP response body of the RI 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 [RFC7159] that MUST conform to [RFC7493] containing a dictionary of key:value pairs. Senders MUST encode all (top level object and sub-object) keys specified in this document in lowercase. Receivers MUST ignore any keys that are unknown or invalid.

The following top level keys are defined along with whether they are applicable to RI requests, RI responses or both:

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 a list of the CDN Provider IDs of previous CDNs that have participated in the request routing for the associated User Agent request. On RI requests it contains the list of previous CDNs that this RI request has passed through. On RI responses it contains the list of CDNs that were involved in obtaining the final redirection included in the RI response. See Section 4.8
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 coarsely 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 and responses MAY contain a cdn-path key. If the max-hops key is not present then there is no limit on the number of CDN hops that the RI request can be propagated along. If the first uCDN does not wish the RI request to be propagated beyond the dCDN it is making the request to, then the uCDN MUST set max-hops to 1.

The `cdn-path` MAY be reflected back in RI responses, although doing so could expose information to the uCDN that a dCDN may not wish to expose (for example, the existence of business relationships between a dCDN and other CDNs).

If the `cdn-path` is reflected back in the RI response it MUST contain the value of `cdn-path` received in the associated RI request with the final dCDN's CDN Provider ID appended. Transit CDNs MAY remove the `cdn-path` from RI responses but MUST NOT modify the `cdn-path` in other ways.

The presence of an error key within a response that also contains either a `dns` or `http` key does not automatically indicate that the RI request was unsuccessful as the error key MAY be used for communicating additional (e.g., debugging) information. When a response contains an error key as well as either a `dns` or `http` key, the error-code SHOULD be `lxx` (e.g., 100). See Section 4.7 for more details of encoding error information in RI responses.

All implementations that support IPv4 addresses MUST support the encoding specified by the '`IPv4address`' rule in Section 3.2.2 of [RFC3986]. Likewise, implementations that support IPv6 addresses MUST support all IPv6 address formats specified in [RFC4291]. Server implementations SHOULD use IPv6 address formats specified in [RFC5952].

4.3. MIME Media Types used by the RI interface

RI requests MUST use a MIME Media Type of `application/cdni` as specified in [RFC7736], with the Payload Type (`ptype`) parameter set to '`redirection-request`'.

RI responses MUST use a MIME Media Type of `application/cdni` as specified in [RFC7736], with the Payload Type (`ptype`) parameter set to '`redirection-response`'.

4.4. DNS redirection

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

4.4.1. 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 uCDN.
- o The type of DNS query made (usually either A or AAAA).
- o The class of DNS query made (usually 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 uCDN).

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. The value of this field SHALL be set to either 'A' or 'AAAA'.
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 (or prefix) of the UA in CIDR format.
dns-only	Boolean	No	If True then dCDN MUST only use DNS redirection and MUST include RTs to one or more surrogates in any successful RI response. CDNs MUST include the dns-only property set to True on any cascaded RI requests. Defaults to False.

An RI request for DNS-based redirection MUST include a dns dictionary. This dns dictionary MUST contain the following keys: resolver-ip, qtype, qclass, qname and the value of each MUST be the value of the appropriate part of the User Agent's DNS query/request. For internationalized domain names containing non-ASCII characters,

the value of the qname field MUST be the ASCII-compatible encoded (ACE) representation (A-label) of the domain name [RFC5890].

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

```
POST /dcdn/ri HTTP/1.1
Host: rrl.dcdn.example.net
Content-Type: application/cdni; ptype=redirection-request
Accept: application/cdni; ptype=redirection-response
```

```
{
  "dns" : {
    "resolver-ip" : "192.0.2.1",
    "c-subnet" : "198.51.100.0/24",
    "qtype" : "A",
    "qclass" : "IN",
    "qname" : "www.example.com"
  },
  "cdn-path": ["AS64496:0"],
  "max-hops": 3
}
```

4.4.2. DNS Redirection responses

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

- o The IP address(es) of (or the CNAME of) RTs that are dCDN surrogates (if the dCDN is performing DNS based redirection directly to a surrogate); or
- o The IP address(es) of (or the CNAME of) RTs that are Request Routers (if the dCDN will perform request redirection itself). A dCDN MUST NOT return a RT which is a Request Router if the dns-only key is set to True in the RI request.

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 (see [RFC6895]).
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).
ttl	Integer	No	TTL in seconds of DNS response. Default is 0.

A successful RI response for DNS-based redirection MUST include a dns dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for DNS-based redirection MUST include an error dictionary. If a dns dictionary is included in the RI response, it MUST include the rcode and name keys and it MUST include at least one of the following keys: a, aaaa, cname. The dns dictionary MAY include both 'a' and 'aaaa' keys. If the dns dictionary contains a cname key it MUST NOT contain either an a or aaaa key. For internationalized domain names containing non-ASCII characters, the value of the cname field MUST be the ASCII-compatible encoded (ACE) representation (A-label) of the domain name.

An example of a successful RI response (dCDN->uCDN) for DNS based redirection with both a and aaaa keys is listed below :

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
```

```
{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "a" : ["203.0.113.200", "203.0.113.201", "203.0.113.202"],
    "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
    "ttl" : 60
  }
}
```


A further example of a successful RI response (dCDN->uCDN) for DNS based redirection is listed below, in this case with a cname key containing the FQDN of the RT.

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
```

```
{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "cname" : ["rr1.dcdn.example"],
    "ttl" : 20
  }
}
```

4.5. HTTP Redirection

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

The dictionary keys used in HTTP Redirection requests and responses use the following conventions for their prefixes:

- o c- is prefixed to keys for information related to the Client (User Agent).
- o cs- is prefixed to keys for information passed by the Client (User Agent) to the Server (uCDN).
- o sc- is prefixed to keys for information to be passed by the Server (uCDN) to the Client (User Agent).

4.5.1. HTTP Redirection requests

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

- o The IP address of the User Agent.
- o The URI requested by the User Agent.
- o The HTTP method requested by the User Agent
- o The HTTP version number requested by the User Agent.

The uCDN may also decide to pass the presence and value of particular HTTP headers included in the User Agent request to the dCDN.

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.
cs-uri	String	Yes	The Effective Request URI [RFC7230] requested by the UA.
cs-method	String	Yes	The method part of the request-line as defined in Section 3.1.1 of [RFC7230].
cs-version	String	Yes	The HTTP-version part of the request-line as defined in Section 3.1.1 of [RFC7230].
cs-(<headername>)	String	No	The field-value of the HTTP header field named <HeaderName> as a string, for example, cs-(cookie) would contain the value of the HTTP Cookie header from the UA request.

An RI request for HTTP-based redirection MUST include an http dictionary. This http dictionary MUST contain the following keys: c-ip, cs-method, cs-version and cs-uri and the value of each MUST be the value of the appropriate part of the User Agent's HTTP request.

The http dictionary of an RI request MUST contain a maximum of one cs-(<headername>) key for each unique header field-name (HTTP header field). <headername> MUST be identical to the equivalent HTTP header field-name encoded in all lowercase.

In the case where the User Agent request includes multiple HTTP header fields with the same field-name, it is RECOMMENDED that the uCDN combines these different HTTP headers into a single value according to Section 3.2.2 of [RFC7230]. However, because of the plurality of already defined HTTP header fields, and inconsistency of some of these header fields concerning the combination mechanism

defined in RFC 7230, the uCDN MAY have to deviate from using the combination mechanism where appropriate. For example, it might only send the contents of the first occurrence of the HTTP Headers instead.

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

```
POST /dcdn/rrri HTTP/1.1
Host: rr1.dcdn.example.net
Content-Type: application/cdni; ptype=redirection-request
Accept: application/cdni; ptype=redirection-response
```

```
{
  "http": {
    "c-ip": "198.51.100.1",
    "cs-uri": "http://www.example.com",
    "cs-version": "HTTP/1.1",
    "cs-method": "GET"
  },
  "cdn-path": ["AS64496:0"],
  "max-hops": 3
}
```

4.5.2. HTTP Redirection responses

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

- o A URI pointing to an RT that is the selected dCDN surrogate(s) (if the dCDN is performing HTTP based redirection directly to a surrogate); or
- o A URI pointing to an RT that is a Request Router (if the dCDN will perform request redirection itself).

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 part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA (usually set to 302).
sc-version	String	Yes	The HTTP-version part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.
sc-reason	String	Yes	The reason-phrase part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.
cs-uri	String	Yes	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-(<headername>)	String	No	The field-value of the HTTP header field named <HeaderName> to return to the UA. For example, sc-(expires) would contain the value of the HTTP Expires header.

Note: The sc-(location) key in the table above is an example of sc-(<headername>) that has been called out separately as its presence is mandatory in RI responses.

A successful RI response for HTTP-based redirection MUST include an http dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for HTTP-based redirection MUST include an error dictionary. If an http dictionary is included in the RI response, it MUST include at least the following keys: sc-status, sc-version, sc-reason, cs-uri and sc-(location).

The http dictionary of an RI response MUST contain a maximum of one sc-(<headername>) key for each unique header field-name (HTTP header field). <headername> MUST be identical to the equivalent HTTP header field-name encoded in all lowercase.

The uCDN MAY decide to not return, override or alter any or all of the HTTP headers defined by sc-(<headername>) keys before sending the HTTP response to the UA. It should be noted that in some cases, sending the HTTP Headers indicated by the dCDN transparently on to the UA might result in, for the uCDN, undesired behaviour. As an example, the dCDN might include sc-(cache-control), sc-(last-modified) and sc-(expires) keys in the http dictionary, through which the dCDN may try to influence the cacheability of the response by the UA. If the uCDN would pass these HTTP headers on to the UA, this could mean that further requests from the uCDN would go directly to the dCDN, bypassing the uCDN and any logging it may perform on incoming requests. The uCDN is therefore recommended to carefully consider which HTTP headers to pass on, and which to either override or not pass on at all.

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/cdni; ptype=redirection-response

{
  "http": {
    "sc-status": 302,
    "sc-version": "HTTP/1.1",
    "sc-reason": "Found",
    "cs-uri": "http://www.example.com"
    "sc-(location)":
      "http://surl.dcdn.example/ucdn/example.com",
  }
}
```

4.6. Cacheability and scope of responses

RI responses may be cacheable. As long as a cached RI response is not stale according to standard HTTP Cache-Control or other applicable mechanisms, it may be reused by the uCDN in response to User Agent requests without sending another RI request to the dCDN.

An RI response MUST NOT be reused unless the request from the User Agent would generate an identical RI request to the dCDN as the one that resulted in the cached RI response (except for the c-ip field

provided that the User Agent's c-ip is covered by the scope in the original RI response, as elaborated upon below).

Additionally, although RI requests only encode a single User Agent request to be redirected there may be cases where a dCDN wishes to indicate to the uCDN that the RI response can be reused for other User Agent requests without the uCDN having to make another request via the RI. For example, a dCDN may know that it will always select the same Surrogates for a given set of User Agent IP addresses and in order to reduce request volume across the RI or to remove the additional latency associated with an RI request, the dCDN may wish to indicate that set of User Agent IP addresses to the uCDN in the initial RI response. This is achieved by including an optional scope dictionary in the RI response.

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 and a UA request comes in for which the User Agent's IP matches with the IP subnets in multiple of these cached responses, the uCDN SHOULD use the most recent cached response when determining the appropriate RI response to use.

The following is an example of a DNS redirection response from Section 4.4.2 that is cacheable by the uCDN for 30 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/24.

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: public, max-age=30
```

```
{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "a" : ["203.0.113.200", "203.0.113.201"],
    "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
    "ttl" : 60
  }
  "scope" : {
    "iprange" : ["198.51.100.0/24"]
  }
}
```

Example of HTTP redirection response from Section 4.5.2 that is cacheable by the uCDN for 60 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/24.

Note: The response to the UA is only valid for 30 seconds, whereas the uCDN can cache the RI response for 60 seconds.

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
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" : ["198.51.100.0/24"]
  }
}
```

4.7. Error responses

From a uCDN perspective, there are two types of errors that can be the result of the transmission of an RI request to a dCDN:

1. An HTTP protocol error signaled via an HTTP status code, indicating a problem with the reception or parsing of the RI request or the generation of the RI response by the dCDN, and
2. An RI-level error specified in an RI response message

This section deals with the latter type. The former type is outside the scope of this document.

There are numerous reasons for a dCDN to be unable to return an affirmative RI response to a uCDN. Reasons may include both dCDN internal issues such as capacity problems, as well as reasons outside the influence of the dCDN, such as a malformed RI request. To aid with diagnosing the cause of errors, RI responses SHOULD include an error dictionary to provide additional information to the uCDN as to the reason/cause of the error. The intention behind the error dictionary is to aid with either manual or automatic diagnosis of issues. The resolution of such issues is outside the scope of this document; this document does not specify any consequent actions a uCDN should take upon receiving a particular error code.

Error information (if present) is encoded as a set of key:value pairs within a JSON-encoded error dictionary as follows:

Key	Value	Mandatory	Description
error-code	Integer	Yes	A three-digit numeric code defined by the server to indicate the error(s) that occurred.
reason	String	No	A string providing further information related to the error.

The first digit of the error-code defines the class of error. There are 5 classes of error distinguished by the first digit of the error-code:

1xx: Informational (no error): The response should not be considered an error by the uCDN, which may proceed by redirecting the UA according to the values in the RI response. The error code and accompanying description may be used for informational purposes, e.g., for logging.

2xx: Reserved.

3xx: Reserved.

4xx: uCDN error: The dCDN can not or will not process the request due to something that is perceived to be a uCDN error, for example, the RI request could not be parsed successfully by the dCDN. The last two-digits may be used to more specifically indicate the source of the problem.

5xx: dCDN error: Indicates that the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason, for example, the dCDN was able to parse the RI request but encountered an error for some reason. Examples include the dCDN not being able to retrieve the associated metadata or the dCDN being out of capacity.

The following error codes are defined and maintained by IANA (see Section 6):

Error codes with a "Reason" of "<reason>" do not have a defined value for their 'reason'-key. Depending on the error-code semantics, the value of this field may be determined dynamically.

Code	Reason	Description
100	<reason> (see Description)	Generic informational error-code meant for carrying a human-readable string
400	<reason> (see Description)	Generic error-code for uCDN errors where the dCDN can not or will not process the request due to something that is perceived to be a uCDN error. The reason field may be used to provide more details about the source of the error.
500	<reason> (see Description)	Generic error-code for dCDN errors where the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason. The reason field may be used to provide more details about the source of the error.
501	Unable to retrieve metadata	The dCDN is unable to retrieve the metadata associated with the content requested by the UA. This may indicate a configuration error or the content requested by the UA not existing.
502	Loop detected	The dCDN detected a redirection loop (see Section 4.8).
503	Maximum hops exceeded	The dCDN detected the maximum number of redirection hops exceeding max-hops (see Section 4.8).
504	Out of capacity	The dCDN does not currently have sufficient capacity to handle the UA request.
505	Delivery protocol not supported	The dCDN does not support the (set of) delivery protocols indicated in the CDNI Metadata of the content requested content by the UA.
506	Redirection protocol not supported	The dCDN does not support the requested redirection protocol. This error-code is also used when the RI request has the dns-only flag set to True and the dCDN is not support or is not prepared to return a RT of a surrogate directly.

Table 1

The following is an example of an unsuccessful RI response (dCDN->uCDN) for a DNS based User Agent request:

```
HTTP/1.1 500 Internal Server Error
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: private, no-cache
```

```
{
  "error" : {
    "error-code" : 504,
    "description" : "Out of capacity"
  }
}
```

The following is an example of a successful RI response (dCDN->uCDN) for a HTTP based User Agent request containing an error dictionary for informational purposes:

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: private, no-cache
```

```
{
  "http": {
    "sc-status": 302,
    "sc-version": "HTTP/1.1",
    "sc-reason": "Found",
    "cs-uri": "http://www.example.com"
    "sc-(location)":
      "http://surl.dcdn.example/ucdn/example.com",
  },
  "error" : {
    "error-code" : 100,
    "description" :
      "This is a human-readable message meant for debugging purposes"
  }
}
```

4.8. 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 MUST check the cdn-path and reject any RI requests which already contain the dCDN's Provider ID in the cdn-path. Transit CDNs MUST NOT propagate to any downstream CDNs if the number of CDN Provider IDs in cdn-path (before adding its 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 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, "AS64496:0".

If a dCDN receives an RI request whose cdn-path already contains that dCDN's Provider ID the dCDN MUST send an RI error response which SHOULD include an error code of 502.

If a dCDN receives an RI request where the number of CDN Provider IDs in cdn-path is greater than max-hops, the dCDN MUST send an RI error response which SHOULD include an error code of 503.

It should be noted that the loop detection & prevention mechanisms described above only cover preventing and detecting loops within the RI itself. Besides loops within the RI itself, there is also the possibility of loops in the data plane, for example, if the IP address(es) or URI(s) returned in RI responses do not resolve directly to a surrogate in the final dCDN there is the possibility that a User Agent may be continuously redirected through a loop of CDNs. The specification of solutions to address data plane request redirection loops between CDNs is outside of the scope of this document.

5. Security Considerations

Information passed over the RI could be considered personal or sensitive, for example, RI requests contain parts of a User Agent's original request and RI responses reveal information about the dCDN's policy for which surrogates should serve which content/user locations.

The RI interface also provides a mechanism whereby a uCDN could probe a dCDN and infer the dCDN's edge topology by making repeated RI requests for different content and/or UA IP addresses and correlating the responses from the dCDN. Additionally the ability for a dCDN to indicate that an RI response applies more widely than the original request (via the scope dictionary) may significantly reduce the number of RI requests required to probe and infer the dCDN's edge topology.

The same information could be obtained in the absence of the RI interface, but it could be more difficult to gather as it would require a distributed set of machines with a range of different IP addresses each making requests directly to the dCDN. However, the RI facilitates easier collection of such information as it enables a

single client to query the dCDN for a redirection/surrogate selection on behalf of any UA IP address.

5.1. Authentication, Authorization, Confidentiality, Integrity Protection

An implementation of the CDNI Redirection interface MUST support TLS transport as per [RFC2818] and [RFC7230]. The use of TLS for transport of the CDNI Redirection 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 Redirection messages to/from an authorized CDN);
- o CDNI Redirection interface messages to be transmitted with confidentiality; and
- o The integrity of the CDNI Redirection interface messages to be protected during the exchange.

In an environment where any such protection is required, mutually authenticated encrypted transport MUST be used to ensure confidentiality of the redirection information, and to do so, TLS MUST be used (including authentication of the remote end) by the server-side (dCDN) and the client-side (uCDN) of the CDNI Redirection interface.

When TLS is used, the general TLS usage guidance in [RFC7525] MUST be followed.

5.2. Privacy

Information passed over the RI ought to be considered personal and sensitive. In particular, parts of a User Agent's original request, most notably the UA's IP address and requested URI, are transmitted over the RI to the dCDN. The use of mutually authenticated TLS, as described in the previous section, prevents any other party than the authorized dCDN from gaining access to this information.

Regardless of whether the uCDN and dCDN use the RI, a successful redirect from a uCDN to a dCDN will make that dCDN aware of the UA's IP address. As such, the fact that this information is transmitted across the RI does not allow the dCDN to learn new information. On the other hand, if a uCDN uses the RI to check with multiple

candidate dCDNs, those candidates that do not end up getting redirected to, do obtain information regarding End User IP addresses and requested URIs that they would not have, had the RI not been used.

While it is technically possible to mask some information in the RI Request, such as the last bits of the UA IP address, it is important to note that this will reduce the effectiveness of the RI in certain cases. CDN deployments need to strike a balance between end-user privacy and the features impacted by such masking. This balance is likely to vary from one deployment to another. As an example, when the UA and its DNS resolver is behind a Carrier-grade NAT, and the RI is used to find an appropriate delivery node behind the same NAT, the full IP address might be necessary. Another potential issue when using IP anonymization is that it is no longer possible to correlate an RI Request with a subsequent UA request.

6. IANA Considerations

6.1. CDNI Payload Type Parameter registrations

The IANA is requested to register the following two new Payload Types in the CDNI Payload Type Parameter registry for use with the application/cdni MIME media type.

[RFC Editor Note: Please replace the references to [RFCthis] below with this document's RFC number before publication.]

Payload Type	Specification
redirection-request	[RFCthis]
redirection-response	[RFCthis]

6.1.1. CDNI RI Redirection Request Payload Type

Purpose: The purpose of this payload type is to distinguish RI request messages.

Interface: RI

Encoding: see Section 4.4.1 and Section 4.5.1

6.1.2. CDNI RI Redirection Response Payload Type

Purpose: The purpose of this payload type is to distinguish RI response messages.

Interface: RI

Encoding: see Section 4.4.2 and Section 4.5.2

6.2. RI Error response registry

IANA is requested to create a new "CDNI RI Error response code" subregistry within the "Content Delivery Network Interconnection (CDNI) Parameters" registry. The "CDNI RI Error response code" namespace defines the valid values for the error-code key in RI error responses. The CDNI RI Error response code MUST be a three digit integer.

Additions to the "RI Error response registry" will be made via "Specification Required" as defined in [RFC5226].

The Designated Expert will verify that new error code registrations do not duplicate existing error code definitions (in name or functionality), ensure that the new error code is in accordance with the error classes defined in section Section 4.7 of this document, prevent gratuitous additions to the namespace, and prevent any additions to the namespace that would impair the interoperability of CDNI implementations.

New registrations are required to provide the following information:

Code: A three-digit numeric error-code, in accordance with the error classes defined in section Section 4.7 of this document.

Reason: A string that provides further information related to the error that will be included in the JSON error dictionary with the 'reason'-key. Depending on the error-code semantics, the value of this field may be determined dynamically. In that case, the registration should set this value to '<reason>' and define its semantics in the description field.

Description: A brief description of the error code semantics.

Specification: Reference to the specification that defines the error code in more detail.

The entries in Table 1 are registered by this document, with the value of the 'Specification' field set to [RFCThis].

7. Contributors

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

The following persons have participated as co-authors to this document:

Wang Danhua, Huawei, Email: wangdanhua@huawei.com

He Xiaoyan, Huawei, Email: hexiaoyan@huawei.com

Ge Chen, China Telecom, Email: cheng@gsta.com

Ni Wei, China Mobile, Email: niwei@chinamobile.com

Yunfei Zhang, Email: hishigh@gmail.com

Spencer Dawkins, Huawei, Email: spencer@wonderhamster.org

8. Acknowledgements

The authors would like to thank Taesang Choi, Francois le Faucheur, Matt Miller, Scott Wainner and Kevin J Ma 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, 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>>.
- [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>>.

- [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>>.
- [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>>.
- [RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895, April 2013, <<http://www.rfc-editor.org/info/rfc6895>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.
- [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>>.
- [RTMP] Adobe Systems Incorporated, "Real-Time Messaging Protocol (RTMP) specification", December 2012, <http://www.adobe.com/go/spec_rtmp>.

9.2. Informative References

- [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>>.
- [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>>.
- [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 (editor)
Nokia
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: ben.niven-jenkins@nokia.com

Ray van Brandenburg (editor)
TNO
Anna van Buerenplein 1
The Hague 2595DA
the Netherlands

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

CDNI
Internet-Draft
Intended status: Standards Track
Expires: March 20, 2015

K. Leung
F. Le Faucheur
Cisco Systems
R. van Brandenburg
TNO
B. Downey
Verizon Labs
M. Fisher
Limelight Networks
September 16, 2014

URI Signing for CDN Interconnection (CDNI)
draft-ietf-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 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 March 20, 2015.

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	4
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
2.5. User Agent Attributes	14
3. Creating the Signed URI	14
3.1. Calculating the URI Signature	15
3.2. Packaging the URI Signature	18
4. Validating a URI Signature	18
4.1. Information Element Extraction	19
4.2. Signature Validation	20
4.3. Distribution Policy Enforcement	22
5. Relationship with CDNI Interfaces	22
5.1. CDNI Control Interface	22
5.2. CDNI Footprint & Capabilities Advertisement Interface . .	23
5.3. CDNI Request Routing Redirection Interface	23
5.4. CDNI Metadata Interface	24
5.5. CDNI Logging Interface	27
6. URI Signing Message Flow	28
6.1. HTTP Redirection	28
6.2. DNS Redirection	30
7. HTTP Adaptive Streaming	33
8. IANA Considerations	34
9. Security Considerations	35
10. Privacy	36

11. Acknowledgements	36
12. References	36
12.1. Normative References	36
12.2. Informative References	37
Authors' Addresses	38

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-SHA256: HMAC instantiation using SHA-256 as the cryptographic hash function.
- o SHA-1: Secure Hash Algorithm 1 (SHA-1) [RFC3174] is 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).

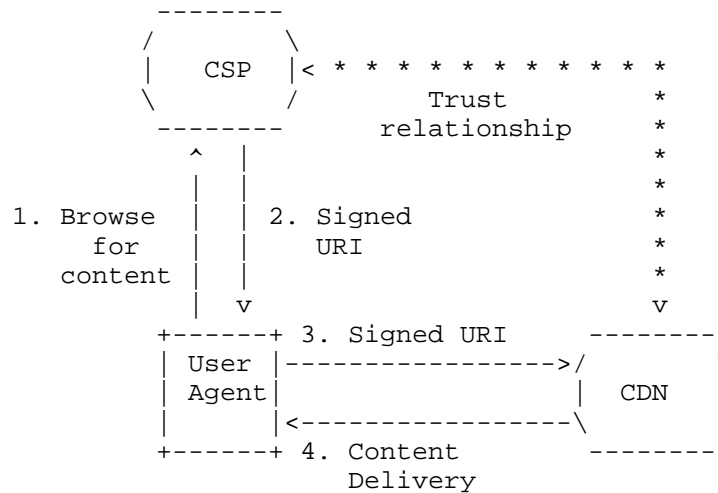


Figure 1: Figure 1: URI Signing in a CDN Environment

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).

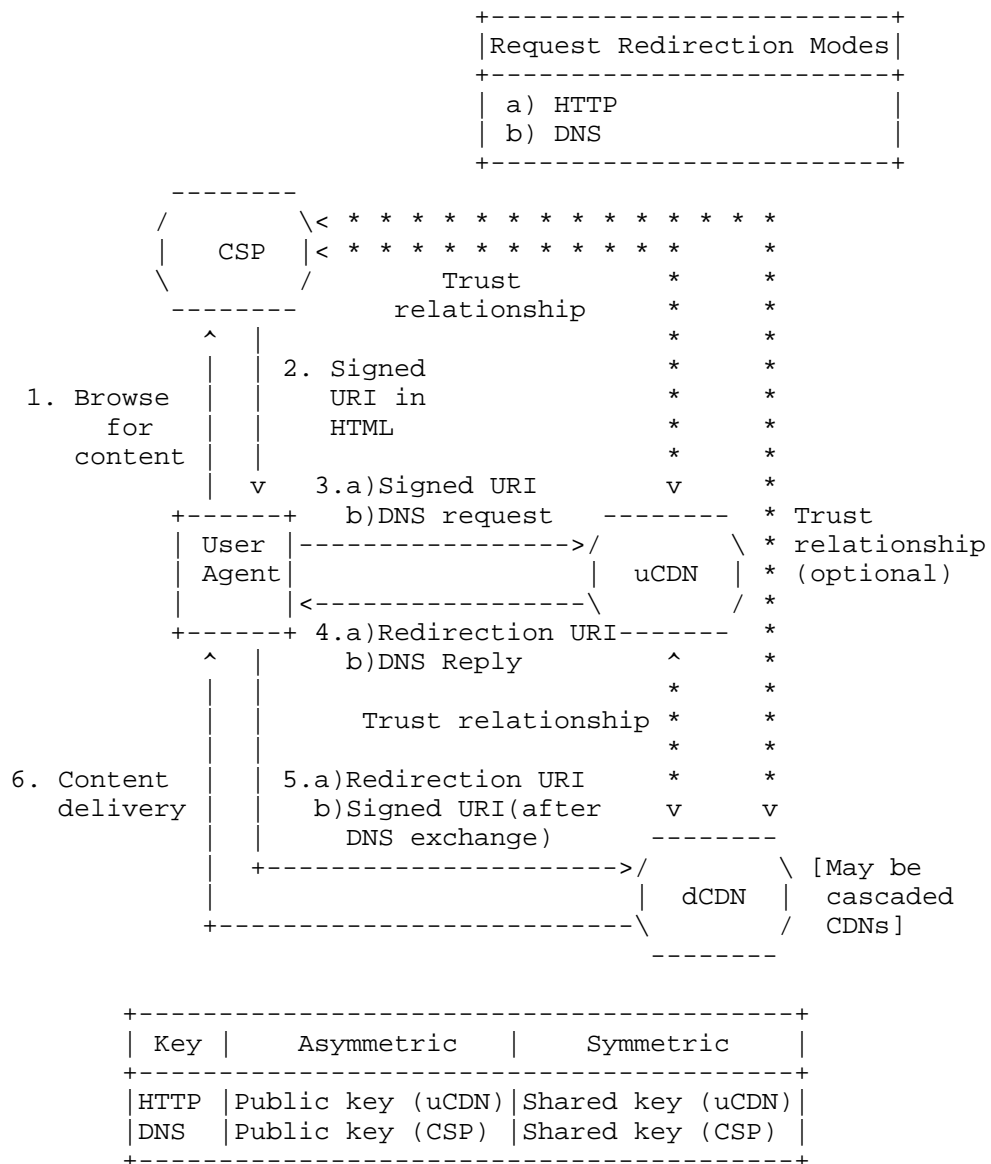


Figure 2: URI Signing in a CDNI Environment

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.

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.

Note that all the URI Signing information elements and the URI query attribute are mandatory to implement, but not mandatory to use.

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 with HMAC. If this Information Element is not present in the URI Signing Package Attribute, the default hash function is SHA-256.
- o Digital Signature Algorithm (DSA) [optional] - Algorithm used to calculate the Digital Signature. 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 of 1. 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. The default HMAC algorithm used is SHA-256.

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)).

2.5. User Agent Attributes

For some use cases, such as logging, it might be useful to allow the UA, or another entity, add one or more attributes to the Signed URI for purposes other than URI Signing without causing URI Signing to fail. In order to do so, such attributes **MUST** be appended after the URI Signing Package Attribute. Any attributes appended in such way after the URI Signature has been calculated are not validated for the purpose of content access authorization. Adding any such attributes to the Signed URI before the URI Signing Package Attribute will cause the URI Signing validation to fail.

Note that a malicious UA might potentially use the ability to append attributes to the Signed URI in order to try to influence the content that is delivered. For example, the UA might append '&quality=HD' to try to make the dCDN deliver an HD version of the requested content. Since such an additional attribute is appended after the URI Signing Package Attribute it is not validated and will not affect the outcome of the URI validation. In order to deal with this vulnerability, a dCDN is **RECOMMENDED** to ignore any query strings appended after the URI Signing Package Attribute for the purpose of content selection.

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: first, calculating the URI Signature and then, 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 (i.e. "1"), skip this step. Otherwise, specify the version by appending the string "VER=#", where '#' represents the new version number. The following steps in the procedure is based on the initial version of URI Signing specified by this document. For other versions, reference the associated RFC for the URI signing procedure.
4. If time window enforcement is not needed, step 4 can be skipped.
 - A. If an information element was added to the message, append an "&" character. Append the string "ET=". Note in the case of re-signing a URI, the information element 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 information element was added to the message, 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 not needed, skip this step. If an information element was added to the message, 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. Optional: If the hash function for the HMAC uses the default value ("SHA-256"), skip this step. If an information element was added to the message, append an "&" character. append the string "HF=". Append the string for the new type of hash function to be used. 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 information element was added to the message, 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=192.0.2.1&KID=example:keys:123&MD=").
5. Compute the message digest using the HMAC algorithm and the default SHA-256 hash function, or another hash function if specified by the HF Information Element, with the shared key and message as the two inputs to the hash function.
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=192.0.2.1&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bddaf").

B. For asymmetric keys, EC DSA is used.

1. Generate the EC private and public key pair. 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 not needed, skip this step. If an information element was added to the message, 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. Optional: If the digital signature algorithm uses the default value ("EC-DSA"), skip this step. If an information element was added to the message, append an "&" character. Append the string "DSA=". Append the string denoting the new digital signature function.
4. If an information element was added to the message, 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=192.0.2.1&KID=http://example.com/public/keys/123&DS=").
5. Compute the message digest using SHA-1 (without a key) for the message. Note: The digital signature generated in the next step is calculated over the SHA-1 message digest, instead of over the cleartype message, to reduce the length of the digital signature, and thereby the length of the URI Signing Package Attribute and the resulting Signed URI. Since SHA-1 is not used for cryptographic purposes here, the security concerns around SHA-1 do not apply.
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=192.0.2.1&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=192.0.2.1&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bddaf").
2. Compute the URI Signing Package Attribute using Base-64 Data Encoding [RFC4648] on the message (e.g. "VkVSPTEmRVQ9MTIwOTQyMjk3NiZDSVA9MTkyLjAuMi4xJktJRD1leGFtcGxlOmtleXM6MTIzJk1EPTFlY2IxNDQ2YTY0MzEzNTJhYWwZmI2ZTBkY2EzMGUzMMDM1NjU5M2E5N2FjYjk3MjIwMjEyMGRjNDgyYmRkYWY="). 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=VkVSPTEmRVQ9MTIwOTQyMjk3NiZDSVA9MTkyLjAuMi4xJktJRD1leGFtcGxlOmtleXM6MTIzJk1EPTFlY2IxNDQ2YTY0MzEzNTJhYWwZmI2ZTBkY2EzMGUzMMDM1NjU5M2E5N2FjYjk3MjIwMjEyMGRjNDgyYmRkYWY="). Note: this is the completed Signed URI.

4. Validating a URI Signature

The process of validating a Signed URI can be divided into three sets of steps: first, extraction of the URI Signing information elements, then validation of the URI signature to ensure the integrity of the Signed URI, and finally, validation of the information elements to ensure proper enforcement of the distribution policy. The integrity of the Signed URI is confirmed before distribution policy enforcement because validation procedure would detect the right event when the URI is tampered with. Note it is possible to use some other algorithm and implementation as long as the same result is achieved.

4.1. Information Element Extraction

Extract the information elements embedded in the URI. Note that some steps are to be skipped if the corresponding URI Signing information elements are not embedded in the Signed URI.

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] to obtain all the URI Signing information elements (e.g. "ET=1209422976&CIP=192.0.2.1&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bddaf").
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 information element 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 "MD" if the information element exists in the query string. The existence of this information element indicates a symmetric key is used.
5. Extract the value from "DS" if the information element exists in the query string. The existence of this information element indicates an asymmetric key is used.
6. 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.
7. Extract the value from "CIP" if the information element exists in the query string. The existence of this information element indicates content delivery is enforced based on client IP address.

8. Extract the value from "ET" if the information element exists in the query string. The existence of this information element indicates content delivery is enforced based on time.
9. Extract the value from the "KID" information element, if it exists. The existence of this information element indicates a key can be referenced.
10. Extract the value from the "HF" information element, if it exists. The existence of this information element indicates a different hash function than the default.
11. Extract the value from the "DSA" information element, if it exists. The existence of this information element indicates a different digital signature algorithm than the default.

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. If "KID" information element exists, validate that the key identifier is in the allowable KID set as listed in the CDNI metadata or configuration. The request is denied when the key identifier is not allowed. Otherwise, the "KID" information element is not in the Signed URI. In this case, obtain the shared key via CDNI metadata or configuration.
 - b. If "HF" information element exists, validate that the hash function is in the allowable "HF" set as listed in the CDNI metadata or configuration. The request is denied when the hash function is not allowed. Otherwise,

the "HF" information element is not in the Signed URI.
In this case, the default hash function is SHA-256.

- 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=192.
0.2.1&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.
 - 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. If "KID" information element exists, validate that the
key identifier is in the allowable KID set as listed in
the CDNI metadata or configuration. The request is
denied when the key identifier is not allowed.
Otherwise, the "KID" information element is not in the
Signed URI. In this case, obtain the public key via CDNI
metadata or configuration.
 - b. If "DSA" information element exists, validate that the
digital signature algorithm is in the allowable "DSA" set
as listed in the CDNI metadata or configuration. The
request is denied when the DSA is not allowed.
Otherwise, the "DSA" information element is not in the
Signed URI. In this case, the default DSA is EC-DSA.
 - 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=192.0.2.1&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.

4.3. Distribution Policy Enforcement

Note the steps are to be skipped if the corresponding URI Signing information elements are not 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. If the "CIP" information element exists, validate that the request came from the same IP address as indicated in the "CIP" information element. If the IP address is incorrect, then the request is denied.
2. If the "ET" information element exists, validate that the request arrived before expiration time based on the "ET" information element. If the time expired, then the request is denied.

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.

In general, new information elements introduced to enhance URI Signing requires a draft and a new version. For Information Elements,

For Enforcement Information Elements, there is no need to advertise the based information elements such as "CIP" and "ET".

For Signature Computation Information Elements:

No need to advertise "VER" Information Element unless it's not "1". In this case, a draft is needed to describe the new version.

Advertise value of the "HF" Information Element (i.e. SHA-256) to indicate support for the hash function; Need IANA assignment for new hash function.

Advertise value of the "DSA" Information Element (i.e. EC-DSA) to indicate support for the DSA; Need IANA assignment for new digital signature algorithm.

Advertise "MD" Information Element (i.e. EC-DSA) to indicate support for symmetric key method; A new draft is needed for an alternative method.

Advertise "DS" Information Element (i.e. EC-DSA) to indicate support for asymmetric key method; A new draft is needed for an alternative method.

For URI Signing Package Attribute, there is no need to advertise the base attribute.

5.3. CDNI Request Routing Redirection Interface

The CDNI Request Routing Redirection Interface [I-D.ietf-cdni-redirection] describes the recursive request redirection method. For URI Signing, the Upstream CDN signs the URI provided by the Downstream CDN. This approach has the following benefits:

Consistency with interative request routing method

URI Signing is fully operational 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 CDNI Metadata Interface [I-D.ietf-cdni-metadata] describes the CDNI metadata distribution in order to enable content acquisition and delivery. For URI Signing, additional CDNI metadata objects are specified. In general, an Empty set means "all". These are the CDNI metadata objects used for URI Signing.

The UriSigning Metadata object contains information to enable URI signing and validation by a dCDN. The UriSigning properties are defined below.

Property: enforce

Description: 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.

Type: Boolean

Mandatory-to-Specify: No. If a UriSigning object is present in the metadata for a piece of content (even if the object is empty), then URI signing should be enforced. If no UriSigning object is present in the metadata for a piece of content, then the URI signature should not be validated.

Property: key-id

Description: Designated key identifier used for URI Signing computation when the Signed URI does not contain the Key ID information element.

Type: String

Mandatory-to-Specify: No. A Key ID is not essential for all implementations of URI signing.

Property: key-id-set

Description: Allowable Key ID set that the Signed URI's Key ID information element can reference.

Type: List of Strings

Mandatory-to-Specify: No. Default is to allow any Key ID.
[Editor's note: security issue with default to allow any? Easy for bad guy to put their own key ID in URI. The default should be key ID must be in the set or configuration?]

Property: hash-function

Description: Designated hash function used for URI Signing computation when the Signed URI does not contain the Hash Function information element.

Type: String (limited to the hash function strings in the registry defined by the IANA Considerations (Section 8) section)

Mandatory-to-Specify: No. Default is SHA-256.

Property: hash-function-set

Description: Allowable Hash Function set that the Signed URI's Hash Function information element can reference.

Type: List of Strings

Mandatory-to-Specify: No. Default is to allow any hash function.

Property: digital-signature-algorithm

Description: Designated digital signature function used for URI Signing computation when the Signed URI does not contain the Digital Signature Algorithm information element.

Type: String (limited to the digital signature algorithm strings in the registry defined by the IANA Considerations (Section 8) section).

Mandatory-to-Specify: No. Default is EC-DSA.

Property: digital-signature-algorithm-set

Description: Allowable digital signature function set that the Signed URI's Digital Signature Algorithm information element can reference.

Type: List of Strings

Mandatory-to-Specify: No. Default is to allow any DSA.

Property: version

Description: Designated version used for URI Signing computation when the Signed URI does not contain the VER attribute.

Type: Integer

Mandatory-to-Specify: No. Default is 1.

Property: version-set

Description: Allowable version set that the Signed URI's VER attribute can reference.

Type: List of Integers

Mandatory-to-Specify: No. Default is to allow any version.

Property: package-attribute

Description: Overwrite the default name for the URL Signing Package Attribute.

Type: String

Mandatory-to-Specify: No. Default is "URISigningPackage".

Note that the Key ID information element 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 or configuration MUST provide the allowable Key ID set to authorize the Key ID information element embedded in the Signed URI.

5.5. CDNI Logging Interface

For URI Signing, the Downstream CDN reports that enforcement of the access control was applied to the request for content delivery. When the request is denied due to enforcement of URI Signing, the reason is logged.

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 (mandatory):
 - * format: 3DIGIT
 - * 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.
- o s-uri-signing-deney-reason (optional):
 - * format: QSTRING
 - * field value: the rejection reason when uri signature performed by the Surrogate on the request. Examples:
 - + "invalid client IP address"
 - + "expired signed URI"
 - + "incorrect URI signature"
 - * occurrence: there MUST be zero or exactly one instance of this field.

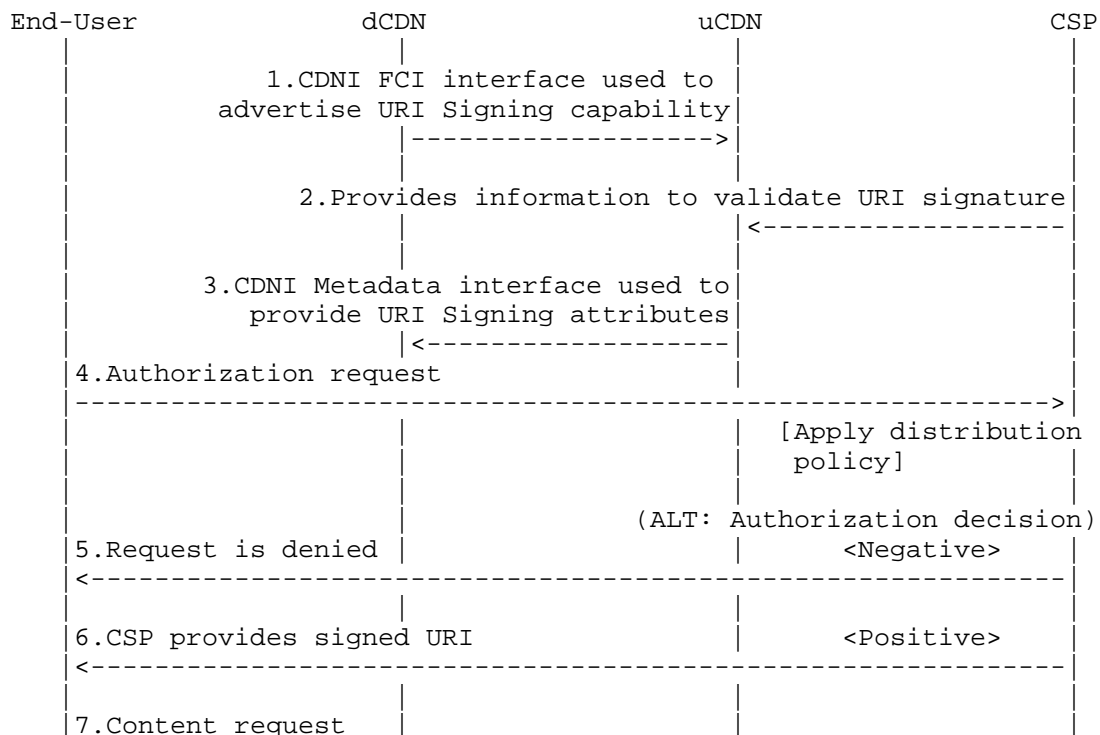
6. URI Signing Message Flow

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.



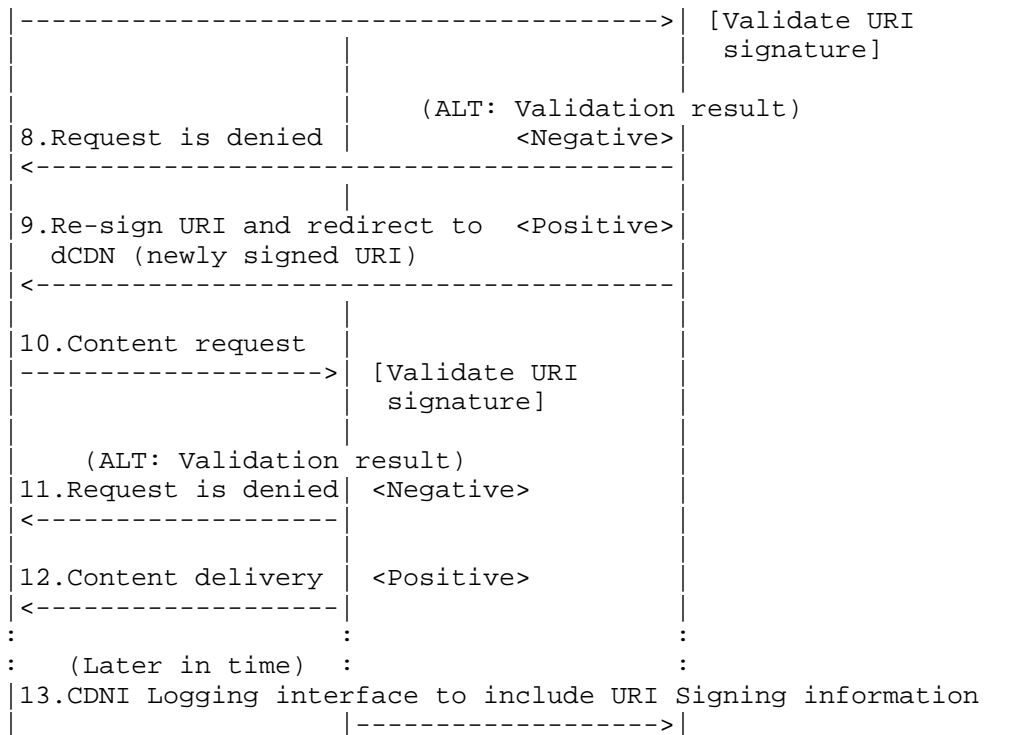


Figure 3: HTTP-based Request Routing with URI Signing

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.

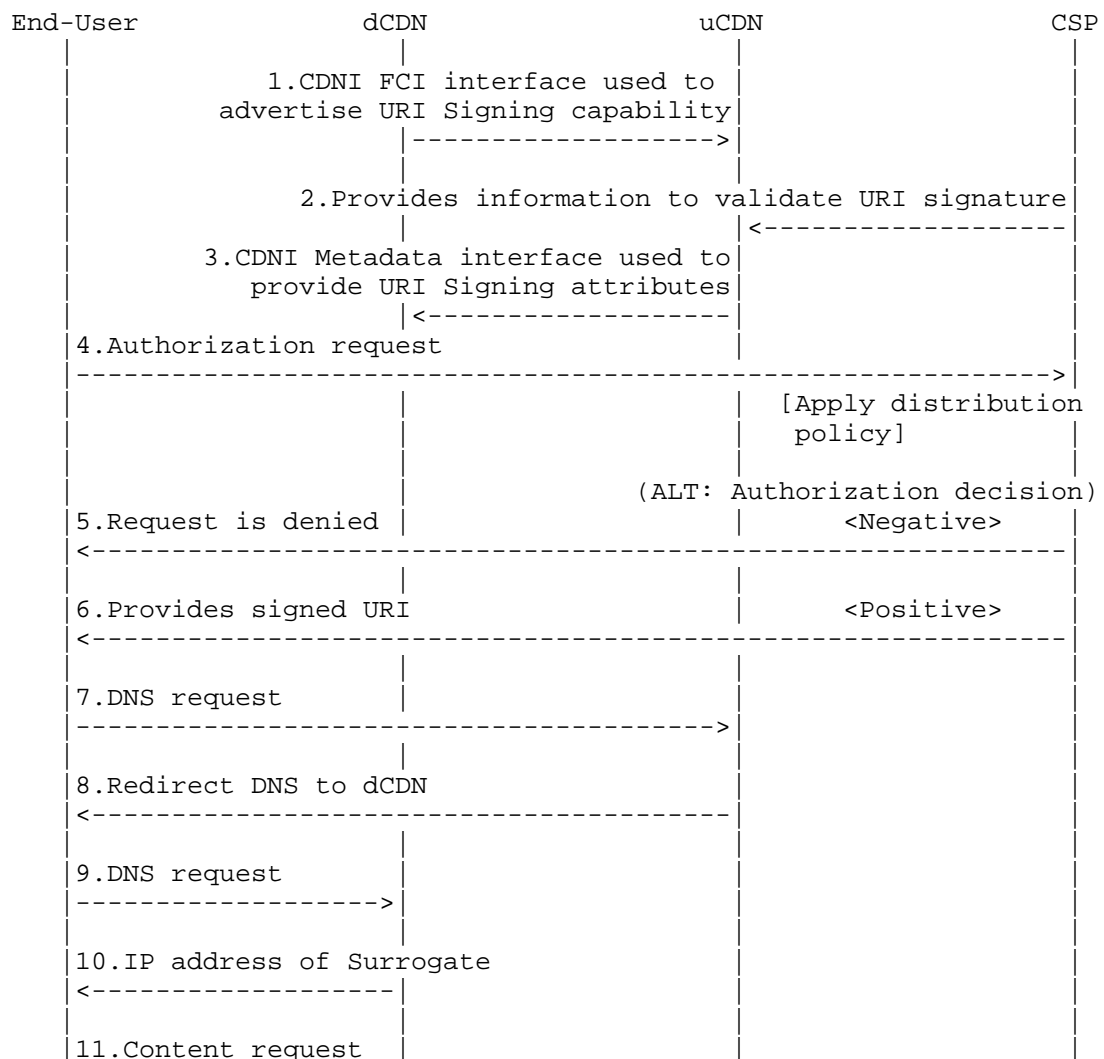
With HTTP-based request routing, URI Signing matches well the general chain of trust model of CDNI both with symmetric key and asymmetric keys because the key information only need to be specific to a pair of adjacent CDNI hops.

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.



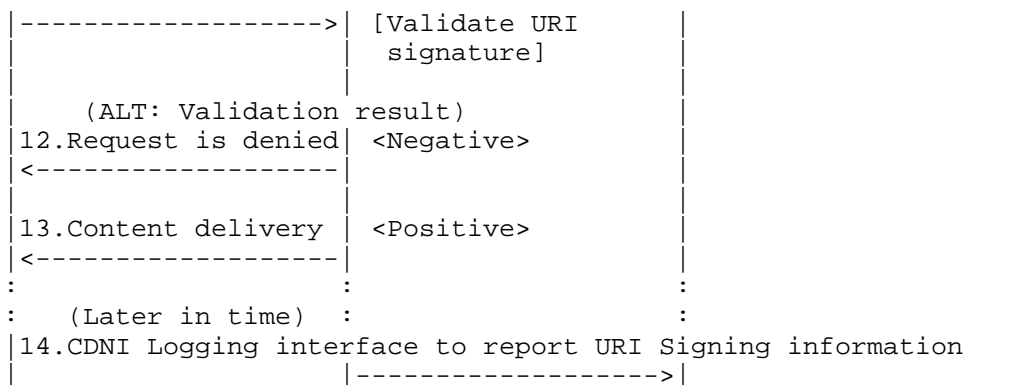


Figure 4: DNS-based Request Routing with URI Signing

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]. [Editor note: DASH draft discussion]

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 URI Signing 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): "SHA-256"
- o DSA (Digital Signature Algorithm): "EC-DSA"

The following URI Signature Information Element names are allocated:

- o MD (Message Digest for Symmetric Key)
- o DS (Digital Signature for Asymmetric Keys)

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 s-url-signing-deny-reason

The IANA is requested to allocate a new entry to the "CDNI GenericMetadata Types" Registry as specified in CDNI Metadata

Interface [I-D.ietf-cdni-metadata] in accordance to the "Specification Required" policy [RFC5226]:

Type name	Specification	Version	MTE	STR
UriSigning	RFCthis	1	true	true

The IANA is also requested to allocate a new MIME type under the IANA MIME Media Type registry for the UriSigning metadata object:

application/cdni.UriSigning.v1

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.

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.

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 value of 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: Scott Leibrand, Kevin Ma, Ben Niven-Jenkins, Thierry Magnien, Dan York, Bhaskar Bhupalam, Matt Caulfield, Samuel Rajakumar and Iuniana Oprescu. In addition, Matt Caulfield provided content for the CDNI Metadata Interface section.

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-13 (work in progress), July 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-14 (work in progress), June 2014.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-07 (work in progress), July 2014.
- [I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection Interface for CDN Interconnection", draft-ietf-cdni-redirection-03 (work in progress), August 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.
- [RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.
- [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.
- [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

Ray van Brandenburg
TNO
Brassersplein 2
Delft 2612CT
the Netherlands

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

Bill Downey
Verizon Labs
60 Sylvan Road
Waltham, Massachusetts 02451
USA

Phone: +1 781 466 2475
Email: william.s.downey@verizon.com

Michel Fisher
Limelight Networks
222 S Mill Ave
Tempe, AZ 85281
USA

Phone: +1 360 419 5185
Email: mfisher@llnw.com

CDNI
Internet-Draft
Intended status: Standards Track
Expires: 23 September 2022

R. van Brandenburg
Tiledmedia
K. Leung

P. Sorber
Apple, Inc.
22 March 2022

URI Signing for Content Delivery Network Interconnection (CDNI)
draft-ietf-cdni-uri-signing-26

Abstract

This document describes how the concept of URI Signing supports the content access control requirements of Content Delivery Network Interconnection (CDNI) and proposes a URI Signing method as a JSON Web Token (JWT) profile.

The proposed URI Signing method specifies the information needed to be included in the URI to transmit the signed JWT, as well as the claims needed by the signed JWT to authorize a User Agent (UA). The mechanism described can be used both in CDNI and single Content Delivery Network (CDN) scenarios.

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 <https://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 23 September 2022.

Copyright Notice

Copyright (c) 2022 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 (<https://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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1.	Introduction	3
1.1.	Terminology	4
1.2.	Background and overview on URI Signing	5
1.3.	CDNI URI Signing Overview	6
1.4.	URI Signing in a non-CDNI context	9
2.	JWT Format and Processing Requirements	9
2.1.	JWT Claims	10
2.1.1.	Issuer (iss) claim	10
2.1.2.	Subject (sub) claim	11
2.1.3.	Audience (aud) claim	11
2.1.4.	Expiry Time (exp) claim	11
2.1.5.	Not Before (nbf) claim	12
2.1.6.	Issued At (iat) claim	12
2.1.7.	JWT ID (jti) claim	12
2.1.8.	CDNI Claim Set Version (cdniv) claim	13
2.1.9.	CDNI Critical Claims Set (cdnicrit) claim	13
2.1.10.	Client IP Address (cdniip) claim	13
2.1.11.	CDNI URI Container (cdniuc) claim	14
2.1.12.	CDNI Expiration Time Setting (cdniets) claim	14
2.1.13.	CDNI Signed Token Transport (cdnistt) claim	14
2.1.14.	CDNI Signed Token Depth (cdnistd) claim	15
2.1.15.	URI Container Forms	15
2.1.15.1.	URI Hash Container (hash:)	16
2.1.15.2.	URI Regular Expression Container (regex:)	16
2.2.	JWT Header	16
3.	URI Signing Token Renewal	17
3.1.	Overview	17
3.2.	Signed Token Renewal mechanism	17
3.2.1.	Required Claims	18
3.3.	Communicating a signed JWTs in Signed Token Renewal	18
3.3.1.	Support for cross-domain redirection	18
4.	Relationship with CDNI Interfaces	19
4.1.	CDNI Control Interface	19
4.2.	CDNI Footprint & Capabilities Advertisement Interface	19
4.3.	CDNI Request Routing Redirection Interface	19
4.4.	CDNI Metadata Interface	19
4.5.	CDNI Logging Interface	21

5. URI Signing Message Flow	22
5.1. HTTP Redirection	22
5.2. DNS Redirection	25
6. IANA Considerations	28
6.1. CDNI Payload Type	28
6.1.1. CDNI UriSigning Payload Type	28
6.2. CDNI Logging Record Type	28
6.2.1. CDNI Logging Record Version 2 for HTTP	29
6.3. CDNI Logging Field Names	29
6.4. CDNI URI Signing Verification Code	30
6.5. CDNI URI Signing Signed Token Transport	31
6.6. JSON Web Token Claims Registration	32
6.6.1. Registry Contents	32
6.7. Expert Review Guidance	33
7. Security Considerations	33
8. Privacy	35
9. Acknowledgements	35
10. Contributors	35
11. References	35
11.1. Normative References	35
11.2. Informative References	37
Appendix A. Signed URI Package Example	39
A.1. Simple Example	40
A.2. Complex Example	40
A.3. Signed Token Renewal Example	41
Authors' Addresses	42

1. Introduction

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of redirection between cooperating CDNs and between a Content Service Provider (CSP) and a CDN. The primary goal of URI Signing is to make sure that only authorized UAs are able to access the content, with a 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 Digital Rights Management (DRM), are more appropriate. In addition to access control, URI Signing also has benefits in reducing the impact of denial-of-service attacks.

The overall problem space for CDN Interconnection (CDNI) is described in CDNI Problem Statement [RFC6707]. This document, along with the CDNI Requirements [RFC7337] document and the CDNI Framework [RFC7336], describes the need for interconnected CDNs to be able to implement an access control mechanism that enforces a CSP's distribution policies.

Specifically, the CDNI Framework [RFC7336] states:

The CSP may also trust the CDN operator to perform actions such as delegating traffic to additional downstream CDNs, and to enforce per-request authorization performed by the CSP using techniques such as URI Signing.

In particular, the following requirement is listed in the CDNI Requirements [RFC7337]:

MI-16 {HIGH} The CDNI Metadata interface shall allow signaling of authorization checks and verification that are to be performed by the Surrogate before delivery. For example, this could potentially include the need to verify information (e.g., Expiry time, Client IP address) required for access authorization.

This document defines a method of signing URIs that allows Surrogates in interconnected CDNs to enforce a per-request authorization initiated by the CSP. Splitting the role of initiating per-request authorization by the CSP and the role of verifying 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 specific CSP distribution policies.

The method is implemented using Signed JSON Web Tokens (JWTs) [RFC7519].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the terminology defined in the CDNI Problem Statement [RFC6707].

This document also uses the terminology of the JSON Web Token (JWT) [RFC7519].

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

- * Signed URI: A URI for which a signed JWT is provided.

- * Target CDN URI: URI created by the CSP to direct a UA towards the Upstream CDN (uCDN). The Target CDN URI can be signed by the CSP and verified by the uCDN and possibly further Downstream CDNs (dCDNs).
- * Redirection URI: URI created by the uCDN to redirect a UA towards the dCDN. The Redirection URI can be signed by the uCDN and verified by the dCDN. In a cascaded CDNI scenario, there can be more than one Redirection URI.
- * Signed Token Renewal: A series of signed JWTs that are used for subsequent access to a set of related resources in a CDN, such as a set of HTTP Adaptive Streaming files. Every time a signed JWT is used to access a particular resource, a new signed JWT is sent along with the resource that can be used to request the next resource in the set. When generating a new signed JWT in Signed Token Renewal, parameters are carried over from one signed JWT to the next.

1.2. Background and overview on URI Signing

A CSP and CDN are assumed to have a trust relationship that enables the CSP to authorize access to a content item, which is realized in practice by including a set of claims in a signed JWT 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 a time window or pattern matching the URI). To prevent the UA from altering the claims the JWT MUST be signed.

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. The signed URI always contains a signed JWT generated by the CSP using a shared secret or private key. Once the UA receives the response with the Signed URI, it sends a new HTTP request using the Signed URI to the CDN (#3). Upon receiving the request, the CDN authenticates the Signed URI by verifying the signed JWT. If applicable, the CDN checks whether the time window is still valid in the Signed URI and the pattern matches the URI of the request. After these claims are verified, the CDN delivers the content (#4).

Note: While using a symmetric shared key is supported, it is NOT RECOMMENDED. See the Security Considerations (Section 7) section about the limitations of shared keys.

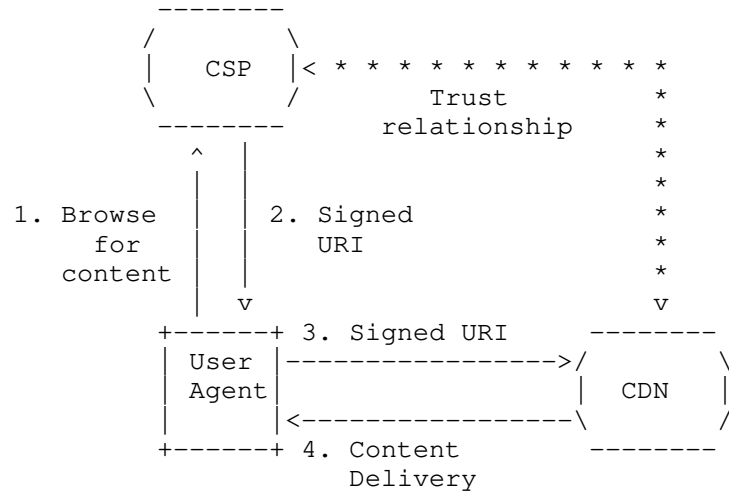


Figure 1: URI Signing in a CDN Environment

1.3. CDNI URI Signing Overview

In a CDNI environment, as shown in Figure 2 below, URI Signing operates the same way in the initial steps #1 and #2, but the later steps involve multiple CDNs delivering the content. The main difference from the single CDN case is a redirection step between the uCDN and the dCDN. In step #3, the UA may send an HTTP request or a DNS request, depending on whether HTTP-based or DNS-based request routing is used. The uCDN responds by directing the UA towards the dCDN using either a Redirection URI (i.e., a Signed URI generated by the uCDN) or a DNS reply, respectively (#4). Once the UA receives the response, it sends the Redirection URI/Target CDN URI to the dCDN (#5). The received URI is verified by the dCDN before delivering the content (#6). Note: The CDNI call flows are covered in Detailed URI Signing Operation (Section 5).

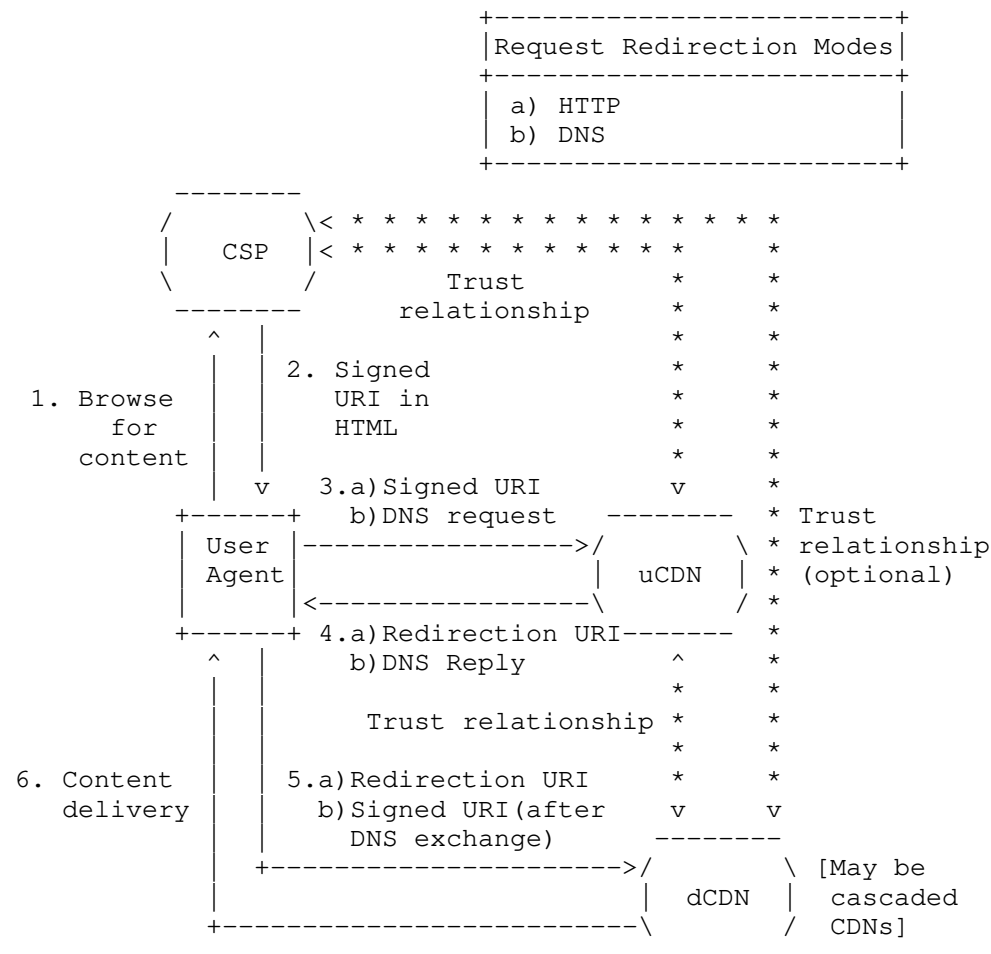


Figure 2: URI Signing in a CDNI Environment

The trust relationships between CSP, uCDN, and dCDN have direct implications for URI Signing. In the case shown in Figure 2, the CSP has a trust relationship with the uCDN. The delivery of the content may be delegated to a dCDN, which has a relationship with the uCDN 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., the Target CDN URI) provided by the CSP reaches the CDN directly. In the case where the dCDN does not have a trust relationship with the CSP, this means that either an asymmetric public/private key method needs to be used for computing the signed JWT (because the CSP and dCDN are not able to exchange symmetric shared secret keys). Shared keys MUST NOT be redistributed.

For HTTP-based request routing, the Signed URI (i.e., the Target CDN URI) provided by the CSP reaches the uCDN. After this URI has been verified by the uCDN, the uCDN creates and signs a new Redirection URI, redirecting the UA to the dCDN. Since this new URI can have a new signed JWT, the relationship between the dCDN and CSP is not relevant. Because a relationship between uCDN and dCDN always exists, either asymmetric public/private keys or symmetric shared secret keys can be used for URI Signing with HTTP-based request routing. Note that the signed Redirection URI MUST maintain HTTPS as the scheme if it was present in the original and it MAY be upgraded from http: to https:.

Two types of keys can be used for URI Signing: asymmetric keys and symmetric shared keys. Asymmetric keys are based on a public/private key pair mechanism and always contain a private key known only 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 and the verifying entity for verifying the Signed URI. Regardless of the type of keys used, the verifying entity has to obtain the key in a manner that allows trust to be placed in the assertions made using that key (either the public or the symmetric key). There are very different requirements (outside the scope of this document) for distributing asymmetric keys and symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent third parties from getting access to the key, since they 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 to sign URIs) and the corresponding private keys are kept secret by the URI signer.

Note: While using a symmetric shared key is supported, it is NOT RECOMMENDED. See the Security Considerations (Section 7) section about the limitations of shared keys.

1.4. URI Signing in a non-CDNI context

While the URI Signing method defined in this document was primarily created for the purpose of allowing URI Signing in CDNI scenarios, i.e., between a uCDN and a dCDN, there is nothing in the defined URI Signing method 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. JWT Format and Processing Requirements

The concept behind URI Signing is based on embedding a signed JSON Web Token (JWT) [RFC7519] in an HTTP or HTTPS URI [RFC7230] (see [RFC7230] Section 2.7). The signed JWT contains a number of claims that can be verified to ensure the UA has legitimate access to the content.

This document specifies the following attribute for embedding a signed JWT in a Target CDN URI or Redirection URI:

- * URI Signing Package (URISigningPackage): The URI attribute that encapsulates all the URI Signing claims in a signed JWT encoded format. This attribute is exposed in the Signed URI as a path-style parameter or a form-style parameter.

The parameter name of the URI Signing Package Attribute is defined in the CDNI Metadata (Section 4.4). If the CDNI Metadata interface is not used, or does not include a parameter name for the URI Signing Package Attribute, the parameter name can be set by configuration (out of scope of this document).

The URI Signing Package will be found by parsing any path-style parameters and form-style parameters looking for a key name matching the URI Signing Package Attribute. Both parameter styles MUST be supported to allow flexibility of operation. The first matching parameter SHOULD be taken to provide the signed JWT, though providing more than one matching key is undefined behavior. Path-style parameters generated in the form indicated by Section 3.2.7 of [RFC6570] and Form-style parameters generated in the form indicated by Sections 3.2.8 and 3.2.9 of [RFC6570] MUST be supported.

The following is an example where the URI Signing Package Attribute name is "token" and the signed JWT is "SIGNEDJWT":

`http://example.com/media/path?come=data&token=SIGNEDJWT&other=data`

2.1. JWT Claims

This section identifies the set of claims that can be used to enforce the CSP distribution policy. New claims can be introduced in the future to extend the distribution policy capabilities.

In order to provide distribution policy flexibility, the exact subset of claims used in a given signed JWT is a runtime decision. Claim requirements are defined in the CDNI Metadata (Section 4.4). If the CDNI Metadata interface is not used, or does not include claim requirements, the claim requirements can be set by configuration (out of scope of this document).

The following claims (where the "JSON Web Token Claims" registry claim name is specified in parentheses below) are used to enforce the distribution policies. All of the listed claims are mandatory to implement in a URI Signing implementation, but are not necessarily mandatory to use in a given signed JWT. (The "optional" and "mandatory" identifiers in square brackets refer to whether or not a given claim **MUST** be present in a URI Signing JWT.)

Note: The time on the entities that generate and verify the signed URI **MUST** be in sync. In the CDNI case, this means that CSP, uCDN, and dCDN servers need to be time-synchronized. It is **RECOMMENDED** to use NTP [RFC5905] for time synchronization.

Note: See the Security Considerations (Section 7) section on the limitations of using an expiration time and client IP address for distribution policy enforcement.

2.1.1. Issuer (iss) claim

Issuer (iss) [optional] - The semantics in [RFC7519] Section 4.1.1 **MUST** be followed. If this claim is used, it **MUST** be used to identify the issuer (signer) of the JWT. In particular, the recipient will have already received, in trusted configuration, a mapping of issuer name to one or more keys used to sign JWTs, and must verify that the JWT was signed by one of those keys. If this claim is used and the CDN verifying the signed JWT does not support Issuer verification, or if the Issuer in the signed JWT does not match the list of known acceptable Issuers, or if the Issuer claim does not match the key used to sign the JWT, the CDN **MUST** reject the request. If the received signed JWT contains an Issuer claim, then any JWT subsequently generated for CDNI redirection **MUST** also contain an Issuer claim, and the Issuer value **MUST** be updated to identify the redirecting CDN. If the received signed JWT does not contain an Issuer claim, an Issuer claim **MAY** be added to a signed JWT generated for CDNI redirection.

2.1.2. Subject (sub) claim

Subject (sub) [optional] - The semantics in [RFC7519] Section 4.1.2 MUST be followed. If this claim is used, it MUST be a JSON Web Encryption (JWE [RFC7516]) Object in compact serialization form, because it contains personally identifiable information. This claim contains information about the subject (for example, a user or an agent) that MAY be used to verify the signed JWT. If the received signed JWT contains a Subject claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Subject claim, and the Subject value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Subject claim if no Subject claim existed in the received signed JWT.

2.1.3. Audience (aud) claim

Audience (aud) [optional] - The semantics in [RFC7519] Section 4.1.3 MUST be followed. This claim is used to ensure that the CDN verifying the JWT is an intended recipient of the request. The claim MUST contain an identity belonging to the chain of entities involved in processing the request (e.g., identifying the CSP or any CDN in the chain) that the recipient is configured to use for the processing of this request. A CDN MAY modify the claim as long it can generate a valid signature.

2.1.4. Expiry Time (exp) claim

Expiry Time (exp) [optional] - The semantics in [RFC7519] Section 4.1.4 MUST be followed, though URI Signing implementations MUST NOT allow for any time synchronization "leeway". If this claim is used and the CDN verifying the signed JWT does not support Expiry Time verification, or if the Expiry Time in the signed JWT corresponds to a time equal to or earlier than the time of the content request, the CDN MUST reject the request. If the received signed JWT contains an Expiry Time claim, then any JWT subsequently generated for CDNI redirection MUST also contain an Expiry Time claim, and the Expiry Time value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add an Expiry Time claim if no Expiry Time claim existed in the received signed JWT.

2.1.5. Not Before (nbf) claim

Not Before (nbf) [optional] - The semantics in [RFC7519] Section 4.1.5 MUST be followed, though URI Signing implementations MUST NOT allow for any time synchronization "leeway". If this claim is used and the CDN verifying the signed JWT does not support Not Before time verification, or if the Not Before time in the signed JWT corresponds to a time later than the time of the content request, the CDN MUST reject the request. If the received signed JWT contains a Not Before time claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Not Before time claim, and the Not Before time value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Not Before time claim if no Not Before time claim existed in the received signed JWT.

2.1.6. Issued At (iat) claim

Issued At (iat) [optional] - The semantics in [RFC7519] Section 4.1.6 MUST be followed. If the received signed JWT contains an Issued At claim, then any JWT subsequently generated for CDNI redirection MUST also contain an Issued At claim, and the Issued At value MUST be updated to identify the time the new JWT was generated. If the received signed JWT does not contain an Issued At claim, an Issued At claim MAY be added to a signed JWT generated for CDNI redirection.

2.1.7. JWT ID (jti) claim

JWT ID (jti) [optional] - The semantics in [RFC7519] Section 4.1.7 MUST be followed. A JWT ID can be used to prevent replay attacks if the CDN stores a list of all previously used values, and verifies that the value in the current JWT has never been used before. If the signed JWT contains a JWT ID claim and the CDN verifying the signed JWT either does not support JWT ID storage or has previously seen the value used in a request for the same content, then the CDN MUST reject the request. If the received signed JWT contains a JWT ID claim, then any JWT subsequently generated for CDNI redirection MUST also contain a JWT ID claim, and the value MUST be the same as in the received signed JWT. If the received signed JWT does not contain a JWT ID claim, a JWT ID claim MUST NOT be added to a signed JWT generated for CDNI redirection. Sizing of the JWT ID is application dependent given the desired security constraints.

2.1.1.8. CDNI Claim Set Version (cdniv) claim

CDNI Claim Set Version (cdniv) [optional] - The CDNI Claim Set Version (cdniv) claim provides a means within a signed JWT to tie the claim set to a specific version of this specification. The cdniv claim is intended to allow changes in and facilitate upgrades across specifications. The type is JSON integer and the value MUST be set to "1", for this version of the specification. In the absence of this claim, the value is assumed to be "1". For future versions this claim will be mandatory. Implementations MUST reject signed JWTs with unsupported CDNI Claim Set versions.

2.1.1.9. CDNI Critical Claims Set (cdnicrit) claim

CDNI Critical Claims Set (cdnicrit) [optional] - The CDNI Critical Claims Set (cdnicrit) claim indicates that extensions to this specification are being used that MUST be understood and processed. Its value is a comma separated listing of claims in the Signed JWT that use those extensions. If any of the listed extension claims are not understood and supported by the recipient, then the Signed JWT MUST be rejected. Producers MUST NOT include claim names defined by this specification, duplicate names, or names that do not occur as claim names within the Signed JWT in the cdnicrit list. Producers MUST NOT use the empty list "" as the cdnicrit value. Recipients MAY consider the Signed JWT to be invalid if the cdnicrit list contains any claim names defined by this specification or if any other constraints on its use are violated. This claim MUST be understood and processed by all implementations.

2.1.1.10. Client IP Address (cdniip) claim

Client IP Address (cdniip) [optional] - The Client IP Address (cdniip) claim holds an IP address or IP prefix for which the Signed URI is valid. This is represented in CIDR notation, with dotted decimal format for IPv4 addresses [RFC0791] or canonical text representation for IPv6 addresses [RFC5952]. The request MUST be rejected if sourced from a client outside the specified IP range. Since the client IP is considered personally identifiable information this field MUST be a JSON Web Encryption (JWE [RFC7516]) Object in compact serialization form. If the CDN verifying the signed JWT does not support Client IP verification, or if the Client IP in the signed JWT does not match the source IP address in the content request, the CDN MUST reject the request. The type of this claim is a JSON string that contains the JWE. If the received signed JWT contains a Client IP claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Client IP claim, and the Client IP value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Client IP claim if no Client IP claim

existed in the received signed JWT.

It should be noted that use of this claim can cause issues, for example, in situations with dual-stack IPv4 and IPv6 networks, MPTCP, QUIC, and mobile clients switching from Wi-Fi to Cellular networks where the client's source address can change, even between address families. This claim exists mainly for legacy feature parity reasons, therefore use of this claim should be done judiciously. An example of a reasonable use case would be making a signed JWT for an internal preview of an asset where the end consumer understands that they must be originated from the same IP for the entirety of the session. Using this claim at large is NOT RECOMMENDED.

2.1.11. CDNI URI Container (cdniuc) claim

URI Container (cdniuc) [mandatory] - The URI Container (cdniuc) holds the URI representation before a URI Signing Package is added. This representation can take one of several forms detailed in Section 2.1.15. If the URI Container used in the signed JWT does not match the URI of the content request, the CDN verifying the signed JWT MUST reject the request. When comparing the URI, the percent encoded form as defined in [RFC3986] Section 2.1 MUST be used. When redirecting a URI, the CDN generating the new signed JWT MAY change the URI Container to comport with the URI being used in the redirection.

2.1.12. CDNI Expiration Time Setting (cdniets) claim

CDNI Expiration Time Setting (cdniets) [optional] - The CDNI Expiration Time Setting (cdniets) claim provides a means for setting the value of the Expiry Time (exp) claim when generating a subsequent signed JWT in Signed Token Renewal. Its type is a JSON numeric value. It denotes the number of seconds to be added to the time at which the JWT is verified that gives the value of the Expiry Time (exp) claim of the next signed JWT. The CDNI Expiration Time Setting (cdniets) SHOULD NOT be used when not using Signed Token Renewal and MUST be present when using Signed Token Renewal.

2.1.13. CDNI Signed Token Transport (cdnistt) claim

CDNI Signed Token Transport (cdnistt) [optional] - The CDNI Signed Token Transport (cdnistt) claim provides a means of signalling the method through which a new signed JWT is transported from the CDN to the UA and vice versa for the purpose of Signed Token Renewal. Its type is a JSON integer. Values for this claim are defined in Section 6.5. If using this claim you MUST also specify a CDNI Expiration Time Setting (cdniets) as noted above.

2.1.14. CDNI Signed Token Depth (cdnistd) claim

CDNI Signed Token Depth (cdnistd) [optional] - The CDNI Signed Token Depth (cdnistd) claim is used to associate a subsequent signed JWT, generated as the result of a CDNI Signed Token Transport claim, with a specific URI subset. Its type is a JSON integer. Signed JWTs MUST NOT use a negative value for the CDNI Signed Token Depth claim.

If the transport used for Signed Token Transport allows the CDN to associate the path component of a URI with tokens (e.g., an HTTP Cookie Path as described in section 4.1.2.4 of [RFC6265]), the CDNI Signed Token Depth value is the number of path segments that should be considered significant for this association. A CDNI Signed Token Depth of zero means that the client SHOULD be directed to return the token with requests for any path. If the CDNI Signed Token Depth is greater than zero, then the CDN SHOULD send the client a token to return for future requests wherein the first CDNI Signed Token Depth segments of the path match the first CDNI Signed Token Depth segments of the signed URI path. This matching MUST use the URI with the token removed, as specified in Section 2.1.15.

If the URI path to match contains fewer segments than the CDNI Signed Token Depth claim, a signed JWT MUST NOT be generated for the purposes of Signed Token Renewal. If the CDNI Signed Token Depth claim is omitted, it means the same thing as if its value were zero. If the received signed JWT contains a CDNI Signed Token Depth claim, then any JWT subsequently generated for CDNI redirection or Signed Token Transport MUST also contain a CDNI Signed Token Depth claim, and the value MUST be the same as in the received signed JWT.

2.1.15. URI Container Forms

The URI Container (cdniuc) claim takes one of the following forms: 'hash:' or 'regex:'. More forms may be added in the future to extend the capabilities.

Before comparing a URI with contents of this container, the following steps MUST be performed:

- * Prior to verification, remove the signed JWT from the URI. This removal is only for the purpose of determining if the URI matches; all other purposes will use the original URI. If the signed JWT is terminated by anything other than a sub-delimiter (as defined in [RFC3986] Section 2.2), everything from the reserved character (as defined in [RFC3986] Section 2.2) that precedes the URI Signing Package Attribute to the last character of the signed JWT will be removed, inclusive. Otherwise, everything from the first character of the URI Signing Package Attribute to the sub-delimiter that terminates the signed JWT will be removed, inclusive.
- * Normalize the URI according to section 2.7.3 [RFC7230] and sections 6.2.2 and 6.2.3 [RFC3986]. This applies to both generation and verification of the signed JWT.

2.1.15.1. URI Hash Container (hash:)

Prefixed with 'hash:', this string is a URL Segment form ([RFC6920] Section 5) of the URI.

2.1.15.2. URI Regular Expression Container (regex:)

Prefixed with 'regex:', this string is any POSIX Section 9 [POSIX.1] Extended Regular Expression compatible regular expression used to match against the requested URI. These regular expressions MUST be evaluated in the POSIX locale (POSIX Section 7.2 [POSIX.1]).

Note: Because '\' has special meaning in JSON [RFC8259] as the escape character within JSON strings, the regular expression character '\' MUST be escaped as '\\'.

An example of a 'regex:' is the following:

```
[^:]*\\://[^/]*/*folder/content/quality_[^/]*/*segment.{3}\\\.mp4(\\?.*)?
```

Note: Due to computational complexity of executing arbitrary regular expressions, it is RECOMMENDED to only execute after verifying the JWT to ensure its authenticity.

2.2. JWT Header

The header of the JWT MAY be passed via the CDNI Metadata interface instead of being included in the URISigningPackage. The header value MUST be transmitted in the serialized encoded form and prepended to the JWT payload and signature passed in the URISigningPackage prior to verification. This reduces the size of the signed JWT token.

3. URI Signing Token Renewal

3.1. Overview

For content that is delivered via HTTP in a segmented fashion, such as MPEG-DASH [MPEG-DASH] or HTTP Live Streaming (HLS) [RFC8216], special provisions need to be made in order to ensure URI Signing can be applied. In general, segmented protocols work by breaking large objects (e.g., videos) into a sequence of small independent segments. Such segments are then referenced by a separate manifest file, which either includes a list of URLs to the segments or specifies an algorithm through which a User Agent can construct the URLs to the segments. Requests for segments therefore originate from the manifest file and, unless the URLs in the manifest file point to the CSP, are not subjected to redirection and URI Signing. This opens up a vulnerability to malicious User Agents sharing the manifest file and deep-linking to the segments.

One method for dealing with this vulnerability would be to include, in the manifest itself, Signed URIs that point to the individual segments. There exist a number of issues with that approach. First, it requires the CDN delivering the manifest to rewrite the manifest file for each User Agent, which would require the CDN to be aware of the exact segmentation protocol used. Secondly, it could also require the expiration time of the Signed URIs to be valid for an extended duration if the content described by the manifest is meant to be consumed in real time. For instance, if the manifest file were to contain a segmented video stream of more than 30 minutes in length, Signed URIs would require to be valid for at least 30 minutes, thereby reducing their effectiveness and that of the URI Signing mechanism in general. For a more detailed analysis of how segmented protocols such as HTTP Adaptive Streaming protocols affect CDNI, see Models for HTTP-Adaptive-Streaming-Aware CDNI [RFC6983].

The method described in this section allows CDNs to use URI Signing for segmented content without having to include the Signed URIs in the manifest files themselves.

3.2. Signed Token Renewal mechanism

In order to allow for effective access control of segmented content, the URI Signing mechanism defined in this section is based on a method through which subsequent segment requests can be linked together. As part of the JWT verification procedure, the CDN can generate a new signed JWT that the UA can use to do a subsequent request. More specifically, whenever a UA successfully retrieves a segment, it receives, in the HTTP 2xx Successful message, a signed JWT that it can use whenever it requests the next segment. As long

as each successive signed JWT is correctly verified before a new one is generated, the model is not broken, and the User Agent can successfully retrieve additional segments. Given the fact that with segmented protocols, it is usually not possible to determine a priori which segment will be requested next (i.e., to allow for seeking within the content and for switching to a different representation), the Signed Token Renewal uses the URI Regular Expression Container scoping mechanisms in the URI Container (cdniuc) claim to allow a signed JWT to be valid for more than one URL.

In order for this renewal of signed JWTs to work, it is necessary for a UA to extract the signed JWT from the HTTP 2xx Successful message of an earlier request and use it to retrieve the next segment. The exact mechanism by which the client does this is outside the scope of this document. However, in order to also support legacy UAs that do not include any specific provisions for the handling of signed JWTs, Section 3.3 defines a mechanism using HTTP Cookies [RFC6265] that allows such UAs to support the concept of renewing signed JWTs without requiring any additional UA support.

3.2.1. Required Claims

The cdnistt claim (Section 2.1.13) and cdniets claim (Section 2.1.12) MUST both be present for Signed Token Renewal. cdnistt MAY be set to a value of '0' to mean no Signed Token Renewal, but there still MUST be a corresponding cdniets that verifies as a JSON number. However, if use of Signed Token Renewal is not desired, it is RECOMMENDED to simply omit both.

3.3. Communicating a signed JWTs in Signed Token Renewal

This section assumes the value of the CDNI Signed Token Transport (cdnistt) claim has been set to 1.

When using the Signed Token Renewal mechanism, the signed JWT is transported to the UA via a 'URISigningPackage' cookie added to the HTTP 2xx Successful message along with the content being returned to the UA, or to the HTTP 3xx Redirection message in case the UA is redirected to a different server.

3.3.1. Support for cross-domain redirection

For security purposes, the use of cross-domain cookies is not supported in some application environments. As a result, the Cookie-based method for transport of the Signed Token described in Section 3.3 might break if used in combination with an HTTP 3xx Redirection response where the target URL is in a different domain. In such scenarios, Signed Token Renewal of a signed JWT SHOULD be

communicated via the query string instead, in a similar fashion to how regular signed JWTs (outside of Signed Token Renewal) are communicated. Note the value of the CDNI Signed Token Transport (cdnistt) claim MUST be set to 2.

Note that the process described herein only works in cases where both the manifest file and segments constituting the segmented content are delivered from the same domain. In other words, any redirection between different domains needs to be carried out while retrieving the manifest file.

4. Relationship with CDNI Interfaces

Some of the CDNI Interfaces need enhancements to support URI Signing. A dCDN that supports URI Signing needs to be able to advertise this capability to the uCDN. The uCDN needs to select a dCDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the uCDN needs to be able to distribute via the CDNI Metadata interface the information necessary to allow the dCDN to verify a Signed URI. Events that pertain to URI Signing (e.g., request denial or delivery after an access authorization decision has been made) need to be included in the logs communicated through the CDNI Logging interface.

4.1. CDNI Control Interface

URI Signing has no impact on this interface.

4.2. CDNI Footprint & Capabilities Advertisement Interface

The CDNI Request Routing: Footprint and Capabilities Semantics document [RFC8008] defines support for advertising CDNI Metadata capabilities, via CDNI Payload Type. The CDNI Payload Type registered in Section 6.1 can be used for capability advertisement.

4.3. CDNI Request Routing Redirection Interface

The CDNI Request Routing Redirection Interface [RFC7975] describes the recursive request redirection method. For URI Signing, the uCDN signs the URI provided by the dCDN. URI Signing therefore has no impact on this interface.

4.4. CDNI Metadata Interface

The CDNI Metadata Interface [RFC8006] describes the CDNI metadata distribution needed to enable content acquisition and delivery. For URI Signing, a new CDNI metadata object is specified.

The UriSigning Metadata object contains information to enable URI Signing and verification by a dCDN. The UriSigning properties are defined below.

Property: enforce

Description: URI Signing enforcement flag. Specifically, this flag indicates if the access to content is subject to URI Signing. URI Signing requires the dCDN to ensure that the URI is signed and verified before delivering content. Otherwise, the dCDN does not perform verification, regardless of whether or not the URI is signed.

Type: Boolean

Mandatory-to-Specify: No. The default is true.

Property: issuers

Description: A list of valid Issuers against which the Issuer claim in the signed JWT may be cross-referenced.

Type: Array of Strings

Mandatory-to-Specify: No. The default is an empty list. An empty list means that any Issuer in the trusted key store of issuers is acceptable.

Property: package-attribute

Description: The attribute name to use for the URI Signing Package.

Type: String

Mandatory-to-Specify: No. The default is "URISigningPackage".

Property: jwt-header

Description: The header part of JWT that is used for verifying a signed JWT when the JWT token in the URI Signing Package does not contain a header part.

Type: String

Mandatory-to-Specify: No. By default, the header is assumed to be included in the JWT token.

The following is an example of a URI Signing metadata payload with all default values:

```
{
  "generic-metadata-type": "MI.UriSigning"
  "generic-metadata-value": {}
}
```

The following is an example of a URI Signing metadata payload with explicit values:

```
{
  "generic-metadata-type": "MI.UriSigning"
  "generic-metadata-value":
    {
      "enforce": true,
      "issuers": ["csp", "ucdn1", "ucdn2"],
      "package-attribute": "usp",
      "jwt-header":
        {
          "alg": "ES256",
          "kid": "P5UpOv0eMqlwcxLf7WxIg09JdSYGYFDOWkldueaImf0"
        }
    }
}
```

4.5. CDNI Logging Interface

For URI Signing, the dCDN reports that enforcement of the access control was applied to the request for content delivery. When the request is denied due to enforcement of URI Signing, the reason is logged.

The following CDNI Logging field for URI Signing SHOULD be supported in the HTTP Request Logging Record as specified in CDNI Logging Interface [RFC7937], using the new "cdni_http_request_v2" record-type registered in Section 6.2.1.

* s-uri-signing (mandatory):

- format: 3DIGIT
- field value: this characterises the URI Signing verification performed by the Surrogate on the request. The allowed values are registered in Section 6.4.

- occurrence: there MUST be zero or exactly one instance of this field.
- * s-uri-signing-deny-reason (optional):
 - format: QSTRING
 - field value: a string for providing further information in case the signed JWT was rejected, e.g., for debugging purposes.
 - occurrence: there MUST be zero or exactly one instance of this field.

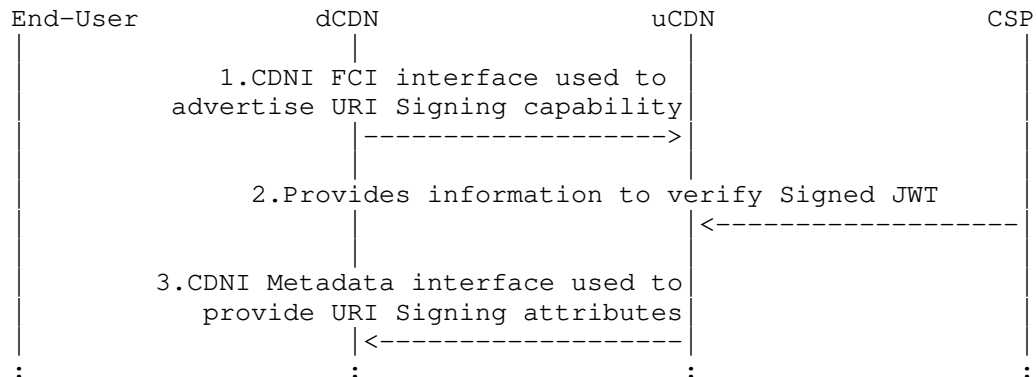
5. URI Signing Message Flow

URI Signing supports both HTTP-based and DNS-based request routing. JSON Web Token (JWT) [RFC7519] defines a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a Signed JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC).

5.1. HTTP Redirection

For HTTP-based request routing, a set of information that is unique to a given end user content request is included in a Signed JWT, using key information that is specific to a pair of adjacent CDNI hops (e.g., between the CSP and the uCDN or between the uCDN and a dCDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI Signing method (assuming HTTP redirection, iterative request routing, and a CDN path with two CDNs) includes the following steps:



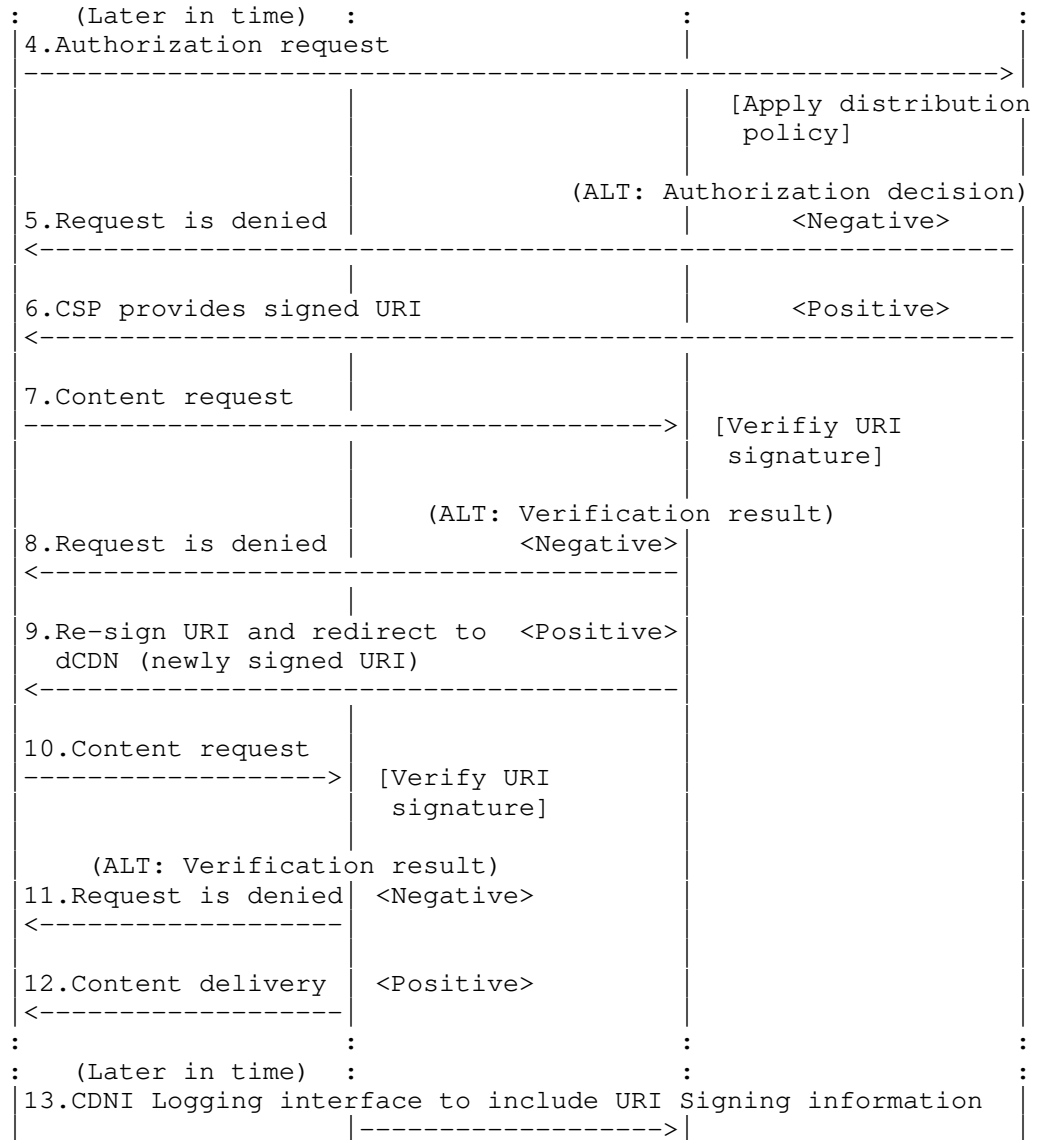


Figure 3: HTTP-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the dCDN advertises its capabilities including URI Signing support to the uCDN.

2. CSP provides to the uCDN the information needed to verify signed URIs from that CSP. For example, this information will include one or more keys used for validation.
3. Using the CDNI Metadata interface, the uCDN communicates to a dCDN the information needed to verify signed URIs from the uCDN for the given CSP. For example, this information may include the URI query string parameter name for the URI Signing Package Attribute in addition to keys used for validation.
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 local distribution policy.
5. If the authorization decision is negative, the CSP rejects the request and sends an error code (e.g., 403 Forbidden) in the HTTP response.
6. If the authorization decision is positive, the CSP computes a Signed JWT 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 uCDN verifies the Signed JWT in the URI using the information provided by the CSP.
8. If the verification result is negative, the uCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.
9. If the verification result is positive, the uCDN computes a Signed JWT that is based on unique parameters of that request and provides it to the end user as the URI to use to further request the content from the dCDN.
10. On receipt of the corresponding content request, the dCDN verifies the Signed JWT in the signed URI using the information provided by the uCDN in the CDNI Metadata.
11. If the verification result is negative, the dCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.
12. If the verification result is positive, the dCDN serves the request and delivers the content.

13. At a later time, the dCDN reports logging events that include URI Signing information.

With HTTP-based request routing, URI Signing matches well the general chain of trust model of CDNI both with symmetric and asymmetric keys because the key information only needs to be specific to a pair of adjacent CDNI hops.

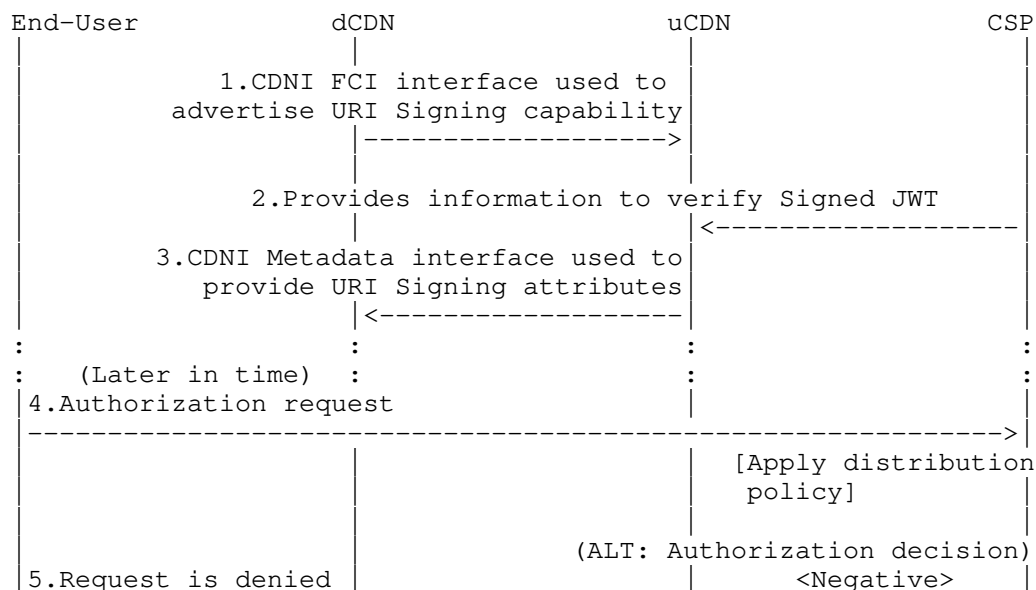
Note: While using a symmetric shared key is supported, it is NOT RECOMMENDED. See the Security Considerations (Section 7) section about the limitations of shared keys.

5.2. DNS Redirection

For DNS-based request routing, the CSP and uCDN 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 dCDNs. However, if a shared secret key is required, then the distribution SHOULD be performed by the CSP directly.

Note: While using a symmetric shared key is supported, it is NOT RECOMMENDED. See the Security Considerations (Section 7) section about the limitations of shared keys.

The URI Signing method (assuming iterative DNS request routing and a CDN path with two CDNs) includes the following steps.



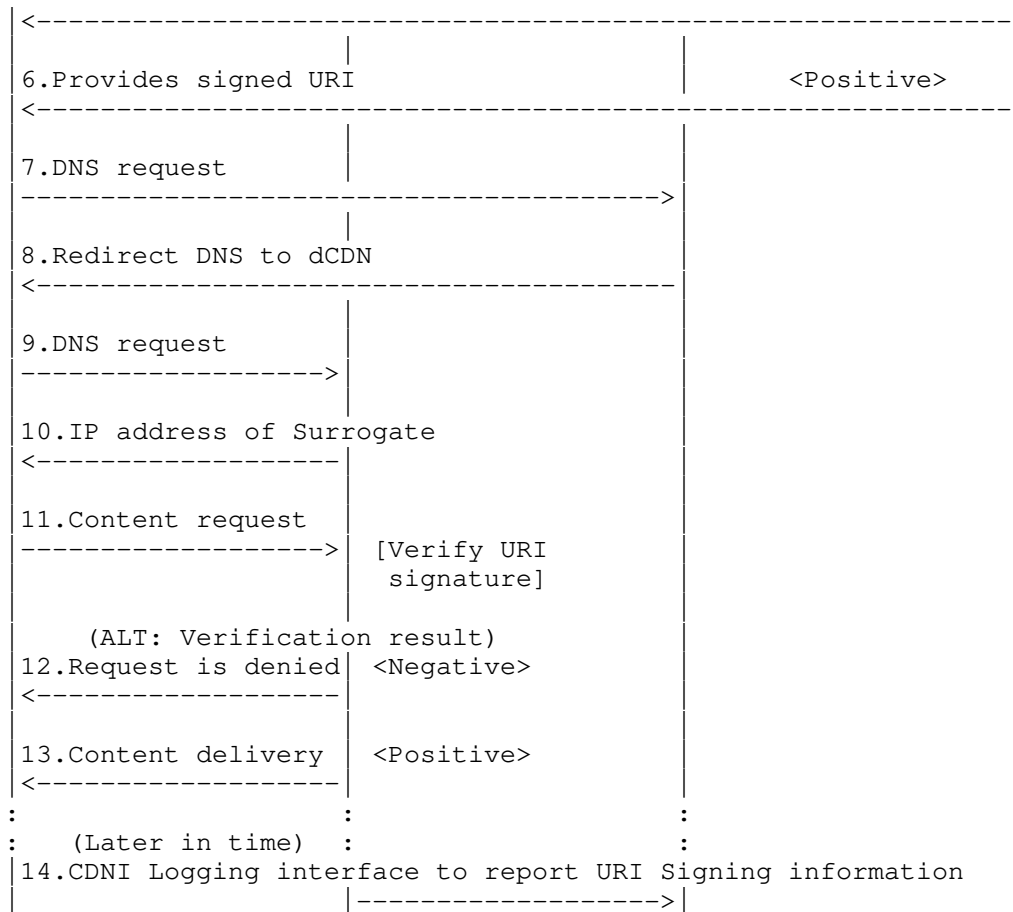


Figure 4: DNS-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the dCDN advertises its capabilities including URI Signing support to the uCDN.
2. CSP provides to the uCDN the information needed to verify Signed JWTs from that CSP. For example, this information will include one or more keys used for validation.
3. Using the CDNI Metadata interface, the uCDN communicates to a dCDN the information needed to verify Signed JWTs from the CSP (e.g., the URI query string parameter name for the URI Signing Package Attribute). In the case of symmetric shared key, the uCDN MUST NOT share the key with a dCDN.

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 local distribution policy.
5. If the authorization decision is negative, the CSP rejects the request and sends an error code (e.g., 403 Forbidden) in the HTTP response.
6. If the authorization decision is positive, the CSP computes a Signed JWT 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 uCDN.
8. On receipt of the DNS request, the uCDN redirects the request to the dCDN.
9. End user sends DNS request to the dCDN.
10. On receipt of the DNS request, the dCDN responds with IP address of one of its Surrogates.
11. On receipt of the corresponding content request, the dCDN verifies the Signed JWT in the URI using the information provided by the uCDN in the CDNI Metadata.
12. If the verification result is negative, the dCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.
13. If the verification result is positive, the dCDN serves the request and delivers the content.
14. At a later time, dCDN 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 needs to be distributed across multiple, possibly untrusted, CDNI hops is the public key, which is generally not confidential.

With DNS-based request routing, URI Signing does not match well with 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, to CDNs with which the CSP may not have a trust relationship. This raises a security concern for applicability

of URI Signing with symmetric keys in case of DNS-based inter-CDN request routing. Due to these flaws, this architecture MUST NOT be implemented.

Note: While using a symmetric shared key is supported, it is NOT RECOMMENDED. See the Security Considerations (Section 7) section about the limitations of shared keys.

6. IANA Considerations

6.1. CDNI Payload Type

This document requests the registration of the following CDNI Payload Type under the IANA "CDNI Payload Types" registry:

Payload Type	Specification
MI.UriSigning	RFcthis

Table 1

[RFC Editor: Please replace RFcthis with the published RFC number for this document.]

6.1.1. CDNI UriSigning Payload Type

Purpose: The purpose of this payload type is to distinguish UriSigning MI objects (and any associated capability advertisement).

Interface: MI/FCI

Encoding: see Section 4.4

6.2. CDNI Logging Record Type

This document requests the registration of the following CDNI Logging record-type under the IANA "CDNI Logging record-types" registry:

record-types	Reference	Description
cdni_http_request_v2	RFCThis	Extension to CDNI Logging Record version 1 for content delivery using HTTP, to include URI Signing logging fields

Table 2

[RFC Editor: Please replace RFCThis with the published RFC number for this document.]

6.2.1. CDNI Logging Record Version 2 for HTTP

The "cdni_http_request_v2" record-type supports all of the fields supported by the "cdni_http_request_v1" record-type [RFC7937] plus the two additional fields "s-uri-signing" and "s-uri-signing-deny-reason", registered by this document in Section 6.3. The name, format, field value, and occurrence information for the two new fields can be found in Section 4.5 of this document.

6.3. CDNI Logging Field Names

This document requests the registration of the following CDNI Logging fields under the IANA "CDNI Logging Field Names" registry:

Field Name	Reference
s-uri-signing	RFCThis
s-uri-signing-deny-reason	RFCThis

Table 3

[RFC Editor: Please replace RFCThis with the published RFC number for this document.]

6.4. CDNI URI Signing Verification Code

The IANA is requested to create a new "CDNI URI Signing Verification Code" subregistry, in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Verification Code" namespace defines the valid values associated with the s-uri-signing CDNI Logging Field. The CDNI URI Signing Verification Code is a 3DIGIT value as defined in Section 4.5. Additions to the CDNI URI Signing Verification Code namespace will conform to the "Specification Required" policy as defined in [RFC8126]. Updates to this subregistry are expected to be infrequent.

Value	Reference	Description
000	RFCThis	No signed JWT verification performed
200	RFCThis	Signed JWT verification performed and verified
400	RFCThis	Signed JWT verification performed and rejected because of incorrect signature
401	RFCThis	Signed JWT verification performed and rejected because of Issuer enforcement
402	RFCThis	Signed JWT verification performed and rejected because of Subject enforcement
403	RFCThis	Signed JWT verification performed and rejected because of Audience enforcement
404	RFCThis	Signed JWT verification performed and rejected because of Expiration Time enforcement
405	RFCThis	Signed JWT verification performed and rejected because of Not Before enforcement
406	RFCThis	Signed JWT verification performed and rejected because only one of CDNI Signed Token Transport or CDNI

		Expiration Time Setting present.
407	RFCthis	Signed JWT verification performed and rejected because of JWT ID enforcement
408	RFCthis	Signed JWT verification performed and rejected because of Version enforcement
409	RFCthis	Signed JWT verification performed and rejected because of Critical Extension enforcement
410	RFCthis	Signed JWT verification performed and rejected because of Client IP enforcement
411	RFCthis	Signed JWT verification performed and rejected because of URI Container enforcement
500	RFCthis	Unable to perform signed JWT verification because of malformed URI

Table 4

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.5. CDNI URI Signing Signed Token Transport

The IANA is requested to create a new "CDNI URI Signing Signed Token Transport" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Signed Token Transport" namespace defines the valid values that may be in the Signed Token Transport (cdnistt) JWT claim. Additions to the Signed Token Transport namespace conform to the "Specification Required" policy as defined in [RFC8126]. Updates to this subregistry are expected to be infrequent.

The following table defines the initial Enforcement Information Elements:

Value	Description	RFC
0	Designates token transport is not enabled	RFCthis
1	Designates token transport via cookie	RFCthis
2	Designates token transport via query string	RFCthis

Table 5

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.6. JSON Web Token Claims Registration

This specification registers the following Claims in the IANA "JSON Web Token Claims" registry [IANA.JWT.Claims] established by [RFC7519].

6.6.1. Registry Contents

- * Claim Name: cdniv
- * Claim Description: CDNI Claim Set Version
- * Change Controller: IESG
- * Specification Document(s): Section 2.1.8 of [[this specification]]
- * Claim Name: cdnicrit
- * Claim Description: CDNI Critical Claims Set
- * Change Controller: IESG
- * Specification Document(s): Section 2.1.9 of [[this specification]]
- * Claim Name: cdniip
- * Claim Description: CDNI IP Address
- * Change Controller: IESG
- * Specification Document(s): Section 2.1.10 of [[this specification]]
- * Claim Name: cdniuc
- * Claim Description: CDNI URI Container
- * Change Controller: IESG
- * Specification Document(s): Section 2.1.11 of [[this specification]]
- * Claim Name: cdniets

- * Claim Description: CDNI Expiration Time Setting for Signed Token Renewal
- * Change Controller: IESG
- * Specification Document(s): Section 2.1.12 of [[this specification]]

- * Claim Name: cdnistt
- * Claim Description: CDNI Signed Token Transport Method for Signed Token Renewal
- * Change Controller: IESG
- * Specification Document(s): Section 2.1.13 of [[this specification]]

- * Claim Name: cdnistd
- * Claim Description: CDNI Signed Token Depth
- * Change Controller: IESG
- * Specification Document(s): Section 2.1.14 of [[this specification]]

6.7. Expert Review Guidance

Generally speaking, we should determine the registration has a rational justification and does not duplicate a previous registration. Early assignment should be permissible as long as there is a reasonable expectation that the specification will become formalized. Expert Reviewers should be empowered to make determinations, but generally speaking they should allow new claims that do not otherwise introduce conflicts with implementation or things that may lead to confusion. They should also follow the guidelines of [RFC8126] Section 5 when sensible.

7. Security Considerations

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of CDNI. The primary goal of URI Signing is to make sure that only authorized UAs are able to access the content, with a 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.

CDNI URI Signing Signed Tokens leverage JSON Web Tokens and thus guidelines in [RFC8725] are applicable for all JWT interactions.

In general, it holds that the level of protection against illegitimate access can be increased by including more claims in the signed JWT. The current version of this document includes claims for enforcing Issuer, Client IP Address, Not Before time, 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:** A (valid) Signed URI may be used to perform replay attacks. The vulnerability to replay attacks can be reduced by picking a relatively short window between the Not Before time and Expiration Time attributes, although this is limited by the fact that any HTTP-based request needs a window of at least a couple of seconds to prevent sudden network issues from denying legitimate UAs access to the content. One may also reduce exposure to replay attacks by including a unique one-time access ID via the JWT ID attribute (jti claim). Whenever the dCDN receives a request with a given unique ID, it adds that ID to the list of 'used' IDs. In the case an illegitimate UA tries to use the same URI through a replay attack, the dCDN can deny the request based on the already-used access ID. This list should be kept bounded. A reasonable approach would be to expire the entries based on the exp claim value. If no exp claim is present then a simple LRU could be used, however this would allow values to eventually be reused.
- * **Illegitimate clients 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 dCDN. This results in the dCDN not being able to distinguish between different users based on Client IP Address which can lead to 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, e.g., attributes that can be found in HTTP headers. However, this may be easily circumvented by a sophisticated attacker.

A shared key distributed between CSP and uCDN is more likely to be compromised. Since this key can be used to legitimately sign a URL for content access authorization, it is important to know the implications of a compromised shared key. While using a shared key scheme can be convenient, this architecture is NOT RECOMMENDED due to the risks associated. It is included for legacy feature parity and is highly discouraged in new implementations.

If a shared key usable for signing is compromised, an attacker can use it to perform a denial-of-service attack by forcing the CDN to evaluate prohibitively expensive regular expressions embedded in a URI Container (cdniuc) claim. As a result, compromised keys should be timely revoked in order to prevent exploitation.

The URI Container (cdniuc) claim can be given a wildcard value. This, combined with the fact that it is the only mandatory claim, means you can effectively make a skeleton key. Doing this does not sufficiently limit the scope of the JWT and is NOT RECOMMENDED. The only way to prevent such a key from being used after it is distributed is to revoke the signing key so it no longer validates.

8. Privacy

The privacy protection concerns described in CDNI Logging Interface [RFC7937] apply when the client's IP address (cdniip) or Subject (sub) is embedded in the Signed URI. For this reason, the mechanism described in Section 2 encrypts the Client IP or Subject before including it in the URI Signing Package (and thus the URL itself).

9. Acknowledgements

The authors would like to thank the following people for their contributions in reviewing this document and providing feedback: Scott Leibrand, Kevin Ma, Ben Niven-Jenkins, Thierry Magnien, Dan York, Bhaskar Bhupalam, Matt Caulfield, Samuel Rajakumar, Iuniana Opreescu, Leif Hedstrom, Gancho Tenev, Brian Campbell, and Chris Lemmons.

10. Contributors

In addition, the authors would also like to make special mentions for certain people who contributed significant sections to this document.

- * Matt Caulfield provided content for the CDNI Metadata Interface section.
- * Emmanuel Thomas provided content for HTTP Adaptive Streaming.
- * Matt Miller provided consultation on JWT usage as well as code to generate working JWT examples.

11. References

11.1. Normative References

- [POSIX.1] "The Open Group Base Specifications Issue 7", IEEE Std 1003.1 2018 Edition, 31 January 2018, <<http://pubs.opengroup.org/onlinepubs/9699919799/>>.
- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <<https://www.rfc-editor.org/info/rfc5952>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [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, <<https://www.rfc-editor.org/info/rfc6707>>.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <<https://www.rfc-editor.org/info/rfc6920>>.

- [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, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC7937] Le Faucheur, F., Ed., Bertrand, G., Ed., Oprescu, I., Ed., and R. Peterkofsky, "Content Distribution Network Interconnection (CDNI) Logging Interface", RFC 7937, DOI 10.17487/RFC7937, August 2016, <<https://www.rfc-editor.org/info/rfc7937>>.
- [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, <<https://www.rfc-editor.org/info/rfc8006>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

11.2. Informative References

- [IANA.JWT.Claims] IANA, "JSON Web Token Claims", <<http://www.iana.org/assignments/jwt>>.

- [MPEG-DASH] ISO, "Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment format", ISO/IEC 23009-1:2014, Edition 2, May 2014, <<http://www.iso.org/standard/65274.html>>.
- [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, DOI 10.17487/RFC6983, July 2013, <<https://www.rfc-editor.org/info/rfc6983>>.
- [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, <<https://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, <<https://www.rfc-editor.org/info/rfc7337>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.
- [RFC7975] Niven-Jenkins, B., Ed. and R. van Brandenburg, Ed., "Request Routing Redirection Interface for Content Delivery Network (CDN) Interconnection", RFC 7975, DOI 10.17487/RFC7975, October 2016, <<https://www.rfc-editor.org/info/rfc7975>>.
- [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, <<https://www.rfc-editor.org/info/rfc8008>>.
- [RFC8216] Pantos, R., Ed. and W. May, "HTTP Live Streaming", RFC 8216, DOI 10.17487/RFC8216, August 2017, <<https://www.rfc-editor.org/info/rfc8216>>.
- [RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <<https://www.rfc-editor.org/info/rfc8725>>.

Appendix A. Signed URI Package Example

This section contains three examples of token usage: a simple example with only the required claim present, a complex example which demonstrates the full JWT claims set, including an encrypted Client IP Address (cdniip), and one that uses a Signed Token Renewal.

Note: All of the examples have whitespace added to improve formatting and readability, but are not present in the generated content.

All examples use the following JWK Set [RFC7517]:

```
{ "keys": [
  {
    "kty": "EC",
    "kid": "P5UpOv0eMqlwcxLf7WxIg09JdSYGYFDOWkldueaImf0",
    "use": "sig",
    "alg": "ES256",
    "crv": "P-256",
    "x": "be807S407dzB6I4hTiCUvmxCI6FuxWba1xYB1LSSsZ8",
    "y": "rOGC4vI69g-WF9AGEVI37sNNwbjIzBxSjLvIL7f3RBA"
  },
  {
    "kty": "EC",
    "kid": "P5UpOv0eMqlwcxLf7WxIg09JdSYGYFDOWkldueaImf0",
    "use": "sig",
    "alg": "ES256",
    "crv": "P-256",
    "x": "be807S407dzB6I4hTiCUvmxCI6FuxWba1xYB1LSSsZ8",
    "y": "rOGC4vI69g-WF9AGEVI37sNNwbjIzBxSjLvIL7f3RBA",
    "d": "yaowezrCLTU6yIwUL5RQw67cHgvZeMTLVZXjUGb1A1M"
  },
  {
    "kty": "oct",
    "kid": "f-WbjxBC3dPuI3d24kP2hfvos7Qz688UTi6aB0hN998",
    "use": "enc",
    "alg": "A128GCM",
    "k": "4uFxxV7fhNmrtiah2dlfFg"
  }
]
```

Note: They are the public signing key, the private signing key, and the shared secret encryption key, respectively. The public and private signing keys have the same fingerprint and only vary by the 'd' parameter that is missing from the public signing key.

A.1. Simple Example

This example is a simple common usage example containing a minimal subset of claims that the authors find most useful.

The JWT Claim Set before signing:

Note: "sha-256;2tderfWPa86Ku7YnzW51YUp7dGUjBS_3SW3ELx4hmWY" is the URL Segment form ([RFC6920] Section 5) of "http://cdni.example/foo/bar".

```
{
  "exp": 1646867369,
  "iss": "uCDN Inc",
  "cdniuc": "hash:sha-256;2tderfWPa86Ku7YnzW51YUp7dGUjBS_3SW3ELx4hmWY"
}
```

The signed JWT:

```
eyJhbGciOiJFUzI1NiIsImtpZCI6IiA1VXBpdjBlTXExd2N4TGZ3V3hJZzA5SmRTWUdZRRkRPV2tsZHVlYUltZjAifQ.eyJleHAiOiJlE2NDY4NjczNjksImIzcyI6InVDRE4gSW5jIiwiaW52RmVjIjoiaGFzaDpzaGEtMjU2OzJ0ZGVyZldQYTg2S3U3WW56VzUxWVVwN2RHVWpCU18zU1czRUx4NGhtV1kifQ.TaNlJM3D96i_9J9Xv1ICO6FUIDFTqt3E2YJkEUOLfcH0b89wYRKtbJ9Yj6h_GRgSoZoQO0cps3yUPcWGK3smCw
```

A.2. Complex Example

This example uses all fields except for those dealing with Signed Token Renewal, including Client IP Address (cdniip) and Subject (sub) which are encrypted. This significantly increases the size of the signed JWT token.

JWE for Client IP Address (cdniip) of [2001:db8::1/32]:

```
eyJlbnMiOiJBMTI4R0NNIiwiaW52RmVjIjoiaGFzaDpzaGEtMjU2OzJ0ZGVyZldQYTg2S3U3WW56VzUxWVVwN2RHVWpCU18zU1czRUx4NGhtV1kifQ.TaNlJM3D96i_9J9Xv1ICO6FUIDFTqt3E2YJkEUOLfcH0b89wYRKtbJ9Yj6h_GRgSoZoQO0cps3yUPcWGK3smCw
```

JWE for Subject (sub) of "UserToken":

```
eyJlbnMiOiJBMTI4R0NNIiwiaW52RmVjIjoiaGFzaDpzaGEtMjU2OzJ0ZGVyZldQYTg2S3U3WW56VzUxWVVwN2RHVWpCU18zU1czRUx4NGhtV1kifQ.TaNlJM3D96i_9J9Xv1ICO6FUIDFTqt3E2YJkEUOLfcH0b89wYRKtbJ9Yj6h_GRgSoZoQO0cps3yUPcWGK3smCw
```

The JWT Claim Set before signing:

```
{
  "aud": "dCDN LLC",
  "sub": "eyJlbmMiOiJBMTI4R0NNIiwiaWxnIjoizGlyIiwia2lkIjoizilXYmp4QkMzZFB1STNkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..CLAu80xclc8Bp-Ui.6PlA3F6ip2Dv.CohdtLLpgBnTvRJQCFuz-g",
  "cdniip": "eyJlbmMiOiJBMTI4R0NNIiwiaWxnIjoizGlyIiwia2lkIjoizilXYmp4QkMzZFB1STNkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..aUDDFEQBIc3nWjOb.bGXWTHPkntmPCKn0pPPNEQ.iyTtnFyb02YBLqwl_YsJA",
  "cdniv": 1,
  "exp": 1646867369,
  "iat": 1646694569,
  "iss": "uCDN Inc",
  "jti": "5DAafLhZAfhsbe",
  "nbf": 1646780969,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\..png"
}
```

The signed JWT:

```
eyJhbGciOiJFUzI1NiIsImtpZCI6IiA1VXBpdjBlTXExd2N4TGy3V3hJZzA5SmRTWUdzRkRPV2tsZHVlYUltZjAiOiJkQ0ROIEExMQyIsInN1YiI6ImV5SmxibUlpT2lkQk1USTRSME5OSWl3aVlXeG5Jam9pWkdseUlpd2lhMmxrSWpvaVppMVhZbXA0UWtNelGQjFTVE5rTWpSc1VESm9ablP2Y3pkUmVqWTRPRlZVYVRaaFFqQm9Uams1T0NKOS4uQ0xBdTgweGNsYzhCcClVaS42UDFBM0Y2aXAyRHYuQ29oZHRMTHBnQm5UdlJKUUNgdXotZyIsImNkbmlpcCI6ImV5SmxibUlpT2lkQk1USTRSME5OSWl3aVlXeG5Jam9pWkdseUlpd2lhMmxrSWpvaVppMVhZbXA0UWtNelGQjFTVE5rTWpSc1VESm9ablP2Y3pkUmVqWTRPRlZVYVRaaFFqQm9Uams1T0NKOS4uYVVEREZFUUJJYzNuV2pPYi5iRlhXVEhQa250bVBDS24wcFBQTkVRLml1SVHR0bkZ5Yk8yWUJMcXdsX1lTakEiLCJjZG5pdilI6MSwiZlXhwIjoXNjQ2ODY3MzY5LCJpYXQiojE2NDY2OTQ1NjksImIzcyI6InVDRlE4gSW5jIiwianRpIjoianURBYWZMaFpBZmhzYmUiLCJpYmYiOiE2NDY3ODANjksImNkbml1YyI6InJlZ2V4Omh0dHA6Ly9jZG5pXWZlXGhZbXBsZS9mb28vYmFyLlswLTldeZnN9XWwucG5nIn0.IjmVX0uD5MYqArc-M08uEsEeoDQn8kuYXZ9HGHDmDDxsHikT0c8jcX8xYD0z3LzQc1MG65i1kT2sRbZ7isUw8w
```

A.3. Signed Token Renewal Example

This example uses fields for Signed Token Renewal.

The JWT Claim Set before signing:

```
{
  "cdniets": 30,
  "cdnistt": 1,
  "cdnistd": 2,
  "exp": 1646867369,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\..ts"
}
```

The signed JWT:

```
eyJhbGciOiJFUzI1NiIsImtpZCI6IiA1VXBpdjBlTXExd2N4TGZ3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJjZG5pZXRzIjozMCI6MSwiY2Ruan0ZCI6MiwiZ
XhwIjoxNjQ2ODY3MzY5LCJjZG5pdWMiOiJyZWdleDpodHRwOi8vY2RuanVxcLmV4YW
lwbGUvZm9vL2Jhcn9bMC05XXszfVxcLnRzIn0.tlPvoKw3BCClw4Lx9PQu7MK6b2IN
55ZoCPSaxovGK0zS53Wpb1MbJBow7G8LiGR39h6-2Iq7PWUSr3MdTIzHYw
```

Once the server verifies the signed JWT it will return a new signed JWT with an updated expiry time (exp) as shown below. Note the expiry time is increased by the expiration time setting (cdniets) value.

The JWT Claim Set before signing:

```
{
  "cdniets": 30,
  "cdnistt": 1,
  "cdnistd": 2,
  "exp": 1646867399,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\..ts"
}
```

The signed JWT:

```
eyJhbGciOiJFUzI1NiIsImtpZCI6IiA1VXBpdjBlTXExd2N4TGZ3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJjZG5pZXRzIjozMCI6MSwiY2Ruan0ZCI6MiwiZ
XhwIjoxNjQ2ODY3MzY5LCJjZG5pdWMiOiJyZWdleDpodHRwOi8vY2RuanVxcLmV4YW
lwbGUvZm9vL2Jhcn9bMC05XXszfVxcLnRzIn0.ivY5d_fKGd-OHTpUs8uJUrnHvt-rdu
zu5H4zM7l67pUUAghub53FqDQ5G16jRYX2sY73mA_uLpYDdb-CPTs8FA
```

Authors' Addresses

Ray van Brandenburg
Tiledmedia
Anna van Buerenplein 1
Den Haag
Phone: +31 88 866 7000
Email: ray@tiledmedia.com

Kent Leung
Email: mail4kentl@gmail.com

Phil Sorber
Apple, Inc.
1800 Wazee Street
Suite 410
Denver, CO 80202
United States
Email: sorber@apple.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 19, 2015

D. Singer
Apple, Inc.
A. Begen
Cisco
October 16, 2014

URLs and HTTP Response Forms for Multicast
draft-singer-appsawg-mcast-url-00

Abstract

This document motivates the need for defining a URL for multicast delivery and provides a proposal for this purpose.

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 19, 2015.

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	2
2. Glossary of Terms	4
3. Use Cases	4
4. Required Information	4
5. Suggested URL Form	5
5.1. Introduction	5
5.2. Information on FLUTE (RFC 6726)	5
5.3. URL Form Requirements	6
5.4. Base URL Form	6
6. FCAST Metainformation field	8
7. HTTP Status Codes and Their Applicability to FCAST	8
7.1. Useful	8
7.2. Unlikely but Possible	9
7.3. Probably Inapplicable	9
8. Operation of the URL Handler	10
9. Security Considerations	10
10. IANA Considerations	11
11. References	11
11.1. Normative References	11
11.2. Informative References	11
Authors' Addresses	11

1. Introduction

Various multicast file-delivery protocols are defined by the IETF and 3GPP (notably File Delivery over Unidirectional Transport (FLUTE) and FCAST). However, they are hard to adopt into other services, partly because they do not follow conventions on how these transports are addressed and what information they deliver. Notably:

1. Much of the Web (Internet) assumes that if a file can be used, it can be referred to by a URL that contains enough information to start to try retrieving it. This is not true for files available over multicast.
2. When a URL form is used, it can be annotated with the information on what it refers to (e.g., a MIME type, a codecs parameter for that MIME type, and so on). If we have no URL, we cannot annotate it.

3. HTTP header responses are widely used to signal the unavailability of an expected resource (404) or that a resource has moved (re-direct), or that there are other choices to retrieve the indicated resource, or to deliver portions of a resource (byte-ranges). Though FCAST uses the metainformation format of HTTP, it misses the status line, so neither unavailability nor re-direct can be signaled. You cannot re-direct from multicast to HTTP, for example.
4. 'Soft' information such as multicast group addresses, transport session identifiers, and so on, are hard-coded into the descriptions (e.g., Session Description Protocol (SDP) files [RFC4566]). In general, the Web/Internet avoids hard-coding such values, preferring to use lookup (e.g., DNS for addresses); lookups can be re-factored as boundaries are crossed.

Traditionally multicasts have been addressed by requiring the client to acquire some pre-knowledge (e.g., an SDP file) by some means out of band. Thus, we require that every protocol that might use multicast be adapted. This is error-prone, limiting, and time-consuming. Instead, if an operating system can have a URL handler for multicast URLs that deliver file objects, with an interface that 'emulates' the interface to HTTP, many (not all) things would 'just work'. Perhaps the most notable is that we might re-direct from HTTP to multicast when the server detects that there is a better way to get the resource (perhaps, at this time and for this client).

The places where URLs occur, and where it would be advantageous to be able to state "this file is available on multicast", are legion. Obvious examples include anything linked into HTML (a Web page or email), especially media (video, audio, images); in HTTP itself where re-direction supplies a URL, and HTTP adaptive streaming systems where many clients could be fetching the same set of content segments. For many of these, operating system support with the same API as HTTP would suffice. Even in the HTTP adaptive streaming case, where it is true that the streaming engine needs to know it is using multicast (as this would make substantial changes to bandwidth estimation, etc.), simplifying the markup and the protocol identification to a URL is a plus. FCAST is closer to HTTP operation than FLUTE; files 'just arrive' and there is no concept of the 'set' as represented by the file delivery table in FLUTE. We therefore focus on FCAST in this document.

2. Glossary of Terms

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

3. Use Cases

Here are two example use cases.

1. The classic stadium. A sports franchise wants everyone in the stadium to be able to watch a few selected camera streams. They multicast the streams over a tuned WiFi system.
2. A network operator (either Internet service or 3G/4G) wants to enable people to see a video mosaic of the top channels, and click through to get to a channel fast.

The simple solutions are:

1. Provide a QR code that embeds a multicast URL linking to the manifest file for the video content at the entrance, in printed material, on posters, etc. When people tune to that multicast URL with their phones, they get the manifest, and it refers to streams that are also multicast. The act of tuning into the session starts the client caching everything that arrives.
2. The mosaic is a multicast URL, and the segments of each program are also multicast but with short cache-times, and using the same URL label as the unicast address (i.e., an HTTP URL). When the user clicks on a program, they fetch the manifest (or perhaps the manifests are also multicast and pre-filled into the cache) and they already have the current segment cached, so startup is effectively instant. As they proceed, there is a good chance the multicast has delivered every segment they need, just in time.

4. Required Information

Currently, tune-in to a multicast involves getting hold of a 'head' file that gives a variety of information. The possible information can be roughly separated into different classes:

1. Information about alternatives that could be supplied as part of the higher-level protocol (e.g., different representations in HTTP adaptive streaming and HTML5 source elements)

2. Information (IP addresses and the like) that is needed to 'bootstrap' the multicast reception
3. Information about where/how the reception is possible (e.g., protocol parameters, time-ranges, and so on)
4. Information that could be acquired later, in-band, such as feedback addresses, the availability of alternatives and unicast repair servers, and so on (or indeed, a fuller description of the multicast itself)

For the sake of simplicity, we propose that we only include (2) and (3) in the URL form.

Ideally, there is something about the multicast itself that allows the client system to assess fairly rapidly whether it is working (the multicast join succeeded, packets are arriving, etc.) and if that fails, the URL handler can give a suitable error indication (maybe an existing one, maybe new).

5. Suggested URL Form

5.1. Introduction

Both FLUTE and FCAST rely on Asynchronous Layered Coding (ALC) [RFC5775] / Layered Coding Transport (LCT) [RFC5651], which in turn has the concept of channels to handle congestion and rate control. We presume the existence of a base channel and indicate how to acquire that.

In an FCAST session, files are identified by URI labels. We suggest that we identify a reserved URN form to indicate 'this is a complete SDP file describing all the sessions'. This allows bootstrapping from the base channel to all of the channels in a session.

FLUTE [RFC6726] is specific about the parameters needed to acquire an ALC/LCT session, and since FCAST [RFC6968] also relies on ALC/LCT, the same analysis applies.

5.2. Information on FLUTE (RFC 6726)

To start receiving a file delivery session, the receiver needs to know transport parameters associated with the session. Interpreting these parameters and starting the reception therefore represents the entry point from which thereafter the receiver operation falls into the scope of this specification. According to [RFC5775], the transport parameters of an ALC/LCT session that the receiver needs to know are:

- o The source IP address.
- o The number of channels in the session.
- o The destination IP address and port number for each channel in the session.
- o The transport session identifier (TSI) of the session.
- o An indication that the session is a FLUTE session. The need to demultiplex objects upon reception is implicit in any use of FLUTE, and this fulfills the ALC requirement of an indication of whether or not a session carries packets for more than one object (all FLUTE sessions carry packets for more than one object).

5.3. URL Form Requirements

We have at least the following requirements:

- o The URLs must be valid according to the RFCs and recent work at the W3C
- o The absolute form must exist (obviously)
- o Relative URLs must also work
- o We should avoid the fragment (#) and query suffices (?) even though, in the latter case, there is no server that the URL is sent to.
- o We should permit the URL to self-declare its validity period (and thus enable rapid time-outs when it is requested outside this period)
- o Ideally, we also allow it to indicate its 'geographic' (operator network) availability scope

5.4. Base URL Form

This suggests a URL form in three parts:

- o A prefix giving the URL scheme and basic parameters
- o A mid-part giving the temporal and geographic scope
- o A suffix that is the label of the desired file

Where the prefix is roughly like

`fcast://destination:port/source:TSI`

with

`destination`: An explicit multicast address (x.y.z.w) or (better) a name that resolves to one (or more) IP multicast address(es) for the base channel.

`port`: Port number for the base channel.

`source`: An explicit IP address (x.y.z.w) or (better) a name that resolves to the source address.

`TSI`: The transport session identifier for the session.

The mid-part has optional terms that are each formatted as / key:value. The keys and their values are:

`start`: the absolute start-time of availability

`end`: likewise, the end time

`network`: an identification of the network(s) on which the multicast is made available (for the indicated time-span, if any)

The start and end times are each optional and if present are expressed exactly as in SDP, i.e. as the decimal representation of Network Time Protocol (NTP) time values in seconds since 1900. If the URL agent determines it is operating outside this time range, a suitable error SHOULD be returned immediately. If either the start or end time are absent, then the multicast starts (or stops) at an indefinite time.

The network attribute takes a list of domain names, joined by the plus sign; if the URL handler is confident that the machine is not on any of the networks, a suitable error SHOULD be returned immediately, as it knows the multicast reception will not succeed.

The suffix starts with the special key /label: and is followed by the label of the desired file. (We retain at least the forward slashes in the path in the clear so that relative URLs work, but perhaps some characters and maybe some instances of slash should be escaped.)

Example:

`fcast://232.0.0.1:5620/broadcast.example.com:527353/start:35776638264/network:media.example.com/label:http://news.example.com/stuff.mp4`

given such a URL, the terminal can (try to) tune into the FCAST session, and retrieve the indicated file.

6. FCAST Metainformation field

In FCAST the metainformation field carries anything that an HTTP metainformation field can carry, but not the status line. This means it is not possible to indicate "this file might be expected here, but it is not here any more" (404) or "this file has moved" (301 or 307) or even that there are multiple choices on where to get this resource (300 'choices'). The most useful, perhaps, is the ability to indicate "you might have expected to get this over this multicast, but it's not here, but over there (re-direct)" perhaps even re-directing back to HTTP, or to another multicast session.

We therefore suggest we define a new form of the FCAST metainformation that also includes a status line formatted exactly as the HTTP status line, but with the HTTP-version replaced by FCAST-version:

Status-Line = FCAST-Version SP Status-Code SP Reason-Phrase CRLF

7. HTTP Status Codes and Their Applicability to FCAST

Here are the status codes available in HTTP 1.1, and a brief statement of whether they could be applicable to FCAST:

7.1. Useful

- o 200 OK: Usual status code when the object is supplied, or when just the metainformation is supplied
- o 203 Non-Authoritative Information
- o 206 Partial Content: Useful to indicate that byte-ranges of the resource are supplied separately
- o 300 Multiple Choices: Useful to indicate that there are also other places to get the content
- o 301 Moved Permanently: The resource might be expected here, but has moved (re-direct)
- o 302 Found
- o 303 See Other

- o 307 Temporary Redirect: The resource might be expected here, but has moved (re-direct)
- o 404 Not Found: The resource might be expected here, but it is no longer available
- o 410 Gone

7.2. Unlikely but Possible

- o 100 Continue: Unlikely to be of use
- o 101 Switching Protocols: Maybe useful
- o 502 Bad Gateway: The multicast was being fed by a gateway that failed
- o 503 Service Unavailable

7.3. Probably Inapplicable

- o 201 Created
- o 202 Accepted
- o 204 No Content
- o 205 Reset Content
- o 304 Not Modified
- o 305 Use Proxy
- o 400 Bad Request
- o 401 Unauthorized
- o 402 Payment Required
- o 403 Forbidden
- o 405 Method Not Allowed
- o 406 Not Acceptable
- o 407 Proxy Authentication Required
- o 408 Request Time-out

- o 409 Conflict
- o 411 Length Required
- o 412 Precondition Failed
- o 413 Request Entity Too Large
- o 414 Request-URI Too Large
- o 415 Unsupported Media Type
- o 416 Requested range not satisfiable
- o 417 Expectation Failed
- o 500 Internal Server Error
- o 501 Not Implemented
- o 504 Gateway Time-out
- o 505 HTTP Version not supported

8. Operation of the URL Handler

When the client-side URL handler gets the first URL for a given session, it would 'tune in that session' and (with luck) start receiving files and metainformation. On the receipt of 'special files' (e.g., an SDP) it can expand its knowledge of the session. Other files not corresponding to the immediate request in hand should be cached, observing the cache control headers. When the indicated file (or at least the requested byte-range of the indicated file) is available, it is returned. If a 404, 410, or 3xx response is received for the indicated file, then an appropriate error is returned, as indeed it is if the URL specifies that the multicast is only available over a given time range, and the request is not or cannot be satisfied in that time range.

9. Security Considerations

TBC.

10. IANA Considerations

This section contains the registration information for the "fcast" URI scheme (in accordance with Section 5.4 of [RFC4395]).

Editor's note: The registration template will be provided in a later revision.

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.
- [RFC6968] Roca, V. and B. Adamson, "FCAST: Object Delivery for the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 6968, July 2013.
- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

11.2. Informative References

- [RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, April 2010.
- [RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, October 2009.
- [RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, November 2012.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

Authors' Addresses

Dave Singer
Apple, Inc.
1 Infinite Loop
Cupertino 95014
USA

EMail: singer@apple.com

Ali Begen
Cisco
181 Bay Street
Toronto, ON M5J 2T3
Canada

EMail: abegen@cisco.com