

INTERNET-DRAFT
Intended Status: Standard Track
Expires: August 23, 2013

TS. Choi
ETRI
YI Seo
KT
JM Lee
SKT
JR Koo
LGU+
JDH Shinn
Solbox Inc.
YH Bang
KAIST
February 19, 2013

CDNi Control - Initialization and Bootstrapping
draft-choi-cdni-control-init-bootstrapping-00

Abstract

This document proposes a mechanism for a CDN to initiate the interconnection across CDNs and bootstrap the other Cdni interfaces.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

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

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

Table of Contents

1	Introduction	4
1.1	Terminology	6
2.	Model for CDNI Triggers	6
2.1.	Timing of Triggered Activity	8
2.2.	Trigger Results	8
3.	Collections of Trigger Status Resources	10
4.	CDNI Trigger interface	10
5.	Properties of Triggers	10
5.1.	Properties of Trigger Requests	10
5.2.	Properties of Trigger Status Resources	10
5.3.	Properties of Trigger Collections	12
5.4.	Trigger Resource Simple Data Type Descriptions	13
5.4.1.	TriggerType	13
5.4.2.	TriggerStatus	13
5.4.3.	General	13
5.4.4.	Contract	13
5.4.5.	Fp_cap	13
5.4.6.	Cdmd	14
5.4.7.	AbsoluteTime	14
6.	JSON Encoding of Objects	14
6.1.	JSON Encoding of Embedded Types	14
6.1.1.	TriggerType	14
6.1.2.	TriggerStatus	14
6.1.3.	General	14
6.1.4.	Fp_cap	14
6.1.5.	Cdmd	14
7.	Examples	15
7.1.	Initialize	15
7.2.	Add	17
7.3.	Update	18
7.4.	Remove	19

7.5. Add Log	21
8 Security Considerations	23
9 IANA Considerations	23
10 References	23
10.1 Normative References	23
10.2 Informative References	23
Authors' Addresses	24

1 Introduction

[I-D.ietf-murray-triggers-01] specifies mechanisms for CDN interconnection control for the "High" and "Medium" priority requirements for the CDNI Control Interface identified in section 4 of [I-D.ietf-cdni-requirements].

This draft concentrates on the remaining "Low" priority requirements for the CDNI Control Interface, reproduced here for convenience:

CNTL-5 [LOW] The CDNI Control interface may allow a CDN to establish, update and terminate a CDN interconnection with another CDN whereby one CDN can act as a Downstream CDN for the other CDN (that acts as an Upstream CDN).

CNTL-6 [LOW] The CDNI Control interface may allow control of the CDNI interconnection between any two CDNs independently for each direction (i.e. For the direction where CDN1 is the Upstream CDN and CDN2 is the Downstream CDN, and for the direction where CDN2 is the Upstream CDN and CDN1 is the Downstream CDN).

CNTL-7 [LOW] The CDNI Control interface may allow bootstrapping of the Request-Routing interface. For example, this can potentially include:

- * negotiation of the Request-Routing method (e.g. DNS vs HTTP, if more than one method is specified)
- * discovery of the Request-Routing protocol endpoints
- * information necessary to establish secure communication between the Request-Routing protocol endpoints.

CNTL-8 [LOW] The CDNI Control interface may allow bootstrapping of the CDNI Metadata interface. This information could, for example, include:

- * discovery of the CDNI Metadata signaling protocol endpoints
- * information necessary to establish secure communication between the CDNI Metadata signaling protocol endpoints.

CNTL-9 [LOW] The CDNI Control interface may allow bootstrapping of the Content Acquisition interface. This could, for example, include exchange and negotiation of the Content Acquisition protocols to be used across the CDNs (e.g. HTTP, HTTPS, FTP, ATIS C2).

CNTL-10 [LOW] The CDNI Control interface may allow exchange and negotiation of delivery authorization mechanisms to be supported across the CDNs (e.g. URI signature based validation).

CNTL-11 [LOW] The CDNI Control interface may allow bootstrapping of the CDNI Logging interface. This information could, for example, include:

- * discovery of the Logging protocol endpoints
 - * information necessary to establish secure communication between the Logging protocol endpoints
 - * negotiation/definition of the log file format and set of fields to be exported through the Logging protocol, with some granularity (e.g. On a per content type basis).
 - * negotiation/definition of parameters related to transaction Logs export (e.g., export protocol, file compression, export frequency, directory).

This document does not consider those parts of the control interface that relate to configuration, bootstrapping or authentication of CDN Interconnect interfaces.

- o Section 2 outlines the model for the Trigger Interface at a high level.
- o Section 3 describes collections of Trigger Resources.
- o Section 4 defines the RESTful web service provided by dCDN.
- o Section 5 lists properties of Trigger Requests and Status Resources.
- o Section 6 defines a JSON encoding for Trigger Requests and Status Resources.
- o Section 7 contains example messages.

1.1 Terminology

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

2. Model for CDNI Triggers

We follow the same model of CDNI triggers defined in section 2 of [I-D.ietf-murray-triggers-01] except that we define additional actions for CDNI control initiation and bootstrapping to meet the requirements described in section 1.

A trigger, sent between CDNs, is a request for the recipient CDN to do some work relating to data from originating CDN.

The trigger may request actions related with initiation and bootstrapping, the following actions can be requested:

- o initialize - used to instruct a CDN to initialize a CDN interconnection with another CDN at boot time.
- o add - used to instruct a CDN to add newly created footprint & capabilities(fp_cap) or content and metadata(cdmd).
- o update - used to instruct a CDN to update the existing fp_cap or cdmd when there are changes.
- o remove - used to instruct a CDN to remove the existing fp_cap or cdmd.
- o negotiate auth - used to instruct a CDN to negotiate delivery authorization mechanism.

The CDNI trigger interface is a RESTful web service offered between CDNs. It allows creation and deletion of triggers, and tracking of the triggered activity. When recipient CDN accepts a trigger it

creates a resource describing status of the triggered activity, a Trigger Status Resource. The originating CDN may poll Trigger Status Resources to monitor progress.

The recipient CDN maintains a collection of Trigger Status Resources for each originating CDN, each originating CDN only has access to its own collection and the location of that collection is shared when CDN interconnection is established.

To trigger activity in recipient CDN, originating CDN will POST to the collection of Trigger Status Resources. If recipient CDN accepts the trigger, it creates a new Trigger Status Resource and returns its location to originating CDN. To monitor progress, originating CDN may GET the Trigger Status Resource. To cancel a trigger, or remove a trigger from the collection once its activity has been completed, originating CDN may DELETE the Trigger Status Resource.

In addition to the collection of all Trigger Status Resources for originating CDN, it shall have access to filtered views of that collection. These filtered views are defined in Section 3 and include collections of active and completed triggers. These collections provide a mechanism for polling the status of multiple jobs.

Figure 1 is an example showing the basic message flow used by the originating CDN to trigger activity in recipient CDN, and for originating CDN to discover the status of that activity. Only successful triggering is shown. Examples of the messages are given in Section 7.

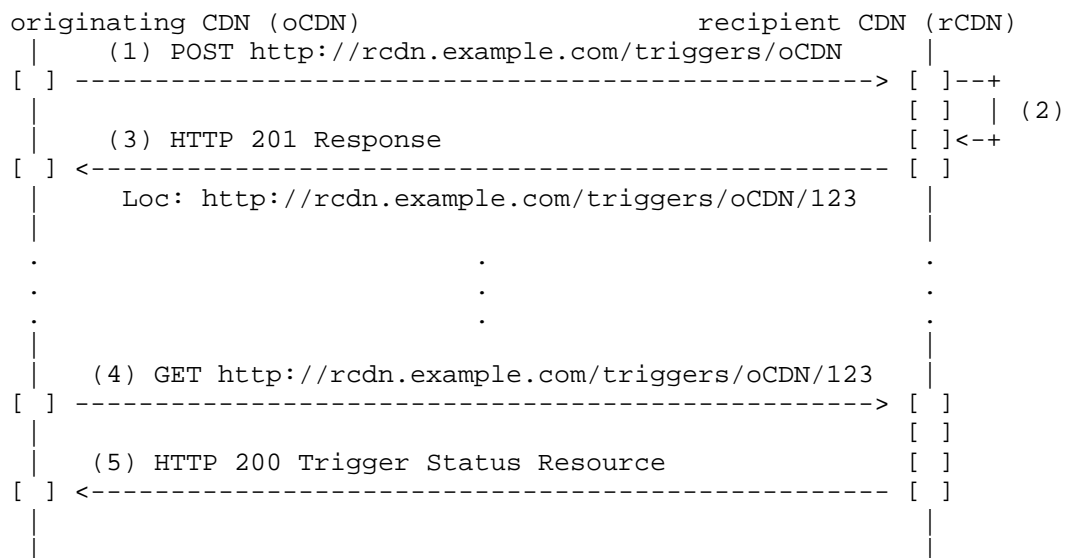


Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are:

1. originating CDN triggers action in recipient CDN by posting to a collection of Trigger Status Resources, "http://dcdn.example.com/triggers/originating CDN". The URL of this was given to originating CDN when the trigger interface was established.
2. recipient CDN authenticates the request, validates the trigger and if it accepts the request, creates a new Trigger Status Resource.
3. recipient CDN responds to originating CDN with an HTTP 201 response status, and the location of the Trigger Status Resource.
4. originating CDN may repeatedly poll the Trigger Status Resource in recipient CDN.
5. recipient CDN responds with the Trigger Status Resource, describing progress or results of the triggered activity.

The remainder of this document describes the messages, Trigger Status Resources, and collections of Trigger Status Resources in more detail.

2.1. Timing of Triggered Activity

Timing of triggered activity is under recipient CDN control, including its start-time and pacing of the activity in the network.

Establish, update and terminate triggers MUST be applied upon the trigger was created in recipient CDN. For other action triggers, recipient CDN MAY apply the triggers upon trigger creation.

2.2. Trigger Results

Each Trigger Request may operate on multiple data items, and may request different actions. The trigger shall be reported as successful only if all actions can be completed successfully.

If any part of the trigger request fails, the trigger shall be reported as failed, and the error property in the Trigger Status Resource will be used to enumerate which actions failed and the reasons for failure.

If a recipient CDN is also acting as uCDN in a cascade environment, it MUST forward triggers to any downstream CDNs that may have data affected by the trigger. The trigger MUST NOT be reported as complete in a CDN until it is complete in all of its downstream CDNs. A trigger MAY be reported as failed as soon as it fails in a CDN or

in any of its downstream CDNs.

3. Collections of Trigger Status Resources

It follows the same mechanism defined in section 3 of [I-D.ietf-murray-triggers-01].

4. CDNI Trigger interface

An interface to enable an originating CDN to trigger defined activities in a recipient CDN is the same as defined in section 4 of [I-D.ietf-murray-triggers-01].

5. Properties of Triggers

5.1. Properties of Trigger Requests

Properties of Trigger Requests for initialization and bootstrapping are defined in the following subsections.

Property: type

Description: This property defines the type of the trigger:

Type: TriggerType

Mandatory: Yes

Property: cdni.general

Description: The general information used to initialize CDN interconnection at boot time.

Type: General

Mandatory: Yes

Property: cdni.contracts

Description: The contract information used to initialize CDN interconnection at boot time.

Type: Contract

Mandatory: Yes

Property: cdni.fp_caps

Description: Footprint and capabilities that need to be added, updated, or removed.

Type: Fp_cap

Mandatory: Yes

Property: cdni.cdmds

Description: Contents and metadata that need to be added, updated, or removed.

Type: Cdmd

Mandatory: Yes

5.2. Properties of Trigger Status Resources

Property: trigger

Description: The properties of trigger request that created this record.

Type: TriggerRequest
Mandatory: Yes

Property: ctime
Description: Time at which the request was received by recipient CDN.
Time is local to recipient CDN, there is no requirement to synchronize clocks between interconnected CDNs.
Type: AbsoluteTime
Mandatory: Yes

Property: mtime
Description: Time at which the resource was last modified.
Time is local to recipient CDN, there is no requirement to synchronize clocks between interconnected CDNs.
Type: AbsoluteTime
Mandatory: Yes

Property: etime
Description: Estimate of the time at which recipient CDN expects to complete the activity. Time is local to recipient CDN, there is no requirement to synchronize clocks between interconnected CDNs.
Type: AbsoluteTime
Mandatory: No

Property: status
Description: Current status of the triggered activity.
Type: TriggerStatus
Mandatory: Yes

Property: error
Description: Error indication.
Type: (To be decided - a set of standard error conditions needs to be defined. The namespace for these errors codes should allow vendor-defined error codes for extension of the protocol. This may allow, for example, for the definition of more specific error codes when two CDNs supplied by the same vendor are interconnected.)
Mandatory: No, and only allowed when "status" is "Failed".

5.3. Properties of Trigger Collections

Property: links
Description: References to Trigger Status Resources in the collection.
Type: List of Relationships.
Mandatory: Yes

Property: staleresourcetime

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

Type: Integer, time in seconds.

Mandatory: Yes, in the collection of all Trigger Status Resources if recipient CDN deletes stale entries.

If the property is present in the filtered collections, it MUST have the same value as in the collection of all Trigger Status Resources.

5.4. Trigger Resource Simple Data Type Descriptions

This section describes the simpler data types that are used for properties of Trigger Status resources.

5.4.1. TriggerType

This type defines the type of action being triggered, permitted actions are initialize, add, update, and negotiate

5.4.2. TriggerStatus

This type describes the current status of a Trigger, possible values are:

- o Pending - the trigger has not yet been acted upon.
- o Active - the trigger is currently being acted upon.
- o Complete - the triggered activity completed successfully.
- o Failed - the triggered activity could not be completed.

5.4.3. General

This type describes a set of general information of a CDN, possible values are cdnname, cdnid, and cdnhostname.

5.4.4. Contract

This type describes a set of CDN information where interconnection contract is established. It consists of cdnname, cdnid, cdnhostname, ucdnflag, and dcdnflag. xcdnflag indicates whether it is capable of either uCDN or dCDN.

5.4.5. Fp_cap

This type describes a set of footprint and capabilities information

needed for bootstrapping. It consists of cdnname, iprange, delay, load, and bandwidth. Note that some attributes of this type need to be aligned with ones defined in the "draft-spp-cdni-rr-foot-cap-semantics" draft.

5.4.6. Cdmd

This type describes a set of contents and metadata information needed for bootstrapping. It consists of hostindex, hostmetadatas, pathmetadatas. Note that attributes of this type is aligned with ones defined in "draft-ietf-cdni-metadata" draft.

5.4.7. AbsoluteTime

Times are expressed in seconds since the UNIX epoch.

6. JSON Encoding of Objects

This encoding is based on that described in [I-D.ietf-murray-triggers-01] and only new types specific to this draft are added.

6.1. JSON Encoding of Embedded Types

6.1.1. TriggerType

Description: One of "initialize", "add", "update", "remove" or "negotiate". Type: string Mandatory: Yes

6.1.2. TriggerStatus

Key: status Description: One of "pending", "active", "failed", "complete" Type: string Mandatory: Yes

6.1.3. General

Keys: general Description: A set of general information of a CDN. Type: General Mandatory: Yes

6.1.4. Fp_cap

Keys: fp_cap Description: A set of footprint and capability information needed for bootstrapping. Type: fp_cap Mandatory: Yes

6.1.5. Cdmd

Keys: cdmd Description: A set of content and metadata information

needed for bootstrapping. Type: cdmd Mandatory: Yes

7. Examples

The following sections provide examples of different CDNI Trigger objects encoded as JSON.

7.1. Initialize

```
REQUEST - init
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: xx
{
  "trigger" : {
    "type": "initialize",
    "cdni.general" : [
      {
        "cdnname": "UCDN1",
        "cdnid": 100000,
        "cdnhostname": "ucdn.example.com"
      }
    ],
    "cdni.contract" : [
      {
        "cdnname": "DCDN1",
        "cdnid": 200000,
        "cdnhostname": "dcdn.example.com"
        "dcdnflag": 1
        "ucdnflag": 0
      }
    ],
    "cdni.fp_cap" : [
      {
        "cdnname": "UCDN1",
        "iprange": 0.0.0.0/32,
        "delay": 10,
        "load": 30,
        "bandwidth": 20
      },
      {
        "cdnname": "UCDN1",
        "iprange": 211.224.204.0/24,
        "delay": 10,
        "load": 30,
        "bandwidth": 20
      }
    ]
  }
}
```

```

    }
  ],
  "cdni.cdmd" : [
    {
      "HostIndex": [
        {
          "hosts": [
            {
              "host": "video.example.com",
              "links": [
                {
                  "rel": "host-metadata",
                  "type": "application/cdni.HostMetadata",
                  "href": "http://metadata.example.ucdn.co
m/video"
                }
              ]
            },
            {
              "host": "images.example.com",
              "links": [
                {
                  "rel": "host-metadata",
                  "type": "application/cdni.HostMetadata",
                  "href": "http://metadata.ucdn.example.co
m/images"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}

```

RESPONSE:

HTTP/1.1 201 Created

Date: Sat, 23 Feb 2013 14:20:08 GMT

Content-Type: application/vnd.cdni.control.trigger.status+json

Location: http://dcdn.example.com/triggers/1

Server: example-server/0.1

Content-Length: xx

```

{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
}

```



```

    "trigger": {
      "cdni.general" [ .. ],
      "cdni.contract" [ .. ],
      "cdni.fp_cap" [ .. ],
      "cdni.cdmd" [ .. ],
      "type": "initialize"
    }
  }

```

7.2. Add

```

REQUEST - add
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: xx
{
  "trigger" : {
    "type": "add",
    "cdni.fp_cap" : [
      {
        "cdnname": "UCDN1",
        "iprange": 211.224.205.0/24,
        "delay": 30,
        "load": 10,
        "bandwidth": 70
      }
    ],
    "cdni.cdmd" : [
      {
        "HostIndex": [
          {
            "hosts": [
              {
                "host": "streaming.example.com",
                "links": [
                  {
                    "rel": "host-metadata",
                    "type": "application/cdni.HostMetad
ata",
                    "href": "http://metadata.ucdn.examp
le.com/streaming"
                  }
                ]
              }
            ]
          }
        ]
      }
    ]
  }
}

```

```
    ]
  }
}
```

RESPONSE:

HTTP/1.1 201 Created

Date: Sat, 23 Feb 2013 14:20:08 GMT

Content-Type: application/vnd.cdni.control.trigger.status+json

Location: http://dcdn.example.com/triggers/2

Server: example-server/0.1

Content-Length: xx

```
{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
  "trigger": {
    "cdni.fp_cap" [ .. ],
    "cdni.cdmd" [ .. ],
    "type": "add"
  }
}
```

7.3. Update

REQUEST - update

POST /triggers HTTP/1.1

User-Agent: example-user-agent/0.1

Host: dcdn.example.com

Accept: */*

Content-Type: application/vnd.cdni.control.trigger.request+json

Content-Length: xx

```
{
  "trigger" : {
    "type": "update",
    "cdni.fp_cap" : [
      {
        "cdnname": "UCDN1",
        "iprange": 211.224.205.0/24,
        "delay": 5,
        "load": 10,
        "bandwidth": 60
      }
    ],
    "cdni.cdmd" : [
      {
        "HostIndex": [
          {
```

```

        "hosts": [
            {
                "host": "streaming.example.com",
                "links": [
                    {
                        "rel": "host-metadata",
                        "type": "application/cdni.HostMetadata",
                        "href": "http://metadata.ucdn.example.co
m/streaming1"
                    }
                ]
            }
        ]
    }
}

```

RESPONSE:

HTTP/1.1 201 Created

Date: Sat, 23 Feb 2013 14:20:08 GMT

Content-Type: application/vnd.cdni.control.trigger.status+json

Location: http://dcdn.example.com/triggers/3

Server: example-server/0.1

Content-Length: xx

```

{
    "ctime": 1361629208,
    "etime": 1361629216,
    "mtime": 1361629208,
    "status": "pending",
    "trigger": {
        "cdni.fp_cap" [ .. ],
        "cdni.cdmd" [ .. ],
        "type": "update"
    }
}

```

7.4. Remove

REQUEST - remove

POST /triggers HTTP/1.1

User-Agent: example-user-agent/0.1

Host: dcdn.example.com

Accept: */*

Content-Type: application/vnd.cdni.control.trigger.request+json

Content-Length: xx

```

{

```

```
"trigger" : {
  "type": "remove",
  "cdni.fp_cap" : [
    {
      "cdnname": "UCDN1",
      "iprange": 211.224.205.0/24,
      "delay": 5,
      "load": 10,
      "bandwidth": 60
    }
  ],
  "cdni.cdmd" : [
    {
      "HostIndex": [
        {
          "hosts": [
            {
              "host": "streaming.example.com",
              "links": [
                {
                  "rel": "host-metadata",
                  "type": "application/cdni.HostMetadata",
                  "href": "http://metadata.ucdn.example.co
m/streaming1"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

RESPONSE:

HTTP/1.1 201 Created

Date: Sat, 23 Feb 2013 14:20:08 GMT

Content-Type: application/vnd.cdni.control.trigger.status+json

Location: http://dcdn.example.com/triggers/4

Server: example-server/0.1

Content-Length: xx

```
{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
  "trigger": {
```

```
    "cdni.fp_cap" [ .. ],
    "cdni.cdmd" [ .. ],
    "type": "remove"
  }
}
```

7.5. Add Log

```
REQUEST - add (log)
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: xx
{
  "trigger" : {
    "type": "add",
    "cdni.loginfo" : [
      {
        "fileformat": "text",
        "encoding": "UTF-8",
        "granularity": "content",
        "protocol": "ftp",
        "compression": "zip",
        "loc": "http://ucdn.example.com/triggers/log",
        "frequency": 60
      }
    ]
  }
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
Location: http://dcdn.example.com/triggers/5
Server: example-server/0.1
Content-Length: xx
```

```
{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
  "trigger": {
    "cdni.loginfo" [ .. ],
    "type": "add"
  }
}
```

}

8 Security Considerations

The recipient CDN must ensure that each originating CDN only has access to its own Trigger Status Resources.

It is anticipated that a common authentication mechanism will be used by this and other CDNI Interconnect interfaces, the mechanism must exist but is not identified in this document.

The recipient CDN must ensure that activity triggered by originating CDN only affects metadata or content originating from that originating CDN.

9 IANA Considerations

TBD.

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.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [I-D.murray-triggers-01] Murray, R., Niven-Jenkins, B, "CDN Interconnect Triggers", draft-murray-cdni-triggers-01 (work in progress), August 30, 2012.

10.2 Informative References

- [I-D.cjlmw-cdni-metadata] Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., and K. Leung, "CDN Interconnect Metadata", draft-cjlmw-cdni-metadata-00 (work in progress), July 2012.
- [I-D.ietf-cdni-framework] Peterson, L. and B. Davie, "Framework for CDN

Interconnection", draft-ietf-cdni-framework-01 (work in progress), July 2012.

[I-D.ietf-cdni-problem-statement]

Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-ietf-cdni-problem-statement-08 (work in progress), June 2012.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[XML-BASE]

Marsh, J., Ed. and R. Tobin, Ed., "XML Base (Second Edition) - <http://www.w3.org/TR/xmlbase/>", January 2009.

Authors' Addresses

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

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

Young-IL Seo
KT Network R&D Laboratory
463-1, Jeonmoin-dong,
Yuseong-gu, Daejeon
Republic of Korea

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

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

Republic of Korea

Phone: 82-10-9429-6260

Email: jminlee@sk.com

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

Phone: 82-10-8080-6115

Email: wjbkoo@lguplus.co.kr

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

Phone: +82-10-3005-4785

Email: eastsky@solbox.com

Yong Hwan Bang
KAIST
8F, N29 Bldg. 373-1 Gusung-Dong
Yousong-Gu, Daejeon
Republic of Korea

Phone: +82-42-350-7516

Email: yjhvyp@gmail.com

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

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

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

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on July 28, 2013.

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

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

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

2. Experimental Setup

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

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

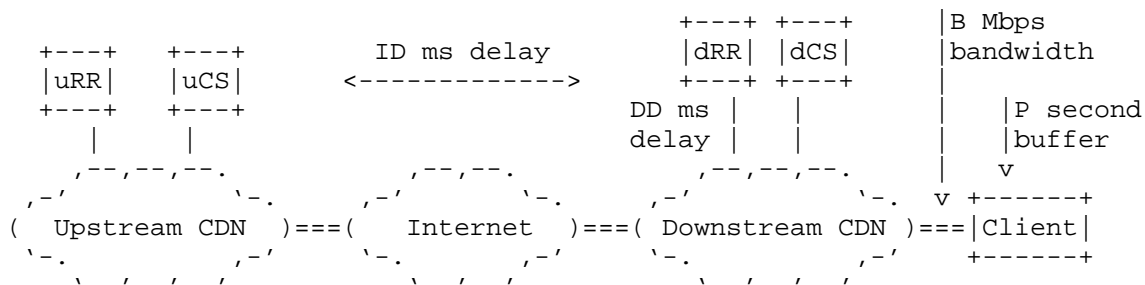


Figure 1: Evaluation scenario and parameters

Three alternative request routing policies are evaluated and compared:

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

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

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

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

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

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

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

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

3. Results

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

3.1. Congested Scenario

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

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

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

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

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

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

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

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

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

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

3.2. Uncongested Scenario

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

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

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

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

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

3.3. Influence of Segment Duration

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

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

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

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

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

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

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

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

4. Conclusion

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

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

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

5. Security Considerations

Not applicable.

6. References

6.1. Normative References

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

6.2. Informative References

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

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

Authors' Addresses

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

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

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

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2013

Danhua. Wang, Ed.
Huawei Technologies
B. Niven-Jenkins, Ed.
Velocix (Alcatel-Lucent)
Xiaoyan. He
Spencer. Dawkins
Huawei
Chen. Ge
China Telecom
Wei. Ni
Yunfei. Zhang
China Mobile
February 23, 2013

Routing Request Redirection for CDN Interconnection
draft-he-cdni-routing-request-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 August 27, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. Interface function and operation overview	4
3.1. Discussion on protocol type for CDNI RRRI	5
4. HTTP based RESTful interface for the Redirection Interface . .	6
4.1. Information passed in RI requests & responses	8
4.2. JSON encoding of RI requests & responses	9
4.3. DNS redirection	11
4.3.1. DNS Redirection requests	11
4.3.2. DNS Redirection responses	12
4.4. HTTP Redirection	13
4.4.1. HTTP Redirection requests	13
4.4.2. HTTP Redirection responses	14
4.5. Indicating the cacheability and scope of responses	15
4.6. Error responses	17
4.7. Loop detection & prevention	18
5. Security Considerations	18
6. IANA Considerations	19
7. Acknowledgements	19
8. Outstanding considerations	19
9. References	20
9.1. Normative References	20
9.2. Informative References	20
Authors' Addresses	20

1. Introduction

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

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

Several real world use cases are described in [RFC6770] which prove the necessity for CDN interconnection. The most frequently mentioned use case is leveraging the collective CDN footprint of interconnected standalone CDNs to achieve the goal of delivering content to additional distributed end users regardless of their location.

[RFC6707] describes the problem area, where CDNs are interconnected as described in [RFC6770] based on the requirements described in [I-D.ietf-cdni-requirements], and using the technology framework described in [I-D.ietf-cdni-framework].

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

2. Terminology

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

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

The following additional terms are introduced by this document:

Application Level Redirection: The act of using an application specific redirection mechanism for the request routing process of a

CDN. The Redirection Target (RT) is the result of the routing decision of a CDN at the time it receives a content request via an application specific protocol response. Examples of an application level redirection are HTTP and RTMP 302 redirections.

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.

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

3. Interface function and operation overview

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

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

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

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

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

queries to.

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

CDNI solutions must support both of the request routing mechanisms illustrated in section 2.1 of [I-D.ietf-cdni-framework], namely Iterative Request Routing and Recursive Request Routing. However, the Iterative Request Routing 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 Routing and so this document will not discuss Iterative Request Routing further.

In the case of Recursive Request Routing, an Upstream CDN forwards a CDNI request routing redirection request from a user agent to a Downstream CDN for the Downstream CDN to select a Redirection Target.

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

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

3.1. Discussion on protocol type for CDNI RRRI

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

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

The possible combinations are shown in Table 1.

Case	uCDN Received Request	dCDN Response	Notes
1	DNS	DNS with IP address/hostname of RT(s)	dCDN uses DNS redirection (via the RI interface) to redirect the UA to one or more RTs (Surroagtes or Request Routers).
2	DNS	DNS with IP address/hostname of itself or another RR as the RT(s)	dCDN uses DNS redirection (via the RI interface) to redirect the UA to a RR (either in the dCDN or in another CDN), which then performs a HTTP redirection to redirect the UA to one or more RTs.
3	HTTP	HTTP 302 redirection	dCDN uses HTTP redirection mode to redirect the UA to a RT.

Recursive Routing Cases

4. HTTP based RESTful interface for the Redirection Interface

This document defines a simple HTTP based RESTful interface for the Redirection Interface, where the attributes of User Agent's requests are encapsulated along with any other data that can aid the downstream CDN in processing the requests. The RI response encapsulates the attributes of the RT(s) that the upstream CDN should return to the User Agent (if it decides to utilize the Downstream CDN for delivery) along with the policy for how the response can be reused.

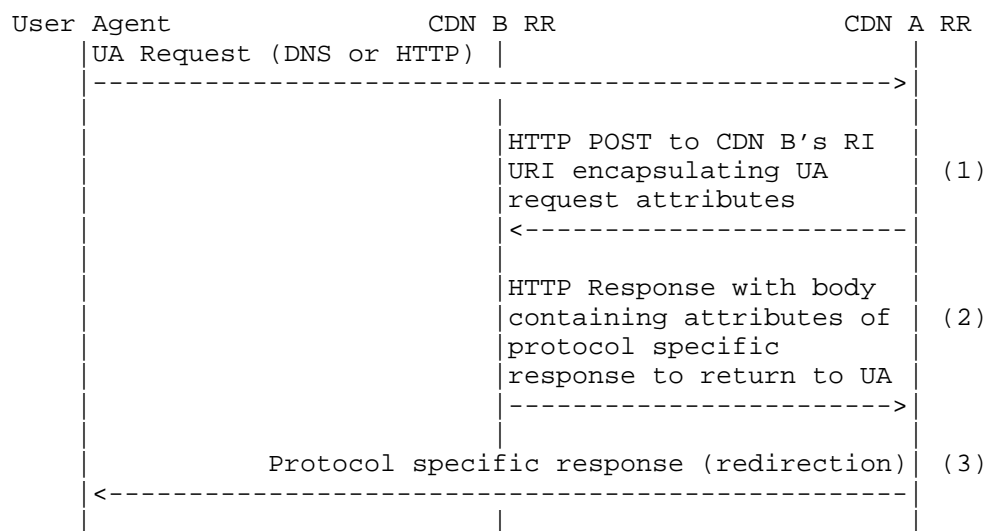
The same RESTful interface is used for both DNS and HTTP redirection of User Agent's requests, although the contents of the RI requests/responses contain data specific to either DNS or HTTP redirection.

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

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

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

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



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

4. The Request Routing System of CDN A sends a protocol specific response (containing the returned attributes) to the User Agent, so that the User Agent's request will be redirected to the RT(s) returned by CDN B.

4.1. Information passed in RI requests & responses

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

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

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

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

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

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

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

These additional policies split into two basic categories:

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

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

4.2. JSON encoding of RI requests & responses

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

The following keys are defined:

Key	Request/Response	Description
dns	Both	The attributes of the UA's DNS request or the attributes of the RT(s) to return in a DNS response.
http	Both	The attributes of the UA's HTTP request or the attributes of the RT to return in a HTTP response.
scope	Response	The scope of the response (if it is cacheable). For example whether the response applies to a wider range of IP addresses than what was included in the RI request.
error	Response	Additional details if the response is an error response.

cdn-path	Request	A List of Strings. Contains the CDN Provider IDs of previous CDNs this RI request has passed through. When cascading a RI request the transit CDN appends its own CDN Provider ID to the list in cdn-path so that downstream CDNs can detect loops in the RI request chain. Transit CDNs should check the cdn-path and not cascade the RI request to downstream CDNs that are already listed in cdn-path.
max-hops	Request	Integer specifying the Maximum Number of hops (CDN Provider IDs) this request is allowed to be propagated along. This allows the uCDN to crudely constrain the latency of the request routing chain.

Top-Level keys in RI requests/responses

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

[[Editor's note: Is this too inflexible? Are they use cases for including both dns & http keys in a RI request? Maybe a use case is responses is if the dCDN wants to indicate it is prepared to have a DNS redirection directly to it or a HTTP redirection from the uCDN RR to the dCDN and wants to indicate both options in a single response?]]

[[Editor's note: Is the cdn-path useful on responses as well?]]

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

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

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

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

[RFC5952].

4.3. DNS redirection

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

4.3.1. DNS Redirection requests

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

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

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

Key	Value	Mandatory	Description
resolver-ip	String	Yes	The IP address of the UA's DNS resolver.
qtype	String	Yes	The type of DNS query made by the UA's DNS resolvers in uppercase (A, AAAA, etc.).
qclass	String	Yes	The class of DNS query made in uppercase (IN, etc.).
qname	String	Yes	The fully qualified domain name being queried.
c-subnet	String	No	The IP address of the UA in CIDR format.

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

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

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

4.3.2. DNS Redirection responses

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

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

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

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

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

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

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

HTTP/1.1 200 OK

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

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

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

4.4. HTTP Redirection

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

4.4.1. HTTP Redirection requests

For HTTP based redirection the uCDN 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.

[[Editor's note: What other attributes of the request should we include? METHOD? Version? other headers?]]

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

Key	Value	Mandatory	Description
c-ip	String	Yes	The IP address of the UA/client

cs-uri String Yes	The URI requested by the UA/client.
-----------------------	-------------------------------------

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

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

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

4.4.2. HTTP Redirection responses

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

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

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

Key	Value	Mandatory	Description
sc-status	Integer	Yes	The status code of the HTTP response to return to the UA (usually 302).
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.
+-----+-----+-----+-----+			

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

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

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

4.5. Indicating the cacheability and scope of responses

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

Cacheability is via the standard HTTP Cache-Control mechanisms.

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

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

[[Editor's note: Maybe add some kind of way to indicate a wider scope on the UA URI, which could be useful in the case where a dCDN wishes to HTTP 302 redirect to its RR first in order to avoid load on the RI interface.]]

Example of DNS redirection response from Section 4.3.2 that is cacheable by the uCDN for 60 seconds and can be returned to any User

Agent with an IPv4 address in 198.51.100.0/16.

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

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

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

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

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


4.6. Error responses

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

RI error response examples.

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

HTTP/1.1 500 Server Error

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

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

Cache-Control: private, no-cache

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

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

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

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

4.7. Loop detection & prevention

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

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

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

5. Security Considerations

[[Editor's note: Not sure if this current text is really security

considerations or whether it is better placed elsewhere in the document.]]

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

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

6. IANA Considerations

This document makes no request of IANA.

7. Acknowledgements

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

8. Outstanding considerations

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

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

9. 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.

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.
- [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.
- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-03 (work in progress), February 2013.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-04 (work in progress), December 2012.
- [I-D.peterson-cdni-strawman]
Peterson, L. and J. Hartman, "A Simple Approach to CDN Interconnection", draft-peterson-cdni-strawman-01 (work in progress), May 2011.

Authors' Addresses

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

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

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

Email: ben@velocix.com

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

Email: hexiaoyan@huawei.com

Spencer Dawkins
Huawei

Email: spencer.dawkins@wondermaster.com

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

Email: cheng@gsta.com

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

Email: niwei@chinamobile.com

Zhang Yunfei
China Mobile

Email: zhangyunfei@chinamobile.com

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

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

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on October 02, 2013.

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

2. Terminology

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

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

The following additional terms are introduced by this document:

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

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

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

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

3. Interface function and operation overview

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

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

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

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

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

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

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

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

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

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

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

4. HTTP based RESTful interface for the Redirection Interface

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

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

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

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

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

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

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

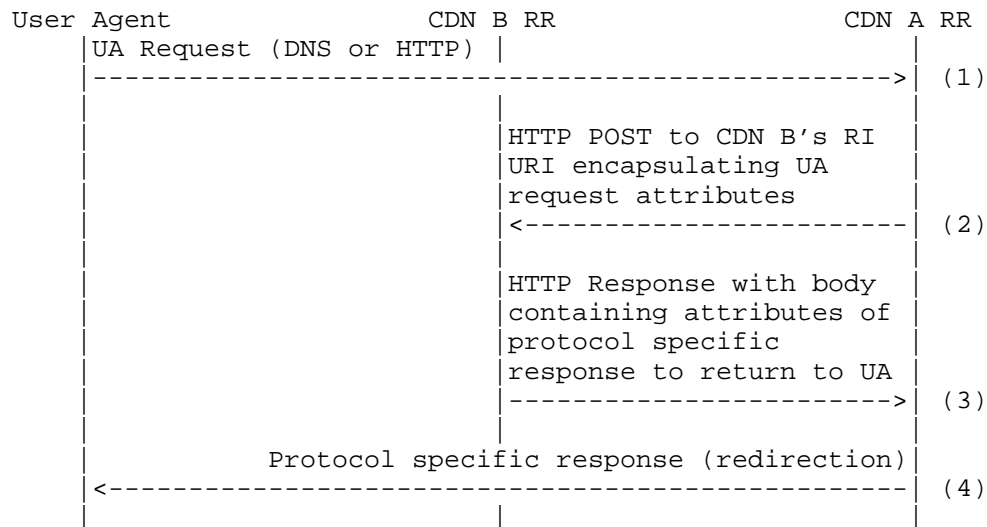


Figure 1: Generic Recursive Request Redirection message flow

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

4.1. Information passed in RI requests & responses

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

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

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

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

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

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

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

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

These additional policies split into two basic categories:

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

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

4.2. JSON encoding of RI requests & responses

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

The following keys are defined:

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


```

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

```

Top-Level keys in RI requests/responses

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

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

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

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

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

4.3. DNS redirection

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

4.3.1. DNS Redirection requests

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

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

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

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

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

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

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

4.3.2. DNS Redirection responses

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

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

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

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

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

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

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

HTTP/1.1 200 OK

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

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

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

```

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

```

4.4. HTTP Redirection

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

4.4.1. HTTP Redirection requests

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

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

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

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

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

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

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

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

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

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

4.4.2. HTTP Redirection responses

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

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

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

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

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

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

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

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

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

4.5. Indicating the cacheability and scope of responses

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

Cacheability is via the standard HTTP Cache-Control mechanisms.

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

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

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

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

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

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

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

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

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

4.6. Error responses

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

RI error response examples.

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

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

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

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

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

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

4.7. Loop detection & prevention

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

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

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

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

5. Security Considerations

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

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

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

6. IANA Considerations

This document makes no request of IANA.

7. Acknowledgements

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

8. Outstanding considerations

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

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

9. Contributing Authors

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

Spencer Dawkins
Huawei

Email: spencer@wonderhamster.org

Yunfei Zhang

Email: hishigh@gmail.com

10. References

10.1. Normative References

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

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

10.2. Informative References

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

Authors' Addresses

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

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

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

Email: ben@velocix.com

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

Email: hexiaoyan@huawei.com

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

Email: cheng@gsta.com

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

Email: niwei@chinamobile.com

Network Working Group
Internet-Draft
Obsoletes: 3466 (if approved)
Intended status: Informational
Expires: August 22, 2013

L. Peterson, Ed.
Akamai Technologies, Inc.
B. Davie
VMware, Inc.
February 18, 2013

Framework for CDN Interconnection
draft-ietf-cdni-framework-03

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on August 22, 2013.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	4
1.1. Terminology	4
1.2. Reference Model	5
1.3. Structure Of This Document	9
2. Building Blocks	9
2.1. Request Redirection	9
2.1.1. DNS Redirection	9
2.1.2. HTTP Redirection	10
3. Overview of CDNI Operation	11
3.1. Preliminaries	13
3.2. Iterative HTTP Redirect Example	14
3.3. Recursive HTTP Redirection Example	19
3.4. Iterative DNS-based Redirection Example	23
3.5. Dynamic Footprint Discovery Example	27
3.6. Content Removal Example	29
3.7. Pre-Positioned Content Acquisition Example	29
3.8. Asynchronous CDNI Metadata Example	31
3.9. Synchronous CDNI Metadata Acquisition Example	33
3.10. Content and Metadata Acquisition with Multiple Upstream CDNs	35
4. Main Interfaces	36
4.1. In-Band versus Out-of-Band Interfaces	37
4.2. Cross Interface Concerns	37
4.3. Request Routing Interface	38
4.4. Logging Interface	39
4.5. Control Interface	41
4.6. Metadata Interface	41
4.7. HTTP Adaptive Streaming Concerns	42
5. Deployment Models	43
5.1. Meshed CDNs	44
5.2. CSP combined with CDN	45
5.3. CSP using CDNI Request Routing Interface	46
5.4. CDN Federations and CDN Exchanges	47
6. Trust Model	50
7. IANA Considerations	51
8. Security Considerations	51
8.1. Security of CDNI Interfaces	52
8.2. Digital Rights Management	53
9. Contributors	53
10. Acknowledgements	53
11. Informative References	53
Authors' Addresses	55

1. Introduction

The interconnection of Content Distribution Networks (CDNs) is motivated by several use cases, such as those described in [I-D.ietf-cdni-use-cases]. The overall problem space for CDN Interconnection is described in RFC 6707. The purpose of this document is to provide an overview of the various components necessary to interconnect CDNs. CDN Interconnection requires the specification of several interfaces and mechanisms to address issues such as request routing, metadata exchange, and the acquisition of content by one CDN from another. The intent of this document is to describe how these interfaces and mechanisms fit together, leaving their detailed specification to other documents. We make extensive use of message flow examples to illustrate the operation of interconnected CDNs, but these examples should be considered illustrative rather than prescriptive.

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

1.1. Terminology

This document draws freely on the core terminology defined in RFC 6707. It also introduces the following terms:

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

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

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

Recursive CDNI Request Redirection: When an Upstream CDN elects to redirect a request towards a Downstream CDN, the Upstream CDN can query the Downstream CDN Request Routing system via the CDNI Request

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

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

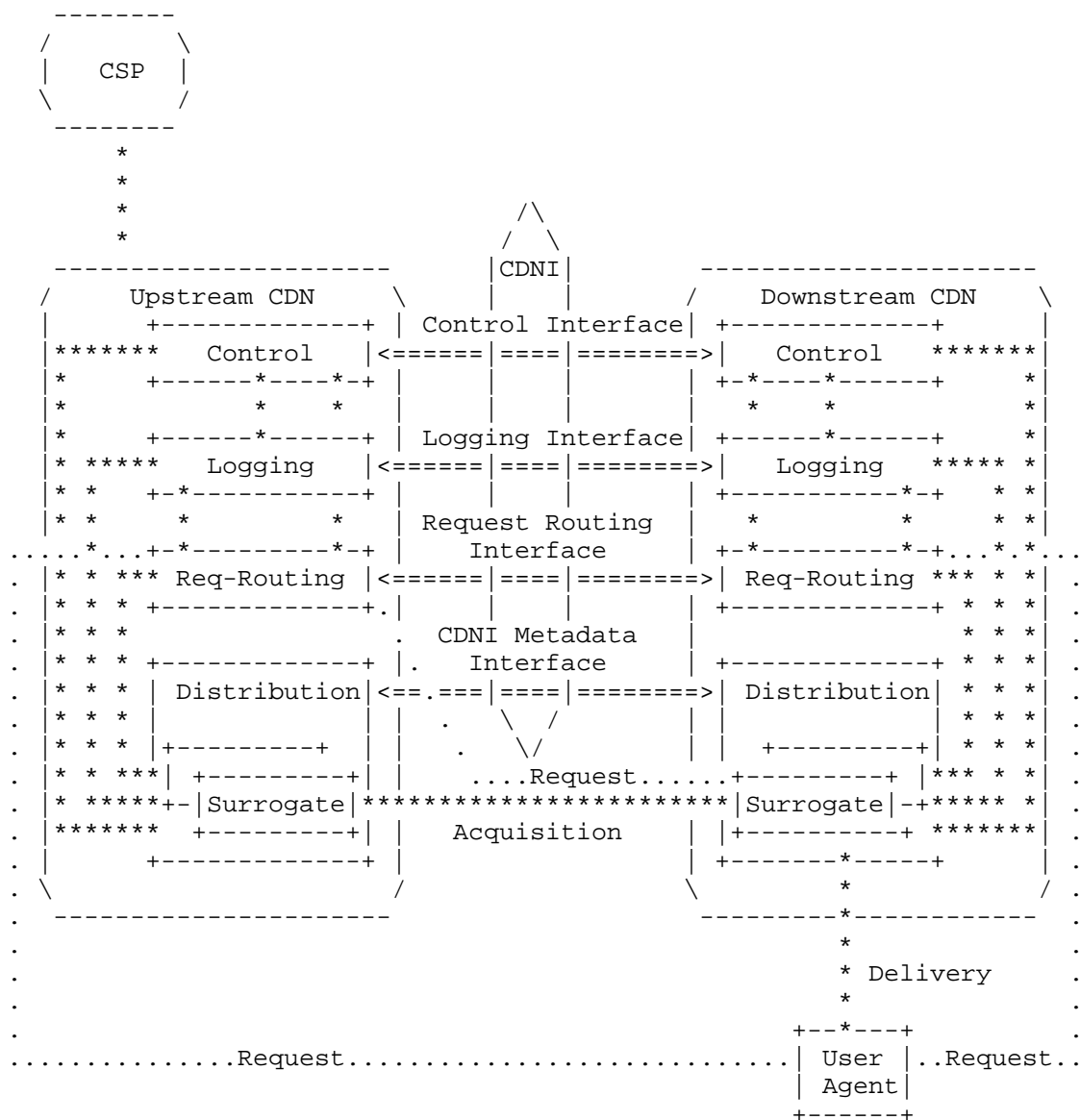
Synchronous CDNI operations: operations between CDNs that happen during the process of servicing a user request, i.e. between the time that the user agent begins its attempt to obtain content and the time at which that request is served.

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

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

1.2. Reference Model

This document uses the reference model in Figure 1 as originally created in RFC 6707.



\Leftarrow interfaces inside the scope of CDNI

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

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

Figure 1: CDNI Model and CDNI Interfaces

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

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

The following briefly describes the four CDNI interfaces, paraphrasing the definitions given in RFC 6707. We discuss these interfaces in more detail in Section 4.

- o CDNI Control Interface (CI): Operations to bootstrap and parameterize the other CDNI interfaces, as well as operations to pre-position, revalidate, and purge both metadata and content. The latter sub-set of operations is sometimes collectively called the "trigger interface."
- o CDNI Request Routing Interface: Operations to determine what CDN (and optionally what surrogate within a CDN) is to serve end-user's requests. Is actually a logical bundling of two separate but related interfaces:
 - * Footprint & Capability Interface (FCI): Asynchronous operations to exchange routing information (e.g., the network footprint and capabilities served by a given CDN) that enables CDN selection for subsequent user requests; and
 - * Request Routing Redirection (RI): Synchronous operations to select a delivery CDN (surrogate) for a given user request.
- o CDNI Metadata Interface (MI): Operations to communicate metadata that governs the how content is delivered by interconnected CDNs. Examples of CDNI metadata include geo-blocking directives, availability windows, access control mechanisms, and purge directives. May include a combination of:
 - * Asynchronous operations to exchange metadata that govern subsequent user requests for content; and
 - * Synchronous operations that govern behavior for a given user request for content.

- o CDNI Logging Interface (LI): Operations that allow interconnected CDNs to exchange relevant activity logs. May include a combination of:
 - * Real-time exchanges, suitable for runtime traffic monitoring; and
 - * Off-line exchanges, suitable for analytics and billing.

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

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

Finally, there is a practical issue that impacts all of the CDNI interfaces, and that is whether or not to optimize CDNI for HTTP Adaptive Streaming (HAS). We highlight specific issues related to delivering HAS content throughout this document, but for a more thorough treatment of the topic, see [I-D.brandenburg-cdni-has].

1.3. Structure Of This Document

The remainder of this document is organized as follows:

- o Section 2 describes some essential building blocks for CDNI, notably the various options for redirecting user requests to a given CDN.
- o Section 3 provides a number of illustrative examples of various CDNI operations.
- o Section 4 describes the functionality of the four main CDNI interfaces.
- o Section 5 shows how various deployment models of CDNI may be achieved using the defined interfaces.
- o Section 6 describes the trust model of CDNI and the issues of transitive trust in particular that CDNI raises.

2. Building Blocks

2.1. Request Redirection

At its core, CDN Interconnection requires the redirection of requests from one CDN to another. For any given request that is received by an upstream CDN, it will either respond to the request directly, or somehow redirect the request to a downstream CDN. Two main mechanisms are available for redirecting a request to a downstream CDN. The first leverages the DNS name resolution process and the second uses in-protocol redirection mechanisms such as the HTTP 302 or 307 redirection response. We discuss these below as background before discussing some examples of their use in Section 3.

2.1.1. DNS Redirection

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

The advantage of DNS redirection is that it is completely transparent to the end user--the user sends a DNS name to the LDNS server and

gets back an IP address. On the other hand, DNS redirection is problematic because the DNS request comes from the LDNS server, not the end-user. This may affect the accuracy of server selection that is based on the user's location. The transparency of DNS redirection is also a problem in that there is no opportunity to take the attributes of the user agent or the URI path component into account. We consider two main forms of DNS redirection: simple and CNAME-based.

In simple DNS redirection, the authoritative DNS server for the name simply returns an IP address from a set of possible IP addresses. The answer is chosen from the set based on characteristics of the set (e.g., the relative loads on the servers) or characteristics of the client (e.g., the location of the client relative to the servers). Simple redirection is straightforward. The only caveats are (1) there is a limit to the number alternate IP addresses a single DNS server can manage; and (2) DNS responses are cached by downstream servers so the TTL on the response must be set to an appropriate value so as to preserve the freshness of the redirection.

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

2.1.2. HTTP Redirection

HTTP redirection makes use of the redirection response of the HTTP protocol (e.g., "302" or "307"). This response contains a new URL that the application should fetch instead of the original URL. By changing the URL appropriately, the server can cause the user to redirect to a different server. The advantages of HTTP redirection are that (1) the server can change the URL fetched by the client to include, for example, both the DNS name of the particular server to use, as well as the original HTTP server that was being accessed; (2) the client sends the HTTP request to the server, so that its IP address is known and can be used in selecting the server; and (3) other attributes (e.g., content type, user-agent type) are visible to the redirection mechanism.

The disadvantages of HTTP redirection are (1) it is visible to the application, so it requires application support and may affect the application behavior (e.g., web browsers will not send cookies if the URL changes to a different domain); (2) HTTP is a heavy-weight

protocol layered on TCP so it has relatively high overhead; and (3) the results of HTTP redirection are not cached so that all redirections must go through to the server.

3. Overview of CDNI Operation

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

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

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

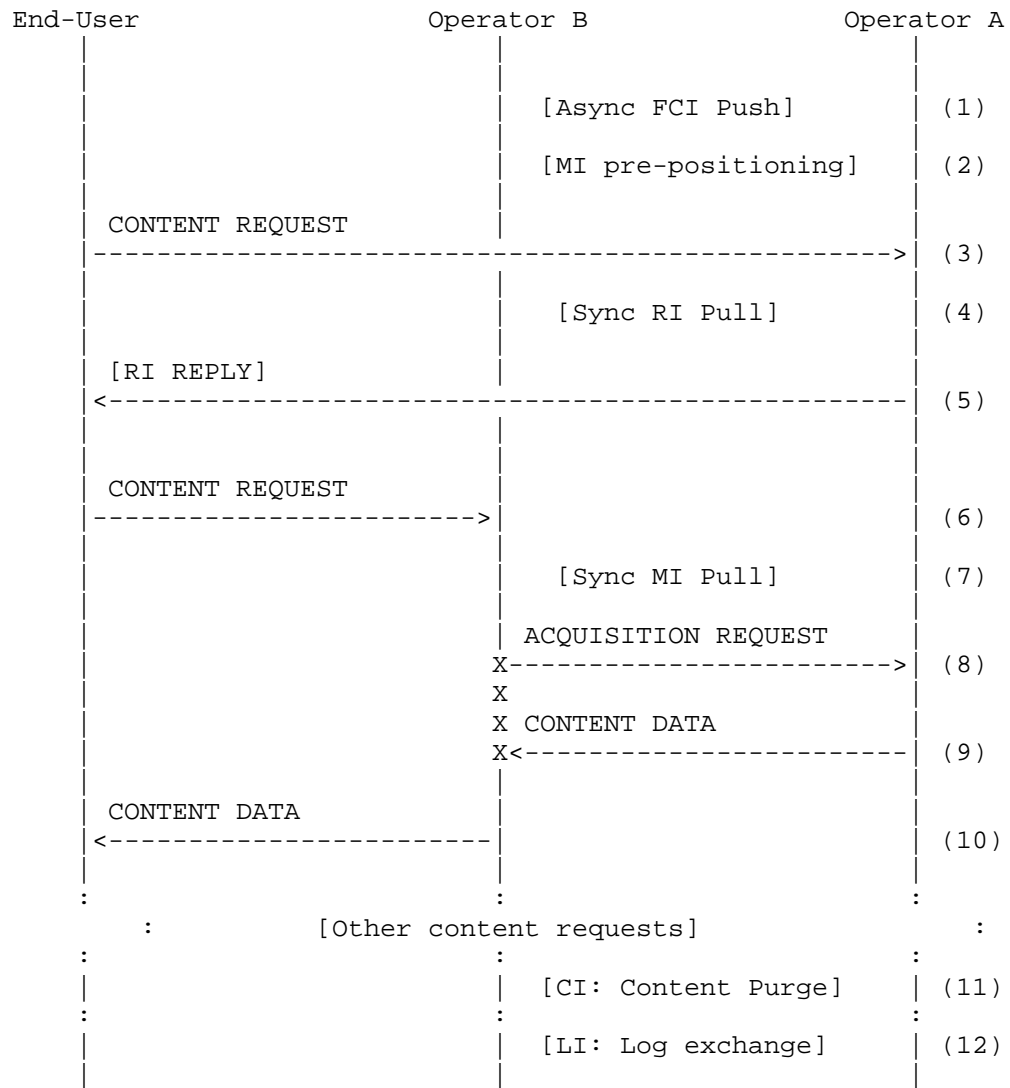


Figure 2: Overview of Operation

The operations shown in the Figure are as follows:

1. dCDN uses the FCI to advertise information relevant to its delivery footprint and capabilities prior to any content requests being redirected.

2. Prior to any content request, the uCDN uses the MI to pre-position CDNI metadata to the dCDN, thereby making that metadata available in readiness for later content requests.
3. A content request from a user agent arrives at uCDN.
4. uCDN may use the RI to synchronously request information from dCDN regarding its delivery capabilities to decide if dCDN is a suitable target for redirection of this request.
5. uCDN redirects the request to dCDN by sending some response (DNS, HTTP) to the user agent.
6. The user agent requests the content from dCDN.
7. dCDN may use the MI to synchronously request metadata related to this content from uCDN, e.g. to decide whether to serve it.
8. If the content is not already in a suitable cache in dCDN, dCDN may acquire it from uCDN.
9. The content is delivered to dCDN from uCDN.
10. The content is delivered to the user agent by dCDN.
11. Some time later, perhaps at the request of the CSP (not shown) uCDN may use the CI to instruct dCDN to purge the content, thereby ensuring it is not delivered again.
12. After one or more content delivery actions by dCDN, a log of delivery actions may be provided to uCDN using the LI.

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

3.1. Preliminaries

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

We assume Operator A provides an upstream CDN that serves content on

behalf of a CSP with CDN-Domain `cdn.csp.com`. We assume that Operator B provides a downstream CDN. An end user at some point makes a request for URL

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

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

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

3.2. Iterative HTTP Redirect Example

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

We assume for this example that Operators A and B have established an agreement to interconnect their CDNs, with A being upstream and B being downstream.

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

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

We assume the operators also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical region or a set of IP address prefixes. This information may again be provided out of band or via a defined CDNI interface.

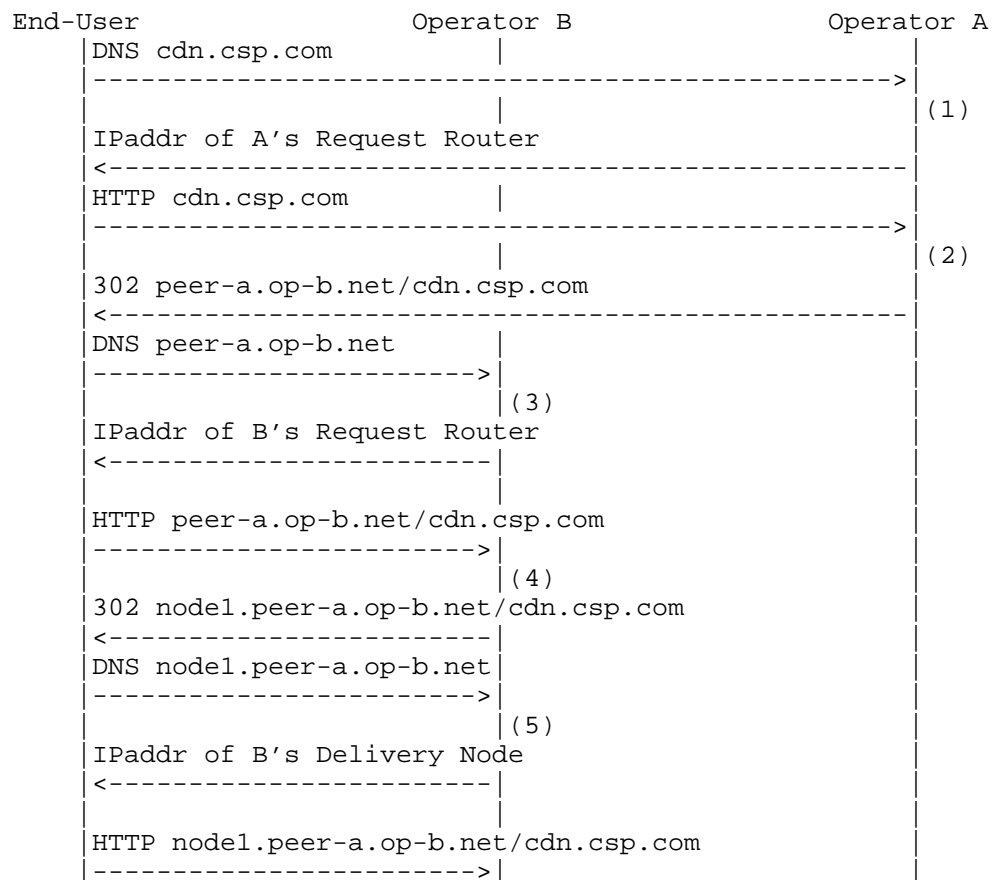
We assume DNS is configured in the following way:

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

Figure 3 illustrates how a client request for

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

is handled.



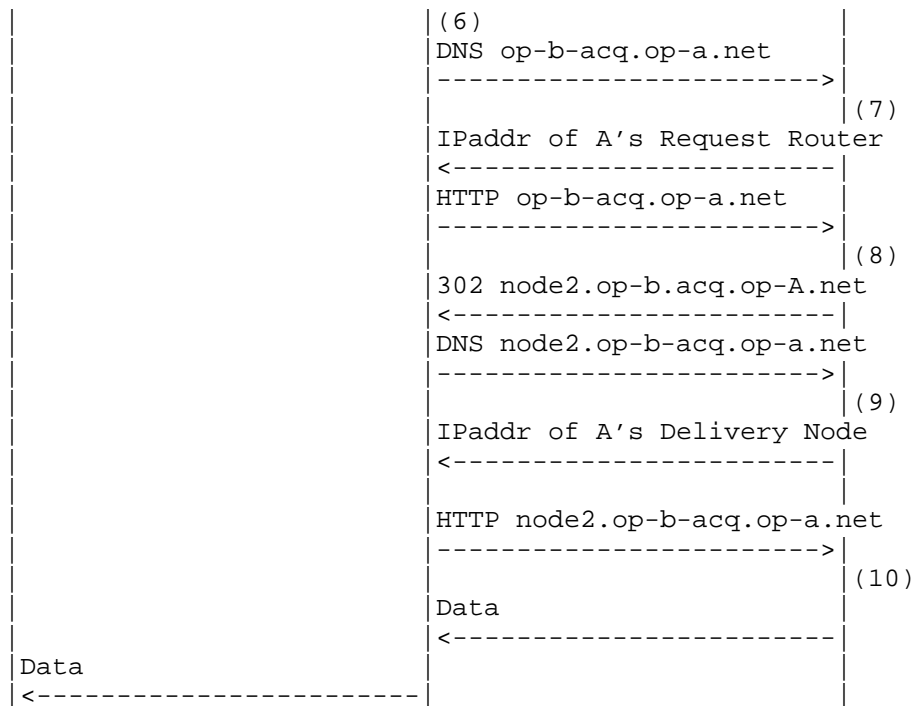


Figure 3: Message Flow for Iterative HTTP Redirection

The steps illustrated in the figure are as follows:

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

4. The request router for Operator B processes the HTTP request and selects a suitable delivery node to serve the end-user request, and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the Operator B's distinguished CDN-Domain that points to the selected delivery node.
5. The end-user does a DNS lookup using Operator B's delivery node subdomain (node1.peer-a.op-b.net). B's DNS resolver returns the IP address of the delivery node.
6. The end-user requests the content from B's delivery node. In the case of a cache hit, steps 6, 7, 8, 9 and 10 below do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDN from uCDN (not the CSP). The distinguished CDN-Domain peer-a.op-b.net indicates to dCDN that this content is to be acquired from uCDN; stripping the CDN-Domain reveals the original CDN-Domain cdn.csp.com and dCDN may verify that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an inter-CDN acquisition CDN-Domain as agreed above (in this case, op-b-acq.op-a.net).
7. Operator A's DNS resolver processes the DNS request and returns the IP address of a request router in operator A.
8. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (op-b-acq.op-a.net). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.
9. Operator A DNS resolver processes the DNS request and returns the IP address of the delivery node in operator A.
10. Operator B requests (acquires) the content from Operator A. Although not shown, Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content

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

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

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

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

We also note that the step of deciding when a request should be

redirected to dCDN rather than served by uCDN has been somewhat glossed over. It may be as simple as checking the client IP address against a list of prefixes, or it may be considerably more complex, involving a wide range of factors, such as the geographic location of the client (perhaps determined from a third party service), CDN load, or specific business rules.

This example uses the "iterative" CDNI request redirection approach. That is, uCDN performs part of the request redirection function by redirecting the client to a request router in the dCDN, which then performs the rest of the redirection function by redirecting to a suitable surrogate. If request routing is performed in the dCDN using HTTP redirection, this translates in the end-user experiencing two successive HTTP redirections. By contrast, the alternative approach of "recursive" CDNI request redirection effectively coalesces these two successive HTTP redirections into a single one, sending the end-user directly to the right delivery node in the dCDN. This "recursive" CDNI request routing approach is discussed in the next section.

3.3. Recursive HTTP Redirection Example

The following example builds on the previous one to illustrate the use of the Request Routing Interface (specifically the CDNI Request Routing Redirection Interface) to enable "recursive" CDNI request routing. We build on the HTTP-based redirection approach because it illustrates the principles and benefits clearly, but it is equally possible to perform recursive redirection when DNS-based redirection is employed.

In contrast to the prior example, the operators need not agree in advance on a CDN-Domain to serve as the target of redirections from uCDN to dCDN. We assume that the operators agree on some distinguished CDN-Domain that will be used for inter-CDN acquisition of CSP's content by dCDN. In this example, we'll use op-b-acq.op-a.net.

We assume the operators also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical region or a set of IP address prefixes. This information may again be provided out of band or via a defined protocol.

We assume DNS is configured in the following way:

- o The content provider is configured to make operator A the authoritative DNS server for cdn.csp.com (or to return a CNAME for cdn.csp.com for which operator A is the authoritative DNS server).

- o Operator A is configured so that a DNS request for op-b-acq.op-a.net returns a request router in Operator A.
- o Operator B is configured so that a request for nodel.op-b.net/cdn.csp.com returns the IP address of a delivery node. Note that there might be a number of such delivery nodes.

Figure 3 illustrates how a client request for

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

is handled.

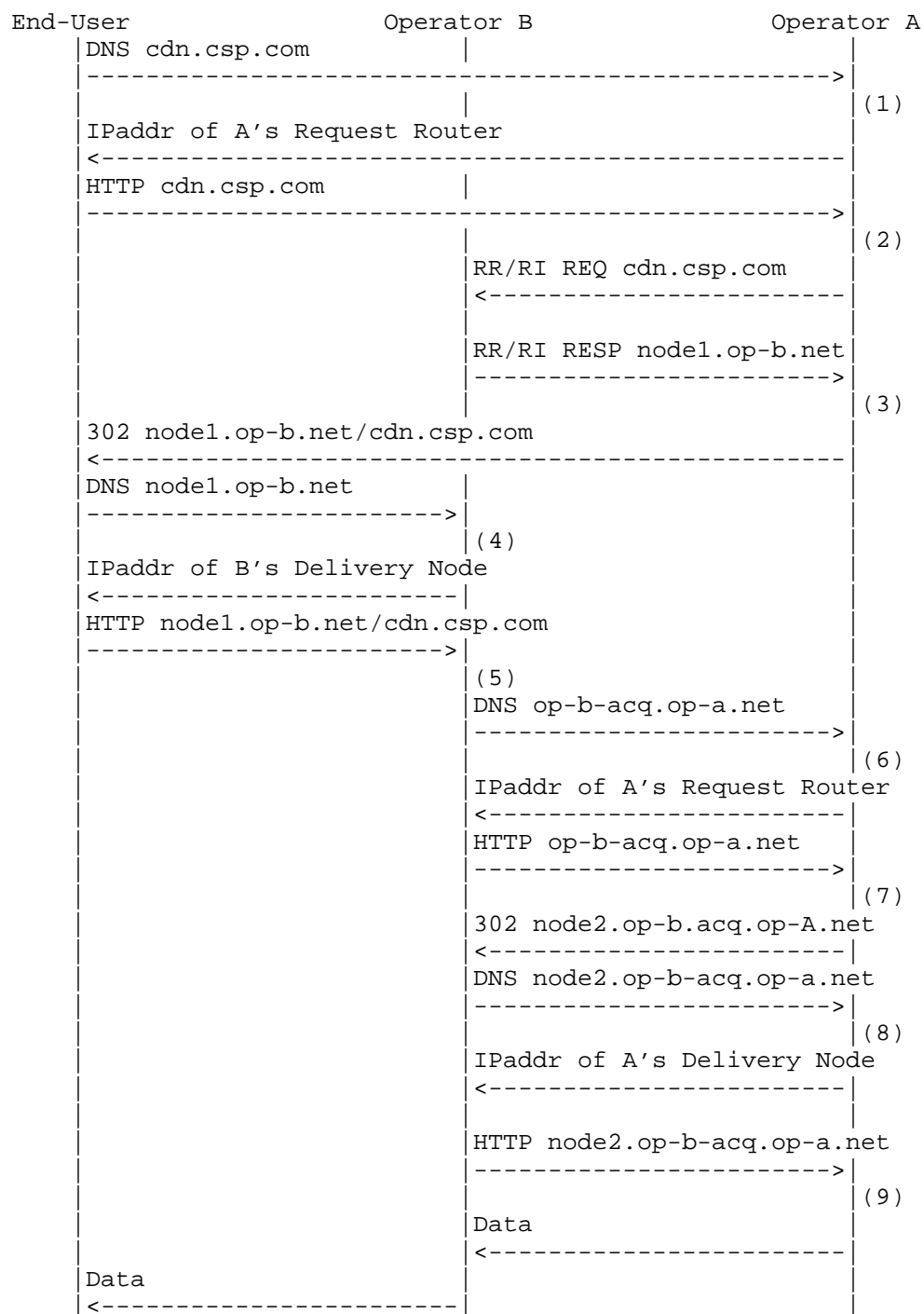


Figure 4: Message Flow for Recursive HTTP Redirection

The steps illustrated in the figure are as follows:

1. A DNS resolver for Operator A processes the DNS request for its customer based on CDN-Domain `cdn.csp.com`. It returns the IP address of a Request Router in Operator A.
2. A Request Router for Operator A processes the HTTP request and recognizes that the end-user is best served by another CDN--specifically one provided by Operator B--and so it queries the CDNI Request Routing Redirection Interface of Operator B, providing a set of information about the request including the URL requested. Operator B replies with the DNS name of a delivery node.
3. Operator A returns a 302 redirect message for a new URL obtained from the Request Routing Interface.
4. The end-user does a DNS lookup using the host name of the URL just provided (`node1.op-b.net`). B's DNS resolver returns the IP address of the corresponding delivery node. Note that, since the name of the delivery node was already obtained from B using the CDNI Request Routing Interface, there should not be any further redirection here (in contrast to the iterative method described above.)
5. The end-user requests the content from B's delivery node, potentially resulting in a cache miss. In the case of a cache miss, the content needs to be acquired from uCDN (not the CSP.) The distinguished CDN-Domain `op-b.net` indicates to dCDN that this content is to be acquired from another CDN; stripping the CDN-Domain reveals the original CDN-Domain `cdn.csp.com`, dCDN may verify that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for the inter-CDN Acquisition "distinguished" CDN-Domain as agreed above (in this case, `op-b-acq.op-a.net`).
6. Operator A DNS resolver processes the DNS request and returns the IP address of a request router in operator A.
7. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (`op-b-acq.op-a.net`). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an

infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname by a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.

8. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-Domain) and so returns the address of a delivery node. (Note that without this specially defined internal domain, Operator A would be at risk of redirecting the request back to Operator B, resulting in an infinite loop.)
9. Operator B requests (acquires) the content from Operator A. Operator A serves content for the requested CDN-Domain to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

Recursive redirection has the advantage over iterative of being more transparent from the end-user's perspective, but the disadvantage of each CDN exposing more of its internal structure (in particular, the addresses of edge caches) to peer CDNs. By contrast, iterative redirection does not require dCDN to expose the addresses of its edge caches to uCDN.

This example happens to use HTTP-based redirection in both CDN A and CDN B, but a similar example could be constructed using DNS-based redirection in either CDN. Hence, the key point to take away here is simply that the end user only sees a single redirection of some type, as opposed to the pair of redirections in the prior (iterative) example.

The use of the Request Routing Interface requires that interface to be appropriately configured and bootstrapped, which is not shown here. More discussion on the bootstrapping of interfaces is provided in Section 4

3.4. Iterative DNS-based Redirection Example

In this section we walk through a simple example using DNS-based redirection for request redirection from uCDN to dCDN (as well as for request routing inside dCDN and uCDN). As noted in Section 2.1, DNS-based redirection has certain advantages over HTTP-based redirection

(notably, it is transparent to the end-user) as well as some drawbacks (notably the client IP address is not visible to the request router).

As before, Operator A has to learn the set of requests that dCDN is willing or able to serve (e.g. which client IP address prefixes or geographic regions are part of the dCDN footprint). We assume Operator has and makes known to operator A some unique identifier that can be used for the construction of a distinguished CDN-Domain, as shown in more detail below. (This identifier strictly needs only to be unique within the scope of Operator A, but a globally unique identifier, such as an AS number assigned to B, is one easy way to achieve that.) Also, Operator A obtains the NS records for Operator B's externally visible redirection servers. Also, as before, a distinguished CDN-Domain, such as op-b-acq.op-a.net, must be assigned for inter-CDN acquisition.

We assume DNS is configured in the following way:

- o The CSP is configured to make Operator A the authoritative DNS server for cdn.csp.com (or to return a CNAME for cdn.csp.com for which operator A is the authoritative DNS server).
- o When uCDN sees a request best served by dCDN, it returns CNAME and NS records for "b.cdn.csp.com", where "b" is the unique identifier assigned to Operator B. (It may, for example, be an AS number assigned to Operator B.)
- o dCDN is configured so that a request for "b.cdn.csp.com" returns a delivery node in dCDN.
- o uCDN is configured so that a request for "op-b-acq.op-a.net" returns a delivery node in uCDN.

Figure 5 depicts the exchange of DNS and HTTP requests. The main differences from Figure 3 are the lack of HTTP redirection and transparency to the end-user.

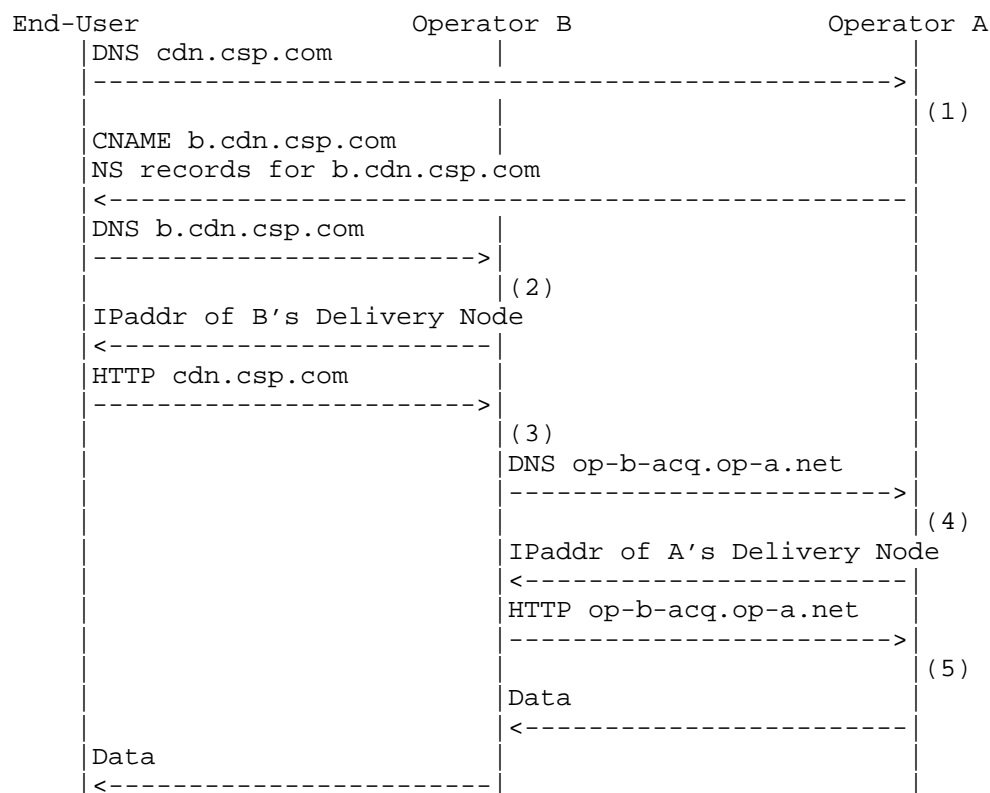


Figure 5: Message Flow for DNS-based Redirection

The steps illustrated in the figure are as follows:

1. Request Router for Operator A processes the DNS request for CDN-Domain `cdn.csp.com` and recognizes that the end-user is best served by another CDN. (This may depend on the IP address of the user's local DNS resolver, or other information discussed below.) The Request Router returns a DNS CNAME response by "stacking" the distinguished identifier for Operator B onto the original CDN-Domain (e.g., `b.cdn.csp.com`), plus an NS record that maps `b.cdn.csp.com` to B's Request Router.
2. The end-user does a DNS lookup using the modified CDN-Domain (i.e., `b.cdn.csp.com`). This causes B's Request Router to respond with a suitable delivery node.
3. The end-user requests the content from B's delivery node. The requested URL contains the name `cdn.csp.com`. (Note that the returned CNAME does not affect the URL.) At this point the

delivery node has the correct IP address of the end-user and can do an HTTP 302 redirect if the redirections in steps 2 and 3 were incorrect. Otherwise B verifies that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an "internal" CDN-Domain as agreed above (op-b-acq.op-a.net).

4. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-Domain) and so returns the address of a delivery node in uCDN.
5. Operator A serves content to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server.

The advantages of this approach are that it is more transparent to the end-user and requires fewer round trips than HTTP-based redirection (in its worst case, i.e., when none of the needed DNS information is cached). A potential problem is that the upstream CDN depends on being able to learn the correct downstream CDN that serves the end-user from the client address in the DNS request. In standard DNS operation, uCDN will only obtain the address of the client's local DNS resolver (LDNS), which is not guaranteed to be in the same network (or geographic region) as the client. If not--e.g., the end-user uses a global DNS service--then the upstream CDN cannot determine the appropriate downstream CDN to serve the end-user. In this case, and assuming the uCDN is capable of detecting that situation, one option is for the upstream CDN to treat the end-user as it would any user not connected to a peer CDN. Another option is for the upstream CDN to "fall back" to a pure HTTP-based redirection strategy in this case (i.e., use the first method). Note that this problem affects existing CDNs that rely on DNS to determine where to redirect client requests, but the consequences are arguably less serious for CDNI since the LDNS is likely in the same network as the dCDN serves. One approach to ensuring that the client's IP address prefix is correctly determined in such situations is described in [I-D.vandergaast-edns-client-subnet].

As with the prior example, this example partially illustrates the various interfaces involved in CDNI. Operator A could learn dynamically from Operator B the set of prefixes or regions that B is willing and able to serve via the Footprint & Capabilities Interface. The distinguished name used for acquisition and the identifier for Operator B that is prepended to the CDN-Domain on redirection are examples of information elements that might also be conveyed by CDNI interfaces (or, alternatively, statically configured). As before,

minimal metadata sufficient to obtain the content is carried "in-band" as part of the redirection process, and standard HTTP is used for inter-CDN acquisition. There is no explicit Logging Interface discussed in this example.

3.5. Dynamic Footprint Discovery Example

There could be situations where being able to dynamically discover the set of requests that a given dCDN is willing and able to serve is beneficial. For example, a CDN might at one time be able to serve a certain set of client IP prefixes, but that set might change over time due to changes in the topology and routing policies of the IP network. The following example illustrates this capability. We have chosen the example of DNS-based redirection, but HTTP-based redirection could equally well use this approach.

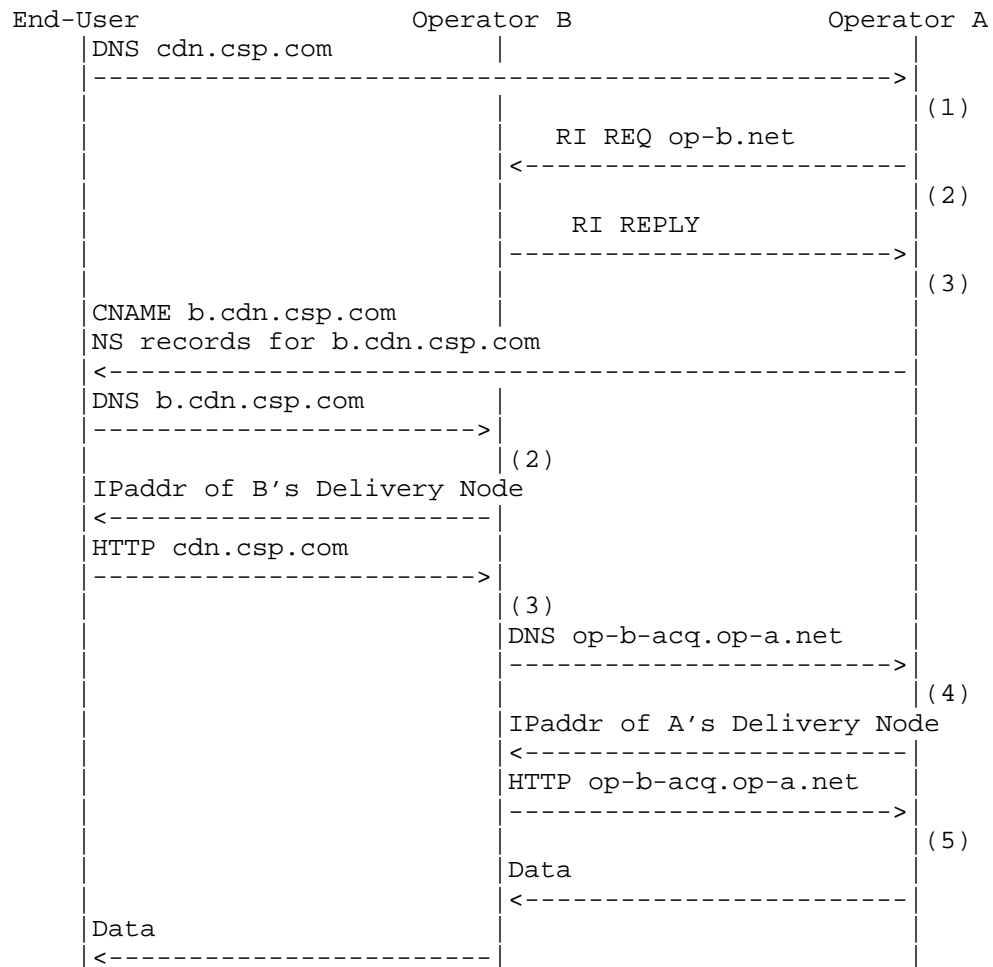


Figure 6: Message Flow for Dynamic Footprint Discovery

This example differs from the one in Figure 5 only in the addition of a CDNI Request Routing Interface Footprint request (step 2) and corresponding response (step 3). The RI REQ could be a message such as "Can you serve clients from this IP Prefix?" or it could be "Provide the list of client IP prefixes you can currently serve". In either case the response might be cached by operator A to avoid repeatedly asking the same question. Alternatively, or in addition, Operator B may spontaneously advertise to Operator A information (or changes) on the set of requests it is willing and able to serve on behalf of operator A; in that case, Operator B may spontaneously issue RR/RI REPLY messages that are not in direct response to a corresponding RR/RI REQ message. (Note that the issues of

determining the client's subnet from DNS requests, as described above, are exactly the same here as in Section 3.4.)

Once Operator A obtains the RI response, it is now able to determine that Operator B's CDN is an appropriate dCDN for this request and therefore a valid candidate dCDN to consider in its Redirection decision. If that dCDN is selected, the redirection and serving of the request proceeds as before (i.e. in the absence of dynamic footprint discovery).

3.6. Content Removal Example

The following example illustrates how the Control Interface may be used to achieve pre-positioning of an item of content in the dCDN. In this example, user requests for a particular content, and corresponding redirection of such requests from Operator A to Operator B CDN, may (or may not) have taken place earlier. Then, at some point in time, the uCDN (for example, in response to a corresponding trigger from the Content Provider) uses the Control Interface to request that content identified by a particular URL be removed from dCDN. The following diagram illustrates the operation.

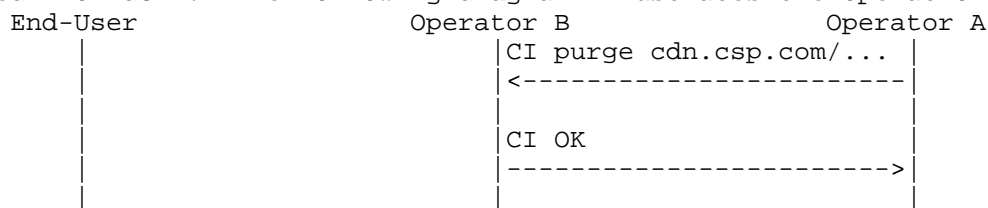


Figure 7: Message Flow for Content Removal

The Control Interface is used to convey the request from uCDN to dCDN that some previously acquired content should be deleted. The URL in the request specifies which content to remove. This example corresponds to a DNS-based redirection scenario such as Section 3.4. If HTTP-based redirection had been used, the URL for removal would be of the form peer-a.op-b.net/cdn.csp.com/...

The dCDN is expected to confirm to the uCDN, as illustrated by the CI OK message, the completion of the removal of the targeted content from all the caches in dCDN.

3.7. Pre-Positioned Content Acquisition Example

The following example illustrates how the Control Interface may be used to pre-position an item of content in the dCDN. In this example, Operator A uses the Metadata Interface to request that

content identified by a particular URL be pre-positioned into Operator B CDN.

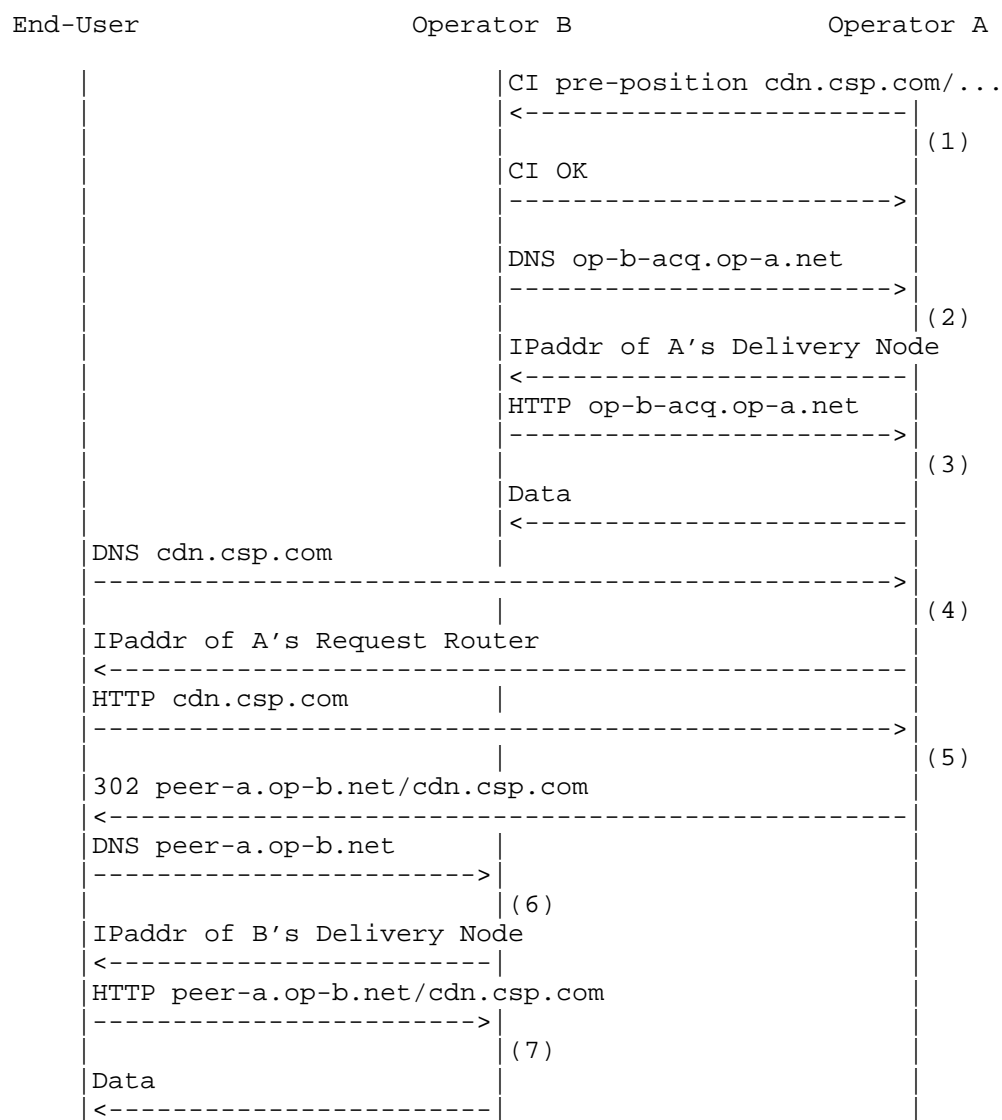


Figure 8: Message Flow for Content Pre-Positioning

The steps illustrated in the figure are as follows:

1. Operator A uses the Control Interface to request that Operator B pre-positions a particular content item identified by its URL.

Operator B responds by confirming that it is willing to perform this operation.

Steps 2 and 3 are exactly the same as steps 5 and 6 of Figure 3, only this time those steps happen as the result of the Pre-positioning request instead of as the result of a cache miss.

Steps 4, 5, 6, 7 are exactly the same as steps 1, 2, 3, 4 of Figure 3, only this time Operator B CDN can serve the end-user request without triggering dynamic content acquisition, since the content has been pre-positioned in dCDN. Note that, depending on dCDN operations and policies, the content pre-positioned in the dCDN may be pre-positioned to all, or a subset of, dCDN caches. In the latter case, intra-CDN dynamic content acquisition may take place inside the dCDN serving requests from caches on which the content has not been pre-positioning; however, such intra-CDN dynamic acquisition would not involve the uCDN.

3.8. Asynchronous CDNI Metadata Example

In this section we walk through a simple example illustrating a scenario of asynchronously exchanging CDNI metadata, where the downstream CDN obtains CDNI metadata for content ahead of a corresponding content request. The example that follows assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used, as in Section 3.3. However, asynchronous exchange of CDNI Metadata is similarly applicable to DNS-based inter-CDN redirection and iterative request routing (in which cases the CDNI metadata may be used at slightly different processing stages of the message flows).

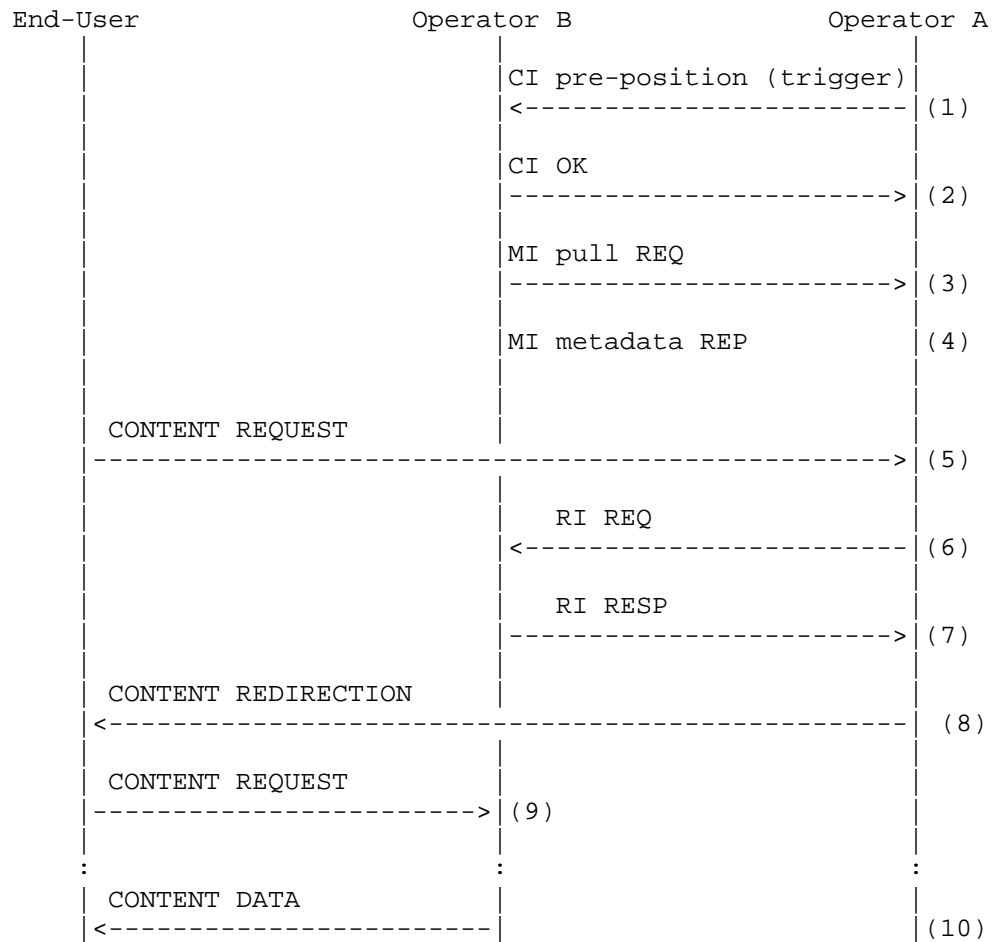


Figure 9: Message Flow for Asynchronous CDNI Metadata

The steps illustrated in the figure are as follows:

1. Operator A uses the Control Interface to trigger to signal the availability of CDNI metadata to Operator B.
2. Operator B acknowledges the receipt of this trigger.
3. Operator B requests the latest metadata from Operator A using the Metadata Interface.
4. Operator A replies with the requested metadata. This document does not constrain how the CDNI metadata information is actually

represented. For the purposes of this example, we assume that Operator A provides CDNI metadata to Operator B indicating that:

- * this CDNI Metadata is applicable to any content referenced by some CDN-Domain.
- * this CDNI metadata consists of a distribution policy requiring enforcement by the delivery node of a specific per-request authorization mechanism (e.g. URI signature or token validation).

5. A Content Request occurs as usual.
6. A CDNI Request Routing Redirection request (RI REQ) is issued by operator A CDN, as discussed in Section 3.3. Operator B's request router can access the CDNI Metadata that are relevant to the requested content and that have been pre-positioned as per Steps 1-4, which may or may not affect the response.
7. Operator B's request router issues a CDNI Request Routing Redirection response (RI RESP) as in Section 3.3.
8. Operator B performs content redirection as discussed in Section 3.3.
9. On receipt of the Content Request by the end user, the delivery node detects that previously acquired CDNI metadata is applicable to the requested content. In accordance with the specific CDNI metadata of this example, the delivery node will invoke the appropriate per-request authorization mechanism, before serving the content. (Details of this authorization are not shown.)
10. Assuming successful per-request authorization, serving of Content Data (possibly preceded by inter-CDN acquisition) proceeds as in Section 3.3.

3.9. Synchronous CDNI Metadata Acquisition Example

In this section we walk through a simple example illustrating a scenario of synchronous CDNI metadata acquisition, in which the downstream CDN obtains CDNI metadata for content at the time of handling a first request for the corresponding content. As in the preceding section, this example assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used (as in Section 3.3), but dynamic CDNI metadata acquisition is applicable to other variations of request routing.

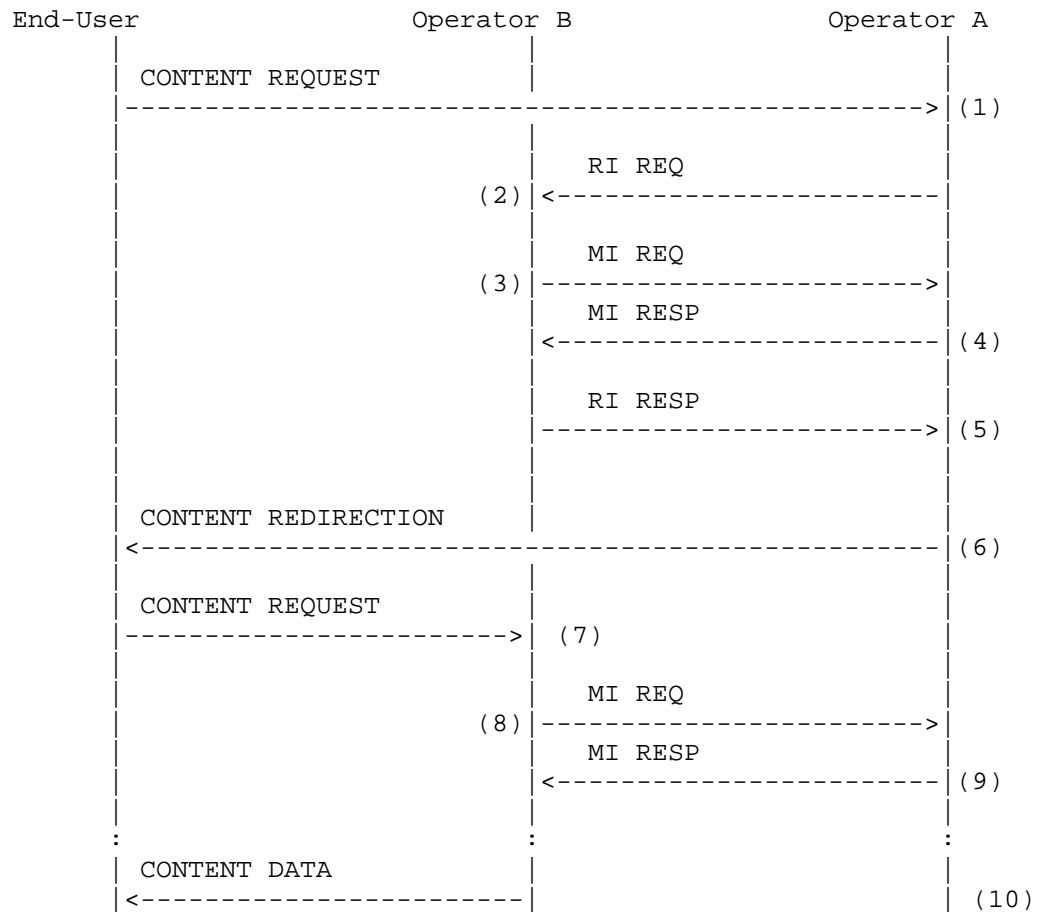


Figure 10: Message Flow for Synchronous CDNI Metadata Acquisition

The steps illustrated in the figure are as follows:

1. A Content Request arrives as normal.
2. A Request Routing Interface request occurs as in the prior example.
3. On receipt of the CDNI Request Routing Request, Operator B's CDN initiates synchronous acquisition of CDNI Metadata that are needed for routing of the end-user request. We assume the URI for the a Metadata server is known ahead of time through some out-of-band means.

4. On receipt of a CDNI Metadata Request, Operator A's CDN responds, making the corresponding CDNI metadata information available to Operator B's CDN. This metadata is considered by operator B's CDN before responding to the Request Routing request. (In a simple case, the metadata could simply be an allow or deny response for this particular request.)
5. Response to the RI request as normal.
6. Redirection message is sent to the end user.
7. A delivery node of Operator B receives the end user request.
8. The delivery node triggers dynamic acquisition of additional CDNI metadata that are needed to process the end-user content request. Note that there may exist cases where this step need not happen, for example because the metadata were already acquired previously.
9. Operator A's CDN responds to the CDNI Metadata Request and makes the corresponding CDNI metadata available to Operator B. This metadata influence how Operator B's CDN processes the end-user request.
10. Content is served (possibly preceded by inter-CDN acquisition) as in Section 3.3.

3.10. Content and Metadata Acquisition with Multiple Upstream CDNs

A single dCDN may receive end-user requests from multiple uCDNs. When a dCDN receives an end-user request, it must determine the identity of the uCDN from which it should acquire the requested content.

Ideally, the acquisition path of an end-user request will follow the redirection path of the request. The dCDN should acquire the content from the same uCDN which redirected the request.

Determining the acquisition path requires the dCDN to reconstruct the redirection path based on information in the end-user request. The method for reconstructing the redirection path differs based on the redirection approach: HTTP or DNS.

With HTTP-redirection, the rewritten URI should include sufficient information for the dCDN to directly or indirectly determine the uCDN when the end-user request is received. The HTTP-redirection approach can be further broken-down based on the how the URL is rewritten during redirection: HTTP-redirection with or without Site

Aggregation. HTTP-redirection with Site Aggregation hides the identity of the original CSP. HTTP-redirection without Site Aggregation does not attempt to hide the identity of the original CSP. With both approaches, the rewritten URI includes enough information to identify the immediate neighbor uCDN.

With DNS-redirection, the dCDN receives the published URI (instead of a rewritten URI) and does not have sufficient information for the dCDN to identify the appropriate uCDN. The dCDN may narrow the set of viable uCDNs by examining the CDNI metadata from each to determine which uCDNs are hosting metadata for the requested content. If there is a single uCDN hosting metadata for the requested content, the dCDN can assume that the request redirection is coming from this uCDN and can acquire content from that uCDN. If there are multiple uCDNs hosting metadata for the requested content, the dCDN may be ready to trust any of these uCDNs to acquire the content (provided the uCDN is in a position to serve it). If the dCDN is not ready to trust any of these uCDNs, it needs to ensure via out of band arrangements that, for a given content, only a single uCDN will ever redirect requests to the dCDN.

Content acquisition may be preceded by content metadata acquisition. If possible, the acquisition path for metadata should also follow the redirection path. Additionally, we assume metadata is indexed based on rewritten URIs in the case of HTTP-redirection and is indexed based on published URIs in the case of DNS-redirection. Thus, the Request Routing Interface and the Metadata Interface are tightly coupled in that the result of request routing (a rewritten URI pointing to the dCDN) serves as an input to metadata lookup. If the content metadata includes information for acquiring the content, then the Metadata Interface is also tightly coupled with the acquisition interface in that the result of the metadata lookup (an acquisition URL likely hosted by the uCDN) should serve as input to the content acquisition.

4. Main Interfaces

Figure 1 illustrates the four main interfaces that are in scope for the CDNI WG, along with several others. The detailed specifications of these interfaces are left to other documents, but see RFC 6707 and [I-D.ietf-cdni-requirements] for some discussion of the interfaces.

One interface that is not shown in Figure 1 is the interface between the user and the CSP. While for the purposes of CDNI that interface is out of scope, it is worth noting that it does exist and can provide useful functions, such as end-to-end performance monitoring and some forms of authentication and authorization.

There is also an important interface between the user and the Request Routing function of both uCDN and dCDN (shown as the "Request" Interface in Figure 1). As we saw in some of the preceding examples, that interface can be used as a way of passing information a subset of metadata such as the minimum information that is required for dCDN to obtain the content from uCDN.

In this section we will provide an overview of the functions performed by each of the CDNI interfaces and discuss how they fit into the overall solution. We also examine some of the design tradeoffs. We begin with an examination of one such tradeoff that affects all the interfaces - the use of in-band or out-of-band communication.

4.1. In-Band versus Out-of-Band Interfaces

Before getting to the individual interfaces, we observe that there is a high-level design choice for each, involving the use of existing in-band communication channels versus defining new out-of-band interfaces.

It is possible that the information needed to carry out various interconnection functions can be communicated between peer CDNs using existing in-band protocols. The use of HTTP 302 redirect is an example of how certain aspects of request routing can be implemented in-band (embedded in URIs). Note that using existing in-band protocols does not imply that the CDNI interfaces are null; it is still necessary to establish the rules (conventions) by which such protocols are used to implement the various interface functions.

There are other opportunities for in-band communication beyond HTTP redirects. For example, many of the HTTP directives used by proxy servers can also be used by peer CDNs to inform each other of caching activity. Of these, one that is particularly relevant is the If-Modified-Since directive, which is used with the GET method to make it conditional: if the requested object has not been modified since the time specified in this field, a copy of the object will not be returned, and instead, a 304 (not modified) response will be returned.

4.2. Cross Interface Concerns

Although the CDNI interfaces are largely independent, there are a set of conventions practiced consistently across all interfaces. Most important among these is how resources are named, for example, how the Metadata and Control Interfaces identify the set of resources to which a given directive applies, or the Logging Interface identifies the set of resources for which a summary record applies.

While in the limit the CDNI interfaces could explicitly identify every individual resource, in practice, they name resource aggregates (sets of URIs) that are to be treated in a similar way. For example, URI aggregates can be identified by a CDN-Domain (i.e., the FQDN at the beginning of a URI) or by a URI-Filter (i.e., a regular expression that matches a subset of URIs contained in some CDN-Doman). In other words, CDN-Domains and URI-Filters provide a uniform means to aggregate sets (and subsets) of URIs for the purpose of defining the scope for some operation in one of the CDNI interfaces.

4.3. Request Routing Interface

The Request Routing Interface comprises two parts: the asynchronous interface used by a dCDN to advertize footprint and capabilities (denoted FCI) to a uCDN, allowing the uCDN to decide whether to redirect particular user requests to that dCDN; and the synchronous interface used by the uCDN to redirect a user request to the dCDN (denoted RI). (These are somewhat analogous to the operations of routing and forwarding in IP.)

As illustrated in Section 3, the RI part of request routing may be implemented in part by DNS and HTTP. Naming conventions may be established by which CDN peers communicate whether a request should be routed or content served.

We also note that RI plays a key role in enabling recursive redirection, as illustrated in Section 3.3. It enables the user to be redirected to the correct delivery node in dCDN with only a single redirection step (as seen by the user). This may be particularly valuable as the chain of interconnected CDNs increases beyond two CDNs.

In support of these redirection requests, it is necessary for CDN peers to exchange additional information with each other, and this is the role of the FCI part of request routing. Depending on the method(s) supported, this might include

- o The operator's unique id (operator-id) or distinguished CDN-Domain (operator-domain);
- o NS records for the operator's set of externally visible request routers;
- o The set of requests the dCDN operator is prepared to serve (e.g. a set of client IP prefixes or geographic regions that may be served by dCDN).

- o Additional capabilities of the dCDN, such as its ability to support different CDNI Metadata requests.

Note that the set of requests that dCDN is willing to serve could in some cases be relatively static (e.g., a set of IP prefixes) which could be exchanged off-line, or might even be negotiated as part of a peering agreement. However, it may also be more dynamic, in which case the exchange supported by FCI would be helpful. A further discussion of the Footprint & Capability Advertisement Interface can be found in [I-D.spp-cdni-rr-foot-cap-semantics].

4.4. Logging Interface

It is necessary for the upstream CDN to have visibility into the delivery of content that it redirected to a downstream CDN. This allows the upstream CDN to properly bill its customers for multiple deliveries of content cached by the downstream CDN, as well as to report accurate traffic statistics to those content providers. This is one role of the Logging Interface.

Other operational data that may be relevant to CDNI can also be exchanged by the Logging Interface. For example, dCDN may report the amount of content it has acquired from uCDN, and how much cache storage has been consumed by content cached on behalf of uCDN.

Traffic logs are easily exchanged off-line. For example, the following traffic log is a small deviation from the Apache log file format, where entries include the following fields:

- o Domain - the full domain name of the origin server
- o IP address - the IP address of the client making the request
- o End time - the ending time of the transfer
- o Time zone - any time zone modifier for the end time
- o Method - the transfer command itself (e.g., GET, POST, HEAD)
- o URL - the requested URL
- o Version - the protocol version, such as HTTP/1.0
- o Response - a numeric response code indicating transfer result
- o Bytes Sent - the number of bytes in the body sent to the client

- o Request ID - a unique identifier for this transfer
- o User agent - the user agent, if supplied
- o Duration - the duration of the transfer in milliseconds
- o Cached Bytes - the number of body bytes served from the cache
- o Referrer - the referrer string from the client, if supplied

Of these, only the Domain field is indirect in the downstream CDN--it is set to the CDN-Domain used by the upstream CDN rather than the actual origin server. This field could then be used to filter traffic log entries so only those entries matching the upstream CDN are reported to the corresponding operator. Further discussion of the Logging Interface can be found in [I-D.bertrand-cdni-logging].

One open question is who does the filtering. One option is that the downstream CDN filters its own logs, and passes the relevant records directly to each upstream peer. This requires that the downstream CDN knows the set of CDN-Domains that belong to each upstream peer. If this information is already exchanged between peers as part of another interface, then direct peer-to-peer reporting is straightforward. If it is not available, and operators do not wish to advertise the set of CDN-Domains they serve to their peers, then the second option is for each CDN to send both its non-local traffic records and the set of CDN-Domains it serves to an independent third-party (i.e., a CDN Exchange), which subsequently filters, merges, and distributes traffic records on behalf of each participating CDN operator.

A second open question is how timely traffic information should be. For example, in addition to off-line traffic logs, accurate real-time traffic monitoring might also be useful, but such information requires that the downstream CDN inform the upstream CDN each time it serves upstream content from its cache. The downstream CDN can do this, for example, by sending a conditional HTTP GET request (If-Modified-Since) to the upstream CDN each time it receives an HTTP GET request from one of its end-users. This allows the upstream CDN to record that a request has been issued for the purpose of real-time traffic monitoring. The upstream CDN can also use this information to validate the traffic logs received later from the downstream CDN.

There is obviously a tradeoff between accuracy of such monitoring and the overhead of the downstream CDN having to go back to the upstream CDN for every request.

Another design tradeoff in the Logging Interface is the degree of

aggregation or summarization of data. One situation that lends itself to summarization is the delivery of HTTP adaptive streaming (HAS), since the large number of individual chunk requests potentially results in large volumes of logging information. This case is discussed below, but other forms of aggregation may also be useful. For example, there may be situations where bulk metrics such as bytes delivered per hour may suffice rather than the detailed per-request logs outlined above. It seems likely that a range of granularities of logging will be needed along with ways to specify the type and degree of aggregation required.

4.5. Control Interface

The Control Interface is initially used to bootstrap the other interfaces. As a simple example, it could be used to provide the address of the logging server in dCDN to uCDN in order to bootstrap the Logging Interface. It may also be used, for example, to establish security associations for the other interfaces.

The other role the Control Interface plays is to allow the uCDN to pre-position, revalidate, or purge metadata and content on a dCDN. These operations, sometimes collectively called the trigger interface, are discussed further in [I-D.murray-cdni-triggers].

4.6. Metadata Interface

The role of the CDNI Metadata Interface is to enable CDNI distribution metadata to be conveyed to the downstream CDN by the upstream CDN. For example, see [I-D.ietf-cdni-metadata]. Such metadata includes geo-blocking restrictions, availability windows, access control policies, and so on. It may also include information to facilitate acquisition of content by dCDN (e.g., alternate sources for the content, authorization information needed to acquire the content from the source).

Some distribution metadata may be partially emulated using in-band mechanisms. For example, in case of any geo-blocking restrictions or availability windows, the upstream CDN can elect to redirect a request to the downstream CDN only if that CDN's advertised delivery footprint is acceptable for the requested URL. Similarly, the request could be forwarded only if the current time is within the availability window. However, such approaches typically come with shortcomings such as inability to prevent from replay outside the time window or inability to make use of a downstream CDN that covers a broader footprint than the geo-blocking restrictions.

Similarly, some forms of access control may also be performed on a per-request basis using HTTP directives. For example, being able to

respond to a conditional GET request gives the upstream CDN an opportunity to influence how the downstream CDN delivers its content. Minimally, the upstream CDN can invalidate (purge) content previously cached by the downstream CDN.

Fine-grain control over how the downstream CDN delivers content on behalf of the upstream CDN is also possible. For example, by including the X-Forwarded-For HTTP header with the conditional GET request, the downstream CDN can report the end-user's IP address to the upstream CDN, giving it an opportunity to control whether the downstream CDN should serve the content to this particular end-user. The upstream CDN would communicate its directive through its response to the conditional GET. The downstream CDN can cache information for a period of time specified by the upstream CDN, thereby reducing control overhead.

All of these in-band techniques serve to illustrate that uCDNs have the option of enforcing some of their access control policies themselves (at the expense of increased inter-CDN signaling load), rather than delegating enforcement to dCDNs using the Metadata Interface. As a consequence, the Metadata Interface should provide a means for the uCDN to express its desire to retain enforcement for itself. For example, this might be done by including a "check with me" flag in the metadata associated with certain content.

4.7. HTTP Adaptive Streaming Concerns

We consider HTTP Adaptive Streaming (HAS) and the impact it has on the CDNI interfaces because large objects (e.g., videos) are broken into a sequence of small, independent chunks. For each of the following, a more thorough discussion, including an overview of the tradeoffs involved in alternative designs, can be found in [I-D.brandenburg-cdni-has].

First, with respect to Content Acquisition and File Management, which are out-of-scope for the CDNI interfaces but nonetheless relevant to the overall operation, we assume no additional measures are required to deal with large numbers of chunks. This means that the dCDN is not explicitly made aware of any relationship between different chunks and the dCDN handles each chunk as if it were an individual and independent content item. The result is that content acquisition between uCDN and dCDN also happens on a per-chunk basis. This approach is in line with the recommendations made in [I-D.brandenburg-cdni-has], which also identifies potential improvements in this area that might be considered in the future.

Second, with respect to Request Routing, we note that HAS manifest files have the potential to interfere with request routing since

manifest files contain URLs pointing to the location of content chunks. To make sure that a manifest file does not hinder CDNI request routing and does not place excessive load on CDNI resources, the use of manifest files could either be limited to those containing relative URLs or the uCDN could modify the URLs in the manifest. Our approach for dealing with these issues is twofold. As a mandatory requirement, CDNs should be able to handle unmodified manifest files containing either relative or absolute URLs. To limit the number of redirects, and thus the load placed on the CDNI Interfaces, as an optional feature uCDNs can use the information obtained through the CDNI Request Routing Redirection Interface to modify the URLs in the manifest file. Since the modification of the manifest file is an optional uCDN-internal process, this does not require any standardization effort beyond being able to communicate chunk locations in the CDNI Request Routing Redirection Interface.

Third, with respect to the Logging Interface, there are several potential issues, including the large number of individual chunk requests potentially resulting in large volumes of logging information, and the desire to correlate logging information for chunk requests that correspond to the same HAS session. For the initial CDNI specification, our approach is to expect participating CDNs to support per-chunk logging (e.g. logging each chunk request as if it were an independent content request) over the CDNI Logging Interface. Optionally, the Logging Interface may include a Content Collection Identifier (CCID) and/or a Session Identifier (SID) as part of the logging fields, thereby facilitating correlation of per-chunk logs into per-session logs for applications benefiting from such session level information (e.g. session-based analytics). This approach is in line with the recommendations made in [I-D.brandenburg-cdni-has], which also identifies potential improvements in this area that might be considered in the future.

Fourth, with respect to the Control Interface, and in particular purging HAS chunks from a given CDN, our approach is to expect each CDN supports per-chunk content purge (e.g. purging of chunks as if they were individual content items). Optionally, a CDN may support content purge on the basis of a "Purge Identifier (Purge-ID)" allowing the removal of all chunks related to a given Content Collection with a single reference. It is possible that this Purge-ID could be merged with the CCID discussed above for HAS Logging, or alternatively, they may remain distinct.

5. Deployment Models

In this section we describe a number of possible deployment models that may be achieved using the CDNI interfaces described above. We

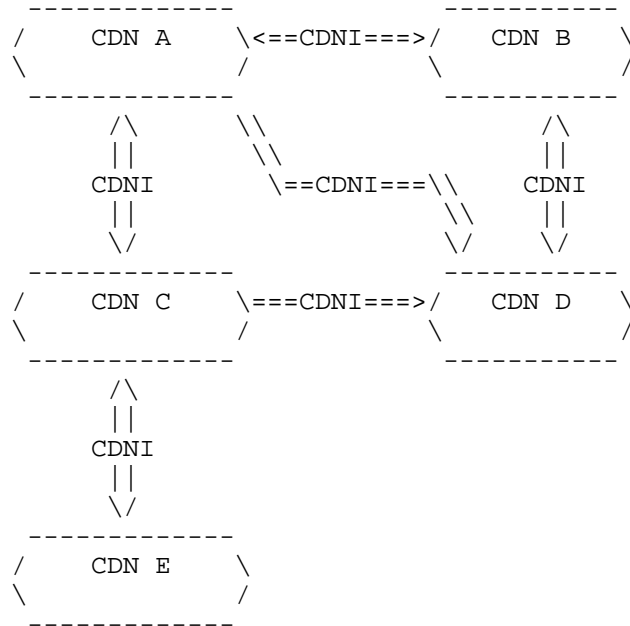
note that these models are by no means exhaustive, and that many other models may be possible.

Although the reference model of Figure 1 shows all CDN functions on each side of the CDNI interface, deployments can rely on entities that are involved in any subset of these functions, and therefore only support the relevant subset of CDNI interfaces. As already noted in Section 3, effective CDNI deployments can be built without necessarily implementing all four interfaces. Some examples of such deployments are shown below.

Note that, while we refer to upstream and downstream CDNs, this distinction applies to specific content items and transactions. That is, a given CDN may be upstream for some transactions and downstream for others, depending on many factors such as location of the requesting client and the particular piece of content requested.

5.1. Meshed CDNs

Although the reference model illustrated in Figure 1 shows a unidirectional CDN interconnection with a single uCDN and a single dCDN, any arbitrary CDNI meshing can be built from this, such as the example meshing illustrated in Figure 11. (Support for arbitrary meshing may or may not be in the initial scope for the working group, but the model allows for it.)



- ==> CDNI interfaces, with right-hand side CDN acting as dCDN to left-hand side CDN
- <==> CDNI interfaces, with right-hand side CDN acting as dCDN to left-hand side CDN and with left-hand side CDN acting as dCDN to right-hand side CDN

Figure 11: CDNI Deployment Model: CDN Meshing Example

5.2. CSP combined with CDN

Note that our terminology refers to functional roles and not economic or business roles. That is, a given organization may be operating as both a CSP and a fully-fledged uCDN when we consider the functions performed, as illustrated in Figure 12.

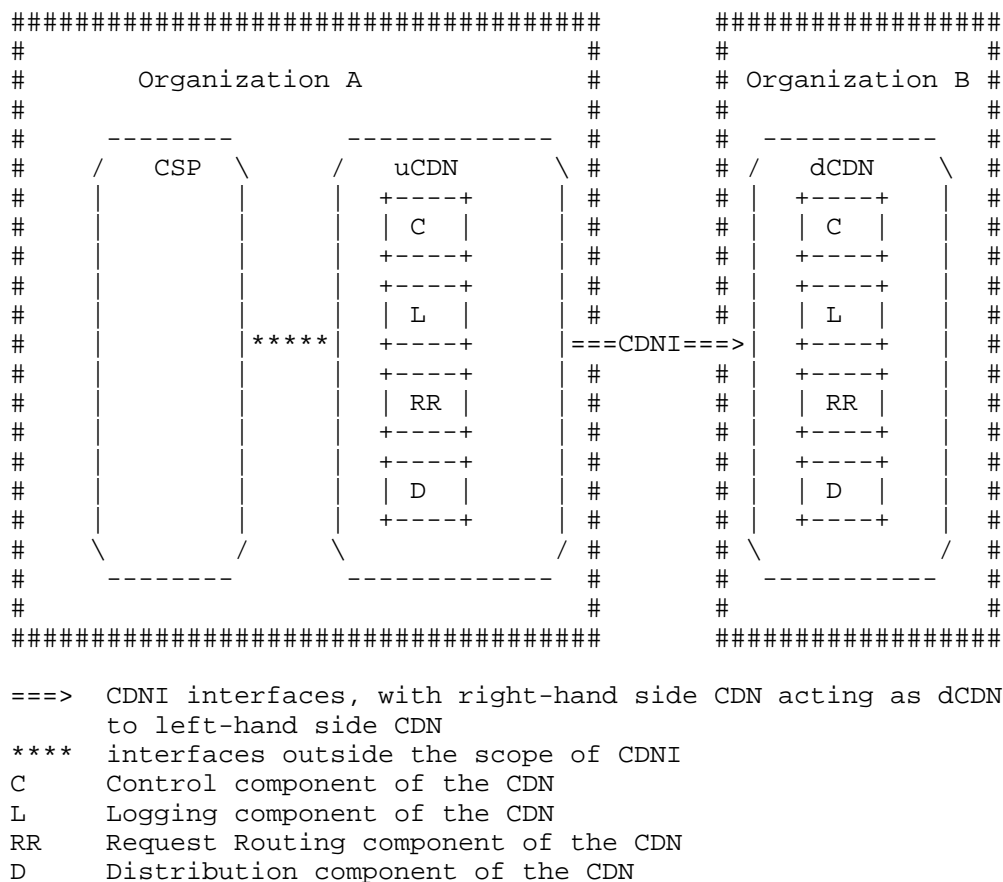


Figure 12: CDNI Deployment Model: Organization combining CSP & uCDN

5.3. CSP using CDNI Request Routing Interface

As another example, a content provider organization may choose to run its own request routing function as a way to select among multiple candidate CDN providers; In this case the content provider may be modeled as the combination of a CSP and of a special, restricted case of a CDN. In that case, as illustrated in Figure 13, the CDNI Request Routing Interfaces can be used between the restricted CDN operated by the content provider Organization and the CDN operated by the full-CDN organization acting as a dCDN in the request routing control plane. Interfaces outside the scope of the CDNI work can be used between the CSP functional entities of the content provider organization and the CDN operated by the full-CDN organization acting as a uCDN) in the CDNI control planes other than the request routing plane (i.e. Control, Distribution, Logging).

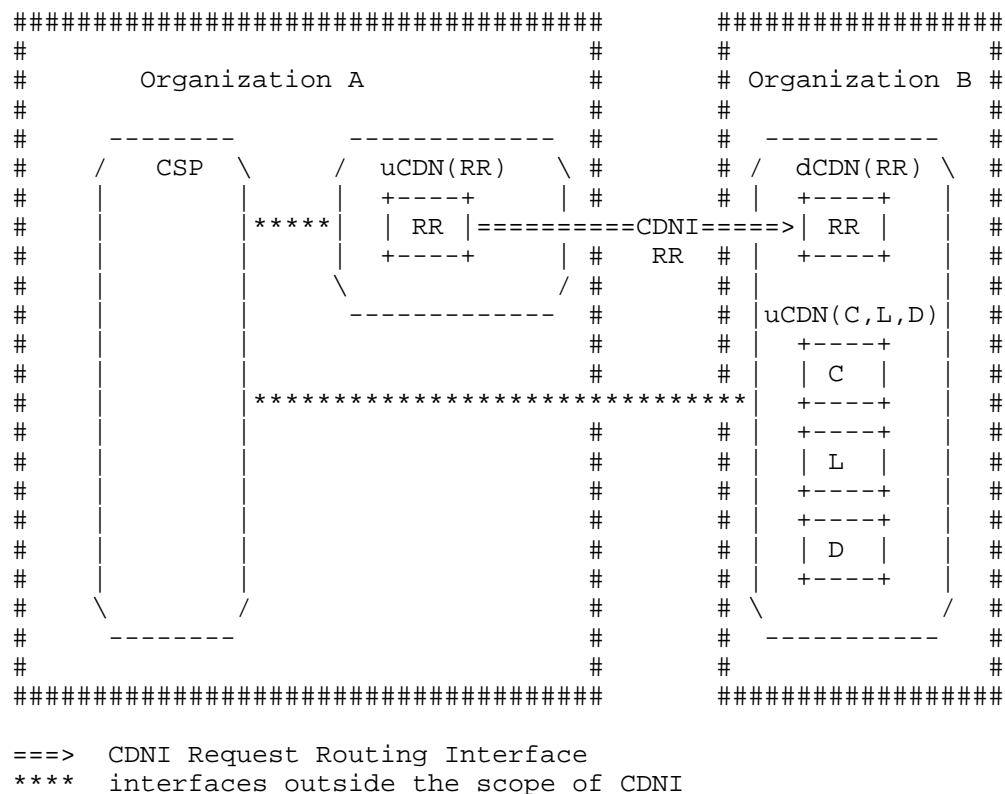


Figure 13: CDNI Deployment Model: Organization combining CSP and partial CDN

5.4. CDN Federations and CDN Exchanges

There are two additional concepts related to, but distinct from CDN Interconnection. The first is CDN Federation. Our view is that CDNI is the more general concept, involving two or more CDNs serving content to each other's users, while federation implies a multi-lateral interconnection arrangement, but other CDN interconnection agreements are also possible (e.g., symmetric bilateral, asymmetric bilateral). An important conclusion is that CDNI technology should not presume (or bake in) a particular interconnection agreement, but should instead be general enough to permit alternative interconnection arrangements to evolve.

The second concept often used in the context of CDN Federation is CDN Exchange--a third party broker or exchange that is used to facilitate a CDN federation. Our view is that a CDN exchange offers valuable machinery to scale the number of CDN operators involved in a multi-

lateral (federated) agreement, but that this machinery is built on top of the core CDNI interconnection mechanisms. For example, as illustrated in Figure 14, the exchange might aggregate and redistribute information about each CDN footprint and capacity, as well as collect, filter, and re-distribute traffic logs that each participant needs for interconnection settlement, but inter-CDN request routing, inter-CDN content distribution (including inter-CDN acquisition) and inter-CDN control which fundamentally involve a direct interaction between an upstream CDN and a downstream CDN--operate exactly as in a pair-wise peering arrangement. Turning to Figure 14, we observe that in this example:

- o each CDN supports a direct CDNI Control Interface to every other CDN
- o each CDN supports a direct CDNI Metadata Interface to every other CDN
- o each CDN supports a CDNI Logging Interface with the CDN Exchange
- o each CDN supports both a CDNI Request Routing Interface with the CDN Exchange (for aggregation and redistribution of dynamic CDN footprint discovery information) and a direct CDNI Request Routing Interface to every other CDN (for actual request redirection).

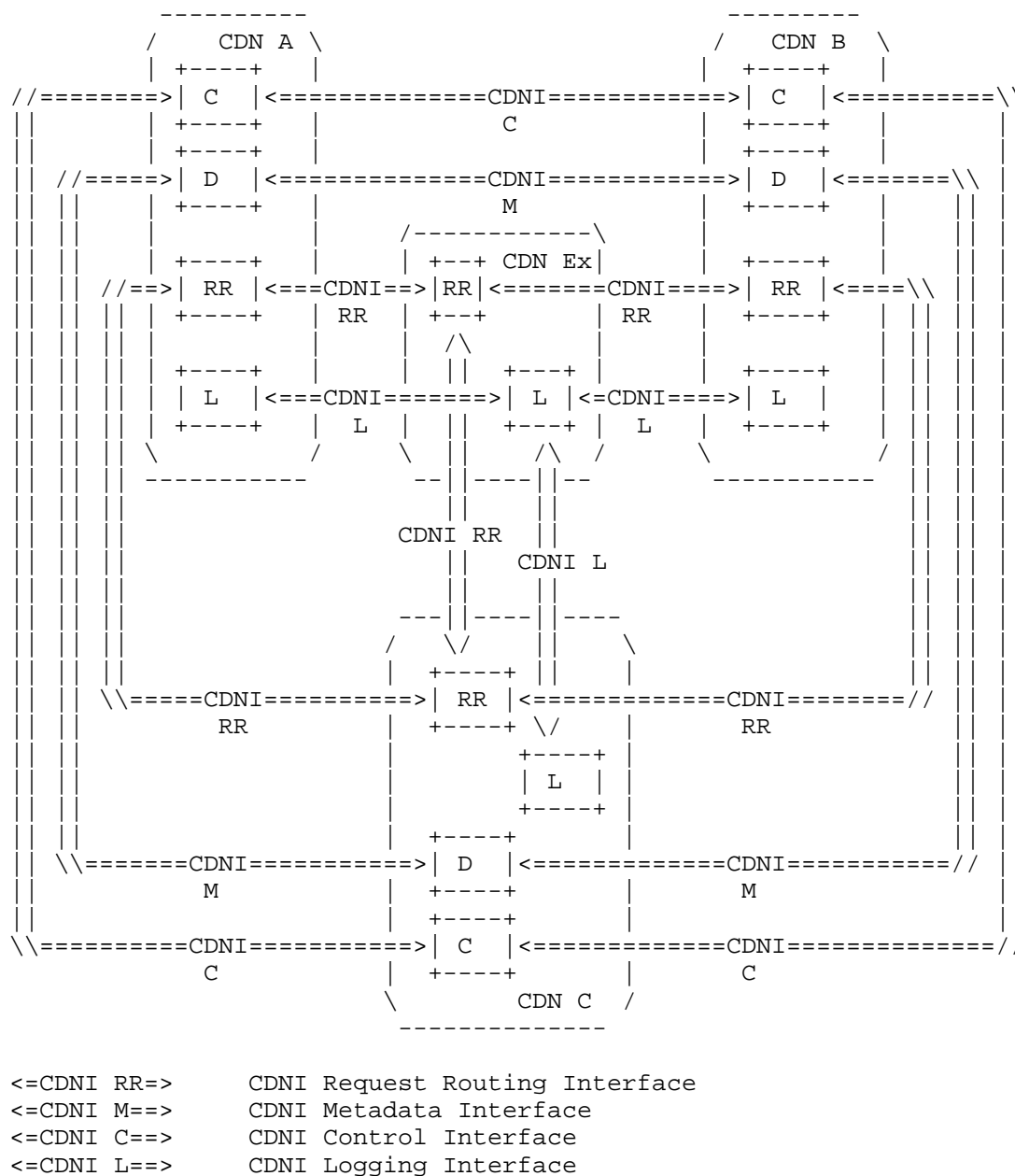


Figure 14: CDNI Deployment Model: CDN Exchange

Note that a CDN exchange may alternatively support a different set of functionality (e.g. Logging only, or Logging and full request

routing, or all the functionality of a CDN including content distribution). All these options are expected to be allowed by the IETF CDNI specifications.

6. Trust Model

There are a number of trust issues that need to be addressed by a CDNI solution. Many of them are in fact similar or identical to those in a simple CDN without interconnection. In a standard CDN environment (without CDNI), the CSP places a degree of trust in a single CDN operator to perform many functions. The CDN is trusted to deliver content with appropriate quality of experience for the end user. The CSP trusts the CDN operator not to corrupt or modify the content. The CSP often relies on the CDN operator to provide reliable accounting information regarding the volume of delivered content. The CSP may also trust the CDN operator to perform actions such as timely invalidation of content and restriction of access to content based on certain criteria such as location of the user and time of day, and to enforce per-request authorization performed by the CSP using techniques such as URI signing.

A CSP also places trust in the CDN not to distribute any information that is confidential to the CSP (e.g., how popular a given piece of content is) or confidential to the end user (e.g., which content has been watched by which user).

A CSP does not necessarily have to place complete trust in a CDN. A CSP will in some cases take steps to protect its content from improper distribution by a CDN, e.g. by encrypting it and distributing keys in some out of band way. A CSP also depends on monitoring (possibly by third parties) and reporting to verify that the CDN has performed adequately. A CSP may use techniques such as client-based metering to verify that accounting information provided by the CDN is reliable. HTTP conditional requests may be used to provide the CSP with some checks on CDN operation. In other words, while a CSP may trust a CDN to perform some functions in the short term, the CSP is able in most cases to verify whether these actions have been performed correctly and to take action (such as moving the content to a different CDN) if the CDN does not live up to expectations.

The main trust issue raised by CDNI is that it introduces transitive trust. A CDN that has a direct relationship with a CSP can now "outsource" the delivery of content to another (downstream) CDN. That CDN may in turn outsource delivery to yet another downstream CDN, and so on.

The top level CDN in such a chain of delegation is responsible for ensuring that the requirements of the CSP are met. Failure to do so is presumably just as serious as in the traditional single CDN case. Hence, an upstream CDN is essentially trusting a downstream CDN to perform functions on its behalf in just the same way as a CSP trusts a single CDN. Monitoring and reporting can similarly be used to verify that the downstream CDN has performed appropriately. However, the introduction of multiple CDNs in the path between CSP and end user complicates the picture. For example, third party monitoring of CDN performance (or other aspects of operation, such as timely invalidation) might be able to identify the fact that a problem occurred somewhere in the chain but not point to the particular CDN at fault.

In summary, we assume that an upstream CDN will invest a certain amount of trust in a downstream CDN, but that it will verify that the downstream CDN is performing correctly, and take corrective action (including potentially breaking off its relationship with that CDN) if behavior is not correct. We do not expect that the trust relationship between a CSP and its "top level" CDN will differ significantly from that found today in single CDN situations. However, it does appear that more sophisticated tools and techniques for monitoring CDN performance and behavior will be required to enable the identification of the CDN at fault in a particular delivery chain.

We expect that the detailed designs for the specific interfaces for CDNI will need to take the transitive trust issues into account. For example, explicit confirmation that some action (such as content removal) has taken place in a downstream CDN may help to mitigate some issues of transitive trust.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

While there is a variety of security issues introduced by a single CDN, we are concerned here specifically with the additional issues that arise when CDNs are interconnected. For example, when a single CDN has the ability to distribute content on behalf of a CSP, there may be concerns that such content could be distributed to parties who are not authorized to receive it, and there are mechanisms to deal with such concerns. Our focus in this section is on how CDN interconnection introduces new security issues not found in the

single CDN case.

Many of the security issues that arise in CDNI are related to the transitivity of trust (or lack thereof) described in Section 6. As noted above, the design of the various interfaces for CDNI must take account of the additional risks posed by the fact that a CDN with whom a CSP has no direct relationship is now potentially distributing content for that CSP. The mechanisms used to mitigate these risks may be similar to those used in the single CDN case, but their suitability in this more complex environment must be validated.

Another concern that arises in any CDN is that information about the behavior of users (what content they access, how much content they consume, etc.) may be gathered by the CDN. This risk certainly exists in inter-connected CDNs, but it should be possible to apply the same techniques to mitigate it as in the single CDN case.

CDNs today offer a variety of means to control access to content, such as time-of-day restrictions, geo-blocking, and URI signing. These mechanisms must continue to function in CDNI environments, and this consideration is likely to affect the design of certain CDNI interfaces (e.g. metadata, request routing.)

Just as with a single CDN, each peer CDN must ensure that it is not used as an "open proxy" to deliver content on behalf of a malicious CSP. Whereas a single CDN typically addresses this problem by having CSPs explicitly register content (or origin servers) that is to be served, simply propagating this information to peer downstream CDNs may be problematic because it reveals more information than the upstream CDN is willing to specify. (To this end, the content acquisition step in the earlier examples force the dCDN to retrieve content from the uCDN rather than go directly to the origin server.)

There are several approaches to this problem. One is for the uCDN to encode a signed token generated from a shared secret in each URL routed to a dCDN, and for the dCDN to validate the request based on this token. Another one is to have each upstream CDN advertise the set of CDN-Domains they serve, where the downstream CDN checks each request against this set before caching and delivering the associated object. Although straightforward, this approach requires operators to reveal additional information, which may or may not be an issue.

8.1. Security of CDNI Interfaces

It is noted in [I-D.ietf-cdni-requirements] that all CDNI interfaces must be able to operate securely over insecure IP networks. Since it is expected that the CDNI interfaces will be implemented using existing application protocols such as HTTP or XMPP, we also expect

that the security mechanisms available to those protocols may be used by the CDNI interfaces. Details of how these interfaces are secured will be specified in the relevant interface documents.

8.2. Digital Rights Management

Issues of digital rights management (DRM, also sometimes called digital restrictions management) is often employed for content distributed via CDNs. In general, DRM relies on the CDN to distribute encrypted content, with decryption keys distributed to users by some other means (e.g. directly from the CSP to the end user.) For this reason, DRM is considered out of scope for the CDNI WG RFC 6707 and does not introduce additional security issues for CDNI.

9. Contributors

The following individuals contributed to this document:

- o Ray Brandenburg
- o Matt Caulfield
- o Francois le Faucheur
- o Aaron Falk
- o David Ferguson
- o John Hartman
- o Ben Niven-Jenkins
- o Kent Leung

10. Acknowledgements

We thank Huw Jones for helpful input to the draft.

11. Informative References

- [I-D.bertrand-cdni-logging]
Bertrand, G., Emile, S., Peterkofsky, R., Faucheur, F.,
and P. Grochocki, "CDNI Logging Interface",
draft-bertrand-cdni-logging-02 (work in progress),

October 2012.

[I-D.brandenburg-cdni-has]

Brandenburg, R., Deventer, O., Faucheur, F., and K. Leung, "Models for adaptive-streaming-aware CDN Interconnection", draft-brandenburg-cdni-has-04 (work in progress), January 2013.

[I-D.ietf-cdni-metadata]

Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata", draft-ietf-cdni-metadata-00 (work in progress), October 2012.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-04 (work in progress), December 2012.

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

Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-10 (work in progress), August 2012.

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

Faucheur, F., Viveganandhan, M., and K. Leung, "CDNI Logging Formats for HTTP and HTTP Adaptive Streaming Deliveries", draft-lefaucheur-cdni-logging-delivery-01 (work in progress), July 2012.

[I-D.murray-cdni-triggers]

Murray, R. and B. Niven-Jenkins, "CDN Interconnect Triggers", draft-murray-cdni-triggers-01 (work in progress), August 2012.

[I-D.seedorf-alto-for-cdni]

Seedorf, J., "ALTO for CDNI Request Routing", draft-seedorf-alto-for-cdni-00 (work in progress), October 2011.

[I-D.spp-cdni-rr-foot-cap-semantics]

Seedorf, J., Peterson, J., and S. Previdi, "CDNI Request Routing: Footprint and Capabilities Semantics", draft-spp-cdni-rr-foot-cap-semantics-02 (work in progress), October 2012.

- [I-D.vandergaast-edns-client-subnet]
Contavalli, C., Gaast, W., Leach, S., and E. Lewis,
"Client subnet in DNS requests",
draft-vandergaast-edns-client-subnet-01 (work in
progress), April 2012.
- [RFC3466] Day, M., Cain, B., Tomlinson, G., and P. Rzewski, "A Model
for Content Internetworking (CDI)", RFC 3466,
February 2003.
- [RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic
Optimization (ALTO) Problem Statement", RFC 5693,
October 2009.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content
Distribution Network Interconnection (CDNI) Problem
Statement", RFC 6707, September 2012.

Authors' Addresses

Larry Peterson (editor)
Akamai Technologies, Inc.
8 Cambridge Center
Cambridge, MA 02142
USA

Email: lapeters@akamai.com

Bruce Davie
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
USA

Email: bdavie@vmware.com

Network Working Group
Internet-Draft
Obsoletes: 3466 (if approved)
Intended status: Informational
Expires: December 8, 2014

L. Peterson
Akamai Technologies, Inc.
B. Davie
VMware, Inc.
R. van Brandenburg, Ed.
TNO
June 6, 2014

Framework for CDN Interconnection
draft-ietf-cdni-framework-14

Abstract

This document presents a framework for Content Distribution Network Interconnection (CDNI). The purpose of the framework is to provide an overall picture of the problem space of CDNI and to describe the relationships among the various components necessary to interconnect CDNs. CDN Interconnection requires the specification of interfaces and mechanisms to address issues such as request routing, distribution metadata exchange, and logging information exchange across CDNs. The intent of this document is to outline what each interface needs to accomplish, and to describe how these interfaces and mechanisms fit together, while leaving their detailed specification to other documents. This document, in combination with RFC 6707, obsoletes RFC 3466.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 8, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Reference Model	5
1.3. Structure Of This Document	9
2. Building Blocks	9
2.1. Request Redirection	9
2.1.1. DNS Redirection	9
2.1.2. HTTP Redirection	11
3. Overview of CDNI Operation	11
3.1. Preliminaries	13
3.2. Iterative HTTP Redirect Example	14
3.3. Recursive HTTP Redirection Example	19
3.4. Iterative DNS-based Redirection Example	23
3.4.1. Notes on using DNSSEC	27
3.5. Dynamic Footprint Discovery Example	28
3.6. Content Removal Example	30
3.7. Pre-Positioned Content Acquisition Example	31
3.8. Asynchronous CDNI Metadata Example	32
3.9. Synchronous CDNI Metadata Acquisition Example	34
3.10. Content and Metadata Acquisition with Multiple Upstream CDNs	36
4. Main Interfaces	37
4.1. In-Band versus Out-of-Band Interfaces	38
4.2. Cross Interface Concerns	38
4.3. Request Routing Interfaces	39
4.4. CDNI Logging Interface	40
4.5. CDNI Control Interface	42
4.6. CDNI Metadata Interface	42
4.7. HTTP Adaptive Streaming Concerns	43
4.8. URI Rewriting	44
5. Deployment Models	45

5.1. Meshed CDNs	46
5.2. CSP combined with CDN	47
5.3. CSP using CDNI Request Routing Interface	47
5.4. CDN Federations and CDN Exchanges	48
6. Trust Model	51
7. IANA Considerations	52
8. Privacy Considerations	52
9. Security Considerations	53
9.1. Security of CDNI Interfaces	54
9.2. Digital Rights Management	54
10. Contributors	54
11. Acknowledgements	54
12. Informative References	55
Authors' Addresses	56

1. Introduction

This document provides an overview of the various components necessary to interconnect CDNs, expanding on the problem statement and use cases introduced in [RFC6770] and [RFC6707]. It describes the necessary interfaces and mechanisms in general terms and outlines how they fit together to form a complete system for CDN Interconnection. Detailed specifications are left to other documents. This document makes extensive use of message flow examples to illustrate the operation of interconnected CDNs, but these examples should be considered illustrative rather than prescriptive.

[RFC3466] uses different terminology and models for "Content Internetworking (CDI)". It is also less prescriptive in terms of interfaces. To avoid confusion, this document obsoletes [RFC3466].

1.1. Terminology

This document uses the core terminology defined in [RFC6707]. It also introduces the following terms:

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

Distinguished CDN-Domain: a CDN-Domain that is allocated by a CDN for the purposes of communication with a peer CDN, but which is not found

in client requests. Such CDN-Domains may be used for inter-CDN acquisition, or as redirection targets, and enable a CDN to distinguish a request from a peer CDN from an end-user request.

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

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

Recursive CDNI Request Redirection: When an upstream CDN elects to redirect a request towards a downstream CDN, the upstream CDN can query the downstream CDN Request Routing system via the CDNI Request Routing Redirection Interface (or use information cached from earlier similar queries) to find out how the downstream CDN wants the request to be redirected. This allows the upstream CDN to factor in the downstream CDN response when redirecting the user agent. This approach is referred to as "Recursive" CDNI Request Redirection. Note that the downstream CDN may elect to have the request redirected directly to a Surrogate inside the downstream CDN, or to any other element in the downstream CDN (or in another CDN) to handle the redirected request appropriately.

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

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

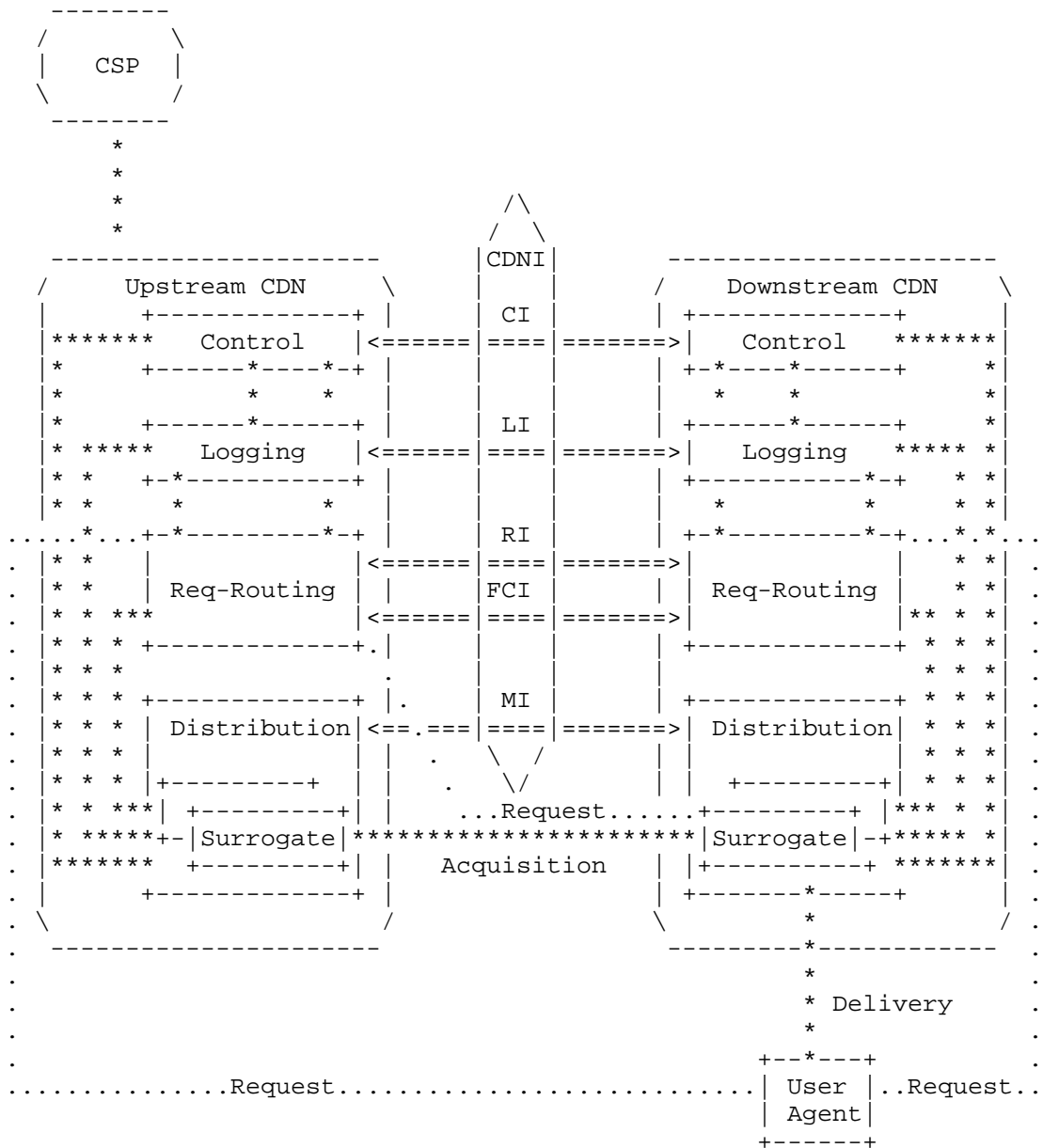
Trigger Interface: a subset of the CDNI Control interface that includes operations to pre-position, revalidate, and purge both metadata and content. These operations are typically called in response to some action (Trigger) by the Content Service Provider (CSP) on the upstream CDN.

We also sometimes use uCDN and dCDN as shorthand for upstream CDN and downstream CDN (see [RFC6707]), respectively.

At various points in this document, the concept of a CDN footprint is used. For a discussion on what constitutes a CDN footprint, the reader is referred to [I-D.ietf-cdni-footprint-capabilities-semantics].

1.2. Reference Model

This document uses the reference model in Figure 1, which expands the reference model originally defined in [RFC6707]. (The difference is that the expanded model splits the Request Routing Interface into its two distinct parts: the Request Routing Redirection interface and the Footprint and Capabilities Advertisement interface, as described below.)



\Leftrightarrow interfaces inside the scope of CDNI

**** and interfaces outside the scope of CDNI

Figure 1: CDNI Expanded Model and CDNI Interfaces

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

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

The following briefly describes the five CDNI interfaces, paraphrasing the definitions given in [RFC6707]. We discuss these interfaces in more detail in Section 4.

- o CDNI Control interface (CI): Operations to bootstrap and parameterize the other CDNI interfaces, as well as operations to pre-position, revalidate, and purge both metadata and content. The latter subset of operations is sometimes collectively called the "Trigger interface."
- o CDNI Request Routing interface: Operations to determine what CDN (and optionally what surrogate within a CDN) is to serve end-user's requests. This interface is actually a logical bundling of two separate but related interfaces:
 - * CDNI Footprint & Capabilities Advertisement interface (FCI): Asynchronous operations to exchange routing information (e.g., the network footprint and capabilities served by a given CDN) that enables CDN selection for subsequent user requests; and
 - * CDNI Request Routing Redirection interface (RI): Synchronous operations to select a delivery CDN (surrogate) for a given user request.
- o CDNI Metadata interface (MI): Operations to communicate metadata that governs how the content is delivered by interconnected CDNs. Examples of CDNI metadata include geo-blocking directives, availability windows, access control mechanisms, and purge directives. It may include a combination of:
 - * Asynchronous operations to exchange metadata that govern subsequent user requests for content; and
 - * Synchronous operations that govern behavior for a given user request for content.

- o CDNI Logging interface (LI): Operations that allow interconnected CDNs to exchange relevant activity logs. It may include a combination of:
 - * Real-time exchanges, suitable for runtime traffic monitoring; and
 - * Offline exchanges, suitable for analytics and billing.

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

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

Finally, there is a practical issue that impacts all of the CDNI interfaces, and that is whether or not to optimize CDNI for HTTP Adaptive Streaming (HAS). We highlight specific issues related to delivering HAS content throughout this document, but for a more thorough treatment of the topic, see [RFC6983].

1.3. Structure Of This Document

The remainder of this document is organized as follows:

- o Section 2 describes some essential building blocks for CDNI, notably the various options for redirecting user requests to a given CDN.
- o Section 3 provides a number of illustrative examples of various CDNI operations.
- o Section 4 describes the functionality of the main CDNI interfaces.
- o Section 5 shows how various deployment models of CDNI may be achieved using the defined interfaces.
- o Section 6 describes the trust model of CDNI and the issues of transitive trust in particular that CDNI raises.

2. Building Blocks

2.1. Request Redirection

At its core, CDN Interconnection requires the redirection of requests from one CDN to another. For any given request that is received by an upstream CDN, it will either respond to the request directly, or somehow redirect the request to a downstream CDN. Two main mechanisms are available for redirecting a request to a downstream CDN. The first leverages the DNS name resolution process and the second uses application-layer redirection mechanisms such as the HTTP 302 or RTSP 302 redirection responses. While there exists a large variety of application-layer protocols that include some form of redirection mechanism, this document will use HTTP (and HTTPS) in its examples. Similar mechanisms can be applied to other application-layer protocols. What follows is a short discussion of both DNS- and HTTP-based redirection, before presenting some examples of their use in Section 3.

2.1.1. DNS Redirection

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

This latter possibility is important because the authoritative DNS server sees only the IP address of the DNS server that queries it, not the IP address of the original end-user.

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

In simple DNS redirection, the authoritative DNS server for the name simply returns an IP address from a set of possible IP addresses. The answer is chosen from the set based on characteristics of the set (e.g., the relative loads on the servers) or characteristics of the client (e.g., the location of the client relative to the servers). Simple redirection is straightforward. The only caveats are (1) there is a limit to the number of alternate IP addresses a single DNS server can manage; and (2) DNS responses are cached by downstream servers so the TTL on the response must be set to an appropriate value so as to preserve the freshness of the redirection.

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

One of the advantages of DNS redirection compared to HTTP redirection is that it can be cached, reducing load on the redirecting CDN's DNS server. However, this advantage can also be a drawback, especially when a given DNS resolver doesn't strictly adhere to the TTL, which is a known problem in some real world environments. In such cases, an end-user might end up at a dCDN without first having passed through the uCDN, which might be an undesirable scenario from a uCDN point of view.

2.1.2. HTTP Redirection

HTTP redirection makes use of the redirection response of the HTTP protocol (e.g., "302" or "307"). This response contains a new URL that the application should fetch instead of the original URL. By changing the URL appropriately, the server can cause the user to redirect to a different server. The advantages of HTTP redirection are that (1) the server can change the URL fetched by the client to include, for example, both the DNS name of the particular server to use, as well as the original HTTP server that was being accessed; (2) the client sends the HTTP request to the server, so that its IP address is known and can be used in selecting the server; and (3) other attributes (e.g., content type, user agent type) are visible to the redirection mechanism.

Just as is the case for DNS redirection, there are some potential disadvantages of using HTTP redirection. For example, it may affect application behavior, e.g. web browsers will not send cookies if the URL changes to a different domain. In addition, although this might also be an advantage, results of HTTP redirection are not cached so that all redirections must go through to the uCDN.

3. Overview of CDNI Operation

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

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

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

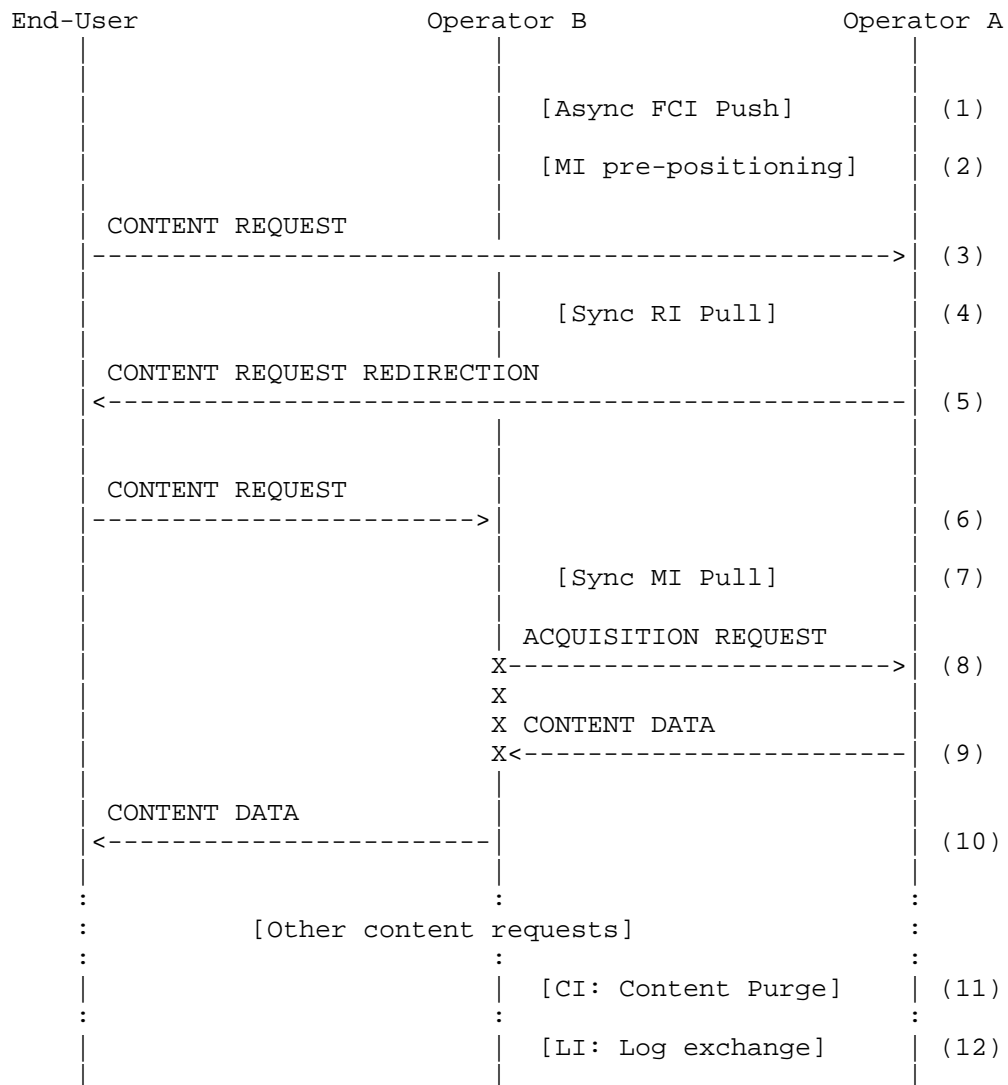


Figure 2: Overview of Operation

The operations shown in the Figure are as follows:

1. dCDN uses the FCI to advertise information relevant to its delivery footprint and capabilities prior to any content requests being redirected.

2. Prior to any content request, the uCDN uses the MI to pre-position CDNI metadata to the dCDN, thereby making that metadata available in readiness for later content requests.
3. A content request from a user agent arrives at uCDN.
4. uCDN may use the RI to synchronously request information from dCDN regarding its delivery capabilities to decide if dCDN is a suitable target for redirection of this request.
5. uCDN redirects the request to dCDN by sending some response (DNS, HTTP) to the user agent.
6. The user agent requests the content from dCDN.
7. dCDN may use the MI to synchronously request metadata related to this content from uCDN, e.g. to decide whether to serve it.
8. If the content is not already in a suitable cache in dCDN, dCDN may acquire it from uCDN.
9. The content is delivered to dCDN from uCDN.
10. The content is delivered to the user agent by dCDN.
11. Some time later, perhaps at the request of the CSP (not shown) uCDN may use the CI to instruct dCDN to purge the content, thereby ensuring it is not delivered again.
12. After one or more content delivery actions by dCDN, a log of delivery actions may be provided to uCDN using the LI.

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

3.1. Preliminaries

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

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

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

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

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

3.2. Iterative HTTP Redirect Example

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

We assume for this example that Operators A and B have established an agreement to interconnect their CDNs, with A being upstream and B being downstream.

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

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

We assume the operators also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical

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

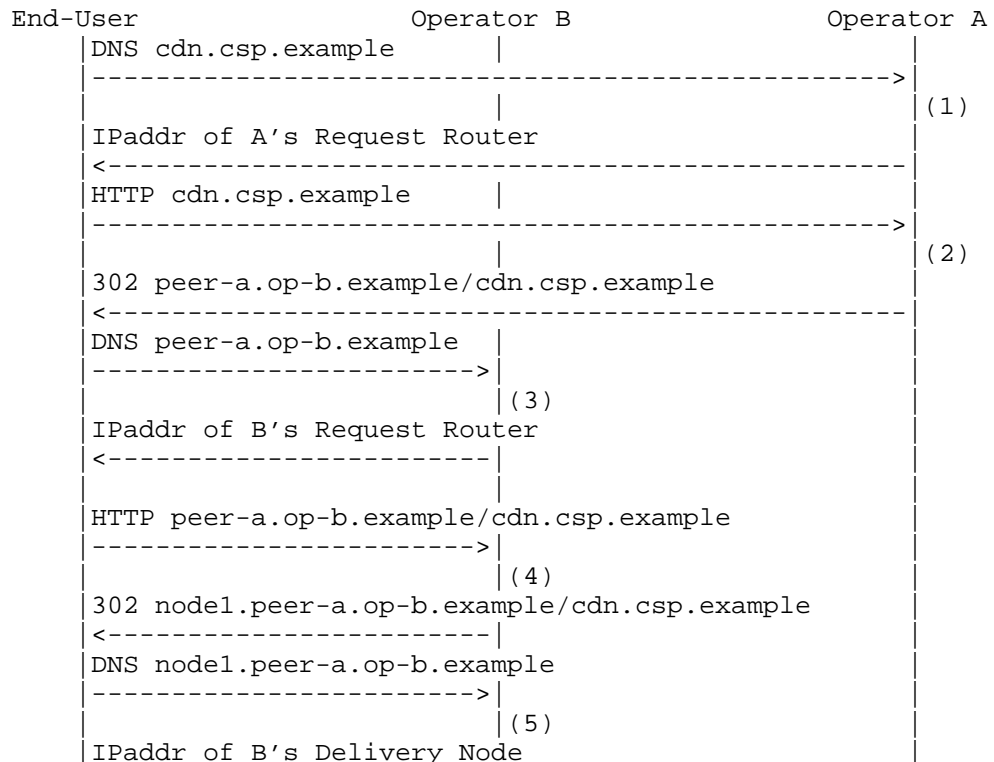
We assume DNS is configured in the following way:

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

Figure 3 illustrates how a client request for

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

is handled.



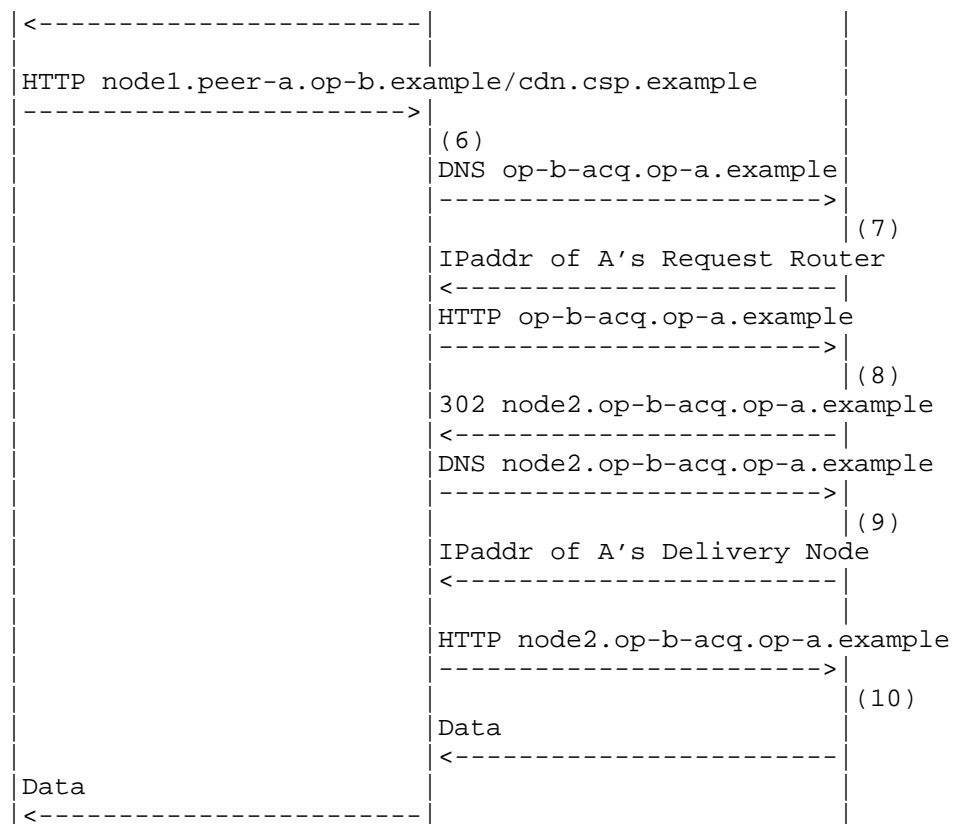


Figure 3: Message Flow for Iterative HTTP Redirection

The steps illustrated in the figure are as follows:

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

IP address of a request router for Operator B. Note that if request routing within dCDN was performed using DNS instead of HTTP redirection, B's DNS resolver would also behave as the request router and directly return the IP address of a delivery node.

4. The request router for Operator B processes the HTTP request and selects a suitable delivery node to serve the end-user request, and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator B's distinguished CDN-Domain that points to the selected delivery node.
5. The end-user does a DNS lookup using Operator B's delivery node subdomain (node1.peer-a.op-b.example). B's DNS resolver returns the IP address of the delivery node.
6. The end-user requests the content from B's delivery node. In the case of a cache hit, steps 6, 7, 8, 9 and 10 below do not happen, and the content data is directly returned by the delivery node to the end-user. In the case of a cache miss, the content needs to be acquired by dCDN from uCDN (not the CSP). The distinguished CDN-Domain peer-a.op-b.example indicates to dCDN that this content is to be acquired from uCDN; stripping the CDN-Domain reveals the original CDN-Domain cdn.csp.example and dCDN may verify that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an inter-CDN acquisition CDN-Domain as agreed above (in this case, op-b-acq.op-a.example).
7. Operator A's DNS resolver processes the DNS request and returns the IP address of a request router in operator A.
8. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (op-b-acq.op-a.example). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.

9. Operator A DNS resolver processes the DNS request and returns the IP address of the delivery node in operator A.
10. Operator B requests (acquires) the content from Operator A. Although not shown, Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

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

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

We note that this example begins to illustrate some of the interfaces that may be required for CDNI, but does not require all of them. For example, obtaining information from dCDN regarding the set of client IP addresses or geographic regions it might be able to serve is an aspect of request routing (specifically of the CDNI Footprint & Capabilities Advertisement interface). Important configuration information such as the distinguished names used for redirection and inter-CDN acquisition could also be conveyed via a CDNI interface (e.g., perhaps the CDNI Control interface). The example also shows how existing HTTP-based methods suffice for the acquisition interface. Arguably, the absolute minimum metadata required for CDNI is the information required to acquire the content, and this information was provided "in-band" in this example by means of the URI handed to the client in the HTTP 302 response. The example also

assumes that the CSP does not require any distribution policy (e.g. time window, geo-blocking) or delivery processing to be applied by the interconnected CDNs. Hence, there is no explicit CDNI Metadata interface invoked in this example. There is also no explicit CDNI Logging interface discussed in this example.

We also note that the step of deciding when a request should be redirected to dCDN rather than served by uCDN has been somewhat glossed over. It may be as simple as checking the client IP address against a list of prefixes, or it may be considerably more complex, involving a wide range of factors, such as the geographic location of the client (perhaps determined from a third party service), CDN load, or specific business rules.

This example uses the "iterative" CDNI request redirection approach. That is, uCDN performs part of the request redirection function by redirecting the client to a request router in the dCDN, which then performs the rest of the redirection function by redirecting to a suitable surrogate. If request routing is performed in the dCDN using HTTP redirection, this translates in the end-user experiencing two successive HTTP redirections. By contrast, the alternative approach of "recursive" CDNI request redirection effectively coalesces these two successive HTTP redirections into a single one, sending the end-user directly to the right delivery node in the dCDN. This "recursive" CDNI request routing approach is discussed in the next section.

While the example above uses HTTP, the iterative HTTP redirection mechanism would work over HTTPS in a similar fashion. In order to make sure an end-user's HTTPS request is not downgraded to HTTP along the redirection path, it is necessary for every request router along the path from the initial uCDN Request Router to the final surrogate in the dCDN to respond to an incoming HTTPS request with an HTTP Redirect containing an HTTPS URL. It should be noted that using HTTPS will have the effect of increasing the total redirection process time and increasing the load on the request routers, especially when the redirection path includes many redirects and thus many TLS/SSL sessions. In such cases, a recursive HTTP redirection mechanism, as described in an example in the next section, might help to reduce some of these issues.

3.3. Recursive HTTP Redirection Example

The following example builds on the previous one to illustrate the use of the request routing interface (specifically the CDNI Request Routing Redirection interface) to enable "recursive" CDNI request routing. We build on the HTTP-based redirection approach because it illustrates the principles and benefits clearly, but it is equally

possible to perform recursive redirection when DNS-based redirection is employed.

In contrast to the prior example, the operators need not agree in advance on a CDN-Domain to serve as the target of redirections from uCDN to dCDN. We assume that the operators agree on some distinguished CDN-Domain that will be used for inter-CDN acquisition of CSP's content by dCDN. In this example, we'll use op-b-acq.op-a.example.

We assume the operators also exchange information regarding which requests dCDN is prepared to serve. For example, dCDN may be prepared to serve requests from clients in a given geographical region or a set of IP address prefixes. This information may again be provided out of band or via a defined protocol.

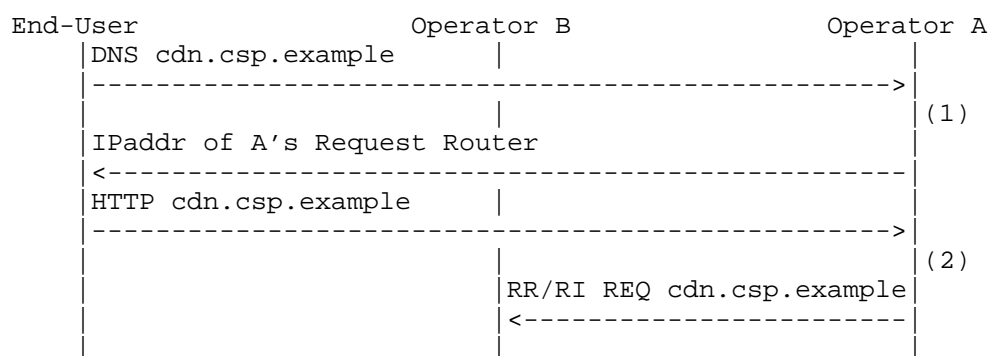
We assume DNS is configured in the following way:

- o The content provider is configured to make operator A the authoritative DNS server for cdn.csp.example (or to return a CNAME for cdn.csp.example for which operator A is the authoritative DNS server).
- o Operator A is configured so that a DNS request for op-b-acq.op-a.example returns a request router in Operator A.
- o Operator B is configured so that a request for node1.op-b.example/cdn.csp.example returns the IP address of a delivery node. Note that there might be a number of such delivery nodes.

Figure 3 illustrates how a client request for

http://cdn.csp.example/...rest of url...

is handled.



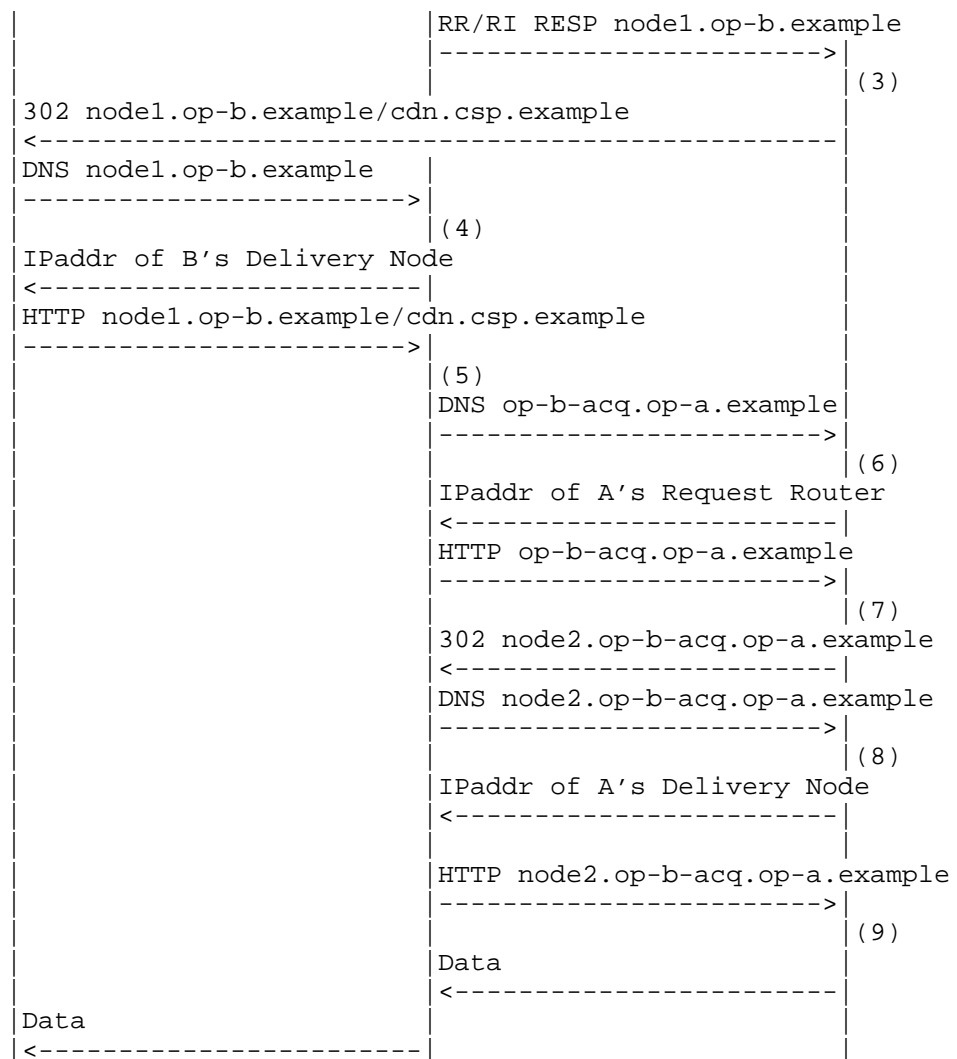


Figure 4: Message Flow for Recursive HTTP Redirection

The steps illustrated in the figure are as follows:

1. A DNS resolver for Operator A processes the DNS request for its customer based on CDN-Domain `cdn.csp.example`. It returns the IP address of a Request Router in Operator A.
2. A Request Router for Operator A processes the HTTP request and recognizes that the end-user is best served by another CDN--specifically one provided by Operator B--and so it queries the

CDNI Request Routing Redirection interface of Operator B, providing a set of information about the request including the URL requested. Operator B replies with the DNS name of a delivery node.

3. Operator A returns a 302 redirect message for a new URL obtained from the RI.
4. The end-user does a DNS lookup using the host name of the URL just provided (node1.op-b.example). B's DNS resolver returns the IP address of the corresponding delivery node. Note that, since the name of the delivery node was already obtained from B using the RI, there should not be any further redirection here (in contrast to the iterative method described above.)
5. The end-user requests the content from B's delivery node, potentially resulting in a cache miss. In the case of a cache miss, the content needs to be acquired from uCDN (not the CSP.) The distinguished CDN-Domain op-b.example indicates to dCDN that this content is to be acquired from another CDN; stripping the CDN-Domain reveals the original CDN-Domain cdn.csp.example, dCDN may verify that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for the inter-CDN Acquisition "distinguished" CDN-Domain as agreed above (in this case, op-b-acq.op-a.example).
6. Operator A DNS resolver processes the DNS request and returns the IP address of a request router in operator A.
7. The request router for Operator A processes the HTTP request from Operator B delivery node. Operator A request router recognizes that the request is from a peer CDN rather than an end-user because of the dedicated inter-CDN acquisition domain (op-b-acq.op-a.example). (Note that without this specially defined inter-CDN acquisition domain, operator A would be at risk of redirecting the request back to operator B, resulting in an infinite loop). The request router for Operator A selects a suitable delivery node in uCDN to serve the inter-CDN acquisition request and returns a 302 redirect message for a new URL constructed by replacing the hostname with a subdomain of the Operator A's distinguished inter-CDN acquisition domain that points to the selected delivery node.
8. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-Domain) and so returns the address of a delivery node. (Note that without this specially defined internal domain, Operator A would be at risk of

redirecting the request back to Operator B, resulting in an infinite loop.)

9. Operator B requests (acquires) the content from Operator A. Operator A serves content for the requested CDN-Domain to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server. It may also perform its own content acquisition steps if needed before returning the content to dCDN.

Recursive redirection has the advantage over iterative of being more transparent from the end-user's perspective, but the disadvantage of each CDN exposing more of its internal structure (in particular, the addresses of edge caches) to peer CDNs. By contrast, iterative redirection does not require dCDN to expose the addresses of its edge caches to uCDN.

This example happens to use HTTP-based redirection in both CDN A and CDN B, but a similar example could be constructed using DNS-based redirection in either CDN. Hence, the key point to take away here is simply that the end user only sees a single redirection of some type, as opposed to the pair of redirections in the prior (iterative) example.

The use of the RI requires that the request routing mechanism be appropriately configured and bootstrapped, which is not shown here. More discussion on the bootstrapping of interfaces is provided in Section 4

3.4. Iterative DNS-based Redirection Example

In this section we walk through a simple example using DNS-based redirection for request redirection from uCDN to dCDN (as well as for request routing inside dCDN and uCDN). As noted in Section 2.1, DNS-based redirection has certain advantages over HTTP-based redirection (notably, it is transparent to the end-user) as well as some drawbacks (notably the client IP address is not visible to the request router).

As before, Operator A has to learn the set of requests that dCDN is willing or able to serve (e.g. which client IP address prefixes or geographic regions are part of the dCDN footprint). We assume Operator B has and makes known to Operator A some unique identifier that can be used for the construction of a distinguished CDN-Domain, as shown in more detail below. (This identifier strictly needs only to be unique within the scope of Operator A, but a globally unique

identifier, such as an AS number assigned to B, is one easy way to achieve that.) Also, Operator A obtains the NS records for Operator B's externally visible redirection servers. Also, as before, a distinguished CDN-Domain, such as `op-b-acq.op-a.example`, must be assigned for inter-CDN acquisition.

We assume DNS is configured in the following way:

- o The CSP is configured to make Operator A the authoritative DNS server for `cdn.csp.example` (or to return a CNAME for `cdn.csp.example` for which operator A is the authoritative DNS server).
- o When uCDN sees a request best served by dCDN, it returns CNAME and NS records for `"b.cdn.csp.example"`, where "b" is the unique identifier assigned to Operator B. (It may, for example, be an AS number assigned to Operator B.)
- o dCDN is configured so that a request for `"b.cdn.csp.example"` returns a delivery node in dCDN.
- o uCDN is configured so that a request for `"op-b-acq.op-a.example"` returns a delivery node in uCDN.

Figure 5 depicts the exchange of DNS and HTTP requests. The main differences from Figure 3 are the lack of HTTP redirection and transparency to the end-user.

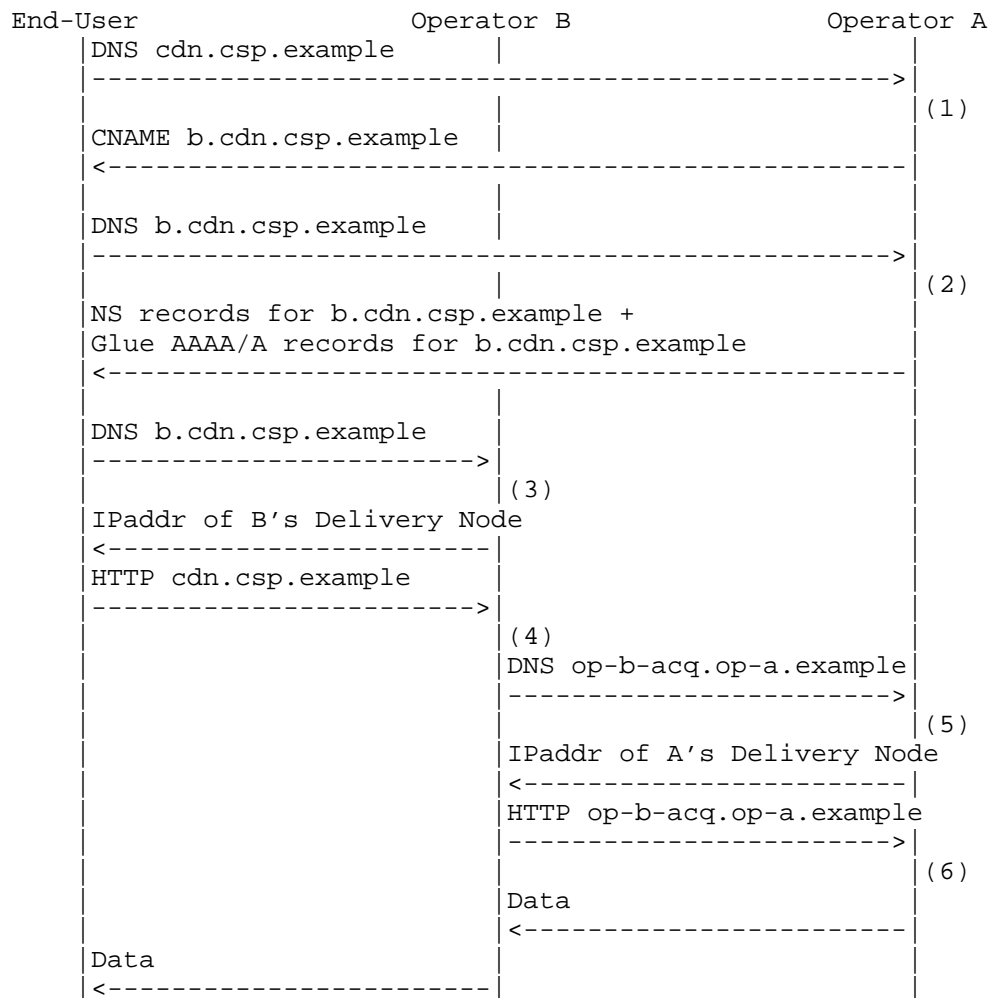


Figure 5: Message Flow for DNS-based Redirection

The steps illustrated in the figure are as follows:

1. Request Router for Operator A processes the DNS request for CDN-Domain `cdn.csp.example` and recognizes that the end-user is best served by another CDN. (This may depend on the IP address of the user's local DNS resolver, or other information discussed below.) The Request Router returns a DNS CNAME response by "stacking" the distinguished identifier for Operator B onto the original CDN-Domain (e.g., `b.cdn.csp.example`).

2. The end-user sends a DNS query for the modified CDN-Domain (i.e. b.cdn.csp.example) to Operator A's DNS server. The Request Router for Operator A processes the DNS request and return a delegation to b.cdn.csp.example by sending an NS record plus glue AAAA/A records pointing to Operator B's DNS server. (This extra step is necessary since typical DNS implementation won't follow an NS record when it is sent together with a CNAME record, thereby necessitating a two-step approach).
3. The end-user sends a DNS query for the modified CDN-Domain (i.e., b.cdn.csp.example) to Operator B's DNS server, using the NS and AAAA/A records received in step 2. This causes B's Request Router to respond with a suitable delivery node.
4. The end-user requests the content from B's delivery node. The requested URL contains the name cdn.csp.example. (Note that the returned CNAME does not affect the URL.) At this point the delivery node has the correct IP address of the end-user and can do an HTTP 302 redirect if the redirections in steps 2 and 3 were incorrect. Otherwise B verifies that this CDN-Domain belongs to a known peer (so as to avoid being tricked into serving as an open proxy). It then does a DNS request for an "internal" CDN-Domain as agreed above (op-b-acq.op-a.example).
5. Operator A recognizes that the DNS request is from a peer CDN rather than an end-user (due to the internal CDN-Domain) and so returns the address of a delivery node in uCDN.
6. Operator A serves content to dCDN. Although not shown, it is at this point that Operator A processes the rest of the URL: it extracts information identifying the origin server, validates that this server has been registered, and determines the content provider that owns the origin server.

The advantages of this approach are that it is more transparent to the end-user and requires fewer round trips than HTTP-based redirection (in its worst case, i.e., when none of the needed DNS information is cached). A potential problem is that the upstream CDN depends on being able to learn the correct downstream CDN that serves the end-user from the client address in the DNS request. In standard DNS operation, uCDN will only obtain the address of the client's local DNS resolver (LDNS), which is not guaranteed to be in the same network (or geographic region) as the client. If not--e.g., the end-user uses a global DNS service--then the upstream CDN cannot determine the appropriate downstream CDN to serve the end-user. In this case, and assuming the uCDN is capable of detecting that situation, one option is for the upstream CDN to treat the end-user as it would any user not connected to a peer CDN. Another option is

for the upstream CDN to "fall back" to a pure HTTP-based redirection strategy in this case (i.e., use the first method). Note that this problem affects existing CDNs that rely on DNS to determine where to redirect client requests, but the consequences are arguably less serious for CDNI since the LDNS is likely in the same network as the dCDN serves.

As with the prior example, this example partially illustrates the various interfaces involved in CDNI. Operator A could learn dynamically from Operator B the set of prefixes or regions that B is willing and able to serve via the CDNI Footprint & Capabilities Advertisement interface. The distinguished name used for acquisition and the identifier for Operator B that is prepended to the CDN-Domain on redirection are examples of information elements that might also be conveyed by CDNI interfaces (or, alternatively, statically configured). As before, minimal metadata sufficient to obtain the content is carried "in-band" as part of the redirection process, and standard HTTP is used for inter-CDN acquisition. There is no explicit CDNI Logging interface discussed in this example.

3.4.1.1. Notes on using DNSSEC

Although it is possible to use DNSSEC in combination with the Iterative DNS-based Redirection mechanism explained above, it is important to note that the uCDN might have to sign records on the fly, since the CNAME returned, and thus the signature provided, can potentially be different for each incoming query. Although there is nothing preventing a uCDN from performing such on-the-fly signing, this might be computationally expensive. In the case where the number of dCDNs, and thus the number of different CNAMEs to return, is relatively stable, an alternative solution would be for the uCDN to pre-generate signatures for all possible CNAMEs. For each incoming query the uCDN would then determine the appropriate CNAME and return it together with the associated pre-generated signature. Note: In the latter case maintaining the serial and signature of SOA might be an issue since technically it should change every time a different CNAME is used. However, since in practice direct SOA queries are relatively rare, a uCDN could defer incrementing the serial and resigning the SOA until it is queried and then do it on-the-fly.

Note also that the NS record and the glue AAAA/A records used in step 2 in the previous section should generally be identical to those of their authoritative zone managed by Operator B. Even if they differ, this will not make the DNS resolution process fail, but the client DNS server will prefer the authoritative data in its cache and use it for subsequent queries. Such inconsistency is a general operational issue of DNS, but it may be more important for this architecture

because the uCDN (operator A) would rely on the consistency to make the resulting redirection work as intended. In general, it is the administrator's responsibility to make them consistent.

3.5. Dynamic Footprint Discovery Example

There could be situations where being able to dynamically discover the set of requests that a given dCDN is willing and able to serve is beneficial. For example, a CDN might at one time be able to serve a certain set of client IP prefixes, but that set might change over time due to changes in the topology and routing policies of the IP network. The following example illustrates this capability. We have chosen the example of DNS-based redirection, but HTTP-based redirection could equally well use this approach.

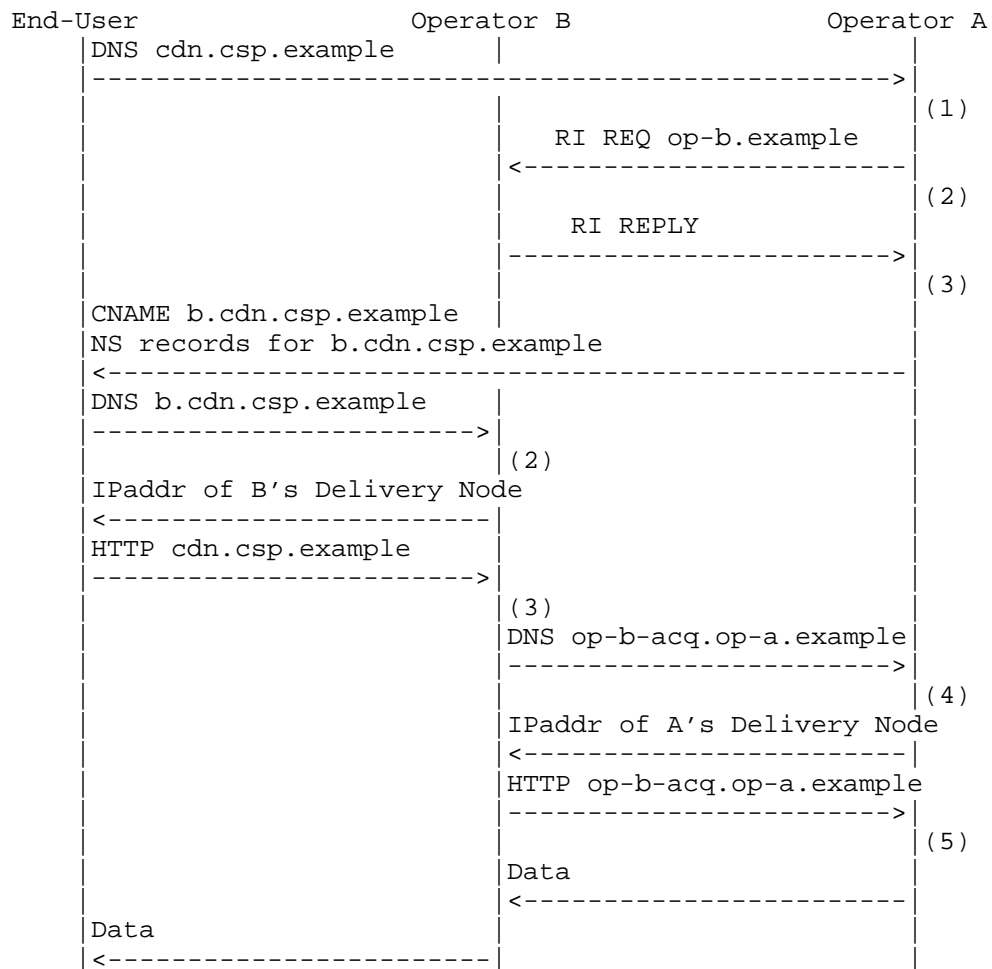


Figure 6: Message Flow for Dynamic Footprint Discovery

This example differs from the one in Figure 5 only in the addition of a RI request (step 2) and corresponding response (step 3). The RI REQ could be a message such as "Can you serve clients from this IP Prefix?" or it could be "Provide the list of client IP prefixes you can currently serve". In either case the response might be cached by operator A to avoid repeatedly asking the same question. Alternatively, or in addition, Operator B may spontaneously advertise to Operator A information (or changes) on the set of requests it is willing and able to serve on behalf of operator A; in that case, Operator B may spontaneously issue RR/RI REPLY messages that are not in direct response to a corresponding RR/RI REQ message. (Note that

the issues of determining the client's subnet from DNS requests, as described above, are exactly the same here as in Section 3.4.)

Once Operator A obtains the RI response, it is now able to determine that Operator B's CDN is an appropriate dCDN for this request and therefore a valid candidate dCDN to consider in its Redirection decision. If that dCDN is selected, the redirection and serving of the request proceeds as before (i.e. in the absence of dynamic footprint discovery).

3.6. Content Removal Example

The following example illustrates how the CDNI Control interface may be used to achieve pre-positioning of an item of content in the dCDN. In this example, user requests for a particular content, and corresponding redirection of such requests from Operator A to Operator B CDN, may (or may not) have taken place earlier. Then, at some point in time, the uCDN (for example, in response to a corresponding Trigger from the Content Provider) uses the CI to request that content identified by a particular URL be removed from dCDN. The following diagram illustrates the operation. It should be noted that a uCDN will typically not know whether a dCDN has cached a given content item, however, it may send the content removal request to make sure no cached versions remain to satisfy any contractual obligations it may have.

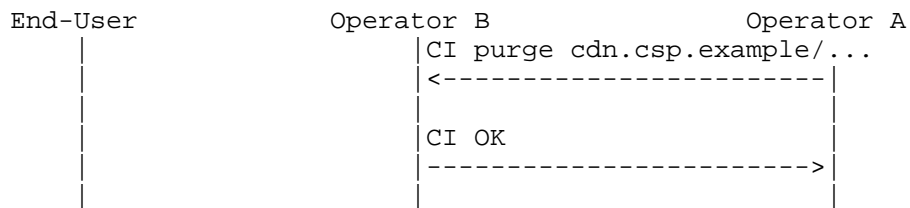


Figure 7: Message Flow for Content Removal

The CI is used to convey the request from uCDN to dCDN that some previously acquired content should be deleted. The URL in the request specifies which content to remove. This example corresponds to a DNS-based redirection scenario such as Section 3.4. If HTTP-based redirection had been used, the URL for removal would be of the form peer-a.op-b.example/cdn.csp.example/...

The dCDN is expected to confirm to the uCDN, as illustrated by the CI OK message, the completion of the removal of the targeted content from all the caches in dCDN.

3.7. Pre-Positioned Content Acquisition Example

The following example illustrates how the CI may be used to pre-position an item of content in the dCDN. In this example, Operator A uses the CDNI Metadata interface to request that content identified by a particular URL be pre-positioned into Operator B CDN.

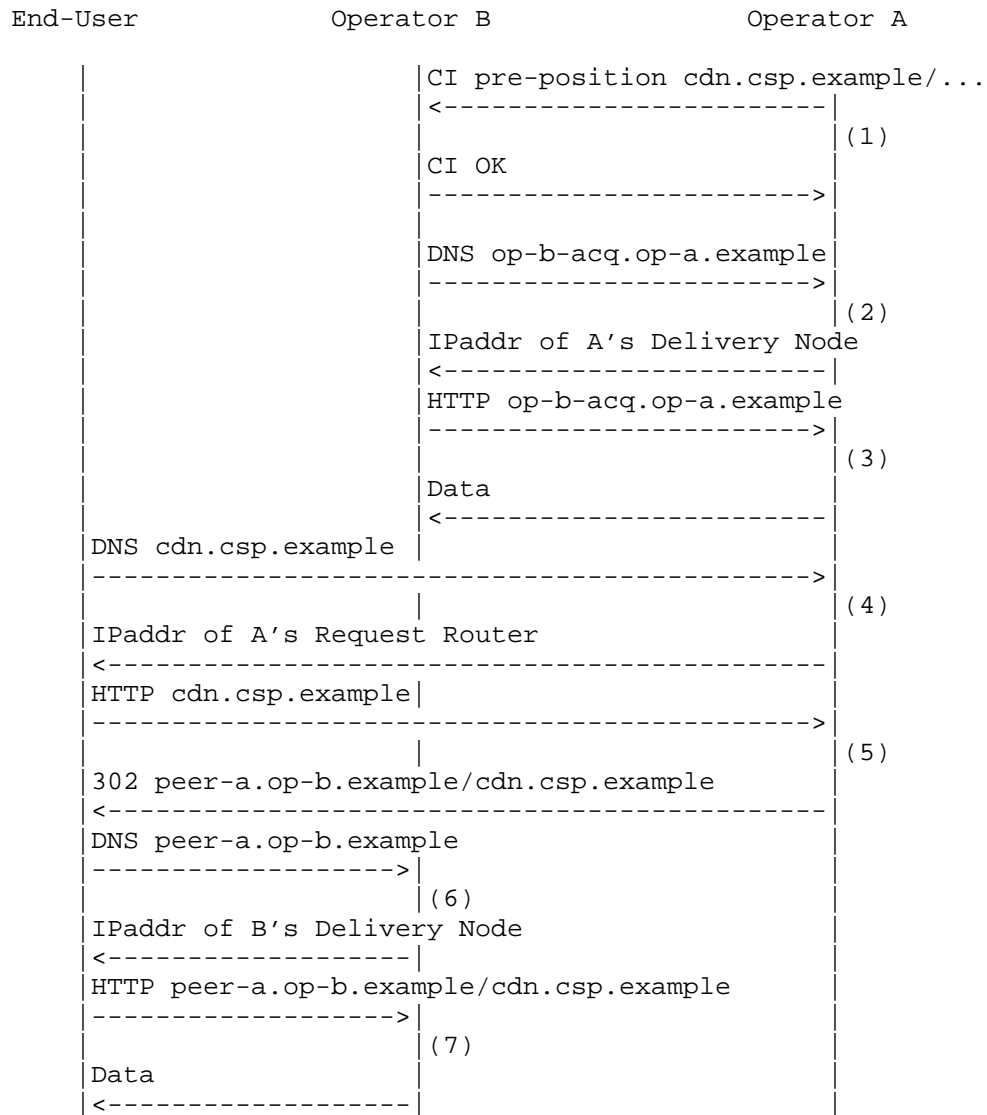


Figure 8: Message Flow for Content Pre-Positioning

The steps illustrated in the figure are as follows:

1. Operator A uses the CI to request that Operator B pre-positions a particular content item identified by its URL. Operator B responds by confirming that it is willing to perform this operation.

Steps 2 and 3 are exactly the same as steps 5 and 6 of Figure 3, only this time those steps happen as the result of the Pre-positioning request instead of as the result of a cache miss.

Steps 4, 5, 6, 7 are exactly the same as steps 1, 2, 3, 4 of Figure 3, only this time Operator B CDN can serve the end-user request without triggering dynamic content acquisition, since the content has been pre-positioned in dCDN. Note that, depending on dCDN operations and policies, the content pre-positioned in the dCDN may be pre-positioned to all, or a subset of, dCDN caches. In the latter case, intra-CDN dynamic content acquisition may take place inside the dCDN serving requests from caches on which the content has not been pre-positioning; however, such intra-CDN dynamic acquisition would not involve the uCDN.

3.8. Asynchronous CDNI Metadata Example

In this section we walk through a simple example illustrating a scenario of asynchronously exchanging CDNI metadata, where the downstream CDN obtains CDNI metadata for content ahead of a corresponding content request. The example that follows assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used, as in Section 3.3. However, Asynchronous exchange of CDNI Metadata is similarly applicable to DNS-based inter-CDN redirection and iterative request routing (in which cases the CDNI metadata may be used at slightly different processing stages of the message flows).

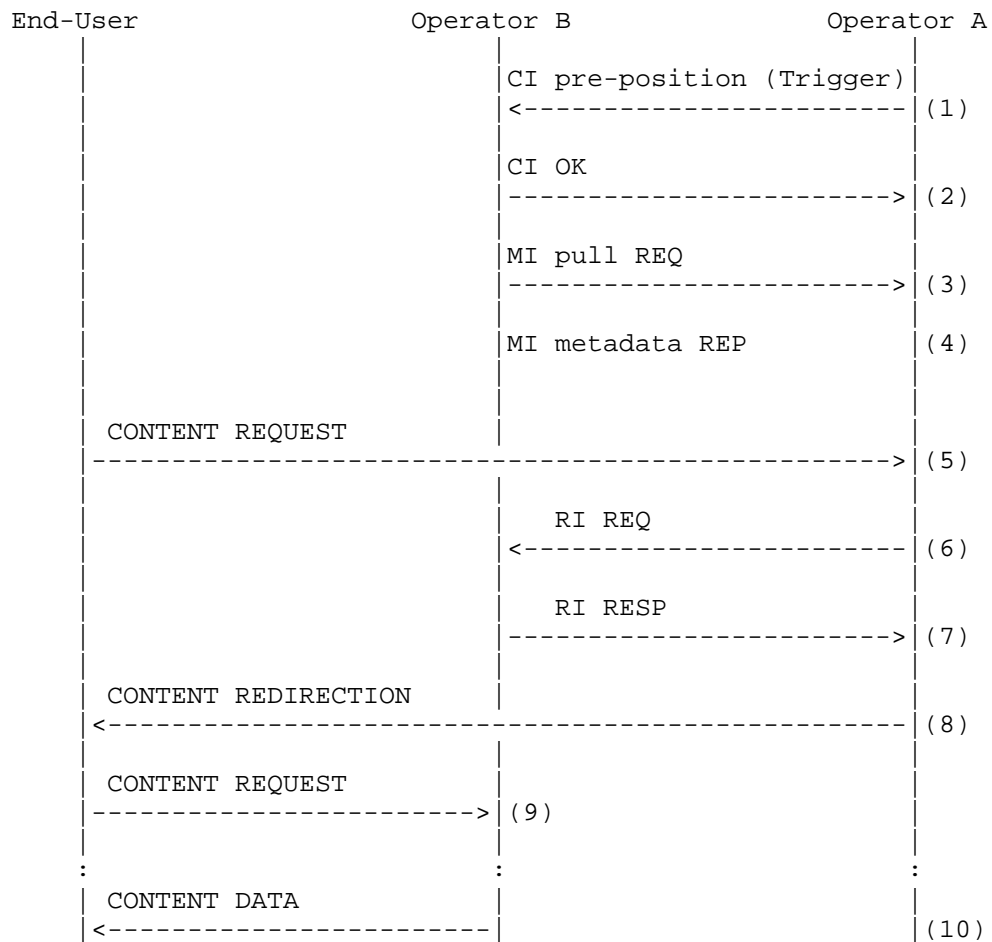


Figure 9: Message Flow for Asynchronous CDNI Metadata

The steps illustrated in the figure are as follows:

1. Operator A uses the CI to Trigger to signal the availability of CDNI metadata to Operator B.
2. Operator B acknowledges the receipt of this Trigger.
3. Operator B requests the latest metadata from Operator A using the MI.
4. Operator A replies with the requested metadata. This document does not constrain how the CDNI metadata information is actually

represented. For the purposes of this example, we assume that Operator A provides CDNI metadata to Operator B indicating that:

- * this CDNI Metadata is applicable to any content referenced by some CDN-Domain.
- * this CDNI metadata consists of a distribution policy requiring enforcement by the delivery node of a specific per-request authorization mechanism (e.g. URI signature or token validation).

5. A Content Request occurs as usual.
6. A CDNI Request Routing Redirection request (RI REQ) is issued by operator A CDN, as discussed in Section 3.3. Operator B's request router can access the CDNI Metadata that are relevant to the requested content and that have been pre-positioned as per Steps 1-4, which may or may not affect the response.
7. Operator B's request router issues a CDNI Request Routing Redirection response (RI RESP) as in Section 3.3.
8. Operator B performs content redirection as discussed in Section 3.3.
9. On receipt of the Content Request by the end user, the delivery node detects that previously acquired CDNI metadata is applicable to the requested content. In accordance with the specific CDNI metadata of this example, the delivery node will invoke the appropriate per-request authorization mechanism, before serving the content. (Details of this authorization are not shown.)
10. Assuming successful per-request authorization, serving of Content Data (possibly preceded by inter-CDN acquisition) proceeds as in Section 3.3.

3.9. Synchronous CDNI Metadata Acquisition Example

In this section we walk through a simple example illustrating a scenario of Synchronous CDNI metadata acquisition, in which the downstream CDN obtains CDNI metadata for content at the time of handling a first request for the corresponding content. As in the preceding section, this example assumes that HTTP-based inter-CDN redirection and recursive CDNI request-routing are used (as in Section 3.3), but dynamic CDNI metadata acquisition is applicable to other variations of request routing.

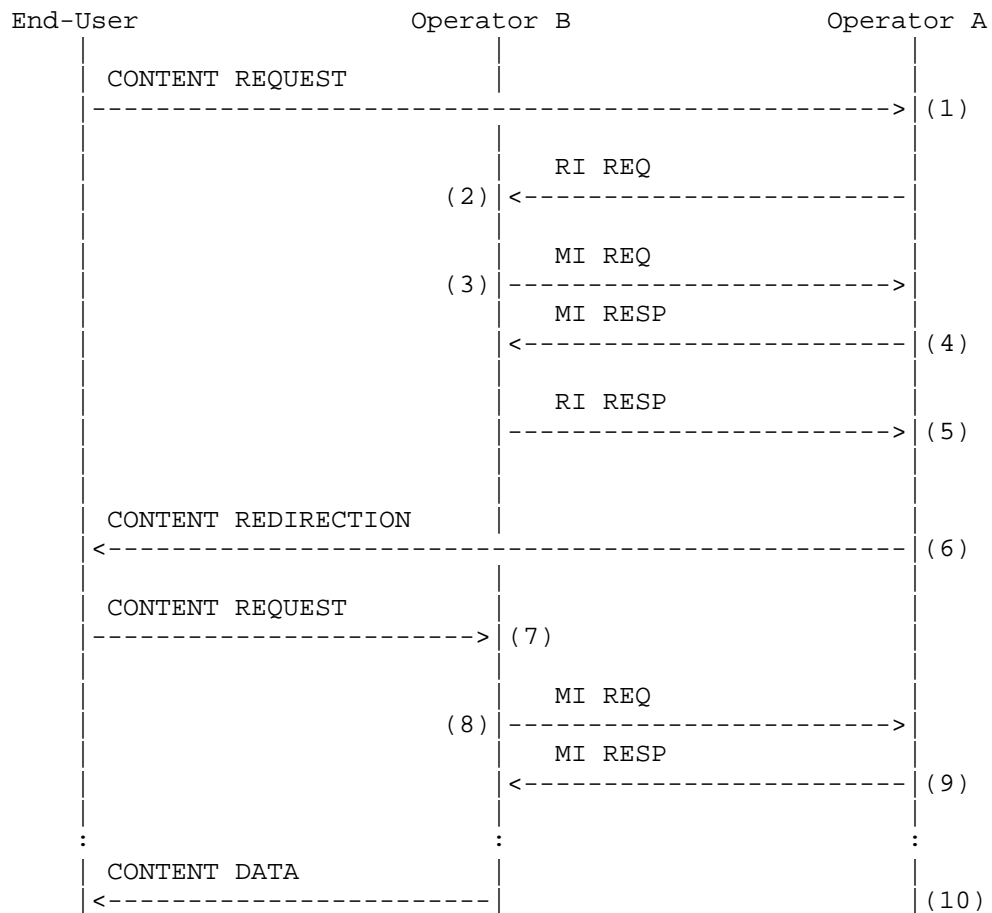


Figure 10: Message Flow for Synchronous CDNI Metadata Acquisition

The steps illustrated in the figure are as follows:

1. A Content Request arrives as normal.
2. An RI request occurs as in the prior example.
3. On receipt of the CDNI Request Routing Request, Operator B's CDN initiates Synchronous acquisition of CDNI Metadata that are needed for routing of the end-user request. We assume the URI for the a Metadata server is known ahead of time through some out-of-band means.

4. On receipt of a CDNI Metadata Request, Operator A's CDN responds, making the corresponding CDNI metadata information available to Operator B's CDN. This metadata is considered by operator B's CDN before responding to the Request Routing request. (In a simple case, the metadata could simply be an allow or deny response for this particular request.)
5. Response to the RI request as normal.
6. Redirection message is sent to the end user.
7. A delivery node of Operator B receives the end user request.
8. The delivery node Triggers dynamic acquisition of additional CDNI metadata that are needed to process the end-user content request. Note that there may exist cases where this step need not happen, for example because the metadata were already acquired previously.
9. Operator A's CDN responds to the CDNI Metadata Request and makes the corresponding CDNI metadata available to Operator B. This metadata influence how Operator B's CDN processes the end-user request.
10. Content is served (possibly preceded by inter-CDN acquisition) as in Section 3.3.

3.10. Content and Metadata Acquisition with Multiple Upstream CDNs

A single dCDN may receive end-user requests from multiple uCDNs. When a dCDN receives an end-user request, it must determine the identity of the uCDN from which it should acquire the requested content.

Ideally, the acquisition path of an end-user request will follow the redirection path of the request. The dCDN should acquire the content from the same uCDN which redirected the request.

Determining the acquisition path requires the dCDN to reconstruct the redirection path based on information in the end-user request. The method for reconstructing the redirection path differs based on the redirection approach: HTTP or DNS.

With HTTP-redirection, the rewritten URI should include sufficient information for the dCDN to directly or indirectly determine the uCDN when the end-user request is received. The HTTP-redirection approach can be further broken-down based on the how the URL is rewritten during redirection: HTTP-redirection with or without Site

Aggregation. HTTP-redirection with Site Aggregation hides the identity of the original CSP. HTTP-redirection without Site Aggregation does not attempt to hide the identity of the original CSP. With both approaches, the rewritten URI includes enough information to identify the immediate neighbor uCDN.

With DNS-redirection, the dCDN receives the published URI (instead of a rewritten URI) and does not have sufficient information for the dCDN to identify the appropriate uCDN. The dCDN may narrow the set of viable uCDNs by examining the CDNI metadata from each to determine which uCDNs are hosting metadata for the requested content. If there is a single uCDN hosting metadata for the requested content, the dCDN can assume that the request redirection is coming from this uCDN and can acquire content from that uCDN. If there are multiple uCDNs hosting metadata for the requested content, the dCDN may be ready to trust any of these uCDNs to acquire the content (provided the uCDN is in a position to serve it). If the dCDN is not ready to trust any of these uCDNs, it needs to ensure via out of band arrangements that, for a given content, only a single uCDN will ever redirect requests to the dCDN.

Content acquisition may be preceded by content metadata acquisition. If possible, the acquisition path for metadata should also follow the redirection path. Additionally, we assume metadata is indexed based on rewritten URIs in the case of HTTP-redirection and is indexed based on published URIs in the case of DNS-redirection. Thus, the RI and the MI are tightly coupled in that the result of request routing (a rewritten URI pointing to the dCDN) serves as an input to metadata lookup. If the content metadata includes information for acquiring the content, then the MI is also tightly coupled with the acquisition interface in that the result of the metadata lookup (an acquisition URL likely hosted by the uCDN) should serve as input to the content acquisition.

4. Main Interfaces

Figure 1 illustrates the main interfaces that are in scope for the CDNI WG, along with several others. The detailed specifications of these interfaces are left to other documents, but see [RFC6707] and [I-D.ietf-cdni-requirements] for some discussion of the interfaces.

One interface that is not shown in Figure 1 is the interface between the user and the CSP. While for the purposes of CDNI that interface is out of scope, it is worth noting that it does exist and can provide useful functions, such as end-to-end performance monitoring and some forms of authentication and authorization.

There is also an important interface between the user and the Request Routing function of both uCDN and dCDN (shown as the "Request" Interface in Figure 1). As we saw in some of the preceding examples, that interface can be used as a way of passing metadata, such as the minimum information that is required for dCDN to obtain the content from uCDN.

In this section we will provide an overview of the functions performed by each of the CDNI interfaces and discuss how they fit into the overall solution. We also examine some of the design tradeoffs, and explore several cross-interface concerns. We begin with an examination of one such tradeoff that affects all the interfaces - the use of in-band or out-of-band communication.

4.1. In-Band versus Out-of-Band Interfaces

Before getting to the individual interfaces, we observe that there is a high-level design choice for each, involving the use of existing in-band communication channels versus defining new out-of-band interfaces.

It is possible that the information needed to carry out various interconnection functions can be communicated between peer CDNs using existing in-band protocols. The use of HTTP 302 redirect is an example of how certain aspects of request routing can be implemented in-band (embedded in URIs). Note that using existing in-band protocols does not imply that the CDNI interfaces are null; it is still necessary to establish the rules (conventions) by which such protocols are used to implement the various interface functions.

There are other opportunities for in-band communication beyond HTTP redirects. For example, many of the HTTP directives used by proxy servers can also be used by peer CDNs to inform each other of caching activity. Of these, one that is particularly relevant is the If-Modified-Since directive, which is used with the GET method to make it conditional: if the requested object has not been modified since the time specified in this field, a copy of the object will not be returned, and instead, a 304 (not modified) response will be returned.

4.2. Cross Interface Concerns

Although the CDNI interfaces are largely independent, there are a set of conventions practiced consistently across all interfaces. Most important among these is how resources are named, for example, how the CDNI Metadata and Control interfaces identify the set of resources to which a given directive applies, or the CDNI Logging

interface identifies the set of resources for which a summary record applies.

While in the limit the CDNI interfaces could explicitly identify every individual resource, in practice, they name resource aggregates (sets of URIs) that are to be treated in a similar way. For example, URI aggregates can be identified by a CDN-Domain (i.e., the FQDN at the beginning of a URI) or by a URI-Filter (i.e., a regular expression that matches a subset of URIs contained in some CDN-Doman). In other words, CDN-Domains and URI-Filters provide a uniform means to aggregate sets (and subsets) of URIs for the purpose of defining the scope for some operation in one of the CDNI interfaces.

4.3. Request Routing Interfaces

The Request Routing interface comprises two parts: the Asynchronous interface used by a dCDN to advertize footprint and capabilities (denoted FCI) to a uCDN, allowing the uCDN to decide whether to redirect particular user requests to that dCDN; and the Synchronous interface used by the uCDN to redirect a user request to the dCDN (denoted RI). (These are somewhat analogous to the operations of routing and forwarding in IP.)

As illustrated in Section 3, the RI part of request routing may be implemented in part by DNS and HTTP. Naming conventions may be established by which CDN peers communicate whether a request should be routed or content served.

We also note that RI plays a key role in enabling recursive redirection, as illustrated in Section 3.3. It enables the user to be redirected to the correct delivery node in dCDN with only a single redirection step (as seen by the user). This may be particularly valuable as the chain of interconnected CDNs increases beyond two CDNs. For further discussion on the RI, see [I-D.ietf-cdni-redirection].

In support of these redirection requests, it is necessary for CDN peers to exchange additional information with each other, and this is the role of the FCI part of request routing. Depending on the method(s) supported, this might include:

- o The operator's unique id (operator-id) or distinguished CDN-Domain (operator-domain);
- o NS records for the operator's set of externally visible request routers;

- o The set of requests the dCDN operator is prepared to serve (e.g. a set of client IP prefixes or geographic regions that may be served by dCDN).
- o Additional capabilities of the dCDN, such as its ability to support different CDNI Metadata requests.

Note that the set of requests that dCDN is willing to serve could in some cases be relatively static (e.g., a set of IP prefixes) which could be exchanged off-line, or might even be negotiated as part of a peering agreement. However, it may also be more dynamic, in which case the exchange supported by FCI would be helpful. A further discussion of the Footprint & Capability Advertisement interface can be found in [I-D.ietf-cdni-footprint-capabilities-semantics].

4.4. CDNI Logging Interface

It is necessary for the upstream CDN to have visibility into the delivery of content that it redirected to a downstream CDN. This allows the upstream CDN to properly bill its customers for multiple deliveries of content cached by the downstream CDN, as well as to report accurate traffic statistics to those content providers. This is one role of the LI.

Other operational data that may be relevant to CDNI can also be exchanged by the LI. For example, dCDN may report the amount of content it has acquired from uCDN, and how much cache storage has been consumed by content cached on behalf of uCDN.

Traffic logs are easily exchanged off-line. For example, the following traffic log is a small deviation from the Apache log file format, where entries include the following fields:

- o Domain - the full domain name of the origin server
- o IP address - the IP address of the client making the request
- o End time - the ending time of the transfer
- o Time zone - any time zone modifier for the end time
- o Method - the transfer command itself (e.g., GET, POST, HEAD)
- o URL - the requested URL
- o Version - the protocol version, such as HTTP/1.0
- o Response - a numeric response code indicating transfer result

- o Bytes Sent - the number of bytes in the body sent to the client
- o Request ID - a unique identifier for this transfer
- o User agent - the user agent, if supplied
- o Duration - the duration of the transfer in milliseconds
- o Cached Bytes - the number of body bytes served from the cache
- o Referer - the referrer string from the client, if supplied

Of these, only the Domain field is indirect in the downstream CDN--it is set to the CDN-Domain used by the upstream CDN rather than the actual origin server. This field could then be used to filter traffic log entries so only those entries matching the upstream CDN are reported to the corresponding operator. Further discussion of the LI can be found in [I-D.ietf-cdni-logging].

One open question is who does the filtering. One option is that the downstream CDN filters its own logs, and passes the relevant records directly to each upstream peer. This requires that the downstream CDN knows the set of CDN-Domains that belong to each upstream peer. If this information is already exchanged between peers as part of another interface, then direct peer-to-peer reporting is straightforward. If it is not available, and operators do not wish to advertise the set of CDN-Domains they serve to their peers, then the second option is for each CDN to send both its non-local traffic records and the set of CDN-Domains it serves to an independent third-party (i.e., a CDN Exchange), which subsequently filters, merges, and distributes traffic records on behalf of each participating CDN operator.

A second open question is how timely traffic information should be. For example, in addition to offline traffic logs, accurate real-time traffic monitoring might also be useful, but such information requires that the downstream CDN inform the upstream CDN each time it serves upstream content from its cache. The downstream CDN can do this, for example, by sending a conditional HTTP GET request (If-Modified-Since) to the upstream CDN each time it receives an HTTP GET request from one of its end-users. This allows the upstream CDN to record that a request has been issued for the purpose of real-time traffic monitoring. The upstream CDN can also use this information to validate the traffic logs received later from the downstream CDN.

There is obviously a tradeoff between accuracy of such monitoring and the overhead of the downstream CDN having to go back to the upstream CDN for every request.

Another design tradeoff in the LI is the degree of aggregation or summarization of data. One situation that lends itself to summarization is the delivery of HTTP adaptive streaming (HAS), since the large number of individual chunk requests potentially results in large volumes of logging information. This case is discussed below, but other forms of aggregation may also be useful. For example, there may be situations where bulk metrics such as bytes delivered per hour may suffice rather than the detailed per-request logs outlined above. It seems likely that a range of granularities of logging will be needed along with ways to specify the type and degree of aggregation required.

4.5. CDNI Control Interface

The CDNI Control interface is initially used to bootstrap the other interfaces. As a simple example, it could be used to provide the address of the logging server in dCDN to uCDN in order to bootstrap the CDNI Logging interface. It may also be used, for example, to establish security associations for the other interfaces.

The other role the CI plays is to allow the uCDN to pre-position, revalidate, or purge metadata and content on a dCDN. These operations, sometimes collectively called the Trigger interface, are discussed further in [I-D.ietf-cdni-control-triggers].

4.6. CDNI Metadata Interface

The role of the CDNI Metadata interface is to enable CDNI distribution metadata to be conveyed to the downstream CDN by the upstream CDN. Such metadata includes geo-blocking restrictions, availability windows, access control policies, and so on. It may also include information to facilitate acquisition of content by dCDN (e.g., alternate sources for the content, authorization information needed to acquire the content from the source). For a full discussion of the CDNI Metadata Interface, see [I-D.ietf-cdni-metadata]

Some distribution metadata may be partially emulated using in-band mechanisms. For example, in case of any geo-blocking restrictions or availability windows, the upstream CDN can elect to redirect a request to the downstream CDN only if that CDN's advertised delivery footprint is acceptable for the requested URL. Similarly, the request could be forwarded only if the current time is within the availability window. However, such approaches typically come with shortcomings such as inability to prevent from replay outside the time window or inability to make use of a downstream CDN that covers a broader footprint than the geo-blocking restrictions.

Similarly, some forms of access control may also be performed on a per-request basis using HTTP directives. For example, being able to respond to a conditional GET request gives the upstream CDN an opportunity to influence how the downstream CDN delivers its content. Minimally, the upstream CDN can invalidate (purge) content previously cached by the downstream CDN.

All of these in-band techniques serve to illustrate that uCDNs have the option of enforcing some of their access control policies themselves (at the expense of increased inter-CDN signaling load), rather than delegating enforcement to dCDNs using the MI. As a consequence, the MI could provide a means for the uCDN to express its desire to retain enforcement for itself. For example, this might be done by including a "check with me" flag in the metadata associated with certain content. The realization of such in-band techniques over the various inter-CDN acquisition protocols (e.g., HTTP) requires further investigation and may require small extensions or semantic changes to the acquisition protocol.

4.7. HTTP Adaptive Streaming Concerns

We consider HTTP Adaptive Streaming (HAS) and the impact it has on the CDNI interfaces because large objects (e.g., videos) are broken into a sequence of small, independent chunks. For each of the following, a more thorough discussion, including an overview of the tradeoffs involved in alternative designs, can be found in RFC 6983.

First, with respect to Content Acquisition and File Management, which are out-of-scope for the CDNI interfaces but nonetheless relevant to the overall operation, we assume no additional measures are required to deal with large numbers of chunks. This means that the dCDN is not explicitly made aware of any relationship between different chunks and the dCDN handles each chunk as if it were an individual and independent content item. The result is that content acquisition between uCDN and dCDN also happens on a per-chunk basis. This approach is in line with the recommendations made in RFC 6983, which also identifies potential improvements in this area that might be considered in the future.

Second, with respect to Request Routing, we note that HAS manifest files have the potential to interfere with request routing since manifest files contain URLs pointing to the location of content chunks. To make sure that a manifest file does not hinder CDNI request routing and does not place excessive load on CDNI resources, the use of manifest files could either be limited to those containing relative URLs or the uCDN could modify the URLs in the manifest. Our approach for dealing with these issues is twofold. As a mandatory requirement, CDNs should be able to handle unmodified manifest files

containing either relative or absolute URLs. To limit the number of redirects, and thus the load placed on the CDNI interfaces, as an optional feature uCDNs can use the information obtained through the CDNI Request Routing Redirection interface to modify the URLs in the manifest file. Since the modification of the manifest file is an optional uCDN-internal process, this does not require any standardization effort beyond being able to communicate chunk locations in the CDNI Request Routing Redirection interface.

Third, with respect to the CDNI Logging interface, there are several potential issues, including the large number of individual chunk requests potentially resulting in large volumes of logging information, and the desire to correlate logging information for chunk requests that correspond to the same HAS session. For the initial CDNI specification, our approach is to expect participating CDNs to support per-chunk logging (e.g. logging each chunk request as if it were an independent content request) over the CDNI Logging interface. Optionally, the LI may include a Content Collection Identifier (CCID) and/or a Session Identifier (SID) as part of the logging fields, thereby facilitating correlation of per-chunk logs into per-session logs for applications benefiting from such session level information (e.g. session-based analytics). This approach is in line with the recommendations made in RFC 6983, which also identifies potential improvements in this area that might be considered in the future.

Fourth, with respect to the CDNI Control interface, and in particular purging HAS chunks from a given CDN, our approach is to expect each CDN supports per-chunk content purge (e.g. purging of chunks as if they were individual content items). Optionally, a CDN may support content purge on the basis of a "Purge Identifier (Purge-ID)" allowing the removal of all chunks related to a given Content Collection with a single reference. It is possible that this Purge-ID could be merged with the CCID discussed above for HAS Logging, or alternatively, they may remain distinct.

4.8. URI Rewriting

When using HTTP redirection, content URIs may be rewritten when redirection takes place within an uCDN, from an uCDN to a dCDN, and within the dCDN. In the case of cascaded CDNs, content URIs may be rewritten at every CDN hop (e.g., between the uCDN and the dCDN acting as the transit CDN, and between the transit CDN and the dCDN serving the request. The content URI used between any uCDN/dCDN pair becomes a common handle that can be referred to without ambiguity by both CDNs in all their inter-CDN communications. This handle allows the uCDN and dCDN to correlate information exchanged using other CDNI

interfaces in both the downstream direction (e.g., when using the MI) and the upstream direction (e.g., when using the LI).

Consider the simple case of a single uCDN/dCDN pair using HTTP redirection. We introduce the following terminology for content URIs to simplify the discussion:

"u-URI" represents a content URI in a request presented to the uCDN;

"ud-URI" is a content URI acting as the common handle across uCDN and dCDN for requests redirected by the uCDN to a specific dCDN;

"d-URI" represents a content URI in a request made within the delegate dCDN.

In our simple pair-wise example, the "ud-URI" effectively becomes the handle that the uCDN/dCDN pair use to correlate all CDNI information. In particular, for a given pair of CDNs executing the HTTP redirection, the uCDN needs to map the u-URI to the ud-URI handle for all MI message exchanges, while the dCDN needs to map the d-URI to the ud-URI handle for all LI message exchanges.

In the case of cascaded CDNs, the transit CDN will rewrite the content URI when redirecting to the dCDN, thereby establishing a new handle between the transit CDN and the dCDN, that is different from the handle between the uCDN and transit CDN. It is the responsibility of the transit CDN to manage its mapping across handles so the right handle for all pairs of CDNs is always used in its CDNI communication.

In summary, all CDNI interfaces between a given pair of CDNs need to always use the "ud-URI" handle for that specific CDN pair as their content URI reference.

5. Deployment Models

In this section we describe a number of possible deployment models that may be achieved using the CDNI interfaces described above. We note that these models are by no means exhaustive, and that many other models may be possible.

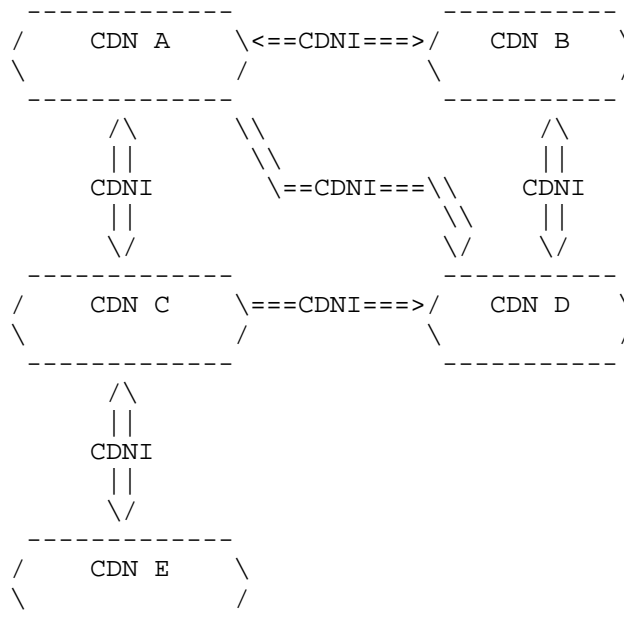
Although the reference model of Figure 1 shows all CDN functions on each side of the CDNI interface, deployments can rely on entities that are involved in any subset of these functions, and therefore only support the relevant subset of CDNI interfaces. As already noted in Section 3, effective CDNI deployments can be built without

necessarily implementing all the interfaces. Some examples of such deployments are shown below.

Note that, while we refer to upstream and downstream CDNs, this distinction applies to specific content items and transactions. That is, a given CDN may be upstream for some transactions and downstream for others, depending on many factors such as location of the requesting client and the particular piece of content requested.

5.1. Meshed CDNs

Although the reference model illustrated in Figure 1 shows a unidirectional CDN interconnection with a single uCDN and a single dCDN, any arbitrary CDNI meshing can be built from this, such as the example meshing illustrated in Figure 11. (Support for arbitrary meshing may or may not be in the initial scope for the working group, but the model allows for it.)



- ==> CDNI interfaces, with right-hand side CDN acting as dCDN to left-hand side CDN
- <==> CDNI interfaces, with right-hand side CDN acting as dCDN to left-hand side CDN and with left-hand side CDN acting as dCDN to right-hand side CDN

Figure 11: CDNI Deployment Model: CDN Meshing Example

5.2. CSP combined with CDN

Note that our terminology refers to functional roles and not economic or business roles. That is, a given organization may be operating as both a CSP and a fully fledged uCDN when we consider the functions performed, as illustrated in Figure 12.

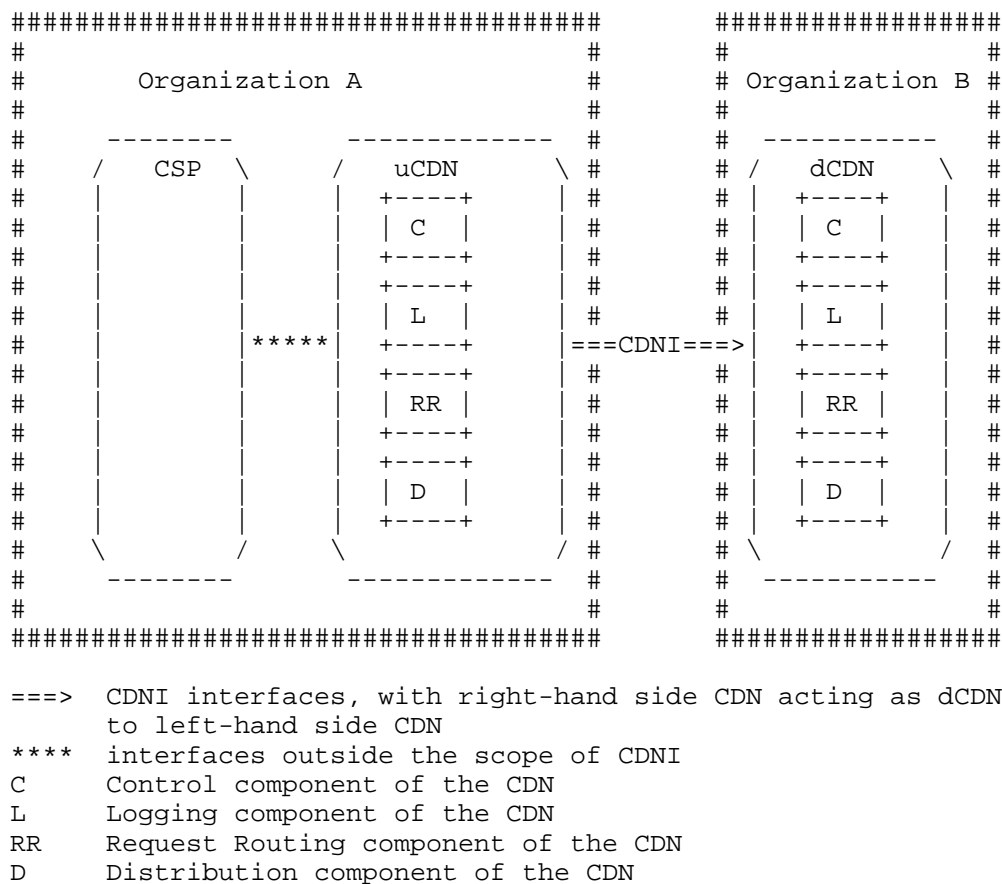


Figure 12: CDNI Deployment Model: Organization combining CSP & uCDN

5.3. CSP using CDNI Request Routing Interface

As another example, a content provider organization may choose to run its own request routing function as a way to select among multiple candidate CDN providers; In this case the content provider may be modeled as the combination of a CSP and of a special, restricted case of a CDN. In that case, as illustrated in Figure 13, the CDNI Request Routing interfaces can be used between the restricted CDN

operated by the content provider Organization and the CDN operated by the full CDN organization acting as a dCDN in the request routing control plane. Interfaces outside the scope of the CDNI work can be used between the CSP functional entities of the content provider organization and the CDN operated by the full CDN organization acting as a uCDN) in the CDNI control planes other than the request routing plane (i.e. Control, Distribution, Logging).

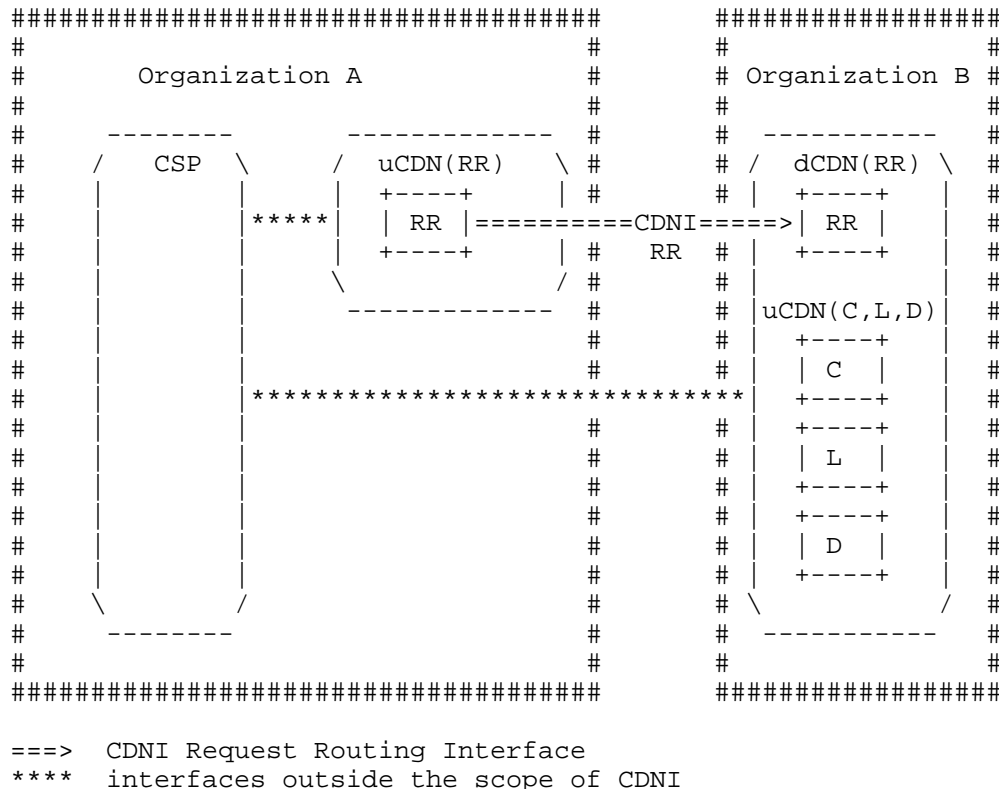


Figure 13: CDNI Deployment Model: Organization combining CSP and partial CDN

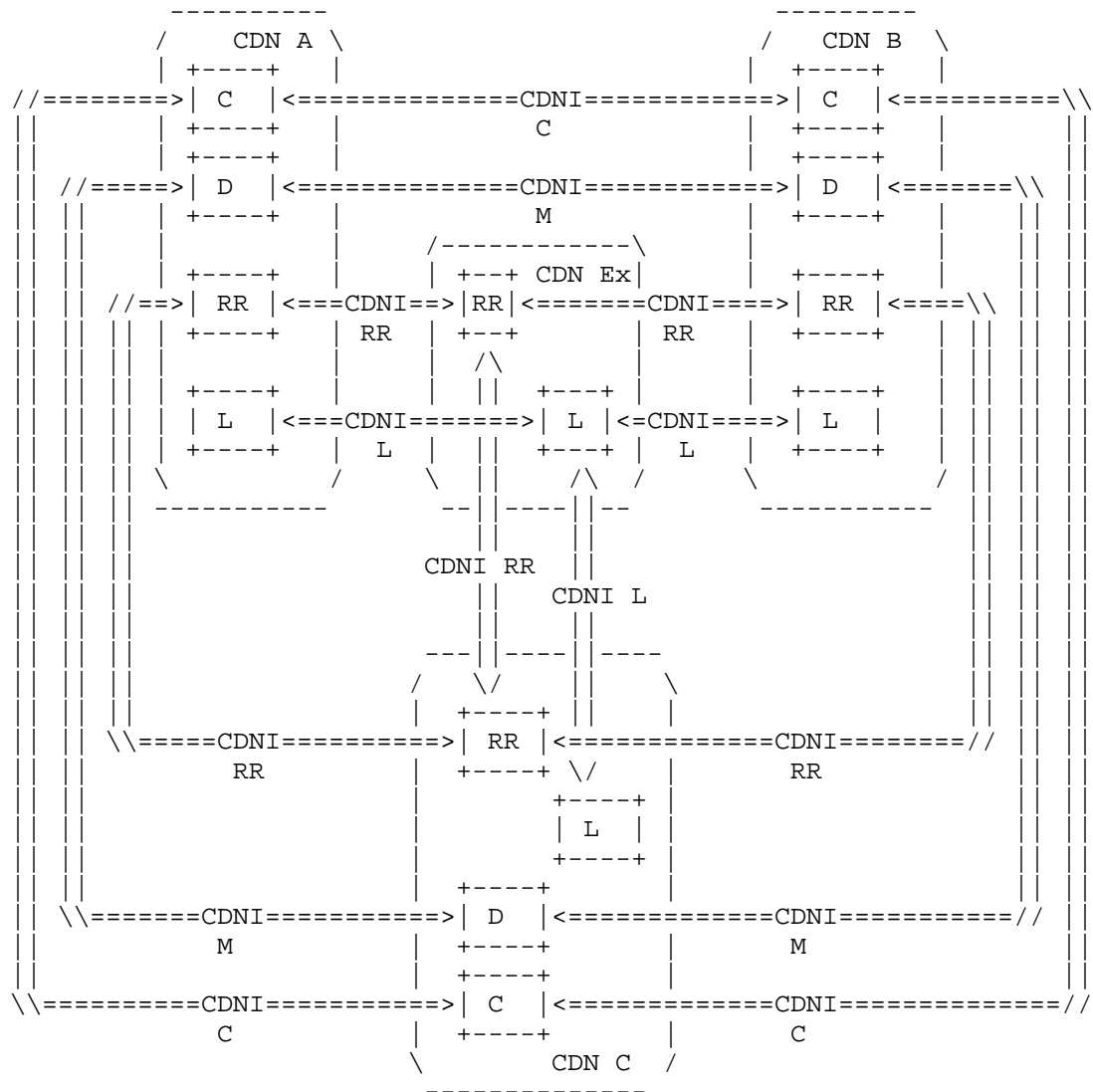
5.4. CDN Federations and CDN Exchanges

There are two additional concepts related to, but distinct from CDN Interconnection. The first is CDN Federation. Our view is that CDNI is the more general concept, involving two or more CDNs serving content to each other's users, while federation implies a multi-lateral interconnection arrangement, but other CDN interconnection agreements are also possible (e.g., symmetric bilateral, asymmetric bilateral). An important conclusion is that CDNI technology should

not presume (or bake in) a particular interconnection agreement, but should instead be general enough to permit alternative interconnection arrangements to evolve.

The second concept often used in the context of CDN Federation is CDN Exchange--a third party broker or exchange that is used to facilitate a CDN federation. Our view is that a CDN exchange offers valuable machinery to scale the number of CDN operators involved in a multi-lateral (federated) agreement, but that this machinery is built on top of the core CDNI interconnection mechanisms. For example, as illustrated in Figure 14, the exchange might aggregate and redistribute information about each CDN footprint and capacity, as well as collect, filter, and redistribute traffic logs that each participant needs for interconnection settlement, but inter-CDN request routing, inter-CDN content distribution (including inter-CDN acquisition) and inter-CDN control which fundamentally involve a direct interaction between an upstream CDN and a downstream CDN--operate exactly as in a pair-wise peering arrangement. Turning to Figure 14, we observe that in this example:

- o each CDN supports a direct CDNI Control interface to every other CDN
- o each CDN supports a direct CDNI Metadata interface to every other CDN
- o each CDN supports a CDNI Logging interface with the CDN Exchange
- o each CDN supports both a CDNI Request Routing interface with the CDN Exchange (for aggregation and redistribution of dynamic CDN footprint discovery information) and a direct RI to every other CDN (for actual request redirection).



<=CDNI RR=> CDNI Request Routing Interface
 <=CDNI M==> CDNI Metadata Interface
 <=CDNI C==> CDNI Control Interface
 <=CDNI L==> CDNI Logging Interface

Figure 14: CDNI Deployment Model: CDN Exchange

Note that a CDN exchange may alternatively support a different set of functionality (e.g. Logging only, or Logging and full request

routing, or all the functionality of a CDN including content distribution). All these options are expected to be allowed by the IETF CDNI specifications.

6. Trust Model

There are a number of trust issues that need to be addressed by a CDNI solution. Many of them are in fact similar or identical to those in a simple CDN without interconnection. In a standard CDN environment (without CDNI), the CSP places a degree of trust in a single CDN operator to perform many functions. The CDN is trusted to deliver content with appropriate quality of experience for the end user. The CSP trusts the CDN operator not to corrupt or modify the content. The CSP often relies on the CDN operator to provide reliable accounting information regarding the volume of delivered content. The CSP may also trust the CDN operator to perform actions such as timely invalidation of content and restriction of access to content based on certain criteria such as location of the user and time of day, and to enforce per-request authorization performed by the CSP using techniques such as URI signing.

A CSP also places trust in the CDN not to distribute any information that is confidential to the CSP (e.g., how popular a given piece of content is) or confidential to the end user (e.g., which content has been watched by which user).

A CSP does not necessarily have to place complete trust in a CDN. A CSP will in some cases take steps to protect its content from improper distribution by a CDN, e.g. by encrypting it and distributing keys in some out of band way. A CSP also depends on monitoring (possibly by third parties) and reporting to verify that the CDN has performed adequately. A CSP may use techniques such as client-based metering to verify that accounting information provided by the CDN is reliable. HTTP conditional requests may be used to provide the CSP with some checks on CDN operation. In other words, while a CSP may trust a CDN to perform some functions in the short term, the CSP is able in most cases to verify whether these actions have been performed correctly and to take action (such as moving the content to a different CDN) if the CDN does not live up to expectations.

One of the trust issues raised by CDNI is transitive trust. A CDN that has a direct relationship with a CSP can now "outsource" the delivery of content to another (downstream) CDN. That CDN may in turn outsource delivery to yet another downstream CDN, and so on.

The top level CDN in such a chain of delegation is responsible for ensuring that the requirements of the CSP are met. Failure to do so

is presumably just as serious as in the traditional single CDN case. Hence, an upstream CDN is essentially trusting a downstream CDN to perform functions on its behalf in just the same way as a CSP trusts a single CDN. Monitoring and reporting can similarly be used to verify that the downstream CDN has performed appropriately. However, the introduction of multiple CDNs in the path between CSP and end user complicates the picture. For example, third party monitoring of CDN performance (or other aspects of operation, such as timely invalidation) might be able to identify the fact that a problem occurred somewhere in the chain but not point to the particular CDN at fault.

In summary, we assume that an upstream CDN will invest a certain amount of trust in a downstream CDN, but that it will verify that the downstream CDN is performing correctly, and take corrective action (including potentially breaking off its relationship with that CDN) if behavior is not correct. We do not expect that the trust relationship between a CSP and its "top level" CDN will differ significantly from that found today in single CDN situations. However, it does appear that more sophisticated tools and techniques for monitoring CDN performance and behavior will be required to enable the identification of the CDN at fault in a particular delivery chain.

We expect that the detailed designs for the specific interfaces for CDNI will need to take the transitive trust issues into account. For example, explicit confirmation that some action (such as content removal) has taken place in a downstream CDN may help to mitigate some issues of transitive trust.

7. IANA Considerations

This memo includes no request to IANA.

8. Privacy Considerations

In general, a CDN has the opportunity to collect detailed information about the behavior of end-users e.g. by logging which files are being downloaded. While the concept of interconnected CDNs as described in this document doesn't necessarily allow any given CDN to gather more information on any specific user, it potentially facilitates sharing of this data by a CDN with more parties. As an example, the purpose of the CDNI Logging Interface is to allow a dCDN to share some of its log records with a uCDN, both for billing purposes as well as for sharing traffic statistics with the Content Provider on which behalf the content was delivered. The fact that the CDNI Interfaces provide mechanisms for sharing such potentially sensitive user data, shows that it is necessary to include in these interface appropriate

privacy and confidentiality mechanisms. The definition of such mechanisms is dealt with in the respective CDN interface documents.

9. Security Considerations

While there are a variety of security issues introduced by a single CDN, we are concerned here specifically with the additional issues that arise when CDNs are interconnected. For example, when a single CDN has the ability to distribute content on behalf of a CSP, there may be concerns that such content could be distributed to parties who are not authorized to receive it, and there are mechanisms to deal with such concerns. Our focus in this section is on how CDN interconnection introduces new security issues not found in the single CDN case. For a more detailed analysis of the security requirements of CDNI, see section 9 of [I-D.ietf-cdni-requirements].

Many of the security issues that arise in CDNI are related to the transitivity of trust (or lack thereof) described in Section 6. As noted above, the design of the various interfaces for CDNI must take account of the additional risks posed by the fact that a CDN with whom a CSP has no direct relationship is now potentially distributing content for that CSP. The mechanisms used to mitigate these risks may be similar to those used in the single CDN case, but their suitability in this more complex environment must be validated.

CDNs today offer a variety of means to control access to content, such as time-of-day restrictions, geo-blocking, and URI signing. These mechanisms must continue to function in CDNI environments, and this consideration is likely to affect the design of certain CDNI interfaces (e.g. metadata, request routing). For more information on URI signing in CDNI, see [I-D.leung-cdni-uri-signing].

Just as with a single CDN, each peer CDN must ensure that it is not used as an "open proxy" to deliver content on behalf of a malicious CSP. Whereas a single CDN typically addresses this problem by having CSPs explicitly register content (or origin servers) that are to be served, simply propagating this information to peer downstream CDNs may be problematic because it reveals more information than the upstream CDN is willing to specify. (To this end, the content acquisition step in the earlier examples force the dCDN to retrieve content from the uCDN rather than go directly to the origin server.)

There are several approaches to this problem. One is for the uCDN to encode a signed token generated from a shared secret in each URL routed to a dCDN, and for the dCDN to validate the request based on this token. Another one is to have each upstream CDN advertise the set of CDN-Domains they serve, where the downstream CDN checks each request against this set before caching and delivering the associated

object. Although straightforward, this approach requires operators to reveal additional information, which may or may not be an issue.

9.1. Security of CDNI Interfaces

It is noted in [I-D.ietf-cdni-requirements] that all CDNI interfaces must be able to operate securely over insecure IP networks. Since it is expected that the CDNI interfaces will be implemented using existing application protocols such as HTTP or XMPP, we also expect that the security mechanisms available to those protocols may be used by the CDNI interfaces. Details of how these interfaces are secured will be specified in the relevant interface documents.

9.2. Digital Rights Management

Issues of digital rights management (DRM, also sometimes called digital restrictions management) is often employed for content distributed via CDNs. In general, DRM relies on the CDN to distribute encrypted content, with decryption keys distributed to users by some other means (e.g. directly from the CSP to the end user.) For this reason, DRM is considered out of scope [RFC6707] and does not introduce additional security issues for CDNI.

10. Contributors

The following individuals contributed to this document:

- o Matt Caulfield
- o Francois le Faucheur
- o Aaron Falk
- o David Ferguson
- o John Hartman
- o Ben Niven-Jenkins
- o Kent Leung

11. Acknowledgements

The authors would like to thank Huw Jones and Jinmei Tatuya for their helpful input to this document. In addition, the authors would like to thank Stephen Farrell, Ted Lemon and Alissa Cooper for their reviews, which have helped to improve this document.

12. Informative References

- [I-D.ietf-cdni-control-triggers]
Murray, R. and B. Niven-Jenkins, "CDNI Control Interface / Triggers", draft-ietf-cdni-control-triggers-02 (work in progress), December 2013.
- [I-D.ietf-cdni-footprint-capabilities-semantics]
Seedorf, J., Peterson, J., Previdi, S., Brandenburg, R., and K. Ma, "CDNI Request Routing: Footprint and Capabilities Semantics", draft-ietf-cdni-footprint-capabilities-semantics-02 (work in progress), February 2014.
- [I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-11 (work in progress), March 2014.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata", draft-ietf-cdni-metadata-06 (work in progress), February 2014.
- [I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection Interface for CDN Interconnection", draft-ietf-cdni-redirection-02 (work in progress), April 2014.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-17 (work in progress), January 2014.
- [I-D.leung-cdni-uri-signing]
Leung, K., Faucheur, F., Downey, B., Brandenburg, R., and S. Leibrand, "URI Signing for CDN Interconnection (CDNI)", draft-leung-cdni-uri-signing-05 (work in progress), March 2014.
- [RFC3466] Day, M., Cain, B., Tomlinson, G., and P. Rzewski, "A Model for Content Internetworking (CDI)", RFC 3466, February 2003.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

- [RFC6770] Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, November 2012.
- [RFC6983] van Brandenburg, R., van Deventer, O., Le Faucheur, F., and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware Content Distribution Network Interconnection (CDNI)", RFC 6983, July 2013.

Authors' Addresses

Larry Peterson
Akamai Technologies, Inc.
8 Cambridge Center
Cambridge, MA 02142
USA

Email: lapeters@akamai.com

Bruce Davie
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
USA

Email: bdavie@vmware.com

Ray van Brandenburg (editor)
TNO
Brassersplein 2
Delft 2612CT
the Netherlands

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

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: August 26, 2013

G. Bertrand, Ed.
I. Oprescu, Ed.
E. Stephan
France Telecom - Orange
R. Peterkofsky
Skytide, Inc.
F. Le Faucheur, Ed.
Cisco Systems
P. Grochocki
Orange Polska
February 22, 2013

CDNI Logging Interface
draft-ietf-cdni-logging-01

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on August 26, 2013.

Copyright Notice

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

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

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

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

Table of Contents

1. Introduction	5
1.1. Terminology	5
1.2. Abbreviations	8
2. CDNI Logging Reference Model	8
2.1. CDNI Logging interactions	8
2.2. Overall Logging Chain	12
2.2.1. Logging Generation and During-Generation Aggregation	13
2.2.2. Logging Collection	14
2.2.3. Logging Filtering	14
2.2.4. Logging Rectification and Post-Generation Aggregation	15
2.2.5. Log-Consuming Applications	15
2.2.5.1. Maintenance/Debugging	15
2.2.5.2. Accounting	16
2.2.5.3. Analytics and Reporting	16
2.2.5.4. Security	16
2.2.5.5. Legal Logging Duties	16
2.2.5.6. Notions common to multiple Log Consuming Applications	16
3. CDNI Logging Transport Requirements	18
3.1. Timeliness	19
3.2. Reliability	19
3.3. Security	19
3.4. Scalability	19
3.5. Consistency between CDNI Logging and CDN Logging	20
3.6. Dispatching/Filtering	20
4. CDNI Logging Information Structure and Transport	20
5. CDNI Logging Fields	22
5.1. Semantics of CDNI Logging Fields	22
5.2. Syntax of CDNI Logging Fields	26
6. CDNI Logging Records	27
6.1. Content Delivery	27
6.2. Content Invalidation and Purging	29
6.3. Request Routing	29
6.4. Logging Extensibility	29
7. CDNI Logging File Format	29
7.1. Logging Files	29
7.2. File Format	29
7.2.1. Headers	30
7.2.2. Body (Logging Records) Format	31
7.2.3. Footer Format	31
8. CDNI Logging File Transport Protocol	31
9. Open Issues	32
10. IANA Considerations	32
11. Security Considerations	32

11.1. Privacy	33
11.2. Non Repudiation	33
12. Acknowledgments	33
13. References	33
13.1. Normative References	33
13.2. Informative References	33
Appendix A. Examples Log Format	34
A.1. W3C Common Log File (CLF) Format	35
A.2. W3C Extended Log File (ELF) Format	35
A.3. National Center for Supercomputing Applications (NCSA) Common Log Format	37
A.4. NCSA Combined Log Format	37
A.5. NCSA Separate Log Format	37
A.6. Squid 2.0 Native Log Format for Access Logs	37
Appendix B. Requirements	38
B.1. Additional Requirements	38
B.2. Compliancy with Requirements draft	39
Appendix C. Analysis of candidate protocols for Logging Transport	39
C.1. Syslog	40
C.2. XMPP	40
C.3. SNMP	40
Authors' Addresses	40

1. Introduction

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

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

- o CDNI problem statement [RFC6707] and framework [I-D.ietf-cdni-framework] identify a Logging interface,
- o Section 7 of [I-D.ietf-cdni-requirements] specifies a set of requirements for Logging,
- o [RFC6770] outlines real world use-cases for interconnecting CDNs. These use cases require the exchange of Logging information between the dCDN and the uCDN.

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

The present document describes:

- o The CDNI Logging reference model (Section 2),
- o The CDNI Logging information structure and Transport (Section 4),
- o The CDNI Logging Fields (Section 5),
- o The CDNI Logging Records (Section 6),
- o The CDNI Logging File format (Section 7),
- o The CDNI Logging File Transport Protocol (Section 8),

In the Appendices, the document provides:

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

1.1. Terminology

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

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

CDN Logging information: logging information generated and collected within a CDN

CDNI Logging information: logging information exchanged across CDNs using the CDNI Logging Interface

Logging information: logging information generated and collected within a CDN or obtained from another CDN using the CDNI Logging Interface

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

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

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

CDNI Logging File: a file containing CDNI Logging Records, as well as additional information facilitating the processing of the CDNI Logging Records.

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

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

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

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

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

Examples:

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

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

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

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

1.2. Abbreviations

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

2. CDNI Logging Reference Model

2.1. CDNI Logging interactions

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

- o customization by the uCDN of the CDNI logging information to be provided by the dCDN to the uCDN (e.g. control of which logging fields are to be communicated to the uCDN for a given task performed by the dCDN, control of which types of events are to be logged). The dCDN takes into account this CDNI logging customization information to determine what logging information to

provide to the uCDN, but it may, or may not, take into account this CDNI logging customization information to influence what CDNI logging information is to be generated and collected within the dCDN (e.g. even if the uCDN requests a restricted subset of the logging information, the dCDN may elect to generate a broader set of logging information). The mechanism to support the customisation by the uCDN of CDNI Logging information is outside the scope of this document and left for further study. We note that the CDNI Control interface or the CDNI Metadata interfaces appear as candidate interfaces on which to potentially build such a customisation mechanism. Before such a mechanism is available, the uCDN and dCDN are expected to agree off-line on what CDNI logging information is to be provided by dCDN to uCDN and rely on management plane actions to configure the CDNI Logging functions to generate (respectively, expect) in dCDN (respectively, in uCDN).

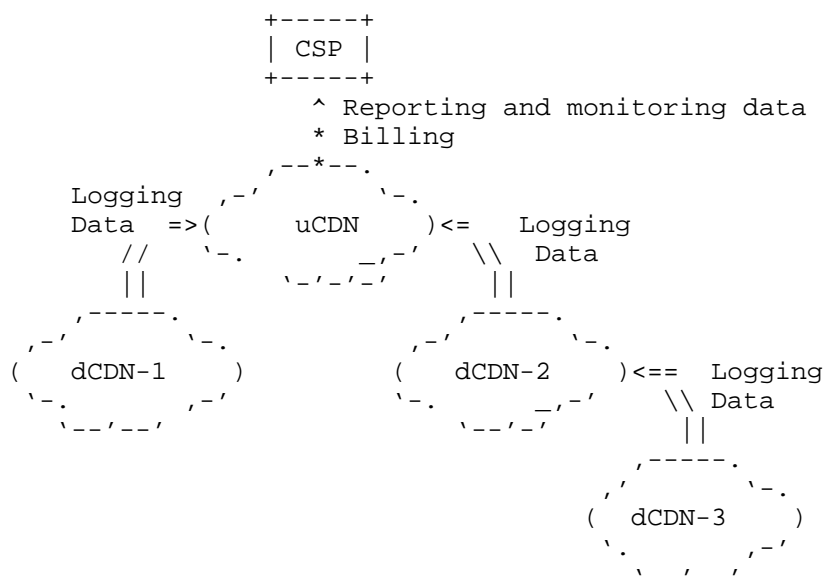
- o generation and collection by the dCDN of logging information related to the completion of any task performed by the dCDN on behalf of the uCDN (e.g., delivery of the content to an end user) or related to events happening in the dCDN that are relevant to the uCDN (e.g., failures or unavailability in dCDN). This takes place within the dCDN and does not directly involve CDNI interfaces.
- o communication by the dCDN to the uCDN of the logging information collected by the dCDN relevant to the uCDN. This is supported by the CDNI Logging interface and in the scope of the present document. For example, the uCDN may use this logging information to charge the CSP, to perform analytics and monitoring for operational reasons, to provide analytics and monitoring views on its content delivery to the CSP or to perform trouble-shooting.
- o customization by the dCDN of the logging to be performed by the uCDN on behalf of the dCDN. The mechanism to support the customisation by the dCDN of CDNI Logging information is outside the scope of this document and left for further study.
- o generation and collection by the uCDN of logging information related to the completion of any task performed by the uCDN on behalf of the dCDN (e.g., serving of content by uCDN to dCDN for acquisition purposes by dCDN) or related to events happening in the uCDN that are relevant to the dCDN. This takes place within the uCDN and does not directly involve CDNI interfaces.
- o communication by the uCDN to the dCDN of the logging information collected by the uCDN relevant to the dCDN. For example, the dCDN might potentially benefit from this information for security

auditing or content acquisition troubleshooting. This is outside the scope of this document and left for further study.

Figure 1 provides an example of CDNI Logging interactions (focusing only on the interactions that are in the scope of this document) in a particular scenario where 4 CDNs are involved in the delivery of content from a given CSP: the uCDN has a CDNI interconnection with dCDN-1 and dCDN-2. In turn, dCDN2 has a CDNI interconnection with dCDN3. In this example, uCDN, dCDN-1, dCDN-2 and dCDN-3 all participate in the delivery of content for the CSP. In this example, the CDNI Logging interface enables the uCDN to obtain logging information from all the dCDNs involved in the delivery. In the example, uCDN uses the Logging data:

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

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



==> CDNI Logging Interface
 ***> outside the scope of CDNI

Figure 1: Interactions in CDNI Logging Reference Model

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

Note that the format of Logging information that a CDN provides over the CDNI interface might be different from the one that the CDN uses internally. In this case, the CDN needs to reformat the Logging information before it provides this information to the other CDN over the CDNI Logging interface. Similarly, a CDN might reformat the Logging data that it receives over the CDNI Logging interface before injecting it into its log-consuming applications or before providing some of this logging information to the CSP. Such reformatting operations introduce latency in the logging distribution chain and introduce a processing burden. Therefore, there are benefits in specifying CDNI Logging format that are suitable for use inside CDNs and also are close to the CDN Log formats commonly used in CDNs today.

2.2. Overall Logging Chain

This section discusses the overall logging chain within and across CDNs to clarify how CDN Logging information is expected to fit in this overall chain. Figure 2 illustrates the overall logging chain within the dCDN, across CDNs using the CDNI Logging interface and within the uCDN. Note that the logging chain illustrated in the Figure is obviously only indicative and varies depending on the specific environments. For example, there may be more or less instantiations of each entity (i.e., there may be 4 Log consuming applications in a given CDN). As another example, there may be one instance of Rectification process per Log Consuming Application instead of a shared one.

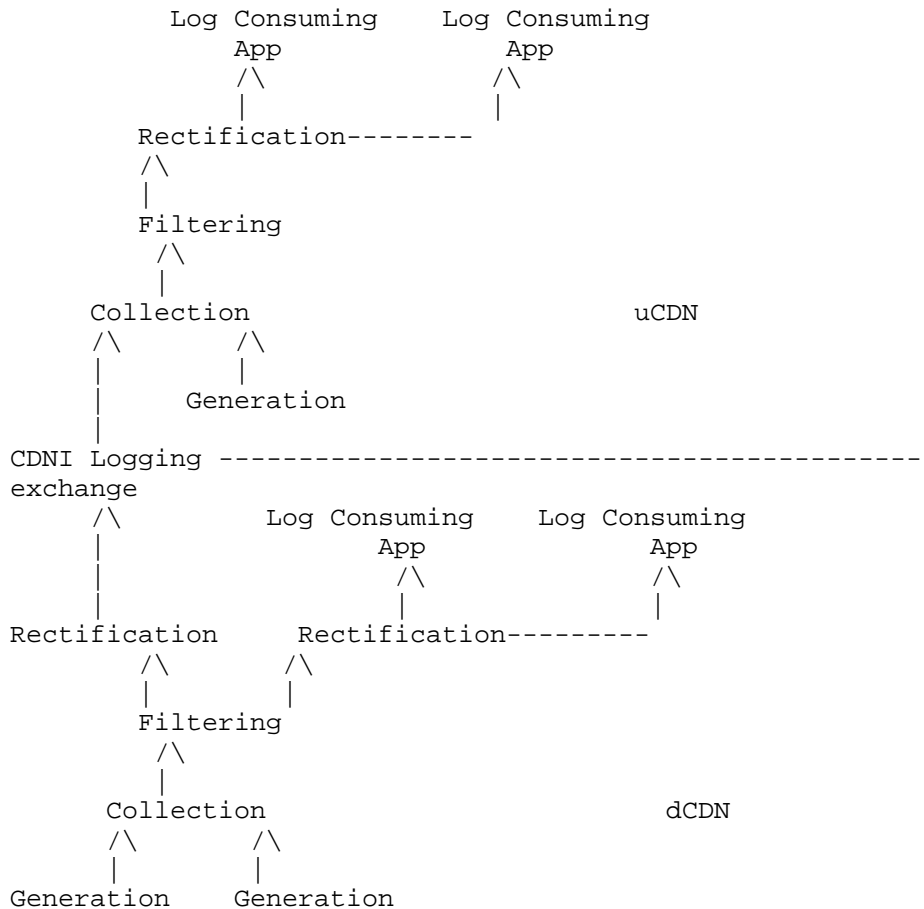


Figure 2: CDNI Logging in the overall Logging Chain

The following subsections describe each of the processes potentially involved in the logging chain of Figure 2.

2.2.1. Logging Generation and During-Generation Aggregation

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

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

[I-D.brandenburg-cdni-has] discusses logging for HTTP Adaptive Streaming (HAS). In accordance with the recommendations articulated there, it is expected that a surrogate will generate separate logging information for delivery of each chunk of HAS content. This ensures that separate logging information can then be provided to interconnected CDNs over the CDNI Logging interface. Still in line with the recommendations of [I-D.brandenburg-cdni-has], the logging information for per-chunck delivery may include some information (a Content Collection IDentifier and a Session IDentifier as discussed in Section 5) intended to facilitate subsequent post-generation aggregation of per-chunck logs into per-session logs. Note that a CDN may also elect to generate aggregate per-session logs when performing HAS delivery, but this needs to be in addition to, and not instead of, the per-chunck delivery logs. We note that this may be revisited in future versions of this document.

Note that in the case of non real-time logging, the trigger of the transmission or generation of the logging file appears to be a synchronous process from a protocol standpoint. The implementation algorithm can choose to enforce a maximum size for the logging file beyound which the transmission is automatically triggered (and thus allow for an asynchrinous transmission process).

2.2.2. Logging Collection

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

In a CDNI environment, in addition to collecting logging information from log-generating entities within the local CDN, the Collection process also collects logging information provided by another CDN, or other CDNs, through the CDNI Logging interface. This is illustrated in Figure 2 where we see that the Collection process of the uCDN collects logging information from log-generating entities within the uCDN as well as logging information coming through CDNI Logging exchange with the dCDN through the CDNI Logging interface.

2.2.3. Logging Filtering

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

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

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

In some use cases described in [RFC6770], the interconnected CDNs do not want to disclose details on their internal topology. The filtering process can then also filter confidential data on the dCDNs' topology (number of servers, location, etc.). In particular, information about the requests served by every Surrogate may be confidential. Therefore, the Logging information must be protected so that data such as Surrogates' hostnames is not disclosed to the uCDN. In the "Inter-Affiliates Interconnection" use case, this information may be disclosed to the uCDN because both the dCDN and the uCDN are operated by entities of the same group.

2.2.4. Logging Rectification and Post-Generation Aggregation

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

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

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

2.2.5. Log-Consuming Applications

2.2.5.1. Maintenance/Debugging

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

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

Logging enables the CDN providers to identify and troubleshoot performance degradations. In particular, Logging enables the

communication of traffic data (e.g., the amount of traffic that has been forwarded by a dCDN on behalf of an uCDN over a given period of time), which is particularly useful for CDN and network planning operations.

2.2.5.2. Accounting

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

2.2.5.3. Analytics and Reporting

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

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

2.2.5.4. Security

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

2.2.5.5. Legal Logging Duties

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

2.2.5.6. Notions common to multiple Log Consuming Applications

2.2.5.6.1. Logging Information Views

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

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

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

2.2.5.6.2. Key Performance Indicators (KPIs)

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

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

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

Here is a list of important examples of KPIs:

- o Number of delivery requests received from End-Users in a given region for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Percentage of delivery successes/failures among the aforementioned requests
- o Number of failures listed by failure type (e.g., HTTP error code) for requests received from End Users in a given region and for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Number and cause of premature delivery termination for End Users in a given region and for each piece of content, during a given period of time (e.g., hour/day/week/month)

- o Maximum and mean number of simultaneous sessions established by End Users in a given region, for a given Delivery Service, and during a given period of time (e.g., hour/day/week/month)
- o Volume of traffic delivered for sessions established by End Users in a given region, for a given Delivery Service, and during a given period of time (e.g., hour/day/week/month)
- o Maximum, mean, and minimum delivery throughput for sessions established by End Users in a given region, for a given Delivery Service, and during a given period of time (e.g., hour/day/week/month)
- o Cache-hit and byte-hit ratios for requests received from End Users in a given region for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Top 10 of the most popularly requested content (during a given day/week/month),
- o Terminal type (mobile, PC, STB, if this information can be acquired from the browser type header, for example).

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

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

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

3. CDNI Logging Transport Requirements

3.1. Timeliness

Some applications consuming CDNI Logging information, such as accounting or trend analytics, only require logging information to be available with a timeliness of the order of a day or the hour. This document focuses on addressing this requirement.

Some applications consuming CDNI Logging information, such as real-time analytics, require logging information to be available in real-time (i.e. of the order of a second after the corresponding event). This document leaves this requirement out of scope.

3.2. Reliability

CDNI logging information must be transmitted reliably. The transport protocol should contain an anti-replay mechanism.

3.3. Security

CDNI logging information exchange must allow authentication, integrity protection, and confidentiality protection. Also, a non-repudiation mechanism is mandatory, the transport protocol should support it.

3.4. Scalability

CDNI logging information exchange must support large scale information exchange, particularly so in the presence of HTTP Adaptive Streaming.

For example, if we consider a client pulling HTTP Progressive Download content with an average duration of 10 minutes, this represents 1/600 CDNI delivery Logging Records per second. If we assume the dCDN is simultaneously serving 100,000 such clients on behalf of the uCDN, the dCDN will be generating 167 Logging Records per second to be communicated to the uCDN over the CDNI Logging interface. Or equivalently, if we assume an average delivery rate of 2Mb/s, the dCDN generates 0.83 CDNI Logging Records per second for every Gb/s of streaming on behalf of the uCDN.

For example, if we consider a client pulling HAS content and receiving a video chunk every 2 seconds, a separate audio chunk every 2 seconds and a refreshed manifest every 10 seconds, this represents 1.1 delivery Logging Record per second. If we assume the dCDN is simultaneously serving 100,000 such clients on behalf of the uCDN, the dCDN will be generating 110,000 Logging Records per second to be communicated to the uCDN over the CDNI Logging interface. Or equivalently, if we assume an average delivery rate of 2Mb/s, the

dCDN generates 550 CDNI Logging Records per second for every Gb/s of streaming on behalf of the uCDN.

3.5. Consistency between CDNI Logging and CDN Logging

There are benefits in using a CDNI logging format as close as possible to intra-CDN logging format commonly used in CDNs today in order to minimize systematic translation at CDN/CDNI boundary.

3.6. Dispatching/Filtering

When a CDN is acting as a dCDN for multiple uCDNs, the dCDN needs to dispatch each CDNI Logging Record to the uCDN that redirected the corresponding request. The CDNI Logging format need to allow, and possibly facilitate, such a dispatching.

4. CDNI Logging Information Structure and Transport

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

This document defines non-real-time transport of CDNI Logging information over the CDNI interface. For such non-real-time transport, this documents defines a third level of structure, the CDNI Logging File, that is a collection of CDNI Logging Records. This structure is described in Figure 3. This document then specifies how to transport such CDNI Logging Files across interconnected CDNs. We observe that this approach can be tuned in a real deployment to achieve near-real time exchange of CDNI Logging information, e.g., by increasing the frequency of logging file creation and distribution throughout the Logging chain, but it is not expected that this approach can support real time transport (e.g., sub-second) of CDNI logging information.

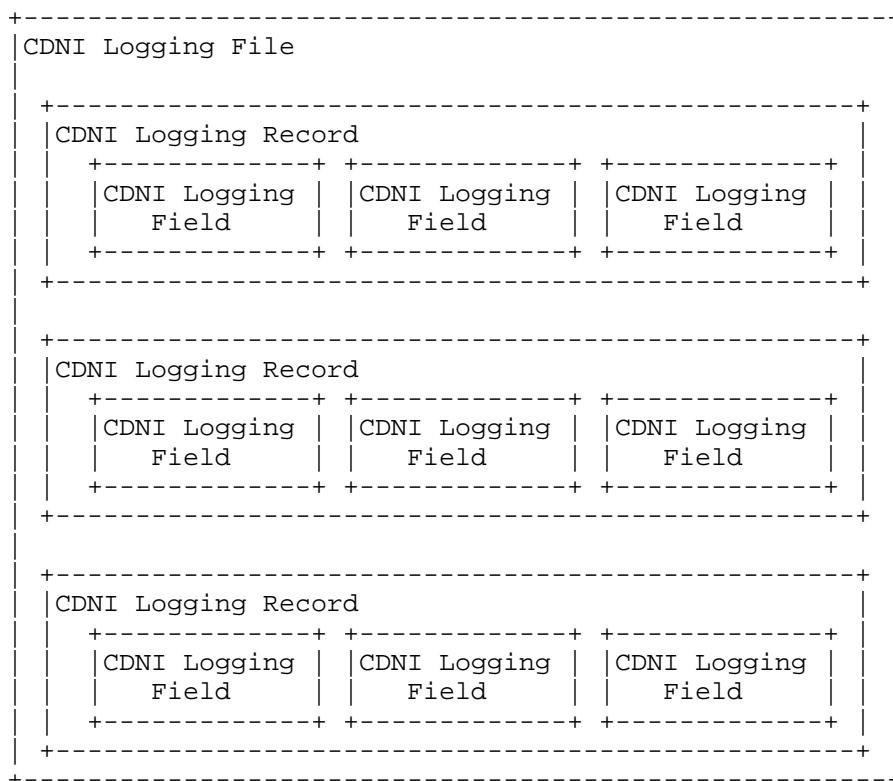


Figure 3: Structure of Logging Files

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

The semantics and encoding of the CDNI Logging fields are specified in Section 5. The semantics and encoding of CDNI Records are specified in Section 6. The CDNI Logging File format is specified in Section 7. The protocol for transport of CDNI Logging File is specified in Section 8.

5. CDNI Logging Fields

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

5.1. Semantics of CDNI Logging Fields

This section specifies the semantics of the CDNI Logging Fields. The specific subset of CDNI Logging fields that can be found in each type of Logging Record is specified in Section 6.

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

Name	Description
Start-time	A start date and time associated with a logged event; for instance, the time at which a Surrogate received a content delivery request or the time at which an origin server received a content acquisition request.
End-time	An end date and time associated with a logged event. For instance, the time at which a Surrogate completed the handling of a content delivery request (e.g., end of delivery or error).
Duration	The duration of an operation in milliseconds. For instance, this field could be used to provide the time it took the Surrogate to send the requested file to the End-User or the time it took the Surrogate to acquire the file on a cache-miss event. In the case where Start-time, End-time, and Duration appear in a Logging Record, the Duration is to be interpreted as a total activity time related to the logged operation.
Client-IP	The IP address of the User Agent that issued the logged request or of a proxy, for instance "203.0.113.1".
Client-port	The source port of the logged request (e.g., 9542)
Destination-IP	The IP address of the host that received the logged request (e.g., 192.0.2.2).
Destination-hostname	The hostname of the host that received the logged request (e.g., Surrogatel.cdna.com).
Destination-port	The destination port of the logged request (e.g., 80).
Operation	The kind of operation that is logged; for instance Delivery or Purging.
URI_full	The full requested URL (e.g., "http://node1.peer-a.op-b.net/cdn.csp.com/movies/potter.avi?param=11&user=toto"). When HTTP request redirection is used, this URI includes the Surrogate FQDN. If the association of requests to Surrogates is confidential, the dCDN can present only URI_part to uCDN.

URI_part	The requested URL path (e.g., /cdn.csp.com/movies/potter.avi?param=11&user=toto if the full request URL was "http://nodel.peer-a.op-b.net/cdn.csp.com/movies/potter.avi?param=11&user=toto"). The URI without host-name typically includes the "CDN domain" (ex.cdn.csp.com) - cf. [I-D.ietf-cdni-framework]: it enables the identification of the CSP service agreed between the CSP and the CDNP operating the uCDN.
Protocol	The protocol and protocol version of the message that triggered the Logging entry (e.g., HTTP/1.1).
Request-method	The protocol method of the request message that triggered the Logging entry.
Status	The protocol status of the reply message related to the Logging entry
Bytes-Sent	The number of bytes at application-layer protocol-level (e.g., HTTP) of the reply message related to the Logging entry. It includes the size of the response headers.
Headers-Sent	The number of bytes corresponding to response headers at application-layer protocol-level (e.g., HTTP) of the reply message related to the Logging entry.
Bytes-received	The number of bytes (headers + body) of the message that triggered the Logging entry.
Referrer	The value of the Referrer header in an HTTP request.
User-Agent	The value of the User Agent header in an HTTP request.
Cookie	The value of the Cookie header in an HTTP request.
Byte-Range	[Ed. note: to be defined]
Cache-control	The value of the cache-control header in an HTTP answer. This header is particularly important for content acquisition logs.
Record-digest	A digest of the Logging Record; it enables detecting corrupted Logging Records.
CCID	A Content Collection Identifier (CCID) eases the correlation of several Logging Records related to a Content Collection (e.g., a movie split in chunks).
SID	A Session Identifier (SID) eases the correlation (and aggregation) of several Logging Records related to a session. The SID is especially relevant for summarizing HAS Logging information [I-D.brandenburg-cdni-has].

uCDN-ID	An element authenticating the operator of the uCDN as the authority having delegated the request to the dCDN.
Delivering-CDN-ID	An identifier (e.g., an aggregation of an IP address and a FQDN) of the Delivering CDN. The Delivering-CDN-ID might be considered as confidential by the dCDN. In such case, the dCDN could either not provide this field to the uCDN or overwrite the Delivering-CDN-ID with its own identifier.
Cache-bytes	The number of body bytes served from caches. This quantity permits the computation of the byte hit ratio.
Action	The Action describes how a given request was treated locally: through which transport protocol, with or without content revalidation, with a cache hit or cache miss, with fresh or stale content, and (if relevant) with which error. Example with Squid format [squid]: "TCP_REFRESH_FAIL_HIT" means that an expired copy of an object requested through TCP was in the cache. Squid attempted to make an If-Modified-Since request, but it failed. The old (stale) object was delivered to the client.
MIME-Type	The MIME-Type of the requested content
dCDN identifier	An element authenticating the operator of the dCDN as the authority requesting the content to the uCDN
Caching_date	Date at which the delivered content was stored in cache
Validity_headers	A copy of all headers related to content validity: Pragma or Cache-Control (no-cache), ETag, Vary, last-modified...
Lookup_duration	Duration of the DNS resolution for resolving the FQDN of (uCDN's or CSP's) origin server.
Delay_to_first_bit	Duration of the operations from the sending of the content acquisition request to the reception of the first bit of the requested content.
Delay_to_last_bit	Duration of the operations from the sending of the content acquisition request to the reception of the last bit of the requested content.

Table 1: Semantics of CDNI Logging Fields

NB: we define three fields related to the timing of logged operations: Start-time, End-time, and Duration. Start-time is typically useful for human readers (e.g., while debugging), however,

some servers log the operation's End-time which corresponds to the time of log record generation. In absence of Logging summarization, only two of these three fields are required to obtain relevant timing information on the operation. However, when some kind of Logging aggregation/summarization is used, it can be advantageous to keep the three fields: for instance, in the case of HAS, keeping the three fields permits computing an average delivery bitrate from a single Logging Record aggregating information on the delivery of multiple consecutive video chunks.

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

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

The URI_full provides information on the Surrogate that provided the content. This information can be relevant, for instance, for the Inter-Affiliates use case described in [RFC6770]. However, in some cases it may be considered as confidential and the dCDN may provide URI_part instead.

Other information that could be logged include operations that refer to the general state of the request, before it gets processed locally. Such information is related to the authorization of the requests, URL rewriting rules enforced, the X-FORWARDED-FOR non standard HTTP header...

[Editor's Note: CDNI Logging information may be used for debugging. Therefore, various CDN operations might be logged, depending on the agreement between the dCDN and the uCDN, such as operations related to Request Routing and Metadata. These may call for a few additional Fields to be defined].

5.2. Syntax of CDNI Logging Fields

This section is intended to contain the specification for the syntax and encoding of the CDNI Logging fields. For now, Table 2 illustrates the definition of some information elements. It provides examples using Apache log format strings [apache] when they exist.

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

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

Table 2: Examples using Apache format

6. CDNI Logging Records

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

This section defines the events for which a CDNI Logging record can be exchanged over the CDNI Logging interface and for each type of Logging Record indicates the allowed set of CDNI Information Elements.

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

6.1. Content Delivery

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

- o Reception by a dCDN Surrogate of a content request

The Logging Record for Content Delivery contains the following set of

CDNI Logging Elements:

Name	Mandatory/Optional
Start-time	Mandatory
Duration	Mandatory
Client-IP	Mandatory
Client-port	Optional
Destination-IP	Mandatory if Destination-Hostname is absent
Destination-Hostname	Mandatory if Destination-IP is absent
Destination-port	Optional
Operation	Optional
URI_full	Mandatory if URI_part is absent
URI_part	Mandatory if URI_full is absent
Protocol	Mandatory if protocol is different to HTTP/1.1
Request-method	Mandatory
Status	Mandatory
Bytes-Sent	Mandatory
Headers-Sent	Optional
Bytes-received	Optional
Referrer	Optional
User-Agent	Optional
Cookie	Optional
Byte-Range	?
Cache-control	Optional
Record-digest	?
CCID	Optional. Only applicable to HTTP Adaptive Streaming delivery.
SID	Optional. Only applicable to HTTP Adaptive Streaming delivery.
Cache-bytes	Optional
Action	Mandatory (in particular re cache Hit/Miss)
MIME-Type	Mandatory

Table 3: CDNI Logging Fields in Delivery Logging Record

In Table 3, "Mandatory" means that this field MUST be included in each Delivery Record and "Optional" means that it can be included based on the agreement between the dCDN and the uCDN as established via mechanism outside the scope of this document (e.g., by human agreement).

6.2. Content Invalidation and Purging

Given that the Purge interface is expected to contain a mechanism to report on completion of the Invalidation/purge request, there is no need to specify separate Log Records for these events.

6.3. Request Routing

[Editor's Note: Is there a requirement for the dCDN to provide logs for request routing events?]

6.4. Logging Extensibility

Future usages might introduce the need for additional Logging fields. In addition, some use-cases such as an Inter-Affiliate Interconnection [RFC6770], might take advantage of extended Logging exchanges. Therefore, it is important to permit CDNs to use additional Logging fields besides the standard ones, if they want. For instance, an "Account-name" identifying the contract enforced by the dCDN for a given request could be provided in extended fields.

The required Logging Records may depend on the considered services. For instance, static file delivery (e.g., pictures) typically does not include any delivery restrictions. By contrast, video delivery typically implies strong content delivery restrictions, as explained in [RFC6770], and Logging could include information about the enforcement of these restrictions. Therefore, to ease the support of varied services as well as of future services, the Logging interface should support optional Logging Records.

7. CDNI Logging File Format

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

7.1. Logging Files

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

7.2. File Format

The Logging file format should be independent from the selected transport protocol, to guarantee a flexible choice of transport

protocols. [Ed. note: for the real time Logging exchanges, this might be hard]

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

7.2.1. Headers

Logging files must include a header with the information described in Figure 4.

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

Figure 4: Logging Headers

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

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

7.2.2. Body (Logging Records) Format

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

Since records for real time information and non-real time information could use different formats, we do not yet solve the problem of real time logging exchanges in this version.

7.2.3. Footer Format

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

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

Figure 5: Logging footers

This digest field permits the detection of corrupted Logging files. This can be useful, for instance, if a problem occurs on the filesystem of the dCDN Logging system and leads to a truncation of a logging file. Additional mechanisms to avoid corrupted Logging files are expected to be provided by the Logging transport protocol, cf. Section 8.

8. CDNI Logging File Transport Protocol

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

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

For managed file transfer, the recommended protocol is SSH File Transfer Protocol (SFTP) [I-D.ietf-secsh-filexfer]. SFTP is widely deployed and it guarantees the respect of the criteria expressed by the CDNI Logging Transport Requirements: timeliness, reliability, security and scalability.

[Ed note: include options for lossless compression]

9. Open Issues

The main remaining tasks on this ID are the following:

- o Finalise the list of CDNI Logging Fields
- o Finalise the encoding of CDNI Logging Fields, Records and File.
- o Identify what can be done (if anything) to maximise reuse of Logging Fields and Logging Records encoding for future support of real-time CDNI Logging exchange

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

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

10. IANA Considerations

TBD

11. Security Considerations

11.1. Privacy

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

11.2. Non Repudiation

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

12. Acknowledgments

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

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

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

13. References

13.1. Normative References

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

13.2. Informative References

- [CLF] A. Luotonen, "The Common Log-file Format, W3C (work in progress)", 1995, <<http://www.w3.org/pub/WWW/Daemon/User/Config/Logging.html>>.
- [ELF] Phillip M. Hallam-Baker and Brian Behlendorf, "Extended Log File Format, W3C (work in progress), WD-logfile-960323", <<http://www.w3.org/TR/WD-logfile.html>>.

- [I-D.brandenburg-cdni-has]
Brandenburg, R., Deventer, O., Faucheur, F., and K. Leung,
"Models for adaptive-streaming-aware CDN Interconnection",
draft-brandenburg-cdni-has-04 (work in progress),
January 2013.
- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN
Interconnection", draft-ietf-cdni-framework-03 (work in
progress), February 2013.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network
Interconnection (CDNI) Requirements",
draft-ietf-cdni-requirements-04 (work in progress),
December 2012.
- [I-D.ietf-secsh-filexfer]
Galbraith, J. and O. Saarenmaa, "SSH File Transfer
Protocol", draft-ietf-secsh-filexfer-13 (work in
progress), July 2006.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content
Distribution Network Interconnection (CDNI) Problem
Statement", RFC 6707, September 2012.
- [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.
- [apache] "Apache 2.2 log files documentation", Feb. 2012,
<<http://httpd.apache.org/docs/current/logs.html>>.
- [squid] "Squid Log-Format documentation", Feb. 2012,
<<http://wiki.squid-cache.org/SquidFaq/SquidLogs>>.

Appendix A. Examples Log Format

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

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

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

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

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

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

The fields are defined as follows [CLF]:

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

Table 4: Information elements in CLF format

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

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

The supported fields are listed below [ELF].

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

Table 5: Information elements in ELF format

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

- o Prefix Description
- o c- Client
- o s- Server
- o r- Remote
- o cs- Client to Server.
- o sc- Server to Client.
- o sr- Server to Remote Server (used by proxies)
- o rs- Remote Server to Server (used by proxies)

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

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

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

This format for Access Logs offers the following fields:

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

A.4. NCSA Combined Log Format

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

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

A.5. NCSA Separate Log Format

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

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

The format of the Agent log is as follows:

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

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

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

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

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

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

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

Table 6: Information elements in Squid format

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

Appendix B. Requirements

B.1. Additional Requirements

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

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

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

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

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

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

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

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

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

B.2. Compliancy with Requirements draft

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

[Ed. note: to be written later]

Appendix C. Analysis of candidate protocols for Logging Transport

This section will be expanded later with an analysis of alternative candidate protocols for transport of CDNI Logging in non-real-time as well as real-time.

C.1. Syslog

[Ed. node: to be written later]

C.2. XMPP

[Ed. node: to be written later]

C.3. SNMP

Authors' Addresses

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

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

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

Phone: +33 6 89 06 92 72
Email: iuniana.oprescu@orange.com

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

Email: emile.stephan@orange.com

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

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

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

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

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

Email: pawel.grochocki@orange.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: December 10, 2016

F. Le Faucheur, Ed.

G. Bertrand, Ed.

I. Oprescu, Ed.

R. Peterkofsky
Google Inc.
June 8, 2016

CDNI Logging Interface
draft-ietf-cdni-logging-27

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on December 10, 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
1.2. Requirements Language	5
2. CDNI Logging Reference Model	5
2.1. CDNI Logging interactions	5
2.2. Overall Logging Chain	8
2.2.1. Logging Generation and During-Generation Aggregation	9
2.2.2. Logging Collection	10
2.2.3. Logging Filtering	10
2.2.4. Logging Rectification and Post-Generation Aggregation	11
2.2.5. Log-Consuming Applications	12
2.2.5.1. Maintenance/Debugging	12
2.2.5.2. Accounting	13
2.2.5.3. Analytics and Reporting	13
2.2.5.4. Content Protection	13
2.2.5.5. Notions common to multiple Log Consuming	
Applications	14
3. CDNI Logging File	16
3.1. Rules	16
3.2. CDNI Logging File Structure	17
3.3. CDNI Logging Directives	20
3.4. CDNI Logging Records	24
3.4.1. HTTP Request Logging Record	25
3.5. CDNI Logging File Extension	36
3.6. CDNI Logging File Examples	36
3.7. Cascaded CDNI Logging Files Example	39
4. Protocol for Exchange of CDNI Logging File After Full	
Collection	42
4.1. CDNI Logging Feed	43
4.1.1. Atom Formatting	43
4.1.2. Updates to Log Files and the Feed	43
4.1.3. Redundant Feeds	44
4.1.4. Example CDNI Logging Feed	44
4.2. CDNI Logging File Pull	46
5. Protocol for Exchange of CDNI Logging File During Collection	47
6. IANA Considerations	48
6.1. CDNI Logging Directive Names Registry	48
6.2. CDNI Logging File version Registry	48
6.3. CDNI Logging record-types Registry	49

6.4. CDNI Logging Field Names Registry	50
6.5. CDNI Logging MIME Media Type	51
7. Security Considerations	52
7.1. Authentication, Authorization, Confidentiality, Integrity Protection	52
7.2. Denial of Service	53
7.3. Privacy	53
8. Acknowledgments	55
9. References	55
9.1. Normative References	55
9.2. Informative References	57
Authors' Addresses	59

1. Introduction

This memo specifies the CDNI Logging interface between a downstream CDN (dCDN) and an upstream CDN (uCDN). First, it describes a reference model for CDNI logging. Then, it specifies the CDNI Logging File format and the actual protocol for exchange of CDNI Logging Files.

The reader should be familiar with the following documents:

- o CDNI problem statement [RFC6707] and framework [RFC7336] identify a Logging interface,
- o Section 8 of [RFC7337] specifies a set of requirements for Logging,
- o [RFC6770] outlines real world use-cases for interconnecting CDNs. These use cases require the exchange of Logging information between the dCDN and the uCDN.

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

The present document describes:

- o The CDNI Logging reference model (Section 2),
- o The CDNI Logging File format (Section 3),
- o The CDNI Logging File Exchange protocol (Section 4).

1.1. Terminology

In this document, the first letter of each CDNI-specific term is capitalized. We adopt the terminology described in [RFC6707] and [RFC7336], and extend it with the additional terms defined below.

Intra-CDN Logging information: logging information generated and collected within a CDN. The format of the Intra-CDN Logging information may be different to the format of the CDNI Logging information.

CDNI Logging information: logging information exchanged across CDNs using the CDNI Logging Interface.

Logging information: logging information generated and collected within a CDN or obtained from another CDN using the CDNI Logging Interface.

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

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

CDNI Logging File: a file containing CDNI Logging Records, as well as additional information facilitating the processing of the CDNI Logging Records.

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

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

1.2. Requirements Language

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 RFC 2119 [RFC2119].

2. CDNI Logging Reference Model

2.1. CDNI Logging interactions

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

- o customization by the uCDN of the CDNI Logging information to be provided by the dCDN to the uCDN (e.g., control of which CDNI Logging fields are to be communicated to the uCDN for a given task performed by the dCDN or control of which types of events are to be logged). The dCDN takes into account this CDNI Logging customization information to determine what Logging information to provide to the uCDN, but it may, or may not, take into account this CDNI Logging customization information to influence what CDN logging information is to be generated and collected within the dCDN (e.g., even if the uCDN requests a restricted subset of the logging information, the dCDN may elect to generate a broader set of logging information). The mechanism to support the customization by the uCDN of CDNI Logging information is outside the scope of this document and left for further study. Until such a mechanism is available, the uCDN and dCDN are expected to agree off-line on what exact set of CDNI Logging information is to be provided by the dCDN to the uCDN, and to rely on management plane actions to configure the CDNI Logging functions in the dCDN to generate this information set and in the uCDN to expect this information set.
- o generation and collection by the dCDN of the intra-CDN Logging information related to the completion of any task performed by the dCDN on behalf of the uCDN (e.g., delivery of the content to an End User) or related to events happening in the dCDN that are relevant to the uCDN (e.g., failures or unavailability in dCDN). This takes place within the dCDN and does not directly involve CDNI interfaces.
- o communication by the dCDN to the uCDN of the Logging information collected by the dCDN relevant to the uCDN. This is supported by the CDNI Logging interface and in the scope of the present document. For example, the uCDN may use this Logging information to charge the CSP, to perform analytics and monitoring for

operational reasons, to provide analytics and monitoring views on its content delivery to the CSP or to perform trouble-shooting. This document exclusively specifies non-real-time exchange of Logging information. Closer to real-time exchange of Logging information (say sub-minute or sub-second) is outside the scope of the present document and left for further study. This document exclusively specifies exchange of Logging information related to content delivery. Exchange of Logging information related to operational events (e.g., dCDN request routing function unavailable, content acquisition failure by dCDN) for audit or operational reactive adjustments by uCDN is outside the scope of the present document and left for further study.

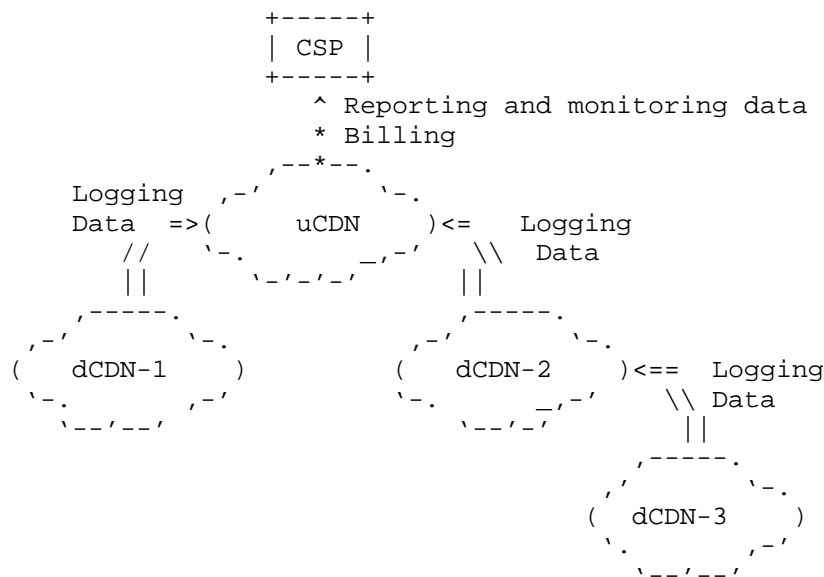
- o customization by the dCDN of the CDNI Logging information to be provided by the uCDN on behalf of the dCDN. The mechanism to support the customization by the dCDN of CDNI Logging information is outside the scope of this document and left for further study.
- o generation and collection by the uCDN of Intra-CDN Logging information related to the completion of any task performed by the uCDN on behalf of the dCDN (e.g., serving of content by uCDN to dCDN for acquisition purposes by dCDN) or related to events happening in the uCDN that are relevant to the dCDN. This takes place within the uCDN and does not directly involve CDNI interfaces.
- o communication by the uCDN to the dCDN of the Logging information collected by the uCDN relevant to the dCDN. For example, the dCDN might potentially benefit from this information for security auditing or content acquisition troubleshooting. This is outside the scope of this document and left for further study.

Figure 1 provides an example of CDNI Logging interactions (focusing only on the interactions that are in the scope of this document) in a particular scenario where four CDNs are involved in the delivery of content from a given CSP: the uCDN has a CDNI interconnection with dCDN-1 and dCDN-2. In turn, dCDN-2 has a CDNI interconnection with dCDN-3, where dCDN-2 is acting as an upstream CDN relative to dCDN-3. In this example, uCDN, dCDN-1, dCDN-2 and dCDN-3 all participate in the delivery of content for the CSP. In this example, the CDNI Logging interface enables the uCDN to obtain Logging information from all the dCDNs involved in the delivery. In the example, the uCDN uses the Logging information:

- o to analyze the performance of the delivery performed by the dCDNs and to adjust its operations after the fact (e.g., request routing) as appropriate,

- o to provide (non-real-time) reporting and monitoring information to the CSP.

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



==> CDNI Logging Interface
 ***> outside the scope of CDNI

Figure 1: Interactions in CDNI Logging Reference Model

A downstream CDN relative to uCDN (e.g., dCDN-2) integrates the relevant Logging information obtained from its own downstream CDNs (i.e., dCDN-3) in the Logging information that it provides to the uCDN, so that the uCDN ultimately obtains all Logging information relevant to a CSP for which it acts as the authoritative CDN. Such aggregation is further discussed in Section 3.7.

Note that the format of Logging information that a CDN provides over the CDNI interface might be different from the one that the CDN uses internally. In this case, the CDN needs to reformat the Logging information before it provides this information to the other CDN over

the CDNI Logging interface. Similarly, a CDN might reformat the Logging information that it receives over the CDNI Logging interface before injecting it into its log-consuming applications or before providing some of this Logging information to the CSP. Such reformatting operations introduce latency in the logging distribution chain and introduce a processing burden. Therefore, there are benefits in specifying CDNI Logging formats that are suitable for use inside CDNs and also are close to the intra-CDN Logging formats commonly used in CDNs today.

2.2. Overall Logging Chain

This section discusses the overall logging chain within and across CDNs to clarify how CDN Logging information is expected to fit in this overall chain. Figure 2 illustrates the overall logging chain within the dCDN, across CDNs using the CDNI Logging interface and within the uCDN. Note that the logging chain illustrated in the Figure is obviously only an example and varies depending on the specific environments. For example, there may be more or fewer instantiations of each entity (e.g., there may be 4 Log consuming applications in a given CDN). As another example, there may be one instance of Rectification process per Log Consuming Application instead of a shared one.

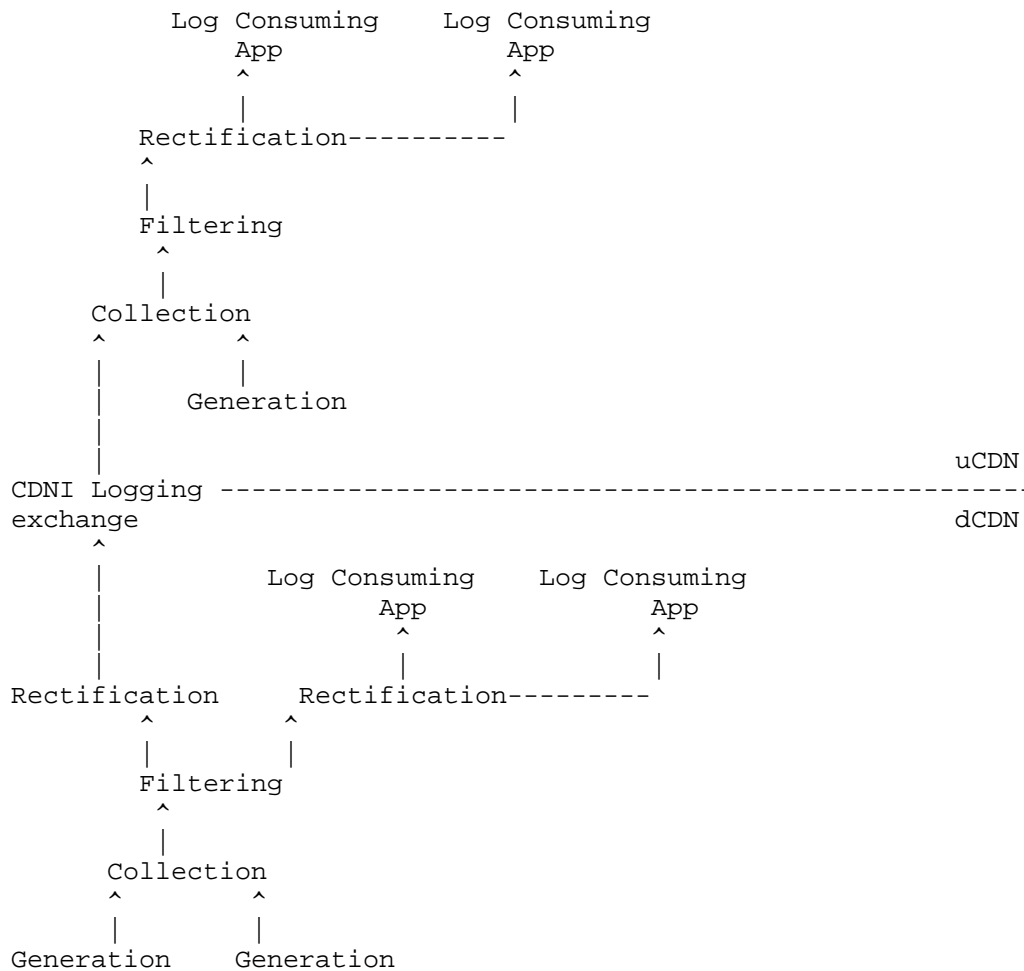


Figure 2: CDNI Logging in the overall Logging Chain

The following subsections describe each of the processes potentially involved in the logging chain of Figure 2.

2.2.1. Logging Generation and During-Generation Aggregation

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

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

[RFC6983] discusses logging for HTTP Adaptive Streaming (HAS). In accordance with the recommendations articulated there, it is expected that a surrogate will generate separate Logging information for delivery of each chunk of HAS content. This ensures that separate Logging information can then be provided to interconnected CDNs over the CDNI Logging interface. Still in line with the recommendations of [RFC6983], the Logging information for per-chunk delivery may include some information (a Content Collection Identifier and a Session Identifier) intended to facilitate subsequent post-generation aggregation of per-chunk logs into per-session logs. Note that a CDN may also elect to generate aggregate per-session logs when performing HAS delivery, but this needs to be in addition to, and not instead of, the per-chunk delivery logs. We note that aggregate per-session logs for HAS delivery are for further study and outside the scope of this document.

2.2.2. Logging Collection

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

In a CDNI environment, in addition to collecting Logging information from log-generating entities within the local CDN, the Collection process also collects Logging information provided by another CDN, or other CDNs, through the CDNI Logging interface. This is illustrated in Figure 2 where we see that the Collection process of the uCDN collects Logging information from log-generating entities within the uCDN as well as Logging information coming from the dCDNs through the CDNI Logging interface.

2.2.3. Logging Filtering

A CDN may be required to only present different subsets of the whole Logging information collected to various log-consuming applications. This is achieved by the Filtering process.

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

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

In some use cases described in [RFC6770], the interconnected CDNs do not want to disclose details on their internal topology. The filtering process can then also filter confidential data on the dCDNs' topology (number of servers, location, etc.). In particular, information about the requests served by each Surrogate may be confidential. Therefore, the Logging information needs to be protected so that data such as Surrogates' hostnames are not disclosed to the uCDN. In the "Inter-Affiliates Interconnection" use case, this information may be disclosed to the uCDN because both the dCDN and the uCDN are operated by entities of the same group.

2.2.4. Logging Rectification and Post-Generation Aggregation

If Logging information is generated periodically, it is important that the sessions that start in one Logging period and end in another are correctly reported. If they are reported in the starting period, then the Logging information of this period will be available only after the end of the session, which delays the Logging information generation. A simple approach is to provide the complete Logging Record for a session in the Logging Period of the session end.

A Logging rectification/update mechanism could be useful to reach a good trade-off between the Logging information generation delay and the Logging information accuracy.

In the presence of HAS, some log-consuming applications can benefit from aggregate per-session logs. For example, for analytics, per-session logs allow display of session-related trends which are much more meaningful for some types of analysis than chunk-related trends. In the case where aggregate logs have been generated directly by the

log-generating entities, those can be used by the applications. In the case where aggregate logs have not been generated, the Rectification process can be extended with a Post-Generation Aggregation process that generates per-session logs from the per-chunk logs, possibly leveraging the information included in the per-chunk logs for that purpose (Content Collection IDentifier and a Session IDentifier). However, in accordance with [RFC6983], this document does not define exchange of such aggregate logs on the CDNI Logging interface. We note that this is for further study and outside the scope of this document.

2.2.5. Log-Consuming Applications

2.2.5.1. Maintenance/Debugging

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

To detect faults, Logging information needs to report success and failure of CDN delivery operations. The uCDN can summarize such information into KPIs. For instance, Logging information needs to allow the computation of the number of times, during a given time period, that content delivery related to a specific service succeeds/fails.

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

Some of these maintenance and debugging applications only require aggregate logging information highly compatible with use of anonymization of IP addresses (as supported by the present document and specified in the definition of the c-groupid field under Section 3.4.1). However, in some situations, it may be useful, where compatible with privacy protection, to access some CDNI Logging Records containing full non-anonymized IP addresses. This is allowed in the definition of the c-groupid (under Section 3.4.1), with very significant privacy protection limitations that are discussed in the definition of the c-groupid field. For example, this may be useful for detailed fault tracking of a particular end user content delivery issue. Where there is a hard requirement by uCDN or CSP to associate a given enduser to individual CDNI Logging Records (e.g., to allow a-posteriori analysis of individual delivery for example in situations of performance-based penalties), instead of using

aggregates containing a single client as discussed in the c-groupid field definition, an alternate approach is to ensure that a client identifier is embedded in the request fields that can be logged in a CDNI Logging Record (for example by including the client identifier in the URI query string or in a HTTP Header). That latter approach offers two strong benefits: first, the aggregate inside the c-groupid can contain more than one client, thereby ensuring stronger privacy protection; second, it allows a reliable identification of the client while IP address does not in many situations (e.g., behind NAT, where dynamic IP addresses are used and reused,...). However, care SHOULD be taken that the client identifiers exposed in other fields of the CDNI Records cannot themselves be linked back to actual users.

2.2.5.2. Accounting

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

2.2.5.3. Analytics and Reporting

The goals of analytics include gathering any relevant information in order to be able to develop statistics on content download, analyze user behavior, and monitor the performance and quality of content delivery. For instance, Logging information enables the CDN providers to report on content consumption (e.g., delivered sessions per content) in a specific geographic area.

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

2.2.5.4. Content Protection

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

2.2.5.5. Notions common to multiple Log Consuming Applications

2.2.5.5.1. Logging Information Views

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

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

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

2.2.5.5.2. Key Performance Indicators (KPIs)

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

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

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

Here is a list of important examples of KPIs:

- o Number of delivery requests received from End Users in a given region for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Percentage of delivery successes/failures among the aforementioned requests
- o Number of failures listed by failure type (e.g., HTTP error code) for requests received from End Users in a given region and for each piece of content, during a given period of time (e.g., hour/day/week/month)

- o Number and cause of premature delivery termination for End Users in a given region and for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Maximum and mean number of simultaneous sessions established by End Users in a given region, for a given Content Provider, and during a given period of time (e.g., hour/day/week/month)
- o Volume of traffic delivered for sessions established by End Users in a given region, for a given Content Provider, and during a given period of time (e.g., hour/day/week/month)
- o Maximum, mean, and minimum delivery throughput for sessions established by End Users in a given region, for a given Content Provider, and during a given period of time (e.g., hour/day/week/month)
- o Cache-hit and byte-hit ratios for requests received from End Users in a given region for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Top 10 most popularly requested contents (during a given day/week/month)
- o Terminal type (mobile, PC, STB, if this information can be acquired from the browser type inferred from the User Agent string, for example).

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

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

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

3. CDNI Logging File

3.1. Rules

This specification uses the Augmented Backus-Naur Form (ABNF) notation and core rules of [RFC5234]. In particular, the present document uses the following rules from [RFC5234]:

CR = %x0D ; carriage return

ALPHA = %x41-5A / %x61-7A ; A-Z / a-z

DIGIT = %x30-39 ; 0-9

DQUOTE = %x22 ; " (Double Quote)

CRLF = CR LF ; Internet standard newline

HEXDIG = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

HTAB = %x09 ; horizontal tab

LF = %x0A ; linefeed

VCHAR = %x21-7E ; visible (printing) characters

OCTET = %x00-FF ; 8 bits of data

The present document also uses the following rules from [RFC3986]:

host = as specified in section 3.2.2 of [RFC3986].

IPv4address = as specified in section 3.2.2 of [RFC3986].

IPv6address = as specified in section 3.2.2 of [RFC3986].

partial-time = as specified in [RFC3339].

The present document also defines the following additional rules:

ADDRESS = IPv4address / IPv6address

ALPHANUM = ALPHA / DIGIT

DATE = 4DIGIT "-" 2DIGIT "-" 2DIGIT

; Dates are encoded as "full-date" specified in [RFC3339].

DEC = 1*DIGIT ["." 1*DIGIT]

NAMEFORMAT = ALPHANUM *(ALPHANUM / "_" / "-")

QSTRING = DQUOTE *(NDQUOTE / PCT-ENCODED) DQUOTE

NDQUOTE = %x20-21 / %x23-24 / %x26-7E / UTF8-2 / UTF8-3 / UTF8-4

; whereby a DQUOTE is conveyed inside a QSTRING unambiguously
by escaping it with PCT-ENCODED.

PCT-ENCODED = "%" HEXDIG HEXDIG

; percent encoding is used for escaping octets that might be
possible in HTTP headers such as bare CR, bare LF, CR LF, HTAB,
SP or null. These octets are rendered with percent encoding in
ABNF as specified by [RFC3986] in order to avoid considering
them as separators for the logging records.

NHTABSTRING = 1*(SP / VCHAR)

TIME = partial-time

USER-COMMENT = * (SP / VCHAR / UTF8-2 / UTF8-3 / UTF8-4)

3.2. CDNI Logging File Structure

As defined in Section 1.1: a CDNI Logging Field is as an atomic logging information element, a CDNI Logging Record is a collection of CDNI Logging fields containing all logging information corresponding to a single logging event, and a CDNI Logging File contains a collection of CDNI Logging Records. This structure is illustrated in Figure 3. The use of a file structure for transfer of CDNI Logging information is selected since this is the most common practise today for exchange of logging information within and across CDNs.

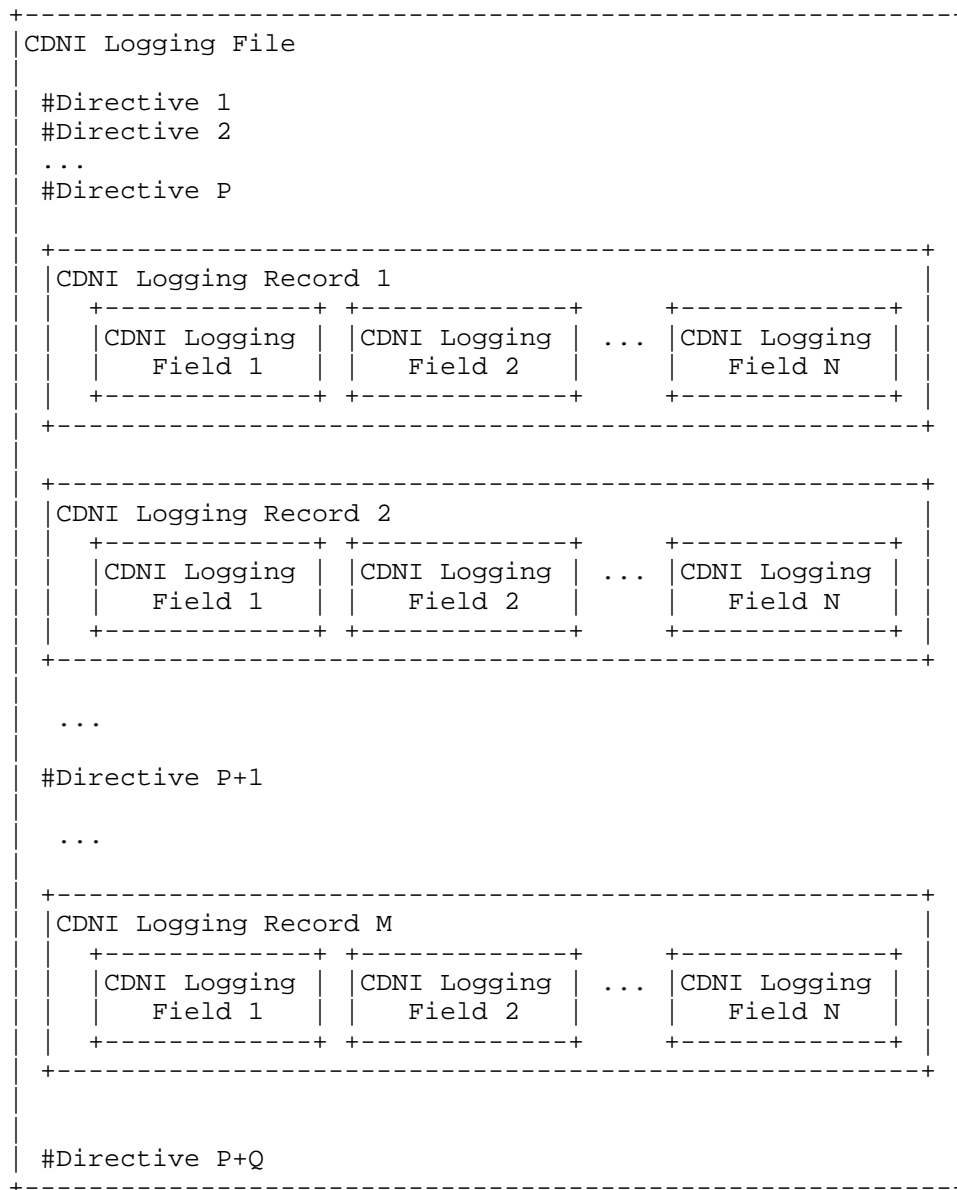


Figure 3: Structure of Logging Files

The CDNI Logging File format is inspired from the W3C Extended Log File Format [ELF]. However, it is fully specified by the present document. Where the present document differs from the W3C Extended

Log File Format, an implementation of the CDNI Logging interface MUST comply with the present document. The W3C Extended Log File Format was used as a starting point, reused where possible and expanded when necessary.

Using a format that resembles the W3C Extended Log File Format is intended to keep CDNI logging format close to the intra-CDN Logging information format commonly used in CDNs today, thereby minimizing systematic translation at CDN/CDNI boundary.

A CDNI Logging File MUST contain a sequence of lines containing US-ASCII characters [CHAR_SET] terminated by CRLF. Each line of a CDNI Logging File MUST contain either a directive or a CDNI Logging Record.

Directives record information about the CDNI Logging process itself. Lines containing directives MUST begin with the "#" character. Directives are specified in Section 3.3.

Logging Records provide actual details of the logged event. Logging Records are specified in Section 3.4.

The CDNI Logging File has a specific structure. It always starts with a directive line and the first directive it contains MUST be the version.

The directive lines form together a group that contains at least one directive line. Each directives group is followed by a group of logging records. The records group contains zero or more actual logging record lines about the event being logged. A record line consists of the values corresponding to all or a subset of the possible Logging fields defined within the scope of the record-type directive. These values MUST appear in the order defined by the fields directive.

Note that future extensions MUST be compliant with the previous description. The following examples depict the structure of a CDNILOGFILE as defined currently by the record-type "cdni_http_request_v1."

DIRLINE = "#" directive CRLF

DIRGROUP = 1*DIRLINE

RECLINE = <any subset of record values that match what is expected according to the fields directive within the immediately preceding DIRGROUP>

```
RECGROUP = *RECLINE
```

```
CDNILOGFILE = 1*(DIRGROUP RECGROUP)
```

3.3. CDNI Logging Directives

A CDNI Logging directive line contains the directive name followed by ":" HTAB and the directive value.

Directive names MUST be of the format NAMEFORMAT. All directive names MUST be registered in the CDNI Logging Directives Names registry. Directive names are case-insensitive as per the basic ABNF([RFC5234]). Unknown directives MUST be ignored. Directive values can have various formats. All possible directive values for the record-type "cdni_http_request_v1" are further detailed in this section.

The following example shows the structure of a directive and enumerates strictly the directive values presently defined in the version "cdni/1.0" of the CDNI Logging File.

```
directive = DIRNAME ":" HTAB DIRVAL
```

```
DIRNAME = NAMEFORMAT
```

```
DIRVAL = NHTABSTRING / QSTRING / host / USER-COMMENT / FIENAME *  
(HTAB FIENAME) / 64HEXDIG
```

An implementation of the CDNI Logging interface MUST support all of the following directives, listed below by their directive name:

- o version:

- * format: NHTABSTRING

- * directive value: indicates the version of the CDNI Logging File format. The entity transmitting a CDNI Logging File as per the present document MUST set the value to "cdni/1.0". In the future, other versions of CDNI Logging File might be specified; those would use a value different to "cdni/1.0" allowing the entity receiving the CDNI Logging File to identify the corresponding version. CDNI Logging File versions are case-insensitive as per the basic ABNF([RFC5234]).

- * occurrence: there MUST be one and only one instance of this directive per CDNI Logging File. It MUST be the first line of the CDNI Logging File.

- * example: "version: HTAB cdni/1.0".
- o UUID:
 - * format: NHTABSTRING
 - * directive value: this a Uniform Resource Name (URN) from the Universally Unique Identifier (UUID) URN namespace specified in [RFC4122]). The UUID contained in the URN uniquely identifies the CDNI Logging File.
 - * occurrence: there MUST be one and only one instance of this directive per CDNI Logging File.
 - * example: "UUID: HTAB NHTABSTRING".
- o claimed-origin:
 - * format: host
 - * directive value: this contains the claimed identification of the entity transmitting the CDNI Logging File (e.g., the host in a dCDN supporting the CDNI Logging interface) or the entity responsible for transmitting the CDNI Logging File (e.g., the dCDN).
 - * occurrence: there MUST be zero or exactly one instance of this directive per CDNI Logging File. This directive MAY be included by the dCDN. It MUST NOT be included or modified by the uCDN.
 - * example: "claimed-origin: HTAB host".
- o established-origin:
 - * format: host
 - * directive value: this contains the identification, as established by the entity receiving the CDNI Logging File, of the entity transmitting the CDNI Logging File (e.g., the host in a dCDN supporting the CDNI Logging interface) or the entity responsible for transmitting the CDNI Logging File (e.g., the dCDN).
 - * occurrence: there MUST be zero or exactly one instance of this directive per CDNI Logging File. This directive MAY be added by the uCDN (e.g., before storing the CDNI Logging File). It MUST NOT be included by the dCDN. The mechanisms used by the

uCDN to establish and validate the entity responsible for the CDNI Logging File is outside the scope of the present document. We observe that, in particular, this may be achieved through authentication mechanisms that are part of the transport layer of the CDNI Logging File pull mechanism (Section 4.2).

- * ABNF example: "established-origin: HTAB host".
- o remark:
 - * format: USER-COMMENT
 - * directive value: this contains comment information. Data contained in this field is to be ignored by analysis tools.
 - * occurrence: there MAY be zero, one or any number of instance of this directive per CDNI Logging File.
 - * example: "remark: HTAB USER-COMMENT".
- o record-type:
 - * format: NAMEFORMAT
 - * directive value: indicates the type of the CDNI Logging Records that follow this directive, until another record-type directive (or the end of the CDNI Logging File). This can be any CDNI Logging Record type registered in the CDNI Logging Record-types registry (Section 6.3). For example this may be "cdni_http_request_v1" as specified in Section 3.4.1. CDNI Logging record-types are case-insensitive as per the basic ABNF([RFC5234]).
 - * occurrence: there MUST be at least one instance of this directive per CDNI Logging File. The first instance of this directive MUST precede a fields directive and MUST precede all CDNI Logging Records.
 - * example: "record-type: HTAB cdni_http_request_v1".
- o fields:
 - * format: FIENAME *(HTAB FIENAME) ; where FIENAME can take any CDNI Logging field name registered in the CDNI Logging Field Names registry (Section 6.4) that is valid for the record type specified in the record-type directive.

- * directive value: this lists the names of all the fields for which a value is to appear in the CDNI Logging Records that follow the instance of this directive (until another instance of this directive). The names of the fields, as well as their occurrences, MUST comply with the corresponding rules specified in the document referenced in the CDNI Logging Record-types registry (Section 6.3) for the corresponding CDNI Logging record-type.
 - * occurrence: there MUST be at least one instance of this directive per record-type directive. The first instance of this directive for a given record-type MUST appear before any CDNI Logging Record for this record-type. One situation where more than one instance of the fields directive can appear within a given CDNI Logging File, is when there is a change, in the middle of a fairly large logging period, in the agreement between the uCDN and the dCDN about the set of fields that are to be exchanged. The multiple occurrences allow records with the old set of fields and records with the new set of fields to be carried inside the same Logging File.
 - * example: "fields: HTAB FIENAME * (HTAB FIENAME)".
- o SHA256-hash:
 - * format: 64HEXDIG
 - * directive value: This directive permits the detection of a corrupted CDNI Logging File. This can be useful, for instance, if a problem occurs on the filesystem of the dCDN Logging system and leads to a truncation of a logging file. The valid SHA256-hash value is included in this directive by the entity that transmits the CDNI Logging File. It MUST be computed by applying the SHA-256 ([RFC6234]) cryptographic hash function on the CDNI Logging File, including all the directives and logging records, up to the SHA256-hash directive itself, excluding the SHA256-hash directive itself. The SHA256-hash value MUST be represented as a US-ASCII encoded hexadecimal number, 64 digits long (representing a 256 bit hash value). The entity receiving the CDNI Logging File also computes in a similar way the SHA-256 hash on the received CDNI Logging File and compares this hash to the value of the SHA256-hash directive. If the two values are equal, then the received CDNI Logging File is to be considered non-corrupted. If the two values are different, the received CDNI Logging File is to be considered corrupted. The behavior of the entity that received a corrupted CDNI Logging File is outside the scope of this specification; we note that the entity MAY attempt to pull again the same CDNI

Logging File from the transmitting entity. If the entity receiving a non-corrupted CDNI Logging File adds an established-origin directive, it MUST then recompute and update the SHA256-hash directive so it also protects the added established-origin directive.

- * occurrence: there MUST be zero or exactly one instance of this directive. There SHOULD be exactly one instance of this directive. One situation where that directive could be omitted is where integrity protection is already provided via another mechanism (for example if an integrity hash is associated to the CDNI Logging File out-of-band through the CDNI Logging Feed (Section 4.1) leveraging ATOM extensions such as those proposed in [I-D.snell-atompub-link-extensions]. When present, the SHA256-hash field MUST be the last line of the CDNI Logging File.
- * example: "SHA256-hash: HTAB 64HEXDIG".

An uCDN-side implementation of the CDNI Logging interface MUST ignore a CDNI Logging File that does not comply with the occurrences specified above for each and every directive. For example, an uCDN-side implementation of the CDNI Logging interface receiving a CDNI Logging file with zero occurrence of the version directive, or with two occurrences of the SHA256-hash, MUST ignore this CDNI Logging File.

An entity receiving a CDNI Logging File with a value set to "cdni/1.0" MUST process the CDNI Logging File as per the present document. An entity receiving a CDNI Logging File with a value set to a different value MUST process the CDNI Logging File as per the specification referenced in the CDNI Logging File version registry (see Section 6.1) if the implementation supports this specification and MUST ignore the CDNI Logging File otherwise.

3.4. CDNI Logging Records

A CDNI Logging Record consists of a sequence of CDNI Logging fields relating to that single CDNI Logging Record.

CDNI Logging fields MUST be separated by the "horizontal tabulation (HTAB)" character.

To facilitate readability, a prefix scheme is used for CDNI Logging field names in a similar way to the one used in W3C Extended Log File Format [ELF]. The semantics of the prefix in the present document is:

- o "c-" refers to the User Agent that issues the request (corresponds to the "client" of W3C Extended Log Format)
- o "d-" refers to the dCDN (relative to a given CDN acting as an uCDN)
- o "s-" refers to the dCDN Surrogate that serves the request (corresponds to the "server" of W3C Extended Log Format)
- o "u-" refers to the uCDN (relative to a given CDN acting as a dCDN)
- o "cs-" refers to communication from the User Agent towards the dCDN Surrogate
- o "sc-" refers to communication from the dCDN Surrogate towards the User Agent

An implementation of the CDNI Logging interface as per the present specification MUST support the CDNI HTTP Request Logging Record as specified in Section 3.4.1.

A CDNI Logging Record contains the corresponding values for the fields that are enumerated in the last fields directive before the current log line. Note that the order in which the field values appear is dictated by the order of the fields names in the fields directive. There SHOULD be no dependency between the various fields values.

3.4.1. HTTP Request Logging Record

This section defines the CDNI Logging Record of record-type "cdni_http_request_v1". It is applicable to content delivery performed by the dCDN using HTTP/1.0([RFC1945]), HTTP/1.1([RFC7230],[RFC7231],[RFC7232],[RFC7233],[RFC7234],[RFC7235]) or HTTPS ([RFC2818],[RFC7230]). We observe that, in the case of HTTPS delivery, there may be value in logging additional information specific to the operation of HTTP over TLS and we note that this is outside the scope of the present document and may be addressed in a future document defining another CDNI Logging Record or another version of the HTTP Request Logging Record.

The "cdni_http_request_v1" record-type is also expected to be applicable to HTTP/2 [RFC7540] since a fundamental design tenet of HTTP/2 is to preserve the HTTP/1.1 semantics. We observe that, in the case of HTTP/2 delivery, there may be value in logging additional information specific to the additional functionality of HTTP/2 (e.g., information related to connection identification, to stream identification, to stream priority and to flow control). We note

that such additional information is outside the scope of the present document and may be addressed in a future document defining another CDNI Logging Record or another version of the HTTP Request Logging Record.

The "cdni_http_request_v1" record-type contains the following CDNI Logging fields, listed by their field name:

- o date:
 - * format: DATE
 - * field value: the date at which the processing of request completed on the Surrogate.
 - * occurrence: there MUST be one and only one instance of this field.
- o time:
 - * format: TIME
 - * field value: the time, which MUST be expressed in Coordinated Universal Time (UTC), at which the processing of request completed on the Surrogate.
 - * occurrence: there MUST be one and only one instance of this field.
- o time-taken:
 - * format: DEC
 - * field value: decimal value of the duration, in seconds, between the start of the processing of the request and the completion of the request processing (e.g., completion of delivery) by the Surrogate.
 - * occurrence: there MUST be one and only one instance of this field.
- o c-groupid:
 - * format: NHTABSTRING
 - * field value: an opaque identifier for an aggregate set of clients, derived from the client IPv4 or IPv6 address in the request received by the Surrogate and/or other network-level

identifying information. The c-groupid serves to group clients into aggregates. Example aggregates include civil geolocation information (the country, second-level administrative division, or postal code from which the client is presumed to make the request based on a geolocation database lookup) or network topological information (e.g., the BGP AS number announcing the prefix containing the address). The c-groupid MAY be structured e.g., US/TN/MEM/38138. Agreement between the dCDN and the uCDN on a mapping between IPv4 and IPv6 addresses and aggregates is presumed to occur out-of-band. The aggregation mapping SHOULD be chosen such that each aggregate contains more than one client.

- + When the aggregate is chosen so that it contains a single client (e.g., to allow more detailed analytics, or to allow a-posteriori analysis of individual delivery for example in situations of performance-based penalties) the c-groupid MAY be structured where some elements identify aggregates and one element identifies the client, e.g., US/TN/MEM/38138/43a5bdd6-95c4-4d62-be65-7410df0021e2. In the case where the aggregate is chosen so that it contains a single client:
 - the element identifying the client SHOULD be algorithmically generated (from the client IPv4 or IPv6 address in the request received by the Surrogate and/or other network-level identifying information) in a way that SHOULD NOT be linkable back to the global addressing context and that SHOULD vary over time (to offer protection against long term attacks).
 - It is RECOMMENDED that the mapping varies at least once every 24 hours.
 - The algorithmic mapping and variation over time can, in some cases, allow the uCDN (with the knowledge of the algorithm and time variation and associated attributes and keys) to reconstruct the actual client IPv4 or IPv6 address and/or other network-level identifying information when required (e.g., to allow a-posteriori analysis of individual delivery for example in situations of performance-based penalties). However, these enduser addresses SHOULD only be reconstructed on-demand and the CDNI Logging File SHOULD only be stored with the anonymised c-groupid value.
 - Allowing reconstruction of client address information carries with it grave risks to end-user privacy. Since

the c-groupid is in this case equivalent in identification power to a client IP address, its use may be restricted by regulation or law as personally identifiable information. For this reason, such use is NOT RECOMMENDED.

- One method for mapping that MAY be supported by implementations relies on a symmetric key that is known only to the uCDN and dCDN and HMAC-based Extract-and-Expand Key Derivation Function (HKDF) key derivation ([RFC5869]), as will be used in TLS 1.3 ([I-D.ietf-tls-rfc5246-bis]). When that method is used:
 - o The uCDN and dCDN need to agree on the "salt" and "input keying material", as described in Section 2.2 of [RFC5869] and the initial "info" parameter (which could be something like the business names of the two organizations in UTF-8, concatenated), as described in Section 2.3 of [RFC5869]. The hash SHOULD be either SHA-2 or SHA-3 [SHA-3] and the encryption algorithm SHOULD be 128-bit AES [AES] in Galois Counter Mode (GCM) [GCM] (AES-GCM) or better. The PRK SHOULD be chosen by both parties contributing alternate random bytes until sufficient length exists. After the initial setup, client-information can be encrypted using the key generated by the "expand" step of Section 2.3 of [RFC5869]. The encrypted value SHOULD be hex encoded or base64 encoded (as specified in section 4 of [RFC4648]). At the agreed-upon expiration time, a new key SHOULD be generated and used. New keys SHOULD be indicated by prefixing the key with a special character such as exclamation point. In this way, shorter lifetimes can be used as needed.
- * occurrence: there MUST be one and only one instance of this field.
- o s-ip:
 - * format: ADDRESS
 - * field value: the IPv4 or IPv6 address of the Surrogate that served the request (i.e., the "server" address).
 - * occurrence: there MUST be zero or exactly one instance of this field.

- o s-hostname:
 - * format: host
 - * field value: the hostname of the Surrogate that served the request (i.e., the "server" hostname).
 - * occurrence: there MUST be zero or exactly one instance of this field.
- o s-port:
 - * format: 1*DIGIT
 - * field value: the destination TCP port (i.e., the "server" port) in the request received by the Surrogate.
 - * occurrence: there MUST be zero or exactly one instance of this field.
- o cs-method:
 - * format: NHTABSTRING
 - * field value: this is the method of the request received by the Surrogate. In the case of HTTP delivery, this is the HTTP method in the request.
 - * occurrence: There MUST be one and only one instance of this field.
- o cs-uri:
 - * format: NHTABSTRING
 - * field value: this is the "effective request URI" of the request received by the Surrogate as specified in [RFC7230]. It complies with the "http" URI scheme or the "https" URI scheme as specified in [RFC7230]). Note that cs-uri can be privacy sensitive. In that case, and where appropriate, u-uri could be used instead of cs-uri.
 - * occurrence: there MUST be zero or exactly one instance of this field.
- o u-uri:
 - * format: NHTABSTRING

- * field value: this is a complete URI, derived from the "effective request URI" ([RFC7230]) of the request received by the Surrogate (i.e., the cs-uri) but transformed by the entity generating or transmitting the CDNI Logging Record, in a way that is agreed upon between the two ends of the CDNI Logging interface, so the transformed URI is meaningful to the uCDN. For example, the two ends of the CDNI Logging interface could agree that the u-uri is constructed from the cs-uri by removing the part of the hostname that exposes which individual Surrogate actually performed the delivery. The details of modification performed to generate the u-uri, as well as the mechanism to agree on these modifications between the two sides of the CDNI Logging interface are outside the scope of the present document.
- * occurrence: there MUST be one and only one instance of this field.
- o protocol:
 - * format: NHTABSTRING
 - * field value: this is value of the HTTP-Version field as specified in [RFC7230] of the Request-Line of the request received by the Surrogate (e.g., "HTTP/1.1").
 - * occurrence: there MUST be one and only one instance of this field.
- o sc-status:
 - * format: 3DIGIT
 - * field value: this is the Status-Code in the response from the Surrogate. In the case of HTTP delivery, this is the HTTP Status-Code in the HTTP response.
 - * occurrence: There MUST be one and only one instance of this field.
- o sc-total-bytes:
 - * format: 1*DIGIT
 - * field value: this is the total number of bytes of the response sent by the Surrogate in response to the request. In the case of HTTP delivery, this includes the bytes of the Status-Line,

the bytes of the HTTP headers and the bytes of the message-body.

- * occurrence: There MUST be one and only one instance of this field.
- o sc-entity-bytes:
 - * format: 1*DIGIT
 - * field value: this is the number of bytes of the message-body in the HTTP response sent by the Surrogate in response to the request. This does not include the bytes of the Status-Line or the bytes of the HTTP headers.
 - * occurrence: there MUST be zero or exactly one instance of this field.
- o cs(insert_HTTP_header_name_here):
 - * format: QSTRING
 - * field value: the value of the HTTP header (identified by the insert_HTTP_header_name_here in the CDNI Logging field name) as it appears in the request processed by the Surrogate, but prepended by a DQUOTE and appended by a DQUOTE. For example, when the CDNI Logging field name (FIENAME) listed in the preceding fields directive is cs(User-Agent), this CDNI Logging field value contains the value of the User-Agent HTTP header as received by the Surrogate in the request it processed, but prepended by a DQUOTE and appended by a DQUOTE. If the HTTP header as it appeared in the request processed by the Surrogate contains one or more DQUOTE, each DQUOTE MUST be escaped with percent encoding. For example, if the HTTP header contains My_Header"value", then the field value of the cs(insert_HTTP_header_name_here) is "My_Header%x22value%x22". The entity transmitting the CDNI Logging File MUST ensure that the respective insert_HTTP_header_name_here of the cs(insert_HTTP_header_name_here) listed in the fields directive comply with HTTP specifications. In particular, this field name does not include any HTAB, since this would prevent proper parsing of the fields directive by the entity receiving the CDNI Logging File.
 - * occurrence: there MAY be zero, one or any number of instance of this field.
- o sc(insert_HTTP_header_name_here):

- * format: QSTRING
 - * field value: the value of the HTTP header (identified by the `insert_HTTP_header_name_here` in the CDNI Logging field name) as it appears in the response issued by the Surrogate to serve the request, but prepended by a DQUOTE and appended by a DQUOTE. If the HTTP header as it appeared in the request processed by the Surrogate contains one or more DQUOTE, each DQUOTE MUST be escaped with percent encoding. For example, if the HTTP header contains `My_Header"value"`, then the field value of the `sc(insert_HTTP_header_name_here)` is `"My_Header%x22value%x22"`. The entity transmitting the CDNI Logging File MUST ensure that the respective `insert_HTTP_header_name_here` of the `cs(insert_HTTP_header_name_here)` listed in the fields directive comply with HTTP specifications. In particular, this field name does not include any HTAB, since this would prevent proper parsing of the fields directive by the entity receiving the CDNI Logging File.
 - * occurrence: there MAY be zero, one or any number of instances of this field. For a given `insert_HTTP_header_name_here`, there MUST be zero or exactly one instance of this field.
- o s-ccid:
 - * format: QSTRING
 - * field value: this contains the value of the Content Collection Identifier (CCID) associated by the uCDN to the content served by the Surrogate via the CDNI Metadata interface ([I-D.ietf-cdni-metadata]), prepended by a DQUOTE and appended by a DQUOTE. If the CCID conveyed in the CDNI Metadata interface contains one or more DQUOTE, each DQUOTE MUST be escaped with percent encoding. For example, if the CCID conveyed in the CDNI Metadata interface is `My_CCIDD"value"`, then the field value of the s-ccid is `"My_CCID%x22value%X22"`.
 - * occurrence: there MUST be zero or exactly one instance of this field. For a given `insert_HTTP_header_name_here`, there MUST be zero or exactly one instance of this field.
 - o s-sid:
 - * format: QSTRING
 - * field value: this contains the value of a Session Identifier (SID) generated by the dCDN for a specific HTTP session, prepended by a DQUOTE and appended by a DQUOTE. In particular,

for HTTP Adaptive Streaming (HAS) session, the Session Identifier value is included in the Logging record for every content chunk delivery of that session in view of facilitating the later correlation of all the per content chunk log records of a given HAS session. See section 3.4.2.2. of [RFC6983] for more discussion on the concept of Session Identifier in the context of HAS. If the SID conveyed contains one or more DQUOTE, each DQUOTE MUST be escaped with percent encoding. For example, if the SID is My_SID"value", then the field value of the s-sid is "My_SID%x22value%x22".

- * occurrence: there MUST be zero or exactly one instance of this field.
- o s-cached:
 - * format: 1DIGIT
 - * field value: this characterises whether the Surrogate served the request using content already stored on its local cache or not. The allowed values are "0" (for miss) and "1" (for hit). "1" MUST be used when the Surrogate did serve the request using exclusively content already stored on its local cache. "0" MUST be used otherwise (including cases where the Surrogate served the request using some, but not all, content already stored on its local cache). Note that a "0" only means a cache miss in the Surrogate and does not provide any information on whether the content was already stored, or not, in another device of the dCDN, i.e., whether this was a "dCDN hit" or "dCDN miss".
 - * occurrence: there MUST be zero or exactly one instance of this field.

CDNI Logging field names are case-insensitive as per the basic ABNF([RFC5234]). The "fields" directive corresponding to a HTTP Request Logging Record MUST contain all the fields names whose occurrence is specified above as "There MUST be one and only one instance of this field". The corresponding fields value MUST be present in every HTTP Request Logging Record.

The "fields" directive corresponding to a HTTP Request Logging Record MAY list all the fields value whose occurrence is specified above as "there MUST be zero or exactly one instance of this field" or "there MAY be zero, one or any number of instances of this field". The set of such field names actually listed in the "fields" directive is selected by the CDN generating the CDNI Logging File based on agreements between the interconnected CDNs established through mechanisms outside the scope of this specification (e.g., contractual

agreements). When such a field name is not listed in the "fields" directive, the corresponding field value MUST NOT be included in the Logging Record. When such a field name is listed in the "fields" directive, the corresponding field value MUST be included in the Logging Record; if the value for the field is not available, this MUST be conveyed via a dash character ("-").

The fields names listed in the "fields" directive MAY be listed in the order in which they are listed in Section 3.4.1 or MAY be listed in any other order.

Logging some specific fields from HTTP requests and responses can introduce serious security and privacy risks. For example, cookies will often contain (months) long lived token values that can be used to log into a service as the relevant user. Similar values may be included in other header fields or within URLs or elsewhere in HTTP requests and responses. Centralising such values in a CDNI Logging File can therefore represent a significant increase in risk both for the user and the web service provider, but also for the CDNs involved. Implementations ought therefore to attempt to lower the probability of such bad outcomes e.g. by only allowing a configured set of headers to be added to CDNI Logging Records, or by not supporting wildcard selection of HTTP request/response fields to add. Such mechanisms can reduce the probability that security (or privacy) sensitive values are centralised in CDNI Logging Files. Also, when agreeing on which HTTP request/response fields are to be provided in CDNI Logging Files, the uCDN and dCDN administrators ought to consider these risks. Furthermore, CDNs making use of c-groupid to identify an aggregate of clients rather than individual clients ought to realize that by logging certain header fields they may create the possibility to re-identify individual clients. In these cases heeding the above advice, or not logging header fields at all, is particularly important if the goal is to provide logs that do not identify individual clients."

A dCDN-side implementation of the CDNI Logging interface MUST implement all the following Logging fields in a CDNI Logging Record of record-type "cdni_http_request_v1", and MUST support the ability to include valid values for each of them:

- o date
- o time
- o time-taken
- o c-groupid

- o s-ip
- o s-hostname
- o s-port
- o cs-method
- o cs-uri
- o u-uri
- o protocol
- o sc-status
- o sc-total-bytes
- o sc-entity-bytes
- o cs(insert_HTTP_header_name_here)
- o sc(insert_HTTP_header_name_here)
- o s-cached

A dCDN-side implementation of the CDNI Logging interface MAY support the following Logging fields in a CDNI Logging Record of record-type "cdni_http_request_v1":

- o s-ccid
- o s-sid

If a dCDN-side implementation of the CDNI Logging interface supports these fields, it MUST support the ability to include valid values for them.

An uCDN-side implementation of the CDNI Logging interface MUST be able to accept CDNI Logging Files with CDNI Logging Records of record-type "cdni_http_request_v1" containing any CDNI Logging Field defined in Section 3.4.1 as long as the CDNI Logging Record and the CDNI Logging File are compliant with the present document.

In case an uCDN-side implementation of the CDNI Logging interface receives a CDNI Logging File with HTTP Request Logging Records that do not contain field values for exactly the set of field names actually listed in the preceding "fields" directive, the

implementation MUST ignore those HTTP Request Logging Records, and MUST accept the other HTTP Request Logging Records.

To ensure that the logging file is correct, the text MUST be sanitized before being logged. Null, bare CR, bare LF and HTAB have to be removed by escaping them through percent encoding to avoid confusion with the logging record separators.

3.5. CDNI Logging File Extension

The CDNI Logging File contains blocks of directives and blocks of corresponding records. The supported set of directives is defined relative to the CDNI Logging File Format version. The complete set of directives for version "cdni/1.0" are defined in Section 3.3. The directive list is not expected to require much extension, but when it does, the new directive MUST be defined and registered in the "CDNI Logging Directive Names" registry, as described in Figure 9, and a new version MUST be defined and registered in the "CDNI Logging File version" registry, as described in Section 6.2. For example, adding a new CDNI Logging Directive, e.g., "foo", to the set of directives defined for "cdni/1.0" in Section 3.3, would require registering both the new CDNI Logging Directive "foo" and a new CDNI Logging File version, e.g., "CDNI/2.0", which includes all of the existing CDNI Logging Directives of "cdni/1.0" plus "foo".

It is expected that as new logging requirements arise, the list of fields to log will change and expand. When adding new fields, the new fields MUST be defined and registered in the "CDNI Logging Field Names" registry, as described in Section 6.4, and a new record-type MUST be defined and registered in the "CDNI Logging record-types" registry, as described in Section 6.3. For example, adding a new CDNI Logging Field, e.g., "c-bar", to the set of fields defined for "cdni_http_request_v1" in Section 3.4.1, would require registering both the new CDNI Logging Field "c-bar" and a new CDNI record-type, e.g., "cdni_http_request_v2", which includes all of the existing CDNI Logging Fields of "cdni_http_request_v1" plus "c-bar".

3.6. CDNI Logging File Examples

Let us consider the upstream CDN and the downstream CDN labelled uCDN and dCDN-1 in Figure 1. When dCDN-1 acts as a downstream CDN for uCDN and performs content delivery on behalf of uCDN, dCDN-1 will include the CDNI Logging Records corresponding to the content deliveries performed on behalf of uCDN in the CDNI Logging Files for uCDN. An example CDNI Logging File communicated by dCDN-1 to uCDN is shown below in Figure 4.

```

#version:<HTAB>cdni/1.0<CRLF>

#UUID:<HTAB>urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6<CRLF>

#claimed-origin:<HTAB>cdni-logging-entity.dcdn-1.example.com<CRLF>

#record-type:<HTAB>cdni_http_request_v1<CRLF>

#fields:<HTAB>date<HTAB>time<HTAB>time-taken<HTAB>c-groupid<HTAB>
cs-method<HTAB>u-uri<HTAB>protocol<HTAB>
sc-status<HTAB>sc-total-bytes<HTAB>cs(User-Agent)<HTAB>
cs(Referer)<HTAB>s-cached<CRLF>

2013-05-17<HTAB>00:38:06.825<HTAB>9.058<HTAB>US/TN/MEM/38138<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/movie100.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>6729891<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>00:39:09.145<HTAB>15.32<HTAB>FR/PACA/NCE/06100<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/movie118.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>15799210<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>00:42:53.437<HTAB>52.879<HTAB>US/TN/MEM/38138<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/picture11.mp4<HTAB>
HTTP/1.0<HTAB>200<HTAB>97234724<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host5.example.com"<HTAB>0<CRLF>

#SHA256-hash:<HTAB> 64-hexadecimal-digit hash value <CRLF>

```

Figure 4: CDNI Logging File Example

If uCDN establishes by some means (e.g., via TLS authentication when pulling the CDNI Logging File) the identity of the entity from which it pulled the CDNI Logging File, uCDN can add to the CDNI Logging an established-origin directive as illustrated below:

```

#established-origin:<HTAB>cdni-logging-entity.dcdn-
1.example.com<CRLF>

```

As illustrated in Figure 2, uCDN will then ingest the corresponding CDNI Logging Records into its Collection process, alongside the Logging Records generated locally by the uCDN itself. This allows uCDN to aggregate Logging Records for deliveries performed by itself (through Records generated locally) as well as for deliveries performed by its downstream CDN(s). This aggregate information can then be used (after Filtering and Rectification, as illustrated in Figure 2) by Log Consuming Applications that take into account deliveries performed by uCDN as well as by all of its downstream CDNs.

We observe that the time between

1. when a delivery is completed in dCDN and
2. when the corresponding Logging Record is ingested by the Collection process in uCDN

depends on a number of parameters such as the Logging Period agreed to by uCDN and dCDN, how much time uCDN waits before pulling the CDNI Logging File once it is advertised in the CDNI Logging Feed, and the time to complete the pull of the CDNI Logging File. Therefore, if we consider the set of Logging Records aggregated by the Collection process in uCDN in a given time interval, there could be a permanent significant timing difference between the CDNI Logging Records received from the dCDN and the Logging Records generated locally. For example, in a given time interval, the Collection process in uCDN may be aggregating Logging Records generated locally by uCDN for deliveries performed in the last hour and CDNI Logging Records generated in the dCDN for deliveries in the hour before last.

Say, that for some reason (for example a Surrogate bug), dCDN-1 could not collect the total number of bytes of the responses sent by the Surrogate (in other words, the value for sc-total-bytes is not available). Then the corresponding CDNI Logging records would contain a dash character ("-") in lieu of the value for the sc-total-bytes field (as specified in Section 3.4.1). In that case, the CDNI Logging File that would be communicated by dCDN-1 to uCDN is shown below in Figure 5.

```

#version:<HTAB>cdni/1.0<CRLF>

#UUID:<HTAB>urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6<CRLF>

#claimed-origin:<HTAB>cdni-logging-entity.dcdn-1.example.com<CRLF>

#record-type:<HTAB>cdni_http_request_v1<CRLF>

#fields:<HTAB>date<HTAB>time<HTAB>time-taken<HTAB>c-groupid<HTAB>
cs-method<HTAB>u-uri<HTAB>protocol<HTAB>
sc-status<HTAB>sc-total-bytes<HTAB>cs(User-Agent)<HTAB>
cs(Referer)<HTAB>s-cached<CRLF>

2013-05-17<HTAB>00:38:06.825<HTAB>9.058<HTAB>US/TN/MEM/38138<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/movie100.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>-<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>00:39:09.145<HTAB>15.32<HTAB>FR/PACA/NCE/06100<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/movie118.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>-<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>00:42:53.437<HTAB>52.879<HTAB>US/TN/MEM/38138<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/picture11.mp4<HTAB>
HTTP/1.0<HTAB>200<HTAB>-<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host5.example.com"<HTAB>0<CRLF>

#SHA256-hash:<HTAB> 64-hexadecimal-digit hash value <CRLF>

```

Figure 5: CDNI Logging File Example With A Missing Field Value

3.7. Cascaded CDNI Logging Files Example

Let us consider the cascaded CDN scenario of uCDN, dCDN-2 and dCDN-3 as depicted in Figure 1. After completion of a delivery by dCDN-3 on behalf of dCDN-2, dCDN-3 will include a corresponding Logging Record in a CDNI Logging File that will be pulled by dCDN-2 and that is illustrated below in Figure 6. In practice, a CDNI Logging File is

likely to contain a very high number of CDNI Logging Records. However, for readability, the example in Figure 6 contains a single CDNI Logging Record.

```
#version:<HTAB>cdni/1.0<CRLF>

#UUID:<HTAB>urn:uuid:65718ef-0123-9876-adce4321bcde<CRLF>

#claimed-origin:<HTAB>cdni-logging-entity.dcdn-3.example.com<CRLF>

#record-type:<HTAB>cdni_http_request_v1<CRLF>

#fields:<HTAB>date<HTAB>time<HTAB>time-taken<HTAB>c-groupid<HTAB>
cs-method<HTAB>u-uri<HTAB>protocol<HTAB>
sc-status<HTAB>sc-total-bytes<HTAB>cs(User-Agent)<HTAB>
cs(Referer)<HTAB>s-cached<CRLF>

2013-05-17<HTAB>00:39:09.119<HTAB>14.07<HTAB>US/CA/SFO/94114<HTAB>
GET<HTAB>
http://cdni-dcdn-2.dcdn-3.example.com/video/movie118.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>15799210<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari /533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

#SHA256-hash:<HTAB> 64-hexadecimal-digit hash value <CRLF>
```

Figure 6: Cascaded CDNI Logging File Example (dCDN-3 to dCDN-2)

If dCDN-2 establishes by some means (e.g., via TLS authentication when pulling the CDNI Logging File) the identity of the entity from which it pulled the CDNI Logging File, dCDN-2 can add to the CDNI Logging an established-origin directive as illustrated below:

```
#established-origin:<HTAB>cdni-logging-entity.dcdn-
3.example.com<CRLF>
```

dCDN-2 (behaving as an upstream CDN from the viewpoint of dCDN-3) will then ingest the CDNI Logging Record for the considered dCDN-3 delivery into its Collection process (as illustrated in Figure 2). This Logging Record may be aggregated with Logging Records generated locally by dCDN-2 for deliveries performed by dCDN-2 itself. Say, for illustration, that the content delivery performed by dCDN-3 on behalf of dCDN-2 had actually been redirected to dCDN-2 by uCDN, and say that another content delivery has just been redirected by uCDN to dCDN-2 and that dCDN-2 elected to perform the corresponding delivery itself. Then after Filtering and Rectification (as illustrated in Figure 2), dCDN-2 will include the two Logging Records corresponding

respectively to the delivery performed by dCDN-3 and the delivery performed by dCDN-2, in the next CDNI Logging File that will be communicated to uCDN. An example of such CDNI Logging File is illustrated below in Figure 7.

```
#version:<HTAB>cdni/1.0<CRLF>

#UUID:<HTAB>urn:uuid:1234567-8fedc-abab-0987654321ff<CRLF>

#claimed-origin:<HTAB>cdni-logging-entity.dcdn-2.example.com<CRLF>

#record-type:<HTAB>cdni_http_request_v1<CRLF>

#fields:<HTAB>date<HTAB>time<HTAB>time-taken<HTAB>c-groupid<HTAB>
cs-method<HTAB>u-uri<HTAB>protocol<HTAB>
sc-status<HTAB>sc-total-bytes<HTAB>cs(User-Agent)<HTAB>
cs(Referer)<HTAB>s-cached<CRLF>

2013-05-17<HTAB>00:39:09.119<HTAB>14.07<HTAB>US/CA/SFO/94114<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-2.example.com/video/movie118.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>15799210<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari /533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>01:42:53.437<HTAB>52.879<HTAB>FR/IDF/PAR/75001<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-2.example.com/video/picture11.mp4<HTAB>
HTTP/1.0<HTAB>200<HTAB>97234724<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari /533.4"<HTAB>
"host5.example.com"<HTAB>0<CRLF>

#SHA256-hash:<HTAB> 64-hexadecimal-digit hash value <CRLF>
```

Figure 7: Cascaded CDNI Logging File Example (dCDN-2 to uCDN)

If uCDN establishes by some means (e.g., via TLS authentication when pulling the CDNI Logging File) the identity of the entity from which it pulled the CDNI Logging File, uCDN can add to the CDNI Logging an established-origin directive as illustrated below:

```
#established-origin:<HTAB>cdni-logging-entity.dcdn-
2.example.com<CRLF>
```

In the example of Figure 7, we observe that:

- o the first Logging Record corresponds to the Logging Record communicated earlier to dCDN-2 by dCDN-3, which corresponds to a delivery redirected by uCDN to dCDN-2 and then redirected by dCDN-2 to dCDN-3. The fields values in this Logging Record are copied from the corresponding CDNI Logging REcord communicated to dCDN2 by dCDN-3, with the exception of the u-uri that now reflects the URI convention between uCDN and dCDN-2 and that presents the delivery to uCDN as if it was performed by dCDN-2 itself. This reflects the fact that dCDN-2 had taken the full responsibility of the corresponding delivery (even if in this case, dCDN-2 elected to redirect the delivery to dCDN-3 so it is actually performed by dCDN-3 on behalf of dCDN-2).
- o the second Logging Record corresponds to a delivery redirected by uCDN to dCDN-2 and performed by dCDN-2 itself. The time of the delivery in this Logging Record may be significantly more recent than the first Logging Record since it was generated locally while the first Logging Record was generated by dCDN-3 and had to be advertised , and then pulled and then ingested into the dCDN-2 Collection process, before being aggregated with the second Logging Record.

4. Protocol for Exchange of CDNI Logging File After Full Collection

This section specifies a protocol for the exchange of CDNI Logging Files as specified in Section 3 after the CDNI Logging File is fully collected by the dCDN.

This protocol comprises:

- o a CDNI Logging feed, allowing the dCDN to notify the uCDN about the CDNI Logging Files that can be retrieved by that uCDN from the dCDN, as well as all the information necessary for retrieving each of these CDNI Logging Files. The CDNI Logging feed is specified in Section 4.1.
- o a CDNI Logging File pull mechanism, allowing the uCDN to obtain from the dCDN a given CDNI Logging File at the uCDN's convenience. The CDNI Logging File pull mechanisms is specified in Section 4.2.

An implementation of the CDNI Logging interface on the dCDN side (the entity generating the CDNI Logging file) MUST support the server side of the CDNI Logging feed (as specified in Section 4.1) and the server side of the CDNI Logging pull mechanism (as specified in Section 4.2).

An implementation of the CDNI Logging interface on the uCDN side (the entity consuming the CDNI Logging file) MUST support the client side

of the CDNI Logging feed (as specified in Section 4.1) and the client side of the CDNI Logging pull mechanism (as specified in Section 4.2).

4.1. CDNI Logging Feed

The server-side implementation of the CDNI Logging feed MUST produce an Atom feed [RFC4287]. This feed is used to advertise log files that are available for the client-side to retrieve using the CDNI Logging pull mechanism.

4.1.1. Atom Formatting

A CDNI Logging feed MUST be structured as an Archived feed, as defined in [RFC5005], and MUST be formatted in Atom [RFC4287]. This means it consists of a subscription document that is regularly updated as new CDNI Logging Files become available, and information about older CDNI Logging files is moved into archive documents. Once created, archive documents are never modified.

Each CDNI Logging File listed in an Atom feed MUST be described in an `atom:entry` container element.

The `atom:entry` MUST contain an `atom:content` element whose `"src"` attribute is a link to the CDNI Logging File and whose `"type"` attribute is the MIME Media Type indicating that the entry is a CDNI logging file. This MIME Media Type is defined as `"application/cdni"` (See [RFC7736]) with the Payload Type (`ptype`) parameter set to `"logging-file"`.

For compatibility with some Atom feed readers the `atom:entry` MAY also contain an `atom:link` entry whose `"href"` attribute is a link to the CDNI Logging File and whose `"type"` attribute is the MIME Media Type indicating that the entry is a CDNI Logging File using the `"application/cdni"` MIME Media Type with the Payload Type (`ptype`) parameter set to `"logging-file"` (See [RFC7736]).

The URI used in the `atom:id` of the `atom:entry` MUST contain the UUID of the CDNI Logging File.

The `atom:updated` in the `atom:entry` MUST indicate the time at which the CDNI Logging File was last updated.

4.1.2. Updates to Log Files and the Feed

CDNI Logging Files MUST NOT be modified by the dCDN once published in the CDNI Logging feed.

The frequency with which the subscription feed is updated, the period of time covered by each CDNI Logging File or each archive document, and timeliness of publishing of CDNI Logging Files are outside the scope of the present document and are expected to be agreed upon by uCDN and dCDN via other means (e.g., human agreement).

The server-side implementation **MUST** be able to set, and **SHOULD** set, HTTP cache control headers on the subscription feed to indicate the frequency at which the client-side is to poll for updates.

The client-side **MAY** use HTTP cache control headers (set by the server-side) on the subscription feed to determine the frequency at which to poll for updates. The client-side **MAY** instead, or in addition, use other information to determine when to poll for updates (e.g., a polling frequency that may have been negotiated between the uCDN and dCDN by mechanisms outside the scope of the present document and that is to override the indications provided in the HTTP cache control headers).

The potential retention limits (e.g., sliding time window) within which the dCDN is to retain and be ready to serve an archive document is outside the scope of the present document and is expected to be agreed upon by uCDN and dCDN via other means (e.g., human agreement). The server-side implementation **MUST** retain, and be ready to serve, any archive document within the agreed retention limits. Outside these agreed limits, the server-side implementation **MAY** indicate its inability to serve (e.g., with HTTP status code 404) an archive document or **MAY** refuse to serve it (e.g., with HTTP status code 403 or 410).

4.1.3. Redundant Feeds

The server-side implementation **MAY** present more than one CDNI Logging feed for redundancy. Each CDNI Logging File **MAY** be published in more than one feed.

A client-side implementation **MAY** support such redundant CDNI Logging feeds. If it supports redundant CDNI Logging feed, the client-side can use the UUID of the CDNI Logging File, presented in the atom:id element of the Atom feed, to avoid unnecessarily pulling and storing a given CDNI Logging File more than once.

4.1.4. Example CDNI Logging Feed

Figure 8 illustrates an example of the subscription document of a CDNI Logging feed.

```

<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title type="text">CDNI Logging Feed</title>
  <updated>2013-03-23T14:46:11Z</updated>
  <id>urn:uuid:663ae677-40fb-e99a-049d-c5642916b8ce</id>
  <link href="https://dcdn.example/logfeeds/ucdn1"
    rel="self" type="application/atom+xml" />
  <link href="https://dcdn.example/logfeeds/ucdn1"
    rel="current" type="application/atom+xml" />
  <link href="https://dcdn.example/logfeeds/ucdn1/201303231400"
    rel="prev-archive" type="application/atom+xml" />
  <generator version="example version 1">CDNI Log Feed
    Generator</generator>
  <author><name>dcdn.example</name></author>
  <entry>
    <title type="text">CDNI Logging File for uCDN at
      2013-03-23 14:15:00</title>
    <id>urn:uuid:12345678-1234-abcd-00aa-01234567abcd</id>
    <updated>2013-03-23T14:15:00Z</updated>
    <content src="https://dcdn.example/logs/ucdn/
      http-requests-20130323141500000000"
      type="application/cdni"
      ptype="logging-file"/>
    <summary>CDNI Logging File for uCDN at
      2013-03-23 14:15:00</summary>
  </entry>
  <entry>
    <title type="text">CDNI Logging File for uCDN at
      2013-03-23 14:30:00</title>
    <id>urn:uuid:87654321-4321-dcba-aa00-dcba7654321</id>
    <updated>2013-03-23T14:30:00Z</updated>
    <content src="https://dcdn.example/logs/ucdn/
      http-requests-20130323143000000000"
      type="application/cdni"
      ptype="logging-file"/>
    <summary>CDNI Logging File for uCDN at
      2013-03-23 14:30:00</summary>
  </entry>
  ...
  <entry>
    ...
  </entry>
</feed>

```

Figure 8: Example subscription document of a CDNI Logging Feed

4.2. CDNI Logging File Pull

A client-side implementation of the CDNI Logging interface MAY pull, at its convenience, a CDNI Logging File that is published by the server-side in the CDNI Logging Feed (in the subscription document or an archive document). To do so, the client-side:

- o MUST implement HTTP/1.1 ([RFC7230],[RFC7231], [RFC7232], [RFC7233], [RFC7234], [RFC7235]), MAY also support other HTTP versions (e.g., HTTP/2 [RFC7540]) and MAY negotiate which HTTP version is actually used. This allows operators and implementers to choose to use later versions of HTTP to take advantage of new features, while still ensuring interoperability with systems that only support HTTP/1.1.
- o MUST use the URI that was associated to the CDNI Logging File (within the "src" attribute of the corresponding atom:content element) in the CDNI Logging Feed;
- o MUST support exchange of CDNI Logging Files with no content encoding applied to the representation;
- o MUST support exchange of CDNI Logging Files with "gzip" content encoding (as defined in [RFC7230]) applied to the representation.

Note that a client-side implementation of the CDNI Logging interface MAY pull a CDNI Logging File that it has already pulled.

The server-side implementation MUST respond to valid pull request by a client-side implementation for a CDNI Logging File published by the server-side in the CDNI Logging Feed (in the subscription document or an archive document). The server-side implementation:

- o MUST implement HTTP/1.1 to handle the client-side request and MAY also support other HTTP versions (e.g., HTTP/2);
- o MUST include the CDNI Logging File identified by the request URI inside the body of the HTTP response;
- o MUST support exchange of CDNI Logging Files with no content encoding applied to the representation;
- o MUST support exchange of CDNI Logging Files with "gzip" content encoding (as defined in [RFC7231]) applied to the representation.

Content negotiation approaches defined in [RFC7231] (e.g., using Accept-Encoding request-header field or Content-Encoding entity-header field) MAY be used by the client-side and server-side

implementations to establish the content-coding to be used for a particular exchange of a CDNI Logging File.

Applying compression content encoding (such as "gzip") is expected to mitigate the impact of exchanging the large volumes of logging information expected across CDNs. This is expected to be particularly useful in the presence of HTTP Adaptive Streaming (HAS) which, as per the present version of the document, will result in a separate CDNI Log Record for each HAS segment delivery in the CDNI Logging File.

The potential retention limits (e.g., sliding time window, maximum aggregate file storage quotas) within which the dCDN is to retain and be ready to serve a CDNI Logging File previously advertised in the CDNI Logging Feed is outside the scope of the present document and is expected to be agreed upon by uCDN and dCDN via other means (e.g., human agreement). The server-side implementation **MUST** retain, and be ready to serve, any CDNI Logging File within the agreed retention limits. Outside these agreed limits, the server-side implementation **MAY** indicate its inability to serve (e.g., with HTTP status code 404) a CDNI Logging File or **MAY** refuse to serve it (e.g., with HTTP status code 403 or 410).

5. Protocol for Exchange of CDNI Logging File During Collection

We note that, in addition to the CDNI Logging File exchange protocol specified in Section 4, implementations of the CDNI Logging interface may also support other mechanisms to exchange CDNI Logging Files. In particular, such mechanisms might allow the exchange of the CDNI Logging File to start before the file is fully collected. This can allow CDNI Logging Records to be communicated by the dCDN to the uCDN as they are gathered by the dCDN without having to wait until all the CDNI Logging Records of the same logging period are collected in the corresponding CDNI Logging File. This approach is commonly referred to as "tailing" of the file.

Such an approach could be used, for example, to exchange logging information with a significantly reduced time-lag (e.g., sub-minute or sub-second) between when the event occurred in the dCDN and when the corresponding CDNI Logging Record is made available to the uCDN. This can satisfy log-consuming applications requiring extremely fresh logging information such as near-real-time content delivery monitoring. Such mechanisms are for further study and outside the scope of this document.

6. IANA Considerations

6.1. CDNI Logging Directive Names Registry

The IANA is requested to create a new "CDNI Logging Directive Names" subregistry under the "Content Delivery Networks Interconnection (CDNI) Parameters" registry.

The initial contents of the CDNI Logging Directives registry comprise the names of the directives specified in Section 3.3 of the present document, and are as follows:

Directive Name	Reference
version	RFC xxxx
UUID	RFC xxxx
claimed-origin	RFC xxxx
established-origin	RFC xxxx
remark	RFC xxxx
record-type	RFC xxxx
fields	RFC xxxx
SHA256-hash	RFC xxxx

Figure 9

[Instructions to IANA: Replace "RFC xxxx" above by the RFC number of the present document]

Within the registry, names are to be allocated by IANA according to the "Specification Required" policy specified in [RFC5226]. Directive names are to be allocated by IANA with a format of NAMEFORMAT (see Section 3.1). All directive names defined in the logging file are case-insensitive as per the basic ABNF([RFC5234]).

Each specification that defines a new CDNI Logging directive needs to contain a description for the new directive with the same set of information as provided in Section 3.3 (i.e., format, directive value and occurrence).

6.2. CDNI Logging File version Registry

The IANA is requested to create a new "CDNI Logging File version" subregistry under the "Content Delivery Networks Interconnection (CDNI) Parameters" registry.

The initial contents of the CDNI Logging Logging File version registry comprise the value "cdni/1.0" specified in Section 3.3 of the present document, and are as follows:

version	Reference	Description
cdni/1.0	RFC xxxx	CDNI Logging File version 1.0 as specified in RFC xxxx

Figure 10

[Instructions to IANA: Replace "RFC xxxx" above by the RFC number of the present document]

Within the registry, version values are to be allocated by IANA according to the "Specification Required" policy specified in [RFC5226]. Version values are to be allocated by IANA with a format of NAMEFORMAT (see Section 3.1). All version values defined in the logging file are case-insensitive as per the basic ABNF([RFC5234]).

6.3. CDNI Logging record-types Registry

The IANA is requested to create a new "CDNI Logging record-types" subregistry under the "Content Delivery Networks Interconnection (CDNI) Parameters" registry.

The initial contents of the CDNI Logging record-types registry comprise the names of the CDNI Logging Record types specified in Section 3.4 of the present document, and are as follows:

record-types	Reference	Description
cdni_http_request_v1	RFC xxxx	CDNI Logging Record version 1 for content delivery using HTTP

Figure 11

[Instructions to IANA: Replace "RFC xxxx" above by the RFC number of the present document]

Within the registry, record-types are to be allocated by IANA according to the "Specification Required" policy specified in [RFC5226]. Record-types are to be allocated by IANA with a format of

NAMEFORMAT (see Section 3.1). All record-types defined in the logging file are case-insensitive as per the basic ABNF([RFC5234]).

Each specification that defines a new record-type needs to contain a description for the new record-type with the same set of information as provided in Section 3.4.1. This includes:

- o a list of all the CDNI Logging fields that can appear in a CDNI Logging Record of the new record-type
- o for all these fields: a specification of the occurrence for each Field in the new record-type
- o for every newly defined Field, i.e., for every Field that results in a registration in the CDNI Logging Field Names Registry (Section 6.4): a specification of the field name, format and field value.

6.4. CDNI Logging Field Names Registry

The IANA is requested to create a new "CDNI Logging Field Names" subregistry under the "Content Delivery Networks Interconnection (CDNI) Parameters" registry.

This registry is intended to be shared across the currently defined record-type (i.e., `cdni_http_request_v1`) as well as potential other CDNI Logging record-types that may be defined in separate specifications. When a Field from this registry is used by another CDNI Logging record-type, it is to be used with the exact semantics and format specified in the document that registered this field and that is identified in the Reference column of the registry. If another CDNI Logging record-type requires a Field with semantics that are not strictly identical, or a format that is not strictly identical then this new Field is to be registered in the registry with a different Field name. When a Field from this registry is used by another CDNI Logging record-type, it can be used with different occurrence rules.

The initial contents of the CDNI Logging fields Names registry comprise the names of the CDNI Logging fields specified in Section 3.4 of the present document, and are as follows:

Field Name	Reference
date	RFC xxxx
time	RFC xxxx
time-taken	RFC xxxx
c-groupid	RFC xxxx
s-ip	RFC xxxx
s-hostname	RFC xxxx
s-port	RFC xxxx
cs-method	RFC xxxx
cs-uri	RFC xxxx
u-uri	RFC xxxx
protocol	RFC xxxx
sc-status	RFC xxxx
sc-total-bytes	RFC xxxx
sc-entity-bytes	RFC xxxx
cs(insert_HTTP_header_name_here)	RFC xxxx
sc(insert_HTTP_header_name_here)	RFC xxxx
s-ccid	RFC xxxx
s-sid	RFC xxxx
s-cached	RFC xxxx

Figure 12

[Instructions to IANA: Replace "RFC xxxx" above by the RFC number of the present document]

Within the registry, names are to be allocated by IANA according to the "Specification Required" policy specified in [RFC5226]. Field names are to be allocated by IANA with a format of NHTABSTRING (see Section 3.1). All field names defined in the logging file are case-insensitive as per the basic ABNF([RFC5234]).

6.5. CDNI Logging MIME Media Type

The IANA is requested to register the following new Payload Type in the CDNI Payload Type 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
logging-file	[RFCthis]

Figure 13: MIME Media Type payload

The purpose of the logging-file payload type is to distinguish between CDNI Logging Files and other CDNI messages.

Interface: LI

Encoding: see Section 3.2, Section 3.3 and Section 3.4

7. Security Considerations

7.1. Authentication, Authorization, Confidentiality, Integrity Protection

An implementation of the CDNI Logging interface MUST support TLS transport of the CDNI Logging feed (Section 4.1) and of the CDNI Logging File pull (Section 4.2) as per [RFC2818] and [RFC7230].

TLS MUST be used by the server-side and the client-side of the CDNI Logging feed, as well as the server-side and the client-side of the CDNI Logging File pull mechanism, including authentication of the remote end, unless alternate methods are used for ensuring the security of the information exchanged over the LI interface (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 Logging feed and CDNI Logging File pull allows:

- o the dCDN and 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 uCDN to authorize each other (to ensure they are transmitting/receiving CDNI Logging File to/from an authorized CDN)
- o the CDNI Logging information to be transmitted with confidentiality

- o the integrity of the CDNI Logging information to be protected during the exchange.

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

The SHA256-hash directive inside the CDNI Logging File provides additional integrity protection, this time targeting potential corruption of the CDNI logging information during the CDNI Logging File generation, storage or exchange. This mechanism does not itself allow restoration of the corrupted CDNI Logging information, but it allows detection of such corruption and therefore triggering of appropriate corrective actions (e.g., discard of corrupted information, attempt to re-obtain the CDNI Logging information). Note that the SHA256-hash does not protect against tampering by a third party, since such a third party could have recomputed and updated the SHA256-hash after tampering. Protection against third party tampering, when the CDNI Logging File is communicated over the CDN Logging Interface, can be achieved as discussed above through the use of TLS.

7.2. Denial of Service

This document does not define specific mechanism to protect against Denial of Service (DoS) attacks on the Logging Interface. However, the CDNI Logging feed and CDNI Logging pull endpoints are typically to be accessed only by a very small number of valid remote endpoints and therefore can be easily protected against DoS attacks through the usual conventional DOS protection mechanisms such as firewalling or use of Virtual Private Networks (VPNs).

Protection of dCDN Surrogates against spoofed delivery requests is outside the scope of the CDNI Logging interface.

7.3. Privacy

CDNs have the opportunity to collect detailed information about the downloads performed by End Users. A dCDN is expected to collect such information into CDNI Logging Files, which are then communicated to an uCDN.

Having detailed CDNI logging information known by the dCDN in itself does not represent a particular privacy concern since the dCDN is obviously fully aware of all information logged since it generated the information in the first place.

Transporting detailed CDNI logging information over the HTTP based CDNI Logging Interface does not represent a particular privacy

concern because it is protected by usual IETF privacy-protection mechanism (e.g., TLS).

When HTTP redirection is used between the uCDN and the dCDN, making detailed CDNI logging information known to the uCDN does not represent a particular privacy concern because the uCDN is already exposed at request redirection time to most of the information that shows up as CDNI logging information (e.g., enduser IP@, URL, HTTP headers). When DNS redirection is used between the uCDN and the dCDN, there are cases where there is no privacy concern in making detailed CDNI logging information known to the uCDN; this may be the case, for example, where (1) it is considered that because the uCDN has the authority (with respect to the CSP) and control on how the requests are delivered (including whether it is served by the uCDN itself or by a dCDN), the uCDN is entitled to access all detailed information related to the corresponding deliveries, and (2) there is no legal reasons to restrict access by the uCDN to all these detailed information. Conversely, still when DNS redirection is used between the uCDN and the dCDN, there are cases where there may be some privacy concern in making detailed CDNI logging information known to the uCDN; this may be the case, for example, because the uCDN is in a different jurisdiction to the dCDN resulting in some legal reasons to restrict access by the uCDN to all the detailed information related to the deliveries. In this latter case, the privacy concern can be taken into account when the uCDN and dCDN agree about which fields are to be conveyed inside the CDNI Logging Files and which privacy protection mechanism is to be used as discussed in the definition of the c-groupid field specified in Section 3.4.1.

Another privacy concern arises from the fact that large volumes of detailed information about content delivery to users, potentially traceable back to individual users, may be collected in CDNI Logging files. These CDNI Logging files represent high-value targets, likely concentrated in a fairly centralised system (although the CDNI Logging architecture does not mandate a particular level of centralisation/distribution) and at risk of potential data exfiltration. Note that the means of such data exfiltration are beyond the scope of the CDNI Logging interface itself (e.g., corrupted employee, corrupted logging storage system,...). This privacy concern calls for some protection.

The collection of large volumes of such information into CDNI Logging Files introduces potential End Users privacy protection concerns. Mechanisms to address these concerns are discussed in the definition of the c-groupid field specified in Section 3.4.1.

The use of mutually authenticated TLS to establish a secure session for the transport of the CDNI Logging feed and CDNI Logging pull as

discussed in Section 7.1 provides confidentiality while the logging information is in transit and prevents any party other than the authorised uCDN to gain access to the logging information.

We also note that the query string portion of the URL that may be conveyed inside the cs-uri and u-uri fields of CDNI Logging Files, or the HTTP cookies([RFC6265]) that may be conveyed as part of the cs(<HTTP-header-name>) field of CDNI Logging files, may contain personal information or information that can be exploited to derive personal information. Where this is a concern, the CDNI Logging interface specification allows the dCDN to not include the cs-uri and to include a u-uri that removes (or hides) the sensitive part of the query string and allows the dCDN to not include the cs(<HTTP-header-name>) fields corresponding to HTTP headers associated with cookies.

8. Acknowledgments

This document borrows from the W3C Extended Log Format [ELF].

Rob Murray significantly contributed into the text of Section 4.1.

The authors thank Ben Niven-Jenkins, Kevin Ma, David Mandelberg and Ray van Brandenburg for their ongoing input.

Brian Trammel and Rich Salz made significant contributions into making this interface privacy-friendly.

Finally, we also thank Sebastien Cubaud, Pawel Grochocki, Christian Jacquenet, Yannick Le Louedec, Anne Marrec, Emile Stephan, Fabio Costa, Sara Oueslati, Yvan Massot, Renaud Edel, Joel Favier and the contributors of the EU FP7 OCEAN project for their input in the early versions of this document.

9. References

9.1. Normative References

- [AES] NIST, "Advanced Encryption Standard (AES)", August 2015, <FIPS 197>.
- [GCM] NIST, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", November 2007, <SP 800-38D>.
- [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>>.

- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<http://www.rfc-editor.org/info/rfc3339>>.
- [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>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<http://www.rfc-editor.org/info/rfc4122>>.
- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <<http://www.rfc-editor.org/info/rfc4287>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.
- [RFC5005] Nottingham, M., "Feed Paging and Archiving", RFC 5005, DOI 10.17487/RFC5005, September 2007, <<http://www.rfc-editor.org/info/rfc5005>>.
- [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>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [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>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<http://www.rfc-editor.org/info/rfc7231>>.

- [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <<http://www.rfc-editor.org/info/rfc7232>>.
- [RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <<http://www.rfc-editor.org/info/rfc7233>>.
- [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>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<http://www.rfc-editor.org/info/rfc7235>>.
- [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>>.
- [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>>.
- [SHA-3] NIST, "SHA-3 STANDARD: PERMUTATION-BASED HASH AND EXTENDABLE OUTPUT FUNCTIONS", November 2001, <<http://dx.doi.org/10.6028/NIST.FIPS.202>>.

9.2. Informative References

- [CHAR_SET] "IANA Character Sets registry", <<http://www.iana.org/assignments/character-sets/character-sets.xml>>.
- [ELF] Phillip M. Hallam-Baker, and Brian Behlendorf, "Extended Log File Format, W3C (work in progress), WD-logfile-960323", <<http://www.w3.org/TR/WD-logfile.html>>.

- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma,
"CDN Interconnection Metadata", draft-ietf-cdni-
metadata-17 (work in progress), May 2016.
- [I-D.ietf-tls-rfc5246-bis]
Dierks, T. and E. Rescorla, "The Transport Layer Security
(TLS) Protocol Version 1.3", draft-ietf-tls-rfc5246-bis-00
(work in progress), April 2014.
- [I-D.snell-atompub-link-extensions]
Snell, J., "Atom Link Extensions", draft-snell-atompub-
link-extensions-09 (work in progress), June 2012.
- [RFC1945] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext
Transfer Protocol -- HTTP/1.0", RFC 1945,
DOI 10.17487/RFC1945, May 1996,
<<http://www.rfc-editor.org/info/rfc1945>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818,
DOI 10.17487/RFC2818, May 2000,
<<http://www.rfc-editor.org/info/rfc2818>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand
Key Derivation Function (HKDF)", RFC 5869,
DOI 10.17487/RFC5869, May 2010,
<<http://www.rfc-editor.org/info/rfc5869>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms
(SHA and SHA-based HMAC and HKDF)", RFC 6234,
DOI 10.17487/RFC6234, May 2011,
<<http://www.rfc-editor.org/info/rfc6234>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265,
DOI 10.17487/RFC6265, April 2011,
<<http://www.rfc-editor.org/info/rfc6265>>.
- [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>>.

- [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, <<http://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, <<http://www.rfc-editor.org/info/rfc7336>>.
- [RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <<http://www.rfc-editor.org/info/rfc7337>>.
- [RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <<http://www.rfc-editor.org/info/rfc7736>>.

Authors' Addresses

Francois Le Faucheur (editor)
FR

Phone: +33 6 19 98 50 90
Email: flefauch@gmail.com

Gilles Bertrand (editor)

Phone: +41 76 675 91 44
Email: gilbertrand@gmail.com

Iuniana Oprescu (editor)
FR

Email: iuniana.oprescu@gmail.com

Roy Peterkofsky
Google Inc.
345 Spear St, 4th Floor
San Francisco CA 94105
USA

Email: peterkofsky@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 29, 2013

B. Niven-Jenkins
R. Murray
G. Watson
Velocix (Alcatel-Lucent)
M. Caulfield
K. Leung
Cisco Systems
K. Ma
Azuki Systems, Inc.
February 25, 2013

CDN Interconnect Metadata
draft-ietf-cdni-metadata-01

Abstract

The CDNI Metadata Interface enables interconnected CDNs to exchange content distribution metadata in order to enable content acquisition and delivery. The CDNI metadata associated with a piece of content provides a downstream CDN with sufficient information for the downstream CDN to service content requests on behalf of an upstream CDN. This document describes both the core set of CDNI metadata and the protocol for exchanging that metadata.

Requirements Language

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

Status of this Memo

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

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

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

This Internet-Draft will expire on August 29, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
1.1. Terminology	4
2. Design Principles	4
3. CDNI Metadata Data Model	5
3.1. HostIndex, HostMetadata & PathMetadata objects	6
3.2. Generic CDNI Metadata Object Properties	9
3.3. Metadata Inheritance and Override	10
3.4. Metadata Naming	11
4. Encoding-Independent CDNI Metadata Object Descriptions	11
4.1. CDNI Metadata Structural Object Descriptions	12
4.1.1. HostIndex	12
4.1.2. HostMatch	12
4.1.3. HostMetadata	12
4.1.4. PathMatch	13
4.1.5. PathMetadata	13
4.1.6. PatternMatch	14
4.1.7. GenericMetadata	14
4.2. CDNI Metadata Property Object Descriptions	15
4.2.1. Source Metadata	15
4.2.1.1. Source	15
4.2.2. LocationACL Metadata	15
4.2.2.1. LocationRule	16
4.2.2.2. Location	16
4.2.3. TimeWindowACL Metadata	16
4.2.3.1. TimeWindowRule	17
4.2.3.2. TimeWindow	17
4.2.4. ProtocolACL Metadata	17
4.2.4.1. ProtocolRule	17
4.2.5. Authorization Metadata	18
4.2.6. Auth	18

4.2.7. Cache	19
4.2.8. Logging	19
4.3. CDNI Metadata Simple Data Type Descriptions	19
4.3.1. Link	19
4.3.2. Protocol	19
4.3.3. RedirectionMethod	20
4.3.4. Endpoint	20
4.3.5. IPRange	20
4.3.6. URI	20
4.3.7. Time	20
5. CDNI Metadata Capabilities	21
5.1. Protocol ACL Capabilities	21
5.2. Authorization Metadata Capabilities	22
5.3. Host Metadata Capabilities	22
6. CDNI Metadata interface	22
6.1. Transport	23
6.2. Retrieval of CDNI Metadata resources	23
6.3. Bootstrapping	24
6.4. Encoding	25
6.4.1. MIME Media Types	25
6.4.2. JSON Encoding of Objects	26
6.4.2.1. JSON Example	26
6.4.3. XML Encoding of Objects	30
6.4.3.1. XML Example	30
6.5. Extensibility	33
6.5.1. Metadata Enforcement	33
6.5.2. Metadata Override	34
6.6. Versioning	34
7. IANA Considerations	34
8. Security Considerations	35
9. Acknowledgements	35
10. References	35
10.1. Normative References	35
10.2. Informative References	36
Appendix A. Relationship to the CDNI Requirements	36
Appendix B. Metadata Rewriting	37
B.1. Example	38
Authors' Addresses	38

1. Introduction

CDNI enables a downstream CDN to service content requests on behalf of an upstream CDN. The CDNI metadata associated with a piece of content (or with a set of contents) provides a downstream CDN with sufficient information for servicing content requests on behalf of an upstream CDN in accordance with the policies defined by the upstream CDN.

The CDNI Metadata Interface is introduced by [RFC6707] along with three other interfaces that may be used to compose a CDNI solution (Control, Request Routing and Logging). [I-D.ietf-cdni-framework] expands on the information provided in [RFC6707] and describes each interface, and the relationships between them, in more detail. The requirements for the CDNI metadata interface are specified in [I-D.ietf-cdni-requirements].

This document focuses on the CDNI Metadata interface which enables a downstream CDN to obtain CDNI Metadata from an upstream CDN so that the downstream CDN can properly process and respond to:

- o Redirection Requests received over the CDNI Request Routing protocol.
- o Content Requests received directly from User Agents.

Specifically this document proposes:

- o A data structure for mapping content requests to CDNI Metadata properties (Section 3).
- o An initial set of CDNI Metadata properties (Section 4.2).
- o A RESTful web service for the transfer of CDNI Metadata (Section 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.

2. Design Principles

The proposed CDNI Metadata Interface was designed to achieve the following objectives:

1. Cacheability of CDNI metadata objects
2. Deterministic mapping from redirection 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. Leverage existing protocols

Cacheability improves the latency of acquiring metadata while maintaining its freshness and therefore improves the latency of serving content requests. The CDNI Metadata Interface uses HTTP to achieve 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. XML, 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, authorize, and deliver content from a downstream CDN. The data model relies on the assumption that these metadata properties may be aggregated based on the hostname of the content and subsequently on the resource path of the content. The data model associates a set of CDNI Metadata properties with a Hostname to form a default set of metadata properties for content delivered for that Hostname. That default set of metadata properties can be overridden by properties that apply to specific paths within a URI.

Different Hostnames and URI paths will 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 by a dCDN once, even if it is referenced by the CDNI Metadata of multiple Hostnames.

Section 3.1 introduces a high level description of the HostIndex, HostMetadata and PathMetadata objects and describes the relationships between those objects.

Section 3.2 introduces a high level description of the CDNI GenericMetadata object which represents the level at which CDNI Metadata override occurs between HostMetadata and PathMetadata objects.

Section 4 describes in detail the specific CDNI Metadata objects and properties which may be contained within a CDNI GenericMetadata object.

3.1. HostIndex, HostMetadata & PathMetadata objects

A HostIndex object contains a list of Hostnames (and/or IP addresses) for which content requests may be delegated to the downstream CDN. The HostIndex is the starting point for accessing the uCDN's CDNI Metadata data store. It enables surrogates in the dCDN to deterministically discover, on receipt of a User Agent request for content, which other CDNI Metadata objects it requires in order to deliver the requested content.

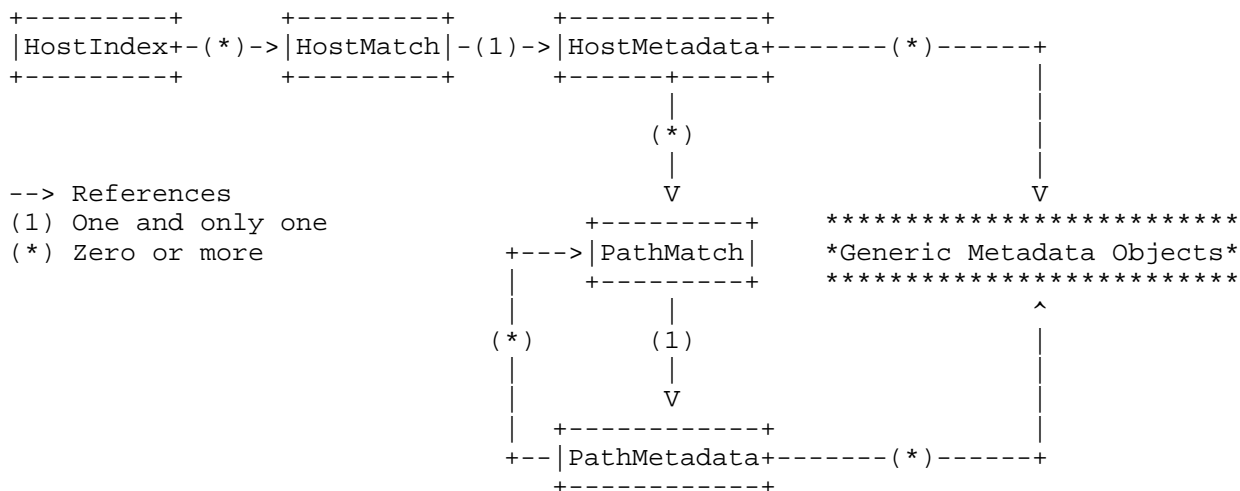
The HostIndex links Hostnames (and/or IP addresses) to HostMetadata objects via HostMatch objects. HostMetadata objects contain (or reference) the default CDNI Metadata required to serve content for that host. When looking up CDNI Metadata, the downstream CDN looks

up the requested Hostname (or IP address) in the HostIndex, from there it can find HostMetadata which describes properties for a host and PathMetadata which may override those properties for given URI paths within the host.

As well as containing the default CDNI Metadata for the specified Hostname, HostMetadata and PathMetadata objects may also contain PathMatch objects which in turn contain PathMetadata objects. PathMatch objects override the CDNI Metadata in the HostMetadata object or one or more preceding PathMetadata objects with more specific CDNI Metadata that applies to content requests matching the pattern defined in that PathMatch object.

For the purposes of retrieving CDNI Metadata all other required CDNI Metadata objects and their properties are discoverable from the appropriate HostMetadata, PathMatch and PathMetadata objects for the requested content.

The relationships between the HostIndex, HostMatch, HostMetadata, PathMatch and PathMetadata objects are described in Figure 1.



Key: ----> = References

Figure 1: Relationships between the HostIndex, HostMetadata & PathMetadata CDNI Metadata Objects

The relationships in Figure 1 are summarised in Table 1 below.

Data Object	Objects it References
HostIndex	0 or more HostMatch objects.
HostMatch	1 HostMetadata object.
HostMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.
PathMatch	1 PathMetadata object.
PathMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.

Table 1: Relationships between CDNI Metadata Objects

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

Data Object	Description
HostIndex	A HostIndex object lists HostMatch objects
HostMatch	A HostMatch object defines a hostname to match against a requested host, and contains or references a HostMetadata object which contains CDNI Metadata objects to be applied when a 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 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.
PathMetadata	A PathMetadata object contains the CDNI GenericMetadata objects for content served with the associated URI path (defined in a PathMatch object). A PathMetadata object may also contain PathMatch objects in order to recursively define more specific URI paths that require different (e.g. more specific) CDNI Metadata to this one. For example, the PathMetadata object which applies to "example.com/movies/*" may describe CDNI Metadata which apply to that resource path and may contain a PathMatch object for "example.com/movies/hd/*" which would reference the corresponding PathMetadata object for the "example.com/movies/hd/" path prefix.
GenericMetadata	A GenericMetadata object contains individual CDNI Metadata objects which define the specific policies and attributes needed to properly deliver the associated content.

Table 2: HostIndex, HostMetadata and PathMetadata CDNI Metadata Objects

3.2. Generic CDNI Metadata Object Properties

The HostMetadata and PathMetadata objects contain or can reference other CDNI Metadata objects that contain properties which describe how User Agent requests for content should be processed, for example where to acquire the content, authorization rules that should be applied, delivery location restrictions and so on. Each such CDNI Metadata object is a specialization of a CDNI GenericMetadata object. The GenericMetadata object abstracts the basic information required for Metadata override and opaque Metadata distribution, from the specifics of any given property (e.g., property semantics, enforcement options, etc.).

The GenericMetadata object defines the type of properties contained within it as well as whether or not the properties are mandatory to enforce. If the dCDN does not understand or support the property type and the property type is mandatory to enforce, the dCDN MUST NOT serve the content to the User Agent. If the dCDN does not understand

or support the property type it is also not going to be able to properly propagate the Metadata for cascaded distribution. If the dCDN does not understand or support the property type and the property type is not mandatory to enforce, then the GenericMetadata object may be safely ignored.

Although a CDN cannot serve content to a User Agent if a mandatory 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 the delivery CDN. For Metadata which does not require customization, the data representation received off the wire MAY be stored and redistributed without being natively understood or supported by the transit CDN. However, for Metadata which require translations, transparent redistribution of the uCDN Metadata values may not be appropriate. Certain Metadata may be safely, though possibly not optimially, redistributed unmodified, e.g., source acquisition address may not be optimal if transparently redistributed, but may still work. Redistribution safety MUST be specified for each GenericMetadata.

3.3. Metadata Inheritance 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:

- the TimeWindowACL defined in the PathMetadata would override the TimeWindowACL defined in the HostMetadata

- the LocationACL defined in the HostMetadata would be inherited for all User Agent requests for content under "example.com/movies".

The PathMetadata defined TimeWindowACL would override the TimeWindowACL defined in the HostMetadata for all User Agent requests for movies.

3.4. Metadata Naming

GenericMetadata objects are identified by their type. The type SHOULD be descriptive, and MAY be hierarchical to support aggregating groups of properties for the purpose of readability and for avoiding name conflicts between vendor extensions. A dotted alpha-numeric notation is suggested for human readability.

Metadata types defined by this document are not hierarchical.

Examples of GenericMetadata object type names:

```
LocationACL
ext.vendor1.featurex
ext.vendor1.featurey
ext.vendor2.featurex
```

[Ed. It is intended that Metadata capability advertisements will allow either individual Metadata names or Metadata bundle identifiers to be used. Need to have a procedure for defining and distributing bundle information to be used in Metadata capability advertisement.]

4. Encoding-Independent CDNI Metadata Object Descriptions

Section 4.1 provides the definitions of each object type declared in Section 3. These objects are described as structural objects as they provide the structure for the inheritance tree and identifying which specific properties apply to a given User Agent content request.

Section 4.2 provides the definitions for the set of core metadata objects which may be contained within a GenericMetadata object. These objects are described as property objects as they define the semantics, enforcement options, and serialization rules for specific properties. These properties govern how User Agent requests for content are handled. Property objects may be composed of or contain references to other objects. In those cases the value of the property can be either an object of that type (the object is embedded) or a Link object that contains a URI and relationship that can be dereferenced to retrieve the CDNI Metadata object that represents the value of that property.

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

object.

4.1. CDNI Metadata Structural Object Descriptions

Each of the sub-sections below describe the structural objects defined in Table 2.

4.1.1. HostIndex

The HostIndex object is the entry point into the CDNI Metadata hierarchy. It contains a list of HostMatch objects. An incoming content request is matched against the hostname inside of 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 a reference to Metadata objects to apply if a match is found.

Property: host

Description: String (hostname or IP address) to match against the requested host.

Type: String

Mandatory-to-Specify: Yes.

Property: host-metadata

Description: CDNI Metadata to apply when delivering content that matches this host.

Type: HostMetadata

Mandatory-to-Specify: Yes.

4.1.3. HostMetadata

The HostMetadata object contains both Metadata that applies to content requests for a particular host and a list of pattern matches for finding more specific Metadata based on the resource path in a content request.

Property: metadata

Description: List of host related metadata.

Type: List of GenericMetadata objects
Mandatory-to-Specify: Yes.
Property: paths
Description: Path specific rules. First match applies.
Type: List of PathMatch objects
Mandatory-to-Specify: No.
Property: modes
Description: Defines which redirection methods are supported.
Type: List of RedirectionMethod
Mandatory-to-Specify: Yes.

4.1.4. PathMatch

The PathMatch object contains an expression given as a PatternMatch object to match against a resource URI path and Metadata objects to apply if a match is found.

Property: path-pattern
Description: Pattern to match against the requested path, i.e. against the [RFC3986] path-absolute.
Type: PatternMatch
Mandatory-to-Specify: Yes.
Property: path-metadata
Description: CDNI Metadata to apply when delivering content that matches this pattern.
Type: PathMetadata
Mandatory-to-Specify: Yes.

4.1.5. PathMetadata

A PathMetadata object contains the CDNI Metadata properties for content served with the associated URI path (defined in a PathMatch object). Note that if CDNI metadata is used as an input to CDNI request routing and DNS-based redirection is employed, then any metadata at the PathMetadata level or below will be inaccessible at request routing time.

Property: metadata
Description: List of path related metadata.
Type: List of GenericMetadata 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 \?
Type: String
Mandatory-to-Specify: Yes.

Property: case-sensitive
Description: Flag indicating whether or not case-sensitive matching should be used.
Type: Boolean
Mandatory-to-Specify: No. Default is case-insensitive match.

Property: match-query-string
Description: Flag indicating whether or not the query string should be included in the pattern match.
Type: Boolean
Mandatory-to-Specify: No. Default is not to include query strings when matching.

4.1.7. GenericMetadata

A GenericMetadata object is a abstraction for managing individual CDNI Metadata properties in an opaque manner.

Property: type
Description: CDNI Metadata property object type.
Type: String
Mandatory-to-Specify: Yes.

Property: value
Description: CDNI Metadata property object.
Type: matches the type property above
Mandatory-to-Specify: Yes.

Property: mandatory-to-enforce
Description: Flag identifying whether or not the enforcement of the property Metadata is required.
Type: Boolean
Mandatory-to-Specify: Yes.

Property: safe-to-redistribute
Description: Flag identifying whether or not the property Metadata may be safely redistributed without modification.

Type: Boolean
Mandatory-to-Specify: No. Default is allow transparent redistribution.

4.2. CDNI Metadata Property Object Descriptions

4.2.1. Source Metadata

Source Metadata provides the dCDN information about content acquisition e.g. how to contact an uCDN Surrogate or an Origin Server 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 priority order.
Type: List of Source objects
Mandatory-to-Specify: No. Default is to use static configuration, out of band of 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: auth
Description: Authentication method to use when requesting content from this source.
Type: Auth
Mandatory-to-Specify: No. Default is no authentication is required.
Property: endpoints
Description: Origins from which the dCDN can acquire content.
Type: List of EndPoint objects
Mandatory-to-Specify: Yes.

4.2.2. LocationACL Metadata

LocationACL Metadata defines location-based restrictions.

Property: locations
Description: Access control list which applies restrictions to delivery based on client location.

Type: List of LocationRule objects

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

4.2.2.1. LocationRule

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

Property: locations

Description: List of locations to which the rule applies.

Type: List of Location objects

Mandatory-to-Specify: Yes.

[Ed: reusing locations as a property name is confusing and should likely be changed]

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. Location

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

Property: iprange

Description: A set of IP Addresses.

Type: List of IPRange objects

Mandatory-to-Specify: Yes.

[Ed: Location as specified above only supports the Class 1a names described in [I-D.jenkins-cdni-names]. Need to add support for Class 1b names to a later version.]

4.2.3. TimeWindowACL Metadata

TimeWindowACL Metadata defines time-based restrictions.

Property: times

Description: Access control list which applies restrictions to delivery based on request time.

Type: List of TimeWindowRule 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: times

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 ACLRule, e.g. Start 09:00AM 01/01/2000 UTC End 17:00PM 01/01/2000 UTC.

Property: start

Description: The start time of the window.

Type: Time

Mandatory-to-Specify: Yes.

Property: end

Description: The end time of the window.

Type: Time

Mandatory-to-Specify: Yes.

4.2.4. ProtocolACL Metadata

ProtocolACL Metadata defines delivery protocol restrictions.

Property: protocols

Description: Access control list which applies restrictions to delivery based on delivery protocol.

Type: List of ProtocolRule 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 allow all protocols.

Property: direction
Description: Defines whether the ProtocolRule specifies protocols for acquisition or delivery.
Type: Enumeration [acquisition|delivery]
Mandatory-to-Specify: No. Default is to apply the rule to both acquisition and delivery.

4.2.5. Authorization Metadata

Authorization Metadata define content authorization methods.

Property: methods
Description: Options for authenticating content requests. All options in the list are equally valid.
Type: List of Auth objects
Mandatory-to-Specify: No. Default is no authorization required.

4.2.6. Auth

An Auth object defines authentication and authorization methods to be used during content delivery and content acquisition, e.g. methods such as tokenization and URL Signing.

[Ed. Need to synchronize authentication configuration with CDNI URL signing draft definitions.]

[Ed. Need to consider how to separate protocol specific method configuration (e.g., HTTP basic/digest authentication), which must match the HostMatch protocol, from protocol agnostic method configurations (e.g., URL signing/tokenization).]

Property: direction
Description: Defines whether the Auth object applies to acquisition or delivery requests.
Type: Enumeration [acquisition|delivery]
Mandatory-to-Specify: No. Default is to apply the rule to both acquisition and delivery.

4.2.7. Cache

[Ed. note: placeholder for cache control metadata - TBD]

4.2.8. Logging

[Ed. note: placeholder for logging metadata - TBD]

4.3. CDNI Metadata Simple Data Type Descriptions

This section describes the simpler data types that are used for properties of CDNI Metadata objects.

4.3.1. Link

A link object may be used in place of any of the objects 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, its rel property is set to the name of the property it is replacing, and its type property is set to the type of the object it is replacing.

Property: href

Description: The URI of the of the addressable object being referenced.

Type: URI

Mandatory: Yes

Property: rel

Description: The Relationship between the referring object and the object it is referencing.

Type: String

Mandatory: Yes

Property: type

Description: The type of the object being referenced.

Type: String

Mandatory: Yes

4.3.2. Protocol

Protocol objects are used to specify registered protocols for content acquisition or delivery.

[Ed. Need to reference protocol registry.]

Type: Enumeration [HTTP|RTSP|RTMP]

4.3.3. RedirectionMethod

RedirectionMethod objects are used to specify registered content redirection modes.

[Ed. Need to reference redirection method registry.]

Type: Enumeration [HTTP-I|HTTP-R|DNS-I|DNS-R]

4.3.4. Endpoint

A hostname (with optional port) or an IP address (with optional port).

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

4.3.5. IPRange

One of:

- o A range of consecutive IP addresses (IPv4 or IPv6) expressed as Address1-Address2 which does not have to be to power of two aligned, for example the range 192.0.2.1-192.0.2.10 is valid. The first Address in the range MUST be 'lower' than the final address in the range.
- o A valid IP subnet (IPv4 or IPv6) expressed using CIDR notation.
- o A single IP address (IPv4 or IPv6).

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

4.3.6. URI

A URI as specified in [RFC3986].

4.3.7. Time

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

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, so there is no reason to delegate those requests. Likewise, for any metadata which may be assigned optional values, it may be useful for the uCDN to know which values the dCDN supports, prior to delegating any content requests to the dCDN. If a 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 there is likely no reason to delegate those requests.

The CDNI Footprint and Capabilities Interface 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 capabilities interface. This section describes the capabilities advertisement requirements for the metadata defined in Section 4.2

5.1. Protocol ACL Capabilities

The ProtocolACL object contains a list of Protocol values. The dCDN MUST advertise which delivery protocols it supports so that the uCDN knows what type of content requests it can redirect to the dCDN. If the dCDN does not support a given acquisition or delivery protocol, the uCDN should not delegate requests requiring those protocols to the dCDN as the dCDN will not be able to properly acquire or deliver the content.

ProtocolRules are defined for either acquisition or delivery. For some CDNs, certain combinations of acquisition and delivery protocols may not make sense (e.g., RTSP acquisition for HTTP delivery), while other CDNs may support customized protocol adaptation. ProtocolACL capabilities are not intended to define which combinations of protocols should be used. ProtocolACL capabilities are only intended to describe which protocols the dCDN does or does not support. Protocol combination restrictions are specified in the metadata itself and associated with specific groups of content assets.

[Ed. Need to register delivery protocol capability ID.]

[Ed. Need to reference protocol registry, and discuss specification of overlapping protocol values.]

5.2. Authorization Metadata Capabilities

The Authorization object contains a list of Auth values. The dCDN MUST advertise which authorization algorithms it supports so that the uCDN knows what type of content requests it can redirect to the dCDN. If the dCDN does not support a given authorization algorithm, the uCDN should not delegate requests requiring that algorithm to the dCDN as the dCDN will not be able to properly acquire the content or enforce delivery restrictions.

[Ed. Need to register authorization algorithm capability ID.]

[Ed. Need to reference auth registry, and discuss specification of overlapping auth values.]

5.3. Host Metadata Capabilities

The HostMetadata object contains a list of redirection method values. The dCDN MUST advertise which redirection modes it supports so that the uCDN knows how to redirect content requests to the dCDN. If the dCDN does not support a given redirection method, the uCDN should not delegate requests to the dCDN using that method as the dCDN will not be able to properly handle the redirection.

[Ed. Need to register redirection method capability ID.]

[Ed. Need to reference redirection method registry.]

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 dynamically as required by the Downstream CDN to process received requests (for example in response to receiving a CDNI Request Routing request from an Upstream CDN or in response to receiving a request for content from a User Agent) or in advance of being required (for example in case of prepositioned CDNI Metadata acquisition).

The CDNI Metadata interface is built on the principles of RESTful web services. This means that requests and responses over the interface are built around the transfer of representations of hyperlinked resources. A resource in the context of the CDNI Metadata interface is any object in the Data Model (as described in Section 3 through Section 4).

In the general case a CDNI Metadata server makes each instance of an addressable CDNI Metadata object available via a unique URI that returns a representation of that instance of that CDNI Metadata object. When an object needs to reference another addressable CDNI Metadata object (for example a HostIndex object referencing a HostMetadata object) it does so by including a link to the referenced object.

CDNI Metadata servers are free to assign whatever structure they desire to the URIs for CDNI Metadata objects and CDNI Metadata clients MUST NOT make any assumptions regarding the structure of CDNI Metadata URIs or the mapping between CDNI Metadata objects and their associated URIs. Therefore any URIs present in the examples below are purely illustrative and are not intended 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. Servers implementing the CDNI Metadata interface MUST support the HTTP GET and HEAD methods.

The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses from servers implementing the CDNI Metadata interface that contain a response body SHOULD include an ETag to enable validation of cached versions of returned resources.

The CDNI Metadata interface specified in this document is a read-only interface. Therefore support for other HTTP methods such as PUT, POST and DELETE etc. is not specified. Server implementations of this interface SHOULD reject all methods other than GET and HEAD.

As the CDNI Metadata interface builds on top of HTTP, CDNI Metadata servers may make use of any HTTP feature when implementing the CDNI Metadata interface, for example a CDNI Metadata server may make use of HTTP's caching mechanisms to indicate that the returned response/representation can be reused without re-contacting the CDNI Metadata server.

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 must decide which upstream CDN's CDNI metadata should be used to handle a particular User Agent request.

When application level redirection (e.g. HTTP 302 redirects) is being used between CDNs, it is expected that the downstream CDN will be able to determine the upstream CDN that redirected a particular request from information contained in the received request (e.g. via the URI). 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 discovered by or configured in the downstream CDN. All other

objects/resources are then discoverable from the HostIndex object by following the links in the HostIndex object and the referenced HostMetadata and PathMetadata objects.

If the URI for the HostIndex object is not manually configured in the downstream CDN then the HostIndex URI could be discovered. A mechanism allowing the downstream CDN to discover the URI of the HostIndex is outside the scope of this document.

6.4. Encoding

Object are resources that may be:

- o Addressable, where the object is a resource that may be retrieved or referenced via its own URI.
- o Embedded, where the object is contained (or inlined) within a property of an addressable object.

In the descriptions of objects we use the term "X contains Y" to mean either Y is directly embedded in X or that Y is linked to by X. It is generally a deployment choice for the uCDN implementation to decide when and which CDNI Metadata objects to embed and which are separately addressable.

6.4.1. MIME Media Types

All MIME types are prefixed with "application/cdni." The MIME type for each object matches the type name of that object as defined by this document. Table 3 lists a few examples of the MIME Media Type for each object (resource) that is retrievable through the CDNI Metadata interface. The MIME type suffix depends on the metadata encoding, either "+xml" or "+json".

Data Object	MIME Media Type
HostIndex	application/cdni.HostIndex
HostMatch	application/cdni.HostMatch
HostMetadata	application/cdni.HostMetadata
PathMatch	application/cdni.PathMatch
PathMetadata	application/cdni.PathMetadata

Table 3: Example MIME Media Types for CDNI Metadata objects

See <http://www.iana.org/assignments/media-types/index.html> for reference.

6.4.2. JSON Encoding of Objects

One possible encoding for a CDNI Metadata object is a JSON object containing a dictionary of (key,value) pairs where the keys are the property names and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource). Likewise, the values associated with each key are dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource).

Dictionary keys in JSON are case sensitive and therefore 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 specific to each object type, the keys defined below may be present in any object.

Key: base

Description: Provides a prefix for any relative URLs in the object. This is similar to the XML base tag [XML-BASE]. If absent, all URLs in the remainder of the document must be absolute URLs.

Type: URI

Mandatory: No

Key: links

Description: The links of this object to other addressable objects. Any property may be replaced by a link to an object with the same type as the property it replaces.

Type: List of Link objects

Mandatory: Yes

6.4.2.1. JSON Example

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

```

{
  "hosts": [
    {
      "host": "video.example.com",
      "links": [
        {
          "rel": "host-metadata",
          "type": "application/cdni.HostMetadata",
          "href": "http://metadata.ucdn.example.com/video"
        }
      ]
    },
    {
      "host": "images.example.com",
      "links": [
        {
          "rel": "host-metadata",
          "type": "application/cdni.HostMetadata",
          "href": "http://metadata.ucdn.example.com/images"
        }
      ]
    }
  ]
}

```

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

```

{
  "metadata": [
    {
      "type": "application/cdni.SourceMetadata",
      "value": {
        "sources": [
          {
            "links": [{
              "rel": "auth",
              "type": "application/cdni.Auth",
              "href": "http://metadata.ucdn.example.com/auth1234"
            }],
            "endpoint": "acql.ucdn.example.com",
            "protocol": "ftp"
          },
          {
            "links": [{
              "rel": "auth",

```

```
        "type": "application/cdni.Auth",
        "href": "http://metadata.ucdn.example.com/auth1234"
      }],
      "endpoint": "acq2.ucdn.example.com",
      "protocol": "http"
    }
  ]
}
},
{
  "type": "application/cdni.LocationACL",
  "value": {
    "locations": [
      {
        "locations": [
          { "iprange": "192.168.0.0/16" }
        ],
        "action": "deny"
      }
    ]
  }
},
{
  "type": "application/cdni.ProtocolACL",
  "value": {
    "protocols": [
      {
        "protocols": [
          "ftp"
        ],
        "action": "deny"
      }
    ]
  }
},
],
"paths": [
  {
    "path-pattern": {
      "pattern": "/videos/trailers/*"
    },
    "links": [{
      "rel": "path-metadata",
      "type": "application/cdni.PathMetadata",
      "href": "http://metadata.ucdn.example.com/videos/trailers"
    }]
  }
],
{
```

```

    "path-pattern": {
      "pattern": "/videos/movies/*"
    },
    "links": [{
      "rel": "pathmetadata",
      "type": "application/cdni.PathMetadata",
      "href": "http://metadata.ucdn.example.com/videos/movies"
    }]
  }
]
}

```

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

```

{
  "metadata": [],
  "paths": [
    {
      "path-pattern": {
        "pattern": "/videos/movies/hd/*"
      },
      "links": [{
        "rel": "pathmetadata",
        "type": "application/cdni.PathMetadata",
        "href": "http://metadata.ucdn.example.com/videos/movies/hd"
      }]
    }
  ]
}

```

Finally, if the path of the requested resource also matches the `"/videos/movies/hd/*"` pattern, the downstream CDN would also fetch the following object from `"http://metadata.ucdn.example.com/videos/movies/hd"` with MIME type `"application/cdni.PathMetadata"`:

```
{
  "metadata": [
    {
      "type": "application/cdni.TimeWindowACL",
      "value": {
        "times": [
          {
            "start": "1213948800",
            "end": "1327393200"
          }
        ],
        "type": "allow"
      }
    }
  ]
}
```

6.4.3. XML Encoding of Objects

Another possible encoding for a CDNI Metadata object is an XML document containing elements with tag names which match property names and values which match the associated property values.

Tag names of elements are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource). Likewise, the values associated with each element are dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource).

Lists are encoded by repeating the singular form of a property name. For example the "hosts" property is a list of "HostMatch" objects. This list would be encoded as multiple "host" elements.

Link objects are a special case. If a Link object replaces a property then a "link" element replaces the expected element. The properties of the Link object are encoded as XML attributes. The type attribute is set to the MIME type of the target object. The href attribute is set to the URI of the target object. The rel attribute is set to the name of the element being replaced.

6.4.3.1. XML Example

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


```
<HostIndex>
  <host>
    <host>video.example.com</host>
    <link rel="host-metadata" type="application/cdni.HostMetadata"
      href="http://metadata.ucdn.example.com/video"/>
  </host>
  <host>
    <host>images.example.com</host>
    <link rel="host-metadata" type="application/cdni.HostMetadata"
      href="http://metadata.ucdn.example.com/images"/>
  </host>
</HostIndex>
```

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

```
<HostMetadata>
  <metadata>
    <type>application/cdni.SourceMetadata</type>
    <value>
      <sources>
        <link rel="auth" type="application/cdni.Auth"
          href="http://metadata.ucdn.example.com/auth1234"/>
        <endpoint>acq1.ucdn.example.com</endpoint>
        <protocol>ftp</protocol>
      </source>
      <source>
        <link rel="auth" type="application/cdni.Auth"
          href="http://metadata.ucdn.example.com/auth1234"/>
        <endpoint>acq2.ucdn.example.com</endpoint>
        <protocol>http</protocol>
      </source>
    </value>
  </metadata>
  <metadata>
    <type>application/cdni.LocationACL</type>
    <value>
      <location>
        <location>
          <iprange>192.168.0.0/16</iprange>
        </location>
        <action>deny</type>
      </location>
    </value>
  </metadata>
</metadata>
```

```

    <type>application/cdni.ProtocolACL</type>
    <value>
      <protocol>
        <protocol>ftp</protocol>
        <action>deny</action>
      </protocol>
    </value>
  </metadata>
  <path>
    <path-pattern>
      <pattern>/videos/trailers/*</pattern>
    </path-pattern>
    <link rel="path-metadata" type="application/cdni.PathMetadata"
      href="http://metadata.ucdn.example.com/videos/trailers"/>
  </path>
  <path>
    <path-pattern>
      <pattern>/videos/movies/*</pattern>
    </path-pattern>
    <link rel="path-metadata" type="application/cdni.PathMetadata"
      href="http://metadata.ucdn.example.com/videos/movies"/>
  </path>
</HostMetadata>

```

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

```

<PathMetadata>
  <path>
    <path-pattern>
      <pattern>/videos/movies/hd/*</pattern>
    </path-pattern>
    <link rel="path-metadata" type="application/cdni.PathMetadata"
      href="http://metadata.ucdn.example.com/videos/movies/hd"/>
  </path>
</PathMetadata>

```

Finally, if the path of the requested resource also matches the `"/videos/movies/hd/*"` pattern, the downstream CDN would also fetch the following object from `"http://metadata.ucdn.example.com/videos/movies/hd"` with MIME type `"application/cdni.PathMetadata"`:

```
<PathMetadata>
  <metadata>
    <type>application/cdni.TimeWindowACL</type>
    <value>
      <time>
        <time>
          <start>1213948800</start>
          <end>1327393200</end>
        </time>
      <type>allow</type>
    </time>
  </metadata>
</PathMetadata>
```

6.5. Extensibility

The set of property Metadata may be extended with proprietary and/or custom property Metadata. The GenericMetadata object defined in Section 4.1.7 allows any Metadata property to be included in either the HostMetadata or PathMetadata lists. As described in Section 3.4, any proprietary and/or custom property Metadata SHOULD be identified by the "ext." prefix in an appropriately descriptive type which conveys the organization defining the property Metadata and the function of the property Metadata.

Note: Identification of the property Metadata defining organization in the property Metadata type decreases the possibility of property Metadata type collision. The fully-qualified domain name of the organization in reverse order may be used for this purpose.

6.5.1. Metadata Enforcement

At any given time, the set of property Metadata supported by the uCDN may not match the set of property Metadata supported by the dCDN. The uCDN may or may not know which property Metadata the dCDN supports. In cases where the uCDN supports Metadata that the dCDN does not, the dCDN MUST be aware of any Metadata marked as "mandatory-to-enforce". If a CDN does not understand or is unable to perform the functions associated with any "mandatory-to-enforce" Metadata, the CDN MUST NOT service any requests for the corresponding content.

Note: Ideally, uCDNs would not delegate content requests to a dCDN which does not support the 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 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 Override

It is possible that new Metadata definitions may obsolete or override existing property Metadata (e.g., a future revision of the CDNI Metadata interface may redefine the Auth Metadata or a custom vendor extension may implement an alternate Auth Metadata option). If multiple Metadata (e.g., cdni.v2.Auth, ext.vendor1.Auth, and ext.vendor2.Auth) all override an existing Metadata (e.g., cdni.Auth) and all are marked as "mandatory-to-enforce", it may be ambiguous which Metadata should be applied, especially if the functionality of the Metadata conflict.

As described in Section 3.3, Metadata override only applies to Metadata objects of the same exact type, found in HostMetadata and nested PathMetadata structures. The CDNI Metadata interface does not support enforcement of dependencies between different Metadata types. It is the responsibility of the CSP and the CDN operators to ensure that Metadata assigned to a given content 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 first version of the CDNI Metadata interface does not include version numbering. Subsequent versions will incorporate a syntax for versioning.

7. IANA Considerations

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

[Ed. Need to consider a registry for Metadata type identifiers.]

8. Security Considerations

The CDNI Metadata Interface is expected to be secured as a function of the transport protocol (e.g. HTTP authentication [RFC2617], HTTPS [RFC2818], or inter-domain IPSec).

If a malicious metadata server is contacted by a downstream CDN, the malicious server may provide metadata to the downstream CDN which denies service for any piece of content to any user agent. The malicious server may also provide metadata which directs a downstream CDN to a malicious origin server instead of the actual origin server. The dCDN is expected to authenticate the server to prevent this situation (e.g. by using HTTPS and validating the server's certificate).

A malicious metadata client could request metadata for a piece of content from an upstream CDN. The metadata information may then be used to glean information regarding the uCDN or to contact an upstream origin server. The uCDN is expected to authenticate client requests to prevent this situation.

9. Acknowledgements

The authors would like to thank David Ferguson and Francois le Faucheur for their valuable comments and input to this document.

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.
- [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.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

10.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-03 (work in progress), February 2013.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-05 (work in progress), February 2013.
- [I-D.zyp-json-schema]
Zyp, K. and G. Court, "A JSON Media Type for Describing the Structure and Meaning of JSON Documents", draft-zyp-json-schema-03 (work in progress), November 2010.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", RFC 4151, October 2005.
- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.
- [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.

Appendix A. Relationship to the CDNI Requirements

Section 6 of [I-D.ietf-cdni-requirements] lists the requirements for the CDNI Metadata Distribution interface. This section outlines which of those requirements are met by the CDNI Metadata interface specified in this document.

All metadata requirements are met either directly or indirectly by the CDNI Metadata Interface described in this document, with the clarifications or exceptions described in the following paragraphs.

Requirements related to pre-positioning of metadata are met by this document on the assumption that other CDNI Interfaces are to be used by the upstream CDN to trigger the pre-positioning of metadata by the downstream CDN via the CDNI Metadata Interface. Triggering metadata pre-positioning is beyond the scope of the CDNI Metadata interface. However, the interface as described by this document supports pulling metadata on-demand for the purpose of pre-positioning.

Requirement META-7 relating to modification of metadata by the upstream CDN is met both by allowing timeouts on the cacheability of metadata objects and by allowing other CDNI interfaces to initiate a refetch or purge of metadata.

Requirement META-18 relating to surrogate cache behavior parameters is supported via extensibility. However, the example parameters in META-18 are not described in this document.

Appendix B. Metadata Rewriting

For some use cases, one CDN in a chain of interconnected CDNs must be able to rewrite CDNI Metadata received from its upstream CDN before presenting that CDNI Metadata to its downstream CDN.

The CDN which is performing the metadata rewriting is referred to as the 'Transit' CDN (tCDN), its upstream CDN as the uCDN and its downstream CDN as the dCDN.

Two (non-exhaustive) examples of when rewriting are:

- Allowing the dCDN is to acquire content from the tCDN instead of (or as well as) the uCDN. The tCDN must modify the appropriate CDNI Source Metadata objects to include itself as a possible source for the content.

- If the tCDN is transforming the original URI as part of CDNI request redirection on-route to the dCDN, the tCDN may need to modify the PatternMatch objects in any PathMetadata to take account of any URI path transformation it has performed.

When performing HTTP redirection between CDNs, the dCDN must be able to map an UA request to a host and path which are meaningful to the tCDN. The dCDN needs only to identify its immediate upstream neighbor and does not need to map (or understand) the entire chain of CDNs that precede the tCDN.

A dCDN may encode the identity of the tCDN in the URI it returns to the UA as part of request redirection (either directly or via the CDNI Request Routing Redirection interface). The exact method the dCDN uses to encode the information it requires is a local

implementation decision provided it enables the dCDN to identify the correct upstream CDN (tCDN) and to map the request to the appropriate host and path so that the dCDN can find and retrieve the correct CDNI Metadata from tCDN.

B.1. Example

The example in this section is not necessarily representative of URL rewriting in practice.

The UA requests the following URI from the uCDN:

`http://video.example/foo/bar`

The uCDN makes a CDNI Request Routing Redirection request to tCDN and tCDN returns a redirection URI of:

`http://tcdn.example/tcdn-prefix/foo/bar`

The tcdn-prefix/ encodes sufficient information for tCDN to identify uCDN as its upstream CDN neighbor. The tCDN makes a CDNI Request Routing Redirection request to dCDN and dCDN returns a redirection URI of:

`http://dcdn.example/dcdn-prefix/tcdn-prefix/foo/bar`

Therefore when dCDN receives a request for:

`http://dcdn.example/dcdn-prefix/tcdn-prefix/foo/bar`

The dCDN can use /dcdn-prefix/ to identify tCDN as its upstream CDN neighbor and reconstruct the URI tCDN expects. The tCDN can in turn use /tcdn-prefix/ to identify uCDN as its upstream CDN neighbour and reconstruct the URI uCDN expects.

Authors' Addresses

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

Email: ben@velocix.com

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: rmurray@velocix.com

Grant Watson
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: gwatson@velocix.com

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

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

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose 95134
USA

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

Kevin J. Ma
Azuki Systems, Inc.
43 Nagog Park
Acton, MA 01720
USA

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 1, 2017

B. Niven-Jenkins
R. Murray
Velocix (Alcatel-Lucent)
M. Caulfield
Cisco Systems
K. Ma
Ericsson
August 28, 2016

CDN Interconnection Metadata
draft-ietf-cdni-metadata-21

Abstract

The Content Delivery Networks Interconnection (CDNI) metadata interface enables interconnected Content Delivery Networks (CDNs) to exchange content distribution metadata in order to enable content acquisition and delivery. The CDNI metadata associated with a piece of content provides a downstream CDN with sufficient information for the downstream CDN to service content requests on behalf of an upstream CDN. This document describes both a base set of CDNI metadata and the protocol for exchanging that metadata.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on March 1, 2017.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
1.1. Terminology	5
1.2. Supported Metadata Capabilities	5
2. Design Principles	6
3. CDNI Metadata object model	7
3.1. HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects	8
3.2. Generic CDNI Metadata Objects	10
3.3. Metadata Inheritance and Override	13
4. CDNI Metadata objects	14
4.1. Definitions of the CDNI structural metadata objects	15
4.1.1. HostIndex	15
4.1.2. HostMatch	15
4.1.3. HostMetadata	17
4.1.4. PathMatch	18
4.1.5. PatternMatch	19
4.1.6. PathMetadata	20
4.1.7. GenericMetadata	21
4.2. Definitions of the initial set of CDNI Generic Metadata objects	23
4.2.1. SourceMetadata	23
4.2.1.1. Source	24
4.2.2. LocationACL Metadata	25
4.2.2.1. LocationRule	27
4.2.2.2. Footprint	27
4.2.3. TimeWindowACL	29
4.2.3.1. TimeWindowRule	30
4.2.3.2. TimeWindow	31
4.2.4. ProtocolACL Metadata	31
4.2.4.1. ProtocolRule	32
4.2.5. DeliveryAuthorization Metadata	33

4.2.6.	Cache	34
4.2.7.	Auth	36
4.2.8.	Grouping	37
4.3.	CDNI Metadata Simple Data Type Descriptions	37
4.3.1.	Link	37
4.3.1.1.	Link Loop Prevention	39
4.3.2.	Protocol	39
4.3.3.	Endpoint	39
4.3.4.	Time	40
4.3.5.	IPv4CIDR	40
4.3.6.	IPv6CIDR	40
4.3.7.	ASN	41
4.3.8.	CountryCode	41
5.	CDNI Metadata Capabilities	41
6.	CDNI Metadata interface	42
6.1.	Transport	42
6.2.	Retrieval of CDNI Metadata resources	43
6.3.	Bootstrapping	44
6.4.	Encoding	44
6.5.	Extensibility	45
6.6.	Metadata Enforcement	46
6.7.	Metadata Conflicts	46
6.8.	Versioning	47
6.9.	Media Types	48
6.10.	Complete CDNI Metadata Example	48
7.	IANA Considerations	52
7.1.	CDNI Payload Types	52
7.1.1.	CDNI MI HostIndex Payload Type	53
7.1.2.	CDNI MI HostMatch Payload Type	53
7.1.3.	CDNI MI HostMetadata Payload Type	54
7.1.4.	CDNI MI PathMatch Payload Type	54
7.1.5.	CDNI MI PatternMatch Payload Type	54
7.1.6.	CDNI MI PathMetadata Payload Type	54
7.1.7.	CDNI MI SourceMetadata Payload Type	54
7.1.8.	CDNI MI Source Payload Type	55
7.1.9.	CDNI MI LocationACL Payload Type	55
7.1.10.	CDNI MI LocationRule Payload Type	55
7.1.11.	CDNI MI Footprint Payload Type	55
7.1.12.	CDNI MI TimeWindowACL Payload Type	55
7.1.13.	CDNI MI TimeWindowRule Payload Type	56
7.1.14.	CDNI MI TimeWindow Payload Type	56
7.1.15.	CDNI MI ProtocolACL Payload Type	56
7.1.16.	CDNI MI ProtocolRule Payload Type	56
7.1.17.	CDNI MI DeliveryAuthorization Payload Type	56
7.1.18.	CDNI MI Cache Payload Type	57
7.1.19.	CDNI MI Auth Payload Type	57
7.1.20.	CDNI MI Grouping Payload Type	57
7.2.	CDNI Metadata Footprint Types Registry	57

7.3. CDNI Metadata Protocol Types Registry	58
8. Security Considerations	58
8.1. Authentication and Integrity	59
8.2. Confidentiality and Privacy	59
8.3. Securing the CDNI Metadata interface	60
9. Acknowledgements	60
10. Contributing Authors	60
11. References	61
11.1. Normative References	61
11.2. Informative References	63
Authors' Addresses	64

1. Introduction

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

The CDNI metadata interface is discussed in [RFC7336] along with four other interfaces that can be used to compose a CDNI solution (CDNI Control interface, CDNI Request Routing Redirection interface, CDNI Footprint & Capabilities Advertisement interface and CDNI Logging interface). [RFC7336] describes each interface and the relationships between them. The requirements for the CDNI metadata interface are specified in [RFC7337].

The CDNI metadata associated with a piece of content (or with a set of content) provides a dCDN with sufficient information for servicing content requests on behalf of an uCDN, in accordance with the policies defined by the uCDN.

This document defines the CDNI metadata interface which enables a dCDN to obtain CDNI metadata from an uCDN so that the dCDN can properly process and respond to:

- o Redirection requests received over the CDNI Request Routing Redirection interface [I-D.ietf-cdni-redirection].
- o Content requests received directly from User Agents.

Specifically, this document specifies:

- o A data structure for mapping content requests and redirection requests to CDNI metadata objects (Section 3 and Section 4.1).
- o An initial set of CDNI Generic metadata objects (Section 4.2).
- o A HTTP web service for the transfer of CDNI metadata (Section 6).

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

Additionally, the following terms are used throughout this document and are defined as follows:

- o Object - a collection of properties.
- o Property - a key and value pair where the key is a property name and the value is the property value or another object.

This document uses the phrase "[Object] A contains [Object] B" for simplicity when a strictly accurate phrase would be "[Object] A contains or references (via a Link object) [Object] B".

1.2. Supported Metadata Capabilities

Only the metadata for a small set of initial capabilities is specified in this document. This set provides the minimum amount of metadata for basic CDN interoperability while still meeting the requirements set forth by [RFC7337].

The following high-level functionality can be configured via the CDNI metadata objects specified in Section 4:

- o Acquisition Source: Metadata for allowing a dCDN to fetch content from a uCDN.
- o Delivery Access Control: Metadata for restricting (or permitting) access to content based on any of the following factors:
 - * Location
 - * Time Window
 - * Delivery Protocol
- o Delivery Authorization: Metadata for authorizing dCDN user agent requests.
- o Cache Control: Metadata for controlling cache behavior of the dCDN.

The metadata encoding described by this document is extensible in order to allow for future additions to this list.

The set of metadata specified in this document covers the initial capabilities above. It is only intended to support CDN interconnection for the delivery of content by a dCDN using HTTP/1.1 [RFC7230] and for a dCDN to be able to acquire content from a uCDN using either HTTP/1.1 or HTTP/1.1 over TLS [RFC2818].

Supporting CDN interconnection for the delivery of content using unencrypted HTTP/2 [RFC7540] (as well as for a dCDN to acquire content using unencrypted HTTP/2 or HTTP/2 over TLS) requires the registration of these protocol names in the CDNI Metadata Protocol Types registry Section 7.3.

Delivery of content using HTTP/1.1 over TLS or HTTP/2 over TLS SHOULD follow the guidelines set forth in [RFC7525]. Offline configuration of TLS parameters between CDNs is beyond the scope of this document.

2. Design Principles

The CDNI metadata interface was designed to achieve the following objectives:

1. Cacheability of CDNI metadata objects;
2. Deterministic mapping from redirection requests and content requests to CDNI metadata properties;
3. Support for DNS redirection as well as application-specific redirection (for example HTTP redirection);
4. Minimal duplication of CDNI metadata; and
5. Leveraging of existing protocols.

Cacheability can decrease the latency of acquiring metadata while maintaining its freshness, and therefore decrease the latency of serving content requests and redirection requests, without sacrificing accuracy. The CDNI metadata interface uses HTTP and its existing caching mechanisms to achieve CDNI metadata cacheability.

Deterministic mappings from content to metadata properties eliminates ambiguity and ensures that policies are applied consistently by all dCDNs.

Support for both HTTP and DNS redirection ensures that the CDNI metadata meets the same design principles for both HTTP and DNS based redirection schemes.

Minimal duplication of CDNI metadata improves storage efficiency in the CDNs.

Leveraging existing protocols avoids reinventing common mechanisms such as data structure encoding (by leveraging I-JSON [RFC7493]) and data transport (by leveraging HTTP [RFC7230]).

3. CDNI Metadata object model

The CDNI metadata object model describes a data structure for mapping redirection requests and content requests to metadata properties. Metadata properties describe how to acquire content from 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.

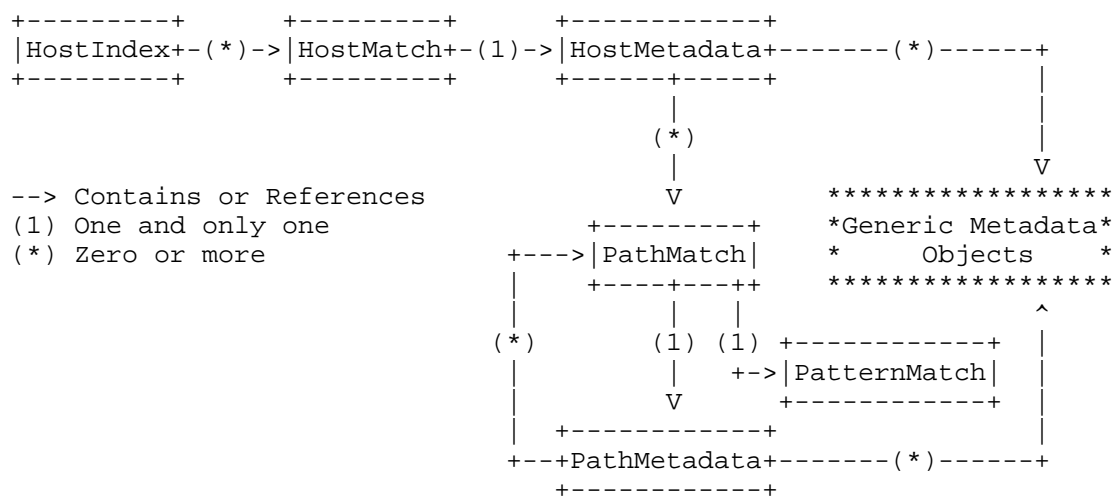


Figure 1: Relationships between CDNI Metadata Objects (Diagram Representation)

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.servicel23.ucdn.example",
            "b.servicel23.ucdn.example"
          ],
          "protocol": "http/1.1"
        },
        {
          "endpoints": ["origin.servicel23.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: Informational
Expires: August 27, 2013

K. Leung, Ed.
Cisco
Y. Lee, Ed.
Comcast
February 23, 2013

Content Distribution Network Interconnection (CDNI) Requirements
draft-ietf-cdni-requirements-05

Abstract

Content Delivery Networks (CDNs) are frequently used for large-scale content delivery. As a result, existing CDN providers are scaling up their infrastructure and many Network Service Providers (NSPs) are deploying their own CDNs. There is a requirement for interconnecting standalone CDNs so that their collective CDN footprint can be leveraged for the end-to-end delivery of content from Content Service Providers (CSPs) to end users. The Content Distribution Network Interconnection (CDNI) working group has been chartered to develop an interoperable and scalable solution for such CDN interconnection.

The goal of the present document is to outline the requirements for the solution and interfaces to be specified by the CDNI working group. This draft is a work in progress and requirements may be added, modified, or removed by the working group.

Requirements Language

The key words "High Priority", "Medium Priority" and "Low Priority" in this document are to be interpreted in the following way:

- o "High Priority" indicates requirements that are to be supported by the CDNI interfaces. A requirement is stated as "High Priority" when it is established by the working group that it can be met without compromising the targeted schedule for WG deliverables, or when it is established that specifying a solution without meeting this requirement would not make sense and would justify re-adjusting the WG schedule, or both. This is tagged as "[HIGH]".
- o "Medium Priority" indicates requirements that are to be supported by the CDNI interfaces unless the WG realizes at a later stage that attempting to meet this requirement would compromise the overall WG schedule (for example it would involve complexities that would result in significantly delaying the deliverables). This is tagged as "[MED]".

- o "Low Priority" indicates requirements that are to be supported by the CDNI interfaces provided that dedicating WG resources to this work does not prevent addressing "High Priority" and "Medium Priority" requirements and that attempting to meet this requirement would not compromise the overall WG schedule. This is tagged as "[LOW]".

Status of this Memo

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

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

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

This Internet-Draft will expire on August 27, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
1.1. Terminology	4
2. CDNI Model and CDNI Interfaces	4
3. Generic CDNI Requirements	6
4. CDNI Control Interface Requirements	7
5. CDNI Request Routing Interface Requirements	10
6. CDNI Metadata Distribution Interface Requirements	14
7. CDNI Logging Interface Requirements	17
8. CDNI Security Requirements	20
9. IANA Considerations	20
10. Security Considerations	21
11. Authors	21
12. Acknowledgements	21
13. References	22
13.1. Normative References	22
13.2. Informative References	22
Authors' Addresses	22

1. Introduction

The volume of video and multimedia content delivered over the Internet is rapidly increasing and expected to continue doing so in the future. In the face of this growth, Content Delivery Networks (CDNs) provide numerous benefits: reduced delivery cost for cacheable content, improved quality of experience for end users, and increased robustness of delivery. For these reasons CDNs are frequently used for large-scale content delivery. As a result, existing CDN providers are scaling up their infrastructure and many Network Service Providers (NSPs) are deploying their own CDNs. It is generally desirable that a given content item can be delivered to an End User regardless of that End User's location or attachment network. However, the footprint of a given CDN in charge of delivering a given content may not expand close enough to the End User's current location or attachment network to realize the cost benefit and user experience that a more distributed CDN would provide. This creates a requirement for interconnecting standalone CDNs so that their collective CDN footprint can be leveraged for the end-to-end delivery of content from Content Service Providers (CSPs) to End Users. The Content Distribution Network Interconnection (CDNI) working group has been chartered to develop an interoperable and scalable solution for such CDN interconnection.

[I-D.ietf-cdni-problem-statement] outlines the problem area that the CDNI working group is chartered to address.

[I-D.ietf-cdni-use-cases] discusses the use cases for CDN Interconnection. [I-D.davie-cdni-framework] discusses the technology framework for the CDNI solution and interfaces.

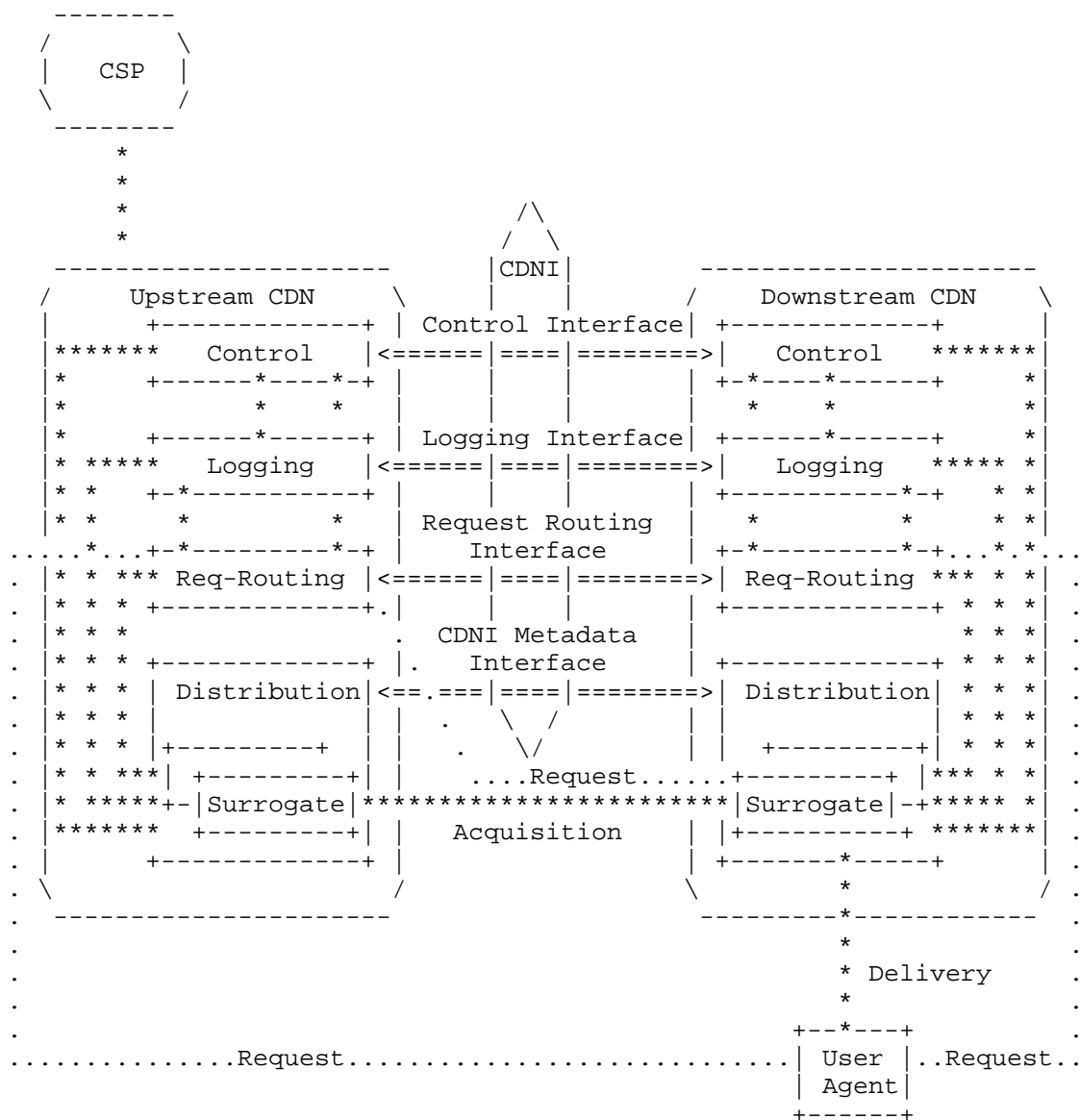
The goal of the present document is to document the requirements for the CDNI solution and interfaces. In order to meet the timelines defined in the working group charter, the present document categorizes the CDNI requirements as "High Priority", "Medium Priority", and "Low Priority".

1.1. Terminology

This document uses the terminology defined in section 1.1 of [I-D.davie-cdni-framework].

2. CDNI Model and CDNI Interfaces

For convenience Figure 1 from [I-D.davie-cdni-framework] illustrating the CDNI problem area and the CDNI protocols is replicated below.



\Leftrightarrow interfaces inside the scope of CDNI

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

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

Figure 1: CDNI Model and CDNI Interfaces

3. Generic CDNI Requirements

This section identifies generic requirements independent of the individual CDNI interfaces. Some of those are expected to affect multiple or all interfaces.

- GEN-1 [MED] Wherever possible, the CDNI interfaces should reuse or leverage existing IETF protocols.
- GEN-2 [HIGH] The CDNI solution shall not require a change, or an upgrade, to the User Agent to benefit from content delivery through interconnected CDNs.
- GEN-3 [HIGH] The CDNI solution shall not require a change, or an upgrade, to the Content Service Provider to benefit from content delivery through interconnected CDNs.
- GEN-4 [HIGH] The CDNI solution shall not require intra-CDN information to be exposed to other CDNs for effective and efficient delivery of the content. Examples of intra-CDN information include surrogate topology, surrogate status, cached content, etc.
- GEN-5 [HIGH] The CDNI solution shall support delivery to the user agent based on HTTP [RFC2616]. (Note that while delivery and acquisition "data plane" protocols are out of the CDNI solution scope, the CDNI solution "control plane" protocols are expected to participate in enabling, selecting or facilitating operations of such acquisition and delivery protocols. Hence it is useful to state requirements on the CDNI solution in terms of which acquisition and delivery protocols).
- GEN-6 [HIGH] The CDNI solution shall support acquisition across CDNs based on HTTP [RFC2616].
- GEN-7 [LOW] The CDNI solution may support delivery to the user agent based on protocols other than HTTP.
- GEN-8 [LOW] The CDNI solution may support acquisition across CDNs based on protocols other than HTTP.
- GEN-9 [MED] The CDNI solution should support cascaded CDN redirection (CDN1 redirects to CDN2 that redirects to CDN3) to an arbitrary number of levels beyond the first level.

- GEN-10 [MED] The CDNI solution should support an arbitrary topology of interconnected CDNs (i.e. the CDN topology cannot be restricted to a tree, a loop-free topology, etc.).
- GEN-11 [HIGH] The CDNI solution shall prevent looping of any CDNI information exchange.
- GEN-12 [HIGH] When making use of third party reference, the CDNI solution shall consider the potential issues associated with the use of various format of third-party references (e.g. NAT or IPv4/IPv6 translation potentially breaking third-party references based on an IP addresses such as URI containing IPv4 or IPv6 address literals, split DNS situations potentially breaking third-party references based on DNS fully qualified domain names) and wherever possible avoid, minimize or mitigate the associated risks based on the specifics of the environments where the reference is used (e.g. likely or unlikely presence of NAT in the path). In particular, this applies to situations where the CDNI solution needs to construct and convey uniform resource identifiers for directing/redirecting a content request, as well as to situations where the CDNI solution needs to pass on a third party reference (e.g. to identify a User Agent) in order to allow another entity to make a more informed decision (e.g. make a more informed request routing decision by attempting to derive location information from the third party reference).
- GEN-13 Removed.
- GEN-14 [HIGH] The CDNI solution shall support HTTP Adaptive Streaming content.

4. CDNI Control Interface Requirements

The primary purpose of the CDNI Control interface is to initiate the interconnection across CDNs, bootstrap the other CDNI interfaces and trigger actions into the Downstream CDN by the Upstream CDN (such as delete object from caches or trigger pre-positioned content acquisition). We observe that while the CDNI Control interface is currently discussed as a single "protocol", further analysis will determine whether the corresponding requirements are to be realized over a single interface and protocol, or over multiple interfaces and protocols.

- CNTL-1 [HIGH] The CDNI Control interface shall allow the Upstream CDN to request that the Downstream CDN (and, if cascaded CDNs are supported by the solution, that the potential cascaded Downstream CDNs) perform the following actions on an object or object set:
- * Mark an object or set of objects and/or its CDNI metadata as "stale" and revalidate them before they are delivered again
 - * Delete an object or set of objects and/or its CDNI metadata from the CDN surrogates and any storage. Only the object(s) and CDNI metadata that pertain to the requesting Upstream CDN are allowed to be purged.
- CNTL-2 [HIGH] The CDNI Control interface shall allow the Downstream CDN to report on the completion of these actions (by itself, and if cascaded CDNs are supported by the solution, by potential cascaded Downstream CDNs), in a manner appropriate for the action (e.g. synchronously or asynchronously). The confirmation receipt should include a success or failure indication. The failure indication is used if the Downstream CDN cannot delete the content in its storage.
- CNTL-3 [HIGH] The CDNI Control interface shall support initiation and control by the Upstream CDN of pre-positioned CDNI metadata acquisition by the Downstream CDN.
- CNTL-4 [MED] The CDNI Control interface should support initiation and control by the Upstream CDN of pre-positioned content acquisition by the Downstream CDN.
- CNTL-5 [LOW] The CDNI Control interface may allow a CDN to establish, update and terminate a CDN interconnection with another CDN whereby one CDN can act as a Downstream CDN for the other CDN (that acts as an Upstream CDN).
- CNTL-6 [LOW] The CDNI Control interface may allow control of the CDNI interconnection between any two CDNs independently for each direction (i.e. For the direction where CDN1 is the Upstream CDN and CDN2 is the Downstream CDN, and for the direction where CDN2 is the Upstream CDN and CDN1 is the Downstream CDN).
- CNTL-7 [LOW] The CDNI Control interface may allow bootstrapping of the Request-Routing interface. For example, this can potentially include:

- * negotiation of the Request-Routing method (e.g. DNS vs HTTP, if more than one method is specified)
 - * discovery of the Request-Routing protocol endpoints
 - * information necessary to establish secure communication between the Request-Routing protocol endpoints.
- CNTL-8 [LOW] The CDNI Control interface may allow bootstrapping of the CDNI Metadata interface. This information could, for example, include:
- * discovery of the CDNI Metadata signaling protocol endpoints
 - * information necessary to establish secure communication between the CDNI Metadata signaling protocol endpoints.
- CNTL-9 [LOW] The CDNI Control interface may allow bootstrapping of the Content Acquisition interface. This could, for example, include exchange and negotiation of the Content Acquisition protocols to be used across the CDNs (e.g. HTTP, HTTPS, FTP, ATIS C2).
- CNTL-10 [LOW] The CDNI Control interface may allow exchange and negotiation of delivery authorization mechanisms to be supported across the CDNs (e.g. URI signature based validation).
- CNTL-11 [LOW] The CDNI Control interface may allow bootstrapping of the CDNI Logging interface. This information could, for example, include:
- * discovery of the Logging protocol endpoints
 - * information necessary to establish secure communication between the Logging protocol endpoints
 - * negotiation/definition of the log file format and set of fields to be exported through the Logging protocol, with some granularity (e.g. On a per content type basis).
 - * negotiation/definition of parameters related to transaction Logs export (e.g., export protocol, file compression, export frequency, directory).

- CNTL-12 [MED] The CDNI Control interface should allow for multiple content items identified by a Content Collection ID to be purged using a single Content Purge action.

5. CDNI Request Routing Interface Requirements

The main function of the Request Routing interface is to allow the Request-Routing systems in interconnected CDNs to communicate to facilitate redirection of the request across CDNs.

- REQ-1 [HIGH] The CDNI Request-Routing interface shall allow the Downstream CDN to communicate to the Upstream CDN coarse information about the Downstream CDN ability and/or willingness to handle requests from the Upstream CDN. For example, this could potentially include a binary signal ("Downstream CDN ready/not-ready to take additional requests from Upstream CDN") to be used in case of excessive load or failure condition in the Downstream CDN.

- REQ-2 [MED] The CDNI Request-Routing interface should allow the Downstream CDN to communicate to the Upstream CDN aggregate information to facilitate CDN selection during request routing, such as Downstream CDN capabilities, resources and affinities (i.e. Preferences or cost). This information could, for example, include:

- * supported content types and delivery protocols
- * footprint (e.g. layer-3 coverage)
- * a set of metrics/attributes (e.g. Streaming bandwidth, storage resources, distribution and delivery priority)
- * a set of affinities (e.g. Preferences, indication of distribution/delivery fees)
- * information to facilitate request redirection (e.g. Reachability information of Downstream CDN Request Routing system).

[Note: Some of this information - such as supported content types and delivery protocols- may also potentially be taken into account by the distribution system in the Upstream CDN for pre-positioning of content and/or metadata in the Downstream CDN in case of pre-positioned content acquisition and/or pre-positioned CDNI metadata acquisition.]

- REQ-3 [MED] In the case of cascaded redirection, the CDNI Request-Routing interface shall allow the Downstream CDN to also include in the information communicated to the Upstream CDN, information on the capabilities, resources and affinities of CDNs to which the Downstream CDN may (in turn) redirect requests received by the Upstream CDN. In that case, the CDNI Request-Routing interface shall prevent looping of such information exchange.
- REQ-4 [LOW] The CDNI Request-Routing interface may allow the Downstream CDN to communicate to the Upstream CDN aggregate information on CDNI administrative limits and policy. This information can be taken into account by the Upstream CDN Request Routing system in its CDN Selection decisions. This information could, for example, include:
- * maximum number of requests redirected by the Upstream CDN to be served simultaneously by the Downstream CDN
 - * maximum aggregate volume of content (e.g. in Terabytes) to be delivered by the Downstream CDN over a time period.
- REQ-5 [HIGH] The CDNI Request-Routing architecture and interface shall support efficient request-routing for small objects. This may, for example, call for a mode of operation (e.g. DNS-based request routing) where freshness and accuracy of CDN/Surrogate selection can be traded-off against reduced request-routing load (e.g. Via lighter-weight queries and caching of request-routing decisions).
- REQ-6 [HIGH] The CDNI Request-Routing architecture and interface shall support efficient request-routing for large objects. This may, for example, call for a mode of operation (e.g. HTTP-based request routing) where freshness and accuracy of CDN/Surrogate selection justifies a per-request decision and a per-request CDNI Request-Routing protocol call.
- REQ-7 [HIGH] The CDNI Request-Routing architecture shall support recursive CDNI request routing.
- REQ-8 [HIGH] The CDNI Request-Routing architecture shall support iterative CDNI request routing.
- REQ-9 [MED] In case of detection of a request redirection loop, the CDNI Request-Routing loop prevention mechanism should allow routing of the request by avoiding the loop (as opposed to the request loop being simply interrupted without routing the request).

- REQ-10 [MED] The CDNI Request-Routing protocol should support a mechanism allowing enforcement of a limit on the number of successive CDN redirections for a given request.
- REQ-11 [LOW] The CDNI Request-Routing protocol may support a mechanism allowing an Upstream CDN to avoid redirecting a request to a Downstream CDN if that is likely to result in the total redirection time exceeding some limit.
- REQ-12 [HIGH] The CDNI Request-Routing protocol shall allow the Upstream CDN to include, in the query to the Downstream CDN, the necessary information to allow the Downstream CDN to process the redirection query. This could, for example, include:
- * information from which the location of the user-agent that originated the request can be inferred (e.g. User Agent fully qualified domain name in case of HTTP-based Request Routing, DNS Proxy fully qualified domain name in case of DNS-based Request Routing)
 - * requested resource information (e.g. Resource URI in case of HTTP-based Request Routing, Resource hostname in case of DNS-based Request Routing)
 - * additional available request information (e.g. request headers in case of HTTP-based Request Routing).
- REQ-13 [LOW] The CDNI Request-Routing protocol may also allow the Upstream CDN to convey information pointing to CDNI metadata applicable (individually or through inheritance) to the requested content. For illustration, the CDNI metadata pointed to could potentially include metadata that is applicable to any content, metadata that is applicable to a content collection (to which the requested content belongs) and/or metadata that is applicable individually to the requested content.
- REQ-14 [HIGH] The CDNI Request-Routing interface shall allow the Downstream CDN to include the following information in the response to the Upstream CDN:
- * status code, in particular indicating acceptance or rejection of request (e.g. Because the Downstream CDN is unwilling or unable to serve the request). In case of rejection, an error code is also to be provided, which allows the Upstream CDN to react appropriately (e.g. Select another Downstream CDN, or serve the request

itself)

- * redirection information (e.g. Resource URI in case of HTTP-based Request Routing, equivalent of a DNS record in case of DNS-based Request Routing).

- REQ-15 [HIGH] The CDNI Request-Routing interface shall allow for per-chunk request routing of HTTP Adaptive Streaming content. [Ed: chunk is treated as any content, is this needed?]
- REQ-16 [MED] The CDNI Request-Routing interface should allow the Upstream CDN to use the information conveyed by the Downstream CDN during the Recursive Request Routing process to rewrite an HTTP Adaptive Streaming manifest file. [Ed: should this be LOW?]
- REQ-17 [MED] The CDNI Request-Routing interface should allow the Upstream CDN to re-sign the invariant portion of the chunk URIs embedded in the HTTP Adaptive Streaming manifest file. [Ed: should this be LOW?]
- REQ-18 [MED] The CDNI Request-routing interface should allow the use of HTTP cookie to associate the chunks with the HTTP Adaptive Stream manifest file (which is verified by the URI signature) based on the Authorization Group ID (which is an identifier used to correlate the manifest file to the related chunks). [Ed: should this be LOW?]
- REQ-19 [MED] The CDNI Request-Routing interface may allow for an efficient method of transferring request routing information for multiple chunks from the Downstream CDN to the Upstream CDN as part of the recursive request routing process. [Ed: should this be LOW?]
- REQ-20 [MED] The CDNI Request-Routing/Footprint and Advertising interface shall support advertisement of the following capabilities:
- * support for customized CDNI Logging
 - * support of Content Collection ID logging
 - * support for Session ID logging

6. CDNI Metadata Distribution Interface Requirements

The primary function of the CDNI Metadata Distribution interface is to allow the Distribution system in interconnected CDNs to communicate to ensure Content Distribution Metadata with inter-CDN scope can be exchanged across CDNs. We observe that while the CDNI Metadata Distribution protocol is currently discussed as a single "protocol", further analysis will determine whether the corresponding requirements are to be realized over a single interface and protocol, or over multiple interfaces and protocols. For example, a subset of the CDNI metadata might be conveyed in-band along with the actual content acquisition across CDNs (e.g. content MD5 in HTTP header) while another subset might require an out-of-band interface & protocol (e.g. geo-blocking information).

- META-1 [HIGH] The CDNI Metadata Distribution interface shall allow the Upstream CDN to provide the Downstream CDN with content distribution metadata of inter-CDN scope.
- META-2 [HIGH] The CDNI Metadata Distribution interface shall support exchange of CDNI metadata for both the dynamic content acquisition model and the pre-positioning content acquisition model.
- META-3 [HIGH] The CDNI Metadata Distribution interface shall support a mode where no, or a subset of, the Metadata is initially communicated to the Downstream CDN along with information about how/where to acquire the rest of the CDNI Metadata (i.e. Dynamic CDNI metadata acquisition).
- META-4 [MED] The CDNI Metadata Distribution interface should support a mode where all the relevant Metadata is initially communicated to the Downstream CDN (i.e. Pre-positioned CDNI metadata acquisition).
- META-5 [HIGH] Whether in the pre-positioned content acquisition model or in the dynamic content acquisition model, the CDNI Metadata Distribution interface shall provide the necessary information to allow the Downstream CDN to acquire the content from an upstream source (e.g. Acquisition protocol and Uniform Resource Identifier in Upstream CDN- or rules to construct this URI).
- META-6 [HIGH] The CDNI metadata shall allow signaling of one or more upstream sources, where each upstream source can be in the Upstream CDN, in another CDN, the CSP origin server or any arbitrary source designated by the Upstream CDN. Note that some upstream sources (e.g. the content origin server)

may or may not be willing to serve the content to the Downstream CDN, if this policy is known to the Upstream CDN then it may omit those sources when exchanging CDNI metadata.

- META-7 [HIGH] The CDNI Metadata Distribution interface shall allow the Upstream CDN to request addition and modification of CDNI Metadata into the Downstream CDN.
- META-8 [HIGH] The CDNI Metadata Distribution interface shall allow removal of obsolete CDNI Metadata from the Downstream CDN (this could, for example, be achieved via an explicit removal request from the Upstream CDN or via expiration of a Time-To-Live associated to the Metadata).
- META-9 [HIGH] The CDNI Metadata Distribution interface shall allow association of CDNI Metadata at the granularity of individual object. This is necessary to achieve fine-grain Metadata distribution at the level of an individual object when necessary.
- META-10 [HIGH] The CDNI Metadata Distribution interface shall allow association of CDNI Metadata at the granularity of an object set. This is necessary to achieve scalable distribution of metadata when a large number of objects share the same distribution policy.
- META-11 [HIGH] The CDNI Metadata Distribution interface shall support multiple levels of inheritance with precedence to more specific metadata. For example, the CDNI Metadata Distribution protocol may support metadata that is applicable to any content, metadata that is applicable to a content collection and metadata that is applicable to an individual content where content level metadata overrides content collection metadata that overrides metadata for any content.
- META-12 [HIGH] The CDNI Metadata Distribution interface shall ensure that conflicting metadata with overlapping scope are prevented or deterministically handled.
- META-13 Removed.
- META-14 [HIGH] The CDNI Metadata Distribution interface shall allow signaling of content distribution control policies. For example, this could potentially include:

- * geo-blocking information (i.e. Information defining geographical areas where the content is to be made available or blocked)
 - * availability windows (i.e. Information defining time windows during which the content is to be made available or blocked; expiration time may also be included to remove content)
 - * delegation whitelist/blacklist (i.e. Information defining which Downstream CDNs the content may/may not be delivered through)
- META-15 [HIGH] The CDNI Metadata interface shall be able to exchange a set of well-accepted metadata elements with specified semantics (e.g. start of time window, end of time window).
- META-16 [HIGH] The CDNI Metadata interface shall allow exchange of opaque metadata element, whose semantic is not defined in CDNI but established by private CDN agreement.
- META-17 [HIGH] The CDNI Metadata Distribution interface shall allow signaling of authorization checks and validation that are to be performed by the surrogate before delivery. For example, this could potentially include:
- * need to validate URI signed information (e.g. Expiry time, Client IP address).
- META-18 [LOW] The CDNI Metadata Distribution interface may allow signaling of CDNI-relevant surrogate cache behavior parameters. For example, this could potentially include:
- * control of whether the query string of HTTP URI is to be ignored by surrogate cache
 - * content revalidation parameters (e.g. TTL)
- META-19 [HIGH] The CDNI Metadata interface shall provide indication of related content (e.g. HTTP Adaptive Bit Rate chunks) by the Content Collection ID (CCID) metadata. This could be used by the Downstream CDN for operations on the group of content. For example, this could potentially include:
- * content acquisition for the entire set of files when one piece of content is requested

- * local file management and storage bundles all the files for the content
- * purging the entire set of files associated with the content
- * logging of the delivery of the content for the session when at least one file in the set was delivered

META-20 [HIGH] The CDNI Metadata Distribution interface shall support an OPTIONAL mechanism allowing the Upstream CDN to indicate to the Downstream CDN which CDNI Log fields are to be provided for all, for specific sets of, or for specific content items delivered using HTTP. A CDNI implementation that does not support this optional CDNI Metadata Distribution Interface mechanism MUST ignore this log format indication and generate CDNI logging format for HTTP Adaptive Streaming using the default set of CDNI Logging fields.

META-21 [MED] The CDNI Metadata Distribution interface shall allow the Upstream CDN to signal to the Downstream CDN the Content Collection ID value for all, for specific sets of, or for specific content items delivered using HTTP. Whenever the Downstream CDN is instructed by the Upstream CDN to report the Content Collection ID field in the log records, the Downstream CDN is to use the value provided through the CDNI Metadata interface for the corresponding content. Note the Session ID field along with Content Collection ID may be used for HTTP Adaptive Streaming content.

META-22 [MED] The CDNI Metadata Distribution interface shall allow the Upstream CDN to signal to the Downstream CDN the Authorization Group ID value for all the related HTTP Adaptive Streamin content (i.e. manifest file and chunks). The authorization result of a content (e.g. manifest file) is transferred over to related content (e.g. chunks). [Ed: need to improve wording?]

7. CDNI Logging Interface Requirements

This section identifies the requirements related to the CDNI Logging interface. We observe that while the CDNI Logging interface is currently discussed as a single "protocol", further analysis will determine whether the corresponding requirements are to be realized over a single interface and protocol, or over multiple interfaces and protocols.

- LOG-1 [HIGH] The CDNI logging architecture and interface shall ensure reliable logging of CDNI events.
- LOG-2 [HIGH] The CDNI Logging interface shall provide logging of deliveries and incomplete deliveries to User Agents performed by the Downstream CDN as a result of request redirection by the Upstream CDN.
- LOG-3 [MED] In the case of cascaded CDNs, the CDNI Logging interface shall allow the Downstream CDN to report to the Upstream CDN logging for deliveries and incomplete deliveries performed by the Downstream CDN itself as well as logging for deliveries and incomplete deliveries performed by cascaded CDNs on behalf of the Downstream CDN.
- LOG-4 Removed.
- LOG-5 [HIGH] The CDNI Logging interface shall support batch/offline exchange of logging records.
- LOG-6 [MED] The CDNI Logging interface should also support additional timing constraints for some types of logging records (e.g. near-real time for monitoring and analytics applications)
- LOG-7 [HIGH] The CDNI Logging interface shall define a log file format and a set of fields to be exported through the Logging protocol, with some granularity (e.g. On a per content type basis).
- LOG-8 [HIGH] The CDNI Logging interface shall define a transport mechanisms to exchange CDNI Logging files.
- LOG-9 [MED] The CDNI Logging interface should allow a CDN to query another CDN for relevant current logging records (e.g. For on-demand access to real-time logging information).
- LOG-10 [LOW] The CDNI Logging interface may support aggregate/summarized logs (e.g. total bytes delivered for a content regardless of individual User Agents to which it was delivered).
- LOG-11 [LOW] The CDNI Logging interface may support logging of performance data for deliveries to User Agents performed by the Downstream CDN as a result of request redirection by the Upstream CDN. Performance data may include various traffic statistics (the specific parameters are to be determined). The CDNI Logging interface may support the Upstream CDN to

indicate the nature and contents of the performance data to be reported by the Downstream CDN.

- LOG-12 [MED] The CDNI Logging interface shall support logging of consumed resources (e.g. storage, bandwidth) to the Upstream CDN for deliveries where content is stored by the Downstream CDN for delivery to User Agents. The information logged may include the type of storage (e.g., Origin, Intermediate, Edge, Cache) as well as the amount of storage (e.g., total GB, GB used, per time period, per content domain) all of which may impact the cost of the services.
- LOG-13 [MED] In the case of cascaded CDNs, the CDNI Logging interface shall support the Downstream CDN to report consumed resources (e.g. storage, bandwidth) to the Upstream CDN where content is stored by the Downstream CDN itself as well as logging for storage resources when content storage is performed by cascaded CDNs on behalf of the Downstream CDN.
- LOG-14 [HIGH] The CDNI Logging interface shall support logging of deleted objects from the Downstream CDN to the Upstream CDN as a result of explicit delete requests on via the CDNI Control interface from the Upstream CDN.
- LOG-15 [HIGH] The CDNI Logging interface shall support extensibility to allow proprietary information fields to be carried. These information fields must be agreed upon ahead of time between the corresponding CDNs.
- LOG-16 [HIGH] The CDNI Logging interface shall support the exchange of extensible log file formats to support proprietary information fields. These information fields must be agreed upon ahead of time between the corresponding CDNs.
- LOG-17 [HIGH] The CDNI Logging interface shall support the notification from Downstream CDN to Upstream CDN for the event that the logging retention duration or maximum size of logging data has exceeded.
- LOG-18 [MED] The CDNI Logging interface should support the ability for the Downstream CDN to include the Content Collection ID and Session ID fields in CDNI log entries generated for HTTP Adaptive Streaming content. This fields can be supported by the "customizable" log format which is expected to be defined independently of HTTP Adaptive Streaming.

8. CDNI Security Requirements

This section identifies the requirements related to the CDNI security. Some of those are expected to affect multiple or all protocols.

- SEC-1 [HIGH] All the CDNI interface shall support secure operation over unsecured IP connectivity (e.g. The Internet). This includes authentication, confidentiality, integrity protection as well as protection against spoofing and replay.
- SEC-2 [HIGH] The CDNI solution shall provide sufficient protection against Denial of Service attacks. This includes protection against spoofed delivery requests sent by user agents directly to a Downstream CDN attempting to appear as if they had been redirected by a given Upstream CDN when they have not.
- SEC-3 [MED] The CDNI solution should be able to ensure that for any given request redirected to a Downstream CDN, the chain of CDN Delegation (leading to that request being served by that CDN) can be established with non-repudiation.
- SEC-4 [MED] The CDNI solution should be able to ensure that the Downstream CDN cannot spoof a transaction log attempting to appear as if it corresponds to a request redirected by a given Upstream CDN when that request has not been redirected by this Upstream CDN. This ensures non-repudiation by the Upstream CDN of transaction logs generated by the Downstream CDN for deliveries performed by the Downstream CDN on behalf of the Upstream CDN.
- SEC-5 [LOW] The CDNI solution may provide a mechanism allowing an Upstream CDN that has credentials to acquire content from the CSP origin server (or another CDN), to allow establishment of credentials authorizing the Downstream CDN to acquire the content from the CSP origin server (or the other CDN) (e.g. In case the content cannot be acquired from the Upstream CDN).

9. IANA Considerations

This document makes no request of IANA.

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

10. Security Considerations

This document discusses CDNI security requirements in Section 8.

11. Authors

This document reflects the contributions from the following authors:

Francois Le Faucheur

Cisco Systems

flefauch@cisco.com

Mahesh Viveganandhan

Cisco Systems

mvittal@cisco.com

Grant Watson

BT

grant.watson@bt.com

12. Acknowledgements

This document leverages the earlier work of the IETF CDI working group in particular as documented in [I-D.cain-request-routing-req], [I-D.amini-cdi-distribution-reqs] and [I-D.gilletti-cdn-aaa-reqs].

The authors would like to thank Gilles Bertrand, Christophe Caillet, Bruce Davie, Phil Eardly, Ben Niven-Jenkins, Agustin Schapira, Emile Stephan, Eric Burger, Susan He, Kevin Ma, and Daryl Malas for their input. Serge Manning along with Robert Streijl, Vishwa Prasad, Percy Tarapore, Mike Geller, and Ramki Krishnan contributed to this document by addressing the requirements of the ATIS Cloud Services Forum.

Ray Brandenburg, Matt Caufield, and Francois Le Faucheur/Gilles Bertrand provided valuable inputs for HTTP Adaptive Streaming, CDNI Metadata interface, and CDNI Logging interface, respectively.

13. References

13.1. Normative References

- [I-D.davie-cdni-framework]
Davie, B. and L. Peterson, "Framework for CDN Interconnection", draft-davie-cdni-framework-01 (work in progress), October 2011.
- [I-D.ietf-cdni-problem-statement]
Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-ietf-cdni-problem-statement-08 (work in progress), June 2012.
- [I-D.ietf-cdni-use-cases]
Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-10 (work in progress), August 2012.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

13.2. Informative References

- [I-D.amini-cdi-distribution-reqs]
Amini, L., "Distribution Requirements for Content Internetworking", draft-amini-cdi-distribution-reqs-02 (work in progress), November 2001.
- [I-D.cain-request-routing-req]
Cain, B., "Request Routing Requirements for Content Internetworking", draft-cain-request-routing-req-03 (work in progress), November 2001.
- [I-D.gilletti-cdn-aaa-reqs]
"CDI AAA Requirements, draft-gilletti-cdn-aaa-reqs-01.txt", June 2001.

Authors' Addresses

Kent Leung (editor)
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
U.S.A.

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

Yiu Lee (editor)
Comcast
One Comcast Center
Philadelphia, PA 19103
U.S.A.

Email: yiulee@cable.comcast.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 3, 2014

K. Leung, Ed.
Cisco
Y. Lee, Ed.
Comcast
Jan 30, 2014

Content Distribution Network Interconnection (CDNI) Requirements
draft-ietf-cdni-requirements-17

Abstract

Content delivery is frequently provided by specifically architected and provisioned Content Delivery Networks (CDNs). As a result of significant growth in content delivered over IP networks, existing CDN providers are scaling up their infrastructure. Many Network Service Providers and Enterprise Service Providers are also deploying their own CDNs. To deliver contents from the Content Service Provider (CSP) to end users, the contents may traverse across multiple CDNs. This creates a need for interconnecting (previously) standalone CDNs so that they can collectively act as a single delivery platform from the CSP to the end users.

The goal of the present document is to outline the requirements for the solution and interfaces to be specified by the CDNI working group.

Status of this Memo

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

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

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

This Internet-Draft will expire on August 3, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. CDNI Model and CDNI Interfaces	4
3. Generic CDNI Requirements	7
4. CDNI Control Interface Requirements	8
5. CDNI Request Routing Redirection Interface Requirements	11
6. CDNI Footprint & Capabilities Advertisement Interface Requirements	13
7. CDNI Metadata Interface Requirements	15
8. CDNI Logging Interface Requirements	19
9. CDNI Security Requirements	21
10. IANA Considerations	22
11. Security Considerations	22
12. Contributors	22
13. Acknowledgements	22
14. References	23
14.1. Normative References	23
14.2. Informative References	23
Authors' Addresses	24

1. Introduction

The volume of video and multimedia content delivered over the Internet is rapidly increasing and expected to continue doing so in the future. In the face of this growth, Content Delivery Networks (CDNs) provide numerous benefits: reduced delivery cost for cacheable content, improved quality of experience for end users, and increased robustness of delivery. For these reasons CDNs are frequently used for large-scale content delivery. As a result of the significant growth in content delivered over IP networks, existing CDN providers are scaling up their infrastructure and many Network Service Providers and Enterprise Service Providers are deploying their own CDNs. Subject to the policy of the Content Service Provider (CSP), it is generally desirable that a given item of content can be delivered to an end user regardless of that end user's location or attachment network. This creates a need for interconnecting (previously) standalone CDNs so they can interoperate and collectively behave as a single delivery infrastructure. The Content Distribution Network Interconnection (CDNI) working group has been chartered to develop an interoperable and scalable solution for such CDN interconnections.

CDNI Problem Statement [RFC6707] outlines the problem area that the CDNI working group is chartered to address. Use Cases for CDNI [RFC6770] discusses the use cases for CDN Interconnection. Framework for CDN Interconnection [I-D.ietf-cdni-framework] discusses the technology framework for the CDNI solution and interfaces.

The goal of the present document is to document the requirements for the CDNI solution and interfaces. In order to meet the timelines defined in the working group charter, the present document categorizes the CDNI requirements as "High Priority", "Medium Priority", and "Low Priority".

1.1. Terminology

This document uses the terminology defined in [RFC6707]. In addition, the key words "High Priority", "Medium Priority" and "Low Priority" in this document are to be interpreted in the following way:

- o "High Priority": When a requirement is tagged as "{HIGH}", it is considered by the working group as an essential function for CDNI and necessary to a deployable solution. This requirement has to be met even if it causes a delay in the delivery by the working group of a deployable solution.

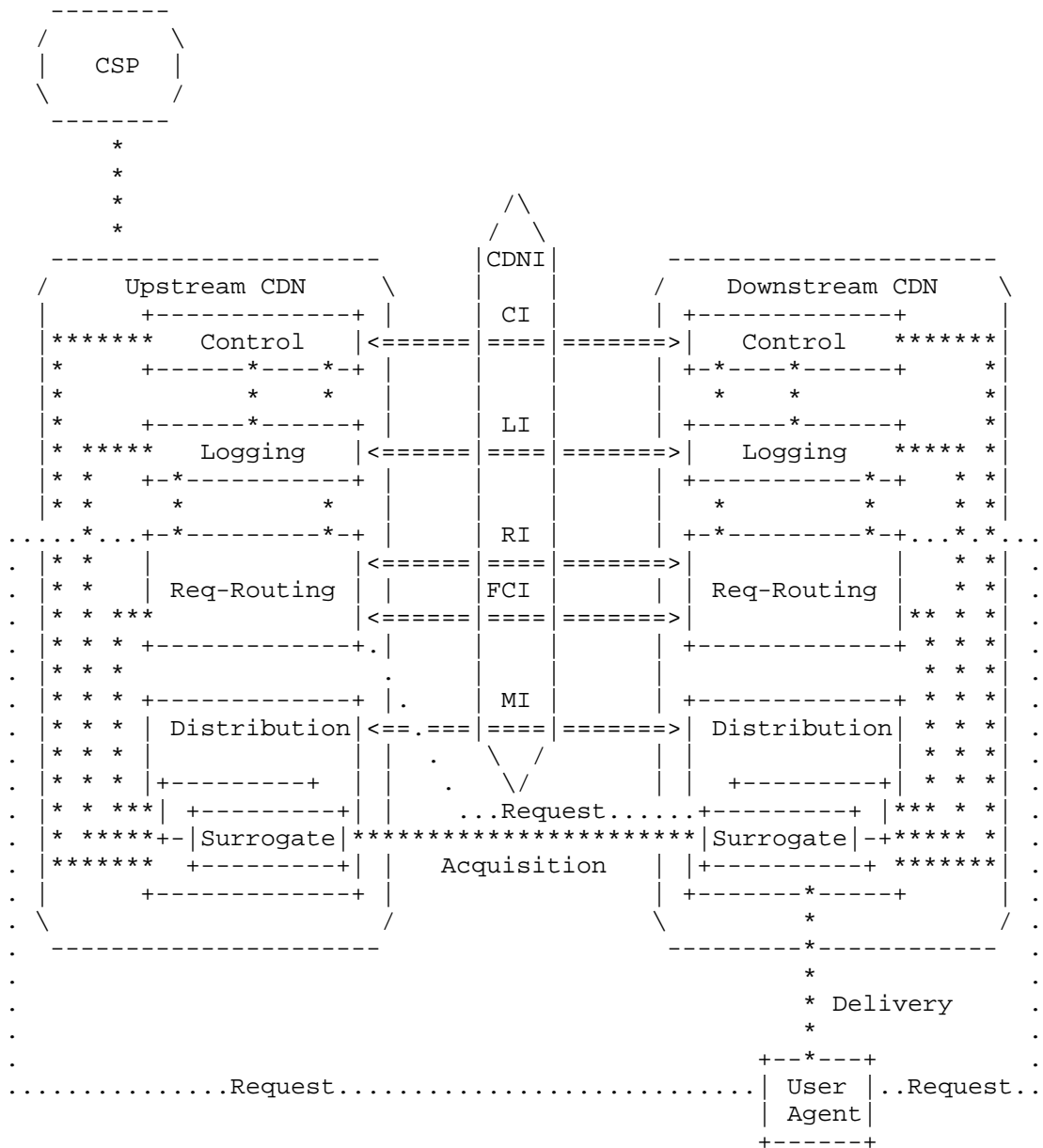
- o "Medium Priority": When a requirement is tagged as "{MED}", it is considered by the working group as an important function for CDNI. This requirement has to be met, unless it is established that attempting to meet this requirement would cause a delay in the delivery by the working group of a deployable solution.
- o "Low Priority": When a requirement is tagged as "{LOW}", it is considered by the working group as a useful function for CDNI. The working group will attempt to meet this requirement as long as it does not prevent meeting the "High Priority" and "Medium Priority" requirements and does not cause a delay in the delivery by the working group of a deployable solution.

2. CDNI Model and CDNI Interfaces

The "CDNI Expanded Model and CDNI Interfaces" figure and brief descriptions of the CDNI interfaces in [I-D.ietf-cdni-framework] are replicated below for convenience. That document contains the definitive reference model and descriptions for the CDNI interfaces.

- o CDNI Control interface (CI): Operations to bootstrap and parameterize the other CDNI interfaces, as well as operations to pre-position, revalidate, and purge both metadata and content. The latter subset of operations is sometimes collectively called the "Trigger interface."
- o CDNI Request Routing interface: Operations to determine what CDN (and optionally what surrogate within a CDN) is to serve end-user's requests. This interface is actually a logical bundling of two separate but related interfaces:
 - * CDNI Footprint & Capabilities Advertisement interface (FCI): Asynchronous operations (as defined in [I-D.ietf-cdni-framework]) to exchange routing information (e.g., the network footprint and capabilities served by a given CDN) that enables CDN selection for subsequent user requests; and
 - * CDNI Request Routing Redirection interface (RI): Synchronous operations (as defined in [I-D.ietf-cdni-framework]) to select a delivery CDN (surrogate) for a given user request.
- o CDNI Metadata interface (MI): Operations to communicate metadata that governs how the content is delivered by interconnected CDNs. Examples of CDNI metadata include geo-blocking directives, availability windows, access control mechanisms, and purge directives. It may include a combination of:

- * Asynchronous operations to exchange metadata that govern subsequent user requests for content; and
- * Synchronous operations that govern behavior for a given user request for content.
- o CDNI Logging interface (LI): Operations that allow interconnected CDNs to exchange relevant activity logs. It may include a combination of:
 - * Real-time exchanges, suitable for runtime traffic monitoring; and
 - * Offline exchanges, suitable for analytics and billing.



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

**** and interfaces outside the scope of CDNI

Figure 1: CDNI Expanded Model and CDNI Interfaces

3. Generic CDNI Requirements

This section identifies generic requirements independent of the individual CDNI interfaces. Some of those are expected to affect multiple or all interfaces. Management is an important aspect of CDNI operation. The fault and performance management is covered in CDNI Logging interface requirements. The other types of management are specific to the CDN provider and not needed for interoperability between CDN providers.

- GEN-1 {MED} Wherever possible, the CDNI interfaces should reuse or leverage existing IETF protocols.

- GEN-2 {HIGH} The CDNI solution shall not require a change, or an upgrade, to the User Agent to benefit from content delivery through interconnected CDNs.

- GEN-3 {HIGH} The CDNI solution shall not require a change, or an upgrade, to the Content Service Provider delivering content through a single CDN, to benefit from content delivery through interconnected CDNs.

- GEN-4 {HIGH} The CDNI solution shall not depend on intra-CDN information to be exposed to other CDNs for effective and efficient delivery of the content. Examples of intra-CDN information include surrogate topology, surrogate status, cached content, etc.

- GEN-5 {HIGH} The CDNI solution shall support CDN interconnection when delivery to the User Agent is based on HTTP [RFC2616]. (Note that while delivery and acquisition "data plane" protocols are out of the CDNI solution scope, the CDNI solution "control plane" protocols are expected to participate in enabling, selecting or facilitating operations of such acquisition and delivery protocols. Hence it is useful to state requirements on the CDNI solution in terms of specifying which acquisition and delivery protocols are to be supported).

- GEN-6 {HIGH} The CDNI solution shall support acquisition across CDNs based on HTTP [RFC2616]. (The note above applies to this requirement too)

- GEN-7 {LOW} The CDNI solution may support delivery to the User Agent based on protocols other than HTTP.

- GEN-8 {LOW} The CDNI solution may support acquisition across CDNs based on protocols other than HTTP.
- GEN-9 {MED} The CDNI solution should support cascaded CDN redirection (CDN1 redirects to CDN2 that redirects to CDN3) to an arbitrary number of levels beyond the first level.
- GEN-10 {MED} The CDNI solution should support an arbitrary topology of interconnected CDNs (i.e. the topology of interconnected CDNs cannot be restricted to a tree, ring, star, etc.).
- GEN-11 {HIGH} The CDNI solution shall prevent looping of any CDNI information exchange.
- GEN-12 {HIGH} When making use of third party reference, the CDNI solution shall consider the potential issues associated with the use of various format of third-party references (e.g. NAT or IPv4/IPv6 translation potentially breaking third-party references based on an IP addresses such as URI containing IPv4 or IPv6 address literals, split DNS situations potentially breaking third-party references based on DNS fully qualified domain names) and wherever possible avoid, minimize or mitigate the associated risks based on the specifics of the environments where the reference is used (e.g. likely or unlikely presence of NAT in the path). In particular, this applies to situations where the CDNI solution needs to construct and convey uniform resource identifiers for directing/redirecting a content request, as well as to situations where the CDNI solution needs to pass on a third party reference (e.g. identify the IP address of a User Agent) in order to allow another entity to make a more informed decision (e.g. make a more informed request routing decision by attempting to derive location information from the third party reference).
- GEN-13 {HIGH} The CDNI solution shall support HTTP Adaptive Streaming content.

4. CDNI Control Interface Requirements

The primary purpose of the CDNI Control interface (CI) is to initiate the interconnection across CDNs, bootstrap the other CDNI interfaces and trigger actions into the Downstream CDN by the Upstream CDN (such as delete object from caches or trigger pre-positioned content acquisition). The working group attempts to align requirements with the appropriate interface; however, solutions to these requirements may apply to a different interface or another interface in addition

to the interface it is associated with.

- CI-1 {HIGH} The CDNI Control interface shall allow the Upstream CDN to request that the Downstream CDN, including downstream cascaded CDNs, delete an object or set of objects and/or its CDNI metadata from the CDN surrogates and any storage. Only the object(s) and CDNI metadata that pertain to the requesting Upstream CDN are allowed to be purged.
- CI-2 {MED} The CDNI Control interface should allow for multiple content items identified by a Content Collection ID to be purged using a single Content Purge action.
- CI-3 {MED} The CDNI Control interface should allow the Upstream CDN to request that the Downstream CDN, including downstream cascaded CDNs, mark an object or set of objects and/or its CDNI metadata as "stale" and revalidate them before they are delivered again.
- CI-4 {HIGH} The CDNI Control interface shall allow the Downstream CDN to report on the completion of these actions (by itself, and including downstream cascaded CDNs), in a manner appropriate for the action (e.g. synchronously or asynchronously). The confirmation receipt should include a success or failure indication. The failure indication and the reason are included if the Downstream CDN cannot delete the content in its storage.
- CI-5 {MED} The CDNI Control interface should support initiation and control by the Upstream CDN of pre-positioned CDNI metadata acquisition by the Downstream CDN.
- CI-6 {MED} The CDNI Control interface should support initiation and control by the Upstream CDN of pre-positioned content acquisition by the Downstream CDN.
- CI-7 {LOW} The CDNI Control interface may allow a CDN to establish, update and terminate a CDN interconnection with another CDN whereby one CDN can act as a Downstream CDN for the other CDN (that acts as an Upstream CDN).
- CI-8 {LOW} The CDNI Control interface may allow control of the CDNI interfaces between any two CDNs independently for each direction (e.g. For the direction where CDN1 is the Upstream CDN and CDN2 is the Downstream CDN, and for the direction where CDN2 is the Upstream CDN and CDN1 is the Downstream CDN).

- CI-9 {LOW} The CDNI Control interface may allow bootstrapping of the CDNI Request Routing interface. For example, this can potentially include:
- * negotiation of the request routing method (e.g. DNS vs HTTP, if more than one method is specified)
 - * discovery of the CDNI Request Routing interface endpoints
 - * information necessary to establish secure communication between the CDNI Request Routing interface endpoints.
- CI-10 {LOW} The CDNI Control interface may allow bootstrapping of the CDNI Metadata interface. This information could, for example, include:
- * discovery of the CDNI Metadata interface endpoints
 - * information necessary to establish secure communication between the CDNI Metadata interface endpoints.
- CI-11 {LOW} The CDNI Control interface may allow bootstrapping of the Content Acquisition interface. This could, for example, include exchange and negotiation of the Content Acquisition methods to be used across the CDNs (e.g. HTTP, HTTPS, FTP, ATIS C2[ATIS-0800042]).
- CI-12 {LOW} The CDNI Control interface may allow bootstrapping of the CDNI Logging interface. This information could, for example, include:
- * discovery of the CDNI Logging interface endpoints
 - * information necessary to establish secure communication between the CDNI Logging interface endpoints
 - * negotiation/definition of the log file format and set of fields to be exported through the logging protocol, with some granularity (e.g. On a per content type basis).
 - * negotiation/definition of parameters related to transaction logs export (e.g., export protocol, file compression, export frequency, directory).

5. CDNI Request Routing Redirection Interface Requirements

The main function of the CDNI Request Routing Redirection interface (RI) is to allow the Request-Routing systems in interconnected CDNs to communicate to facilitate redirection of the request across CDNs.

- RI-1 {HIGH} The CDNI Request Routing Redirection interface shall support efficient request routing for small objects. This may, for example, call for a mode of operation (e.g. DNS-based request routing) where freshness and accuracy of CDN/Surrogate selection can be traded-off against reduced request routing load (e.g. Via lighter-weight queries and caching of request routing decisions).
- RI-2 {HIGH} The CDNI Request Routing Redirection interface shall support efficient request routing for large objects. This may, for example, call for a mode of operation (e.g. HTTP-based request routing) where freshness and accuracy of CDN/Surrogate selection justifies a per-request decision and a per-request CDNI Request-Routing protocol call.
- RI-3 {HIGH} The CDNI Request Routing Redirection interface shall support recursive CDNI request routing.
- RI-4 {HIGH} The CDNI Request Routing Redirection interface shall support iterative CDNI request routing.
- RI-5 {MED} In case of detection of a request redirection loop, the CDNI Request Routing Redirection Interface's loop prevention mechanism should allow redirection of the request on an alternate CDN path (as opposed to the request not being redirected at all).
- RI-6 {MED} The CDNI Request Routing Redirection interface should support a mechanism allowing enforcement of a limit on the number of successive CDN redirections for a given request.
- RI-7 {LOW} The CDNI Request Routing Redirection interface may support a mechanism allowing an Upstream CDN to avoid redirecting a request to a Downstream CDN if that is likely to result in the total redirection time exceeding some limit.
- RI-8 {HIGH} The CDNI Request Routing Redirection interface shall allow the Upstream CDN to include, in the query to the Downstream CDN, the necessary information to allow the Downstream CDN to process the redirection query. This could, for example, include:

- * information from which the geographic region pertaining to the IP address of the User Agent that originated the request can be inferred (e.g. User Agent fully qualified domain name in case of HTTP-based Request Routing, DNS Proxy fully qualified domain name in case of DNS-based Request Routing)
 - * requested resource information (e.g. Resource URI in case of HTTP-based Request Routing, Resource hostname in case of DNS-based Request Routing)
 - * additional available request information (e.g. request headers in case of HTTP-based Request Routing).
- RI-9 {LOW} The CDNI Request Routing Redirection interface may also allow the Upstream CDN to convey information pointing to CDNI metadata applicable (individually or through inheritance) to the requested content. For illustration, the CDNI metadata pointed to could potentially include metadata that is applicable to any content, metadata that is applicable to a content collection (to which the requested content belongs) and/or metadata that is applicable individually to the requested content.
- RI-10 {HIGH} The CDNI Request Routing Redirection interface shall allow the Downstream CDN to include the following information in the response to the Upstream CDN:
- * status code, in particular indicating acceptance or rejection of request (e.g. Because the Downstream CDN is unwilling or unable to serve the request). In case of rejection, an error code is also to be provided, which allows the Upstream CDN to react appropriately (e.g. Select another Downstream CDN, or serve the request itself)
 - * redirection information (e.g. Resource URI in case of HTTP-based Request Routing, equivalent of a DNS record in case of DNS-based Request Routing).
- RI-11 {HIGH} The CDNI Request Routing Redirection interface shall allow for per-chunk request routing of HTTP Adaptive Streaming content.
- RI-12 {LOW} The CDNI Request Routing Redirection interface may allow the Upstream CDN to use the information conveyed by the Downstream CDN during the Recursive Request Routing process to rewrite an HTTP Adaptive Streaming manifest file.

- RI-13 {LOW} The CDNI Request-Routing interface may allow the Upstream CDN to re-compute the message digest or digital signature over the invariant portion of the chunk URIs embedded in the HTTP Adaptive Streaming manifest file.
- RI-14 {MED} The CDNI Request Routing Redirection interface should correlate the HTTP Adaptive Stream manifest file to the related chunks referenced in the manifest file.
- RI-15 {MED} The CDNI Request Routing Redirection interface should allow for an efficient method of transferring request routing information for multiple chunks from the Downstream CDN to the Upstream CDN as part of the recursive request routing process.

6. CDNI Footprint & Capabilities Advertisement Interface Requirements

The main function of the CDNI Footprint & Capabilities Advertisement interface (FCI) is to allow the Downstream CDN to advertise the information regarding its footprint and capabilities to the Upstream CDN.

- FCI-1 {HIGH} The CDNI Footprint & Capabilities Advertisement interface shall allow the Downstream CDN to communicate to the Upstream CDN coarse information about the Downstream CDN ability and/or willingness to handle requests from the Upstream CDN. For example, this could potentially include a binary signal ("Downstream CDN ready/not-ready to take additional requests from Upstream CDN") to be used in case of excessive load or failure condition in the Downstream CDN.
- FCI-2 {MED} The CDNI Footprint & Capabilities Advertisement interface should allow the Downstream CDN to communicate to the Upstream CDN aggregate information to facilitate CDN selection during request routing, such as Downstream CDN capabilities, resources and affinities (i.e. Preferences or cost). This information could, for example, include:
- * supported content types and delivery protocols
 - * footprint (e.g. layer-3 coverage)
 - * a set of metrics/attributes (e.g. Streaming bandwidth, storage resources, distribution and delivery priority)
 - * a set of affinities (e.g. Preferences, indication of distribution/delivery fees)

- * information to facilitate request redirection (e.g. Reachability information of Downstream CDN Request Routing system).

[Note: Some of this information - such as supported content types and delivery protocols- may also potentially be taken into account by the distribution system in the Upstream CDN for pre-positioning of content and/or metadata in the Downstream CDN in case of pre-positioned content acquisition and/or pre-positioned CDNI metadata acquisition.]

FCI-3 {MED} In the case of cascaded redirection, the CDNI Footprint & Capabilities Advertisement interface should allow the Downstream CDN to also include in the information communicated to the Upstream CDN, information on the capabilities, resources and affinities of CDNs to which the Downstream CDN may (in turn) redirect requests received by the Upstream CDN. In that case, the CDNI Request-Routing interface shall prevent looping of such information exchange.

FCI-4 {LOW} The CDNI Footprint & Capabilities Advertisement interface may allow the Downstream CDN to communicate to the Upstream CDN aggregate information on CDNI administrative limits and policy. This information can be taken into account by the Upstream CDN Request Routing system in its CDN Selection decisions. This information could, for example, include:

- * maximum number of requests redirected by the Upstream CDN to be served simultaneously by the Downstream CDN
- * maximum aggregate volume of content (e.g. in Terabytes) to be delivered by the Downstream CDN over a time period.

FCI-5 {MED} The CDNI Footprint & Capabilities Advertisement interface should support advertisement of the following types of capabilities:

- * delivery protocol (e.g., HTTP vs. RTMP)
- * acquisition protocol (for acquiring content from an Upstream CDN)
- * redirection mode (e.g., DNS Redirection vs. HTTP Redirection)
- * capabilities related to CDNI Logging (e.g., supported logging mechanisms)

- * capabilities related to CDNI Metadata (e.g., authorization algorithms or support for proprietary vendor metadata)

- FCI-6 {LOW} The CDNI Control interface may allow exchange and negotiation of delivery authorization mechanisms to be supported across the CDNs (e.g. URI signature based validation).
- FCI-7 {HIGH} The CDNI Footprint & Capabilities Advertisement interface shall support extensible fields used to convey the CDN capabilities and methods to indicate the footprint in the advertisement from the Downstream CDN to the Upstream CDN.

7. CDNI Metadata Interface Requirements

The primary function of the CDNI Metadata interface (MI) is to allow the Distribution system in interconnected CDNs to communicate to ensure Content Distribution Metadata with inter-CDN scope can be exchanged across CDNs. We observe that while the CDNI Metadata Distribution protocol is currently discussed as a single "protocol", further analysis will determine whether the corresponding requirements are to be realized over a single interface and protocol, or over multiple interfaces and protocols. For example, a subset of the CDNI metadata might be conveyed in-band along with the actual content acquisition across CDNs (e.g. content MD5 in HTTP header) while another subset might require an out-of-band interface & protocol (e.g. geo-blocking information).

- MI-1 {HIGH} The CDNI Metadata interface shall allow the Upstream CDN to provide the Downstream CDN with content distribution metadata of inter-CDN scope.
- MI-2 {HIGH} The CDNI Metadata interface shall support exchange of CDNI metadata for both the dynamic content acquisition model and the pre-positioning content acquisition model.
- MI-3 {HIGH} The CDNI Metadata interface shall support a mode where no, or a subset of, the Metadata is initially communicated to the Downstream CDN along with information about how/where to acquire the rest of the CDNI Metadata (i.e. Dynamic CDNI metadata acquisition).
- MI-4 {MED} The CDNI Metadata interface should support a mode where all the relevant Metadata is initially communicated to the Downstream CDN (i.e. Pre-positioned CDNI metadata acquisition).

- MI-5 {HIGH} Whether in the pre-positioned content acquisition model or in the dynamic content acquisition model, the CDNI Metadata interface shall provide the necessary information to allow the Downstream CDN to acquire the content from an upstream source (e.g. Acquisition protocol and Uniform Resource Identifier in Upstream CDN- or rules to construct this URI).
- MI-6 {HIGH} The CDNI metadata shall allow signaling of one or more upstream sources, where each upstream source can be in the Upstream CDN, in another CDN, the CSP origin server or any arbitrary source designated by the Upstream CDN. Note that some upstream sources (e.g. the content origin server) may or may not be willing to serve the content to the Downstream CDN, if this policy is known to the Upstream CDN then it may omit those sources when exchanging CDNI metadata.
- MI-7 {HIGH} The CDNI Metadata interface (possibly in conjunction with the CDNI Control interface) shall allow the Upstream CDN to request addition and modification of CDNI Metadata into the Downstream CDN.
- MI-8 {HIGH} The CDNI Metadata interface (possibly in conjunction with the CDNI Control interface) shall allow removal of obsolete CDNI Metadata from the Downstream CDN (this could, for example, be achieved via an explicit removal request from the Upstream CDN or via expiration of a Time-To-Live associated to the Metadata).
- MI-9 {HIGH} The CDNI Metadata interface shall allow association of CDNI Metadata at the granularity of individual object. This is necessary to achieve fine-grain Metadata distribution at the level of an individual object when necessary.
- MI-10 {HIGH} The CDNI Metadata interface shall allow association of CDNI Metadata at the granularity of an object set. This is necessary to achieve scalable distribution of metadata when a large number of objects share the same distribution policy.
- MI-11 {HIGH} The CDNI Metadata interface shall support multiple levels of inheritance with precedence to more specific metadata. For example, the CDNI Metadata Distribution protocol may support metadata that is applicable to any content, metadata that is applicable to a content collection and metadata that is applicable to an individual content where content level metadata overrides content collection metadata that overrides metadata for any content.

- MI-12 {HIGH} The CDNI Metadata interface shall ensure that conflicting metadata with overlapping scope are prevented or deterministically handled.
- MI-13 {HIGH} The CDNI Metadata interface shall allow signaling of content distribution control policies. For example, this could potentially include:
- * geo-blocking information (i.e. Information defining geographical areas where the content is to be made available or blocked)
 - * availability windows (i.e. Information defining time windows during which the content is to be made available or blocked; expiration time may also be included to remove content)
 - * delegation whitelist/blacklist (i.e. Information defining which Downstream CDNs the content may/may not be delivered through)
- MI-14 {HIGH} The CDNI Metadata interface shall be able to exchange a set of metadata elements with specified semantics (e.g. start of time window, end of time window).
- MI-15 {HIGH} The CDNI Metadata interface shall allow exchange of opaque metadata element, whose semantic is not defined in CDNI but established by private CDN agreement.
- MI-16 {HIGH} The CDNI Metadata 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 the need to validate information (e.g. Expiry time, Client IP address) required for access authorization.
- MI-17 {MED} The CDNI Metadata interface should allow signaling of CDNI-relevant surrogate cache behavior parameters. For example, this could potentially include:
- * control of whether the query string of HTTP URI is to be ignored by surrogate cache
 - * enforcement of caching directives by Downstream CDN that are different than the ones signalled in the HTTP headers (e.g. "Expires" field)

- * rate-pacing by Downstream CDN for content delivery (e.g. Progressive Download)
- MI-18 {HIGH} The CDNI Metadata interface shall provide indication of related content (e.g. HTTP Adaptive Bit Rate chunks) by the Content Collection ID (CCID) metadata. This could be used by the Downstream CDN for operations on the group of content. For example, this could potentially include:
- * content acquisition for the entire set of files when one piece of content is requested
 - * local file management and storage bundles all the files for the content
 - * purging the entire set of files associated with the content
 - * logging of the delivery of the content for the session when at least one file in the set was delivered
- MI-19 {MED} The CDNI Metadata interface should support an optional mechanism allowing the Upstream CDN to indicate to the Downstream CDN which CDNI Log fields are to be provided for all content items, for specific sets of content items, or for specific content items delivered using HTTP. A CDNI implementation that does not support this optional CDNI Metadata Distribution interface mechanism shall ignore this log format indication and generate CDNI logging format for HTTP Adaptive Streaming using the default set of CDNI Logging fields. (Note: This function may be part of the CDNI Metadata interface or the CDNI Control interface.)
- MI-20 {MED} The CDNI Metadata interface should allow the Upstream CDN to signal to the Downstream CDN the Content Collection ID value for all, for specific sets of, or for specific content items delivered using HTTP. Whenever the Downstream CDN is instructed by the Upstream CDN to report the Content Collection ID field in the log records, the Downstream CDN is to use the value provided through the CDNI Metadata interface for the corresponding content. Note the Session ID field along with Content Collection ID may be used for HTTP Adaptive Streaming content.
- MI-21 {MED} The CDNI Metadata interface should allow the Upstream CDN to signal to the Downstream CDN the Authorization Group ID value for all the related HTTP Adaptive Streaming content (i.e. manifest file and chunks). The authorization result of

a content (e.g. manifest file) is transferred over to related content (e.g. chunks).

- MI-22 {HIGH} The CDNI Metadata interface shall support extensible format for CDNI metadata delivery from the Upstream CDN to the Downstream CDN.

8. CDNI Logging Interface Requirements

This section identifies the requirements related to the CDNI Logging interface (LI). We observe that while the CDNI Logging interface is currently discussed as a single "protocol", further analysis will determine whether the corresponding requirements are to be realized over a single interface and protocol, or over multiple interfaces and protocols.

- LI-1 {HIGH} The CDNI logging architecture and interface shall ensure reliable transfer of CDNI logging information across CDNs.
- LI-2 {HIGH} The CDNI Logging interface shall provide logging of deliveries and incomplete deliveries to User Agents performed by the Downstream CDN as a result of request redirection by the Upstream CDN.
- LI-3 {MED} In the case of cascaded CDNs, the CDNI Logging interface should allow the Downstream CDN to report to the Upstream CDN logging for deliveries and incomplete deliveries performed by the Downstream CDN itself as well as logging for deliveries and incomplete deliveries performed by cascaded CDNs on behalf of the Downstream CDN.
- LI-4 {HIGH} The CDNI Logging interface shall support batch/offline exchange of logging records.
- LI-5 {MED} The CDNI Logging interface should also support an additional mechanism taking into account the timing constraints for some types of logging records (e.g. near-real time for monitoring and analytics applications).
- LI-6 {HIGH} The CDNI Logging interface shall define a log file format and a set of fields to be exported for various CDNI logging events.

- LI-7 {HIGH} The CDNI Logging interface shall define a transport mechanism to exchange CDNI Logging files.
- LI-8 {MED} The CDNI Logging interface should allow a CDN to query another CDN for relevant current logging records (e.g. For on-demand access to real-time logging information).
- LI-9 {LOW} The CDNI Logging interface may support aggregate/summarized logs (e.g. total bytes delivered for a content regardless of individual User Agents to which it was delivered).
- LI-10 {LOW} The CDNI Logging interface may support logging of performance data for deliveries to User Agents performed by the Downstream CDN as a result of request redirection by the Upstream CDN. Performance data may include various traffic statistics (the specific parameters are to be determined). The CDNI Logging interface may support the Upstream CDN to indicate the nature and contents of the performance data to be reported by the Downstream CDN.
- LI-11 {MED} The CDNI Logging interface should support logging of consumed resources (e.g. storage, bandwidth) to the Upstream CDN for deliveries where content is stored by the Downstream CDN for delivery to User Agents. The information logged may include the type of storage (e.g., Origin, Intermediate, Edge, Cache) as well as the amount of storage (e.g., total GB, GB used, per time period, per content domain) all of which may impact the cost of the services.
- LI-12 {MED} In the case of cascaded CDNs, the CDNI Logging interface should support the Downstream CDN to report consumed resources (e.g. storage, bandwidth) to the Upstream CDN where content is stored by the Downstream CDN itself as well as logging for storage resources when content storage is performed by cascaded CDNs on behalf of the Downstream CDN.
- LI-13 {HIGH} The CDNI Logging interface shall support logging of deleted objects from the Downstream CDN to the Upstream CDN as a result of explicit delete requests on via the CDNI Control interface from the Upstream CDN.
- LI-14 {HIGH} The CDNI Logging interface shall support the exchange of extensible log file formats to support proprietary information fields. These information fields shall be agreed upon ahead of time between the corresponding CDNs.

- LI-15 {HIGH} The CDNI Logging interface shall allow a CDN to notify another CDN about which CDNI logging information is available for transfer and/or no longer available (e.g. it exceeded some logging retention period or some logging retention volume).
- LI-16 {MED} The CDNI Logging interface should support the ability for the Downstream CDN to include the Content Collection ID and Session ID fields in CDNI log entries generated for HTTP Adaptive Streaming content.
- LI-17 {MED} The CDNI Logging interface should provide privacy protection by not disclosing information that can be used to identify the user (e.g. method that anonymizes the IP address carried in the logging field). The use of the privacy protection mechanism is optional.

9. CDNI Security Requirements

This section identifies the requirements related to the CDNI security. Some of these are expected to affect multiple or all protocols.

- SEC-1 {HIGH} All the CDNI interface shall support secure operation over unsecured IP connectivity (e.g. The Internet). This includes authentication, confidentiality, integrity protection as well as protection against spoofing and replay.
- SEC-2 {HIGH} The CDNI solution shall provide sufficient protection against Denial of Service attacks. This includes protection against spoofed delivery requests sent by User Agents directly to a Downstream CDN attempting to appear as if they had been redirected by a given Upstream CDN when they have not.
- SEC-3 {MED} The CDNI solution should be able to ensure that for any given request redirected to a Downstream CDN, the Downstream CDN can determine the Upstream CDN that redirected the request directly to the Downstream CDN (leading to that request being served by that CDN, or being further redirected).
- SEC-4 {MED} The CDNI solution should be able to ensure that for any given transaction log generated by the Downstream CDN and communicated to an Upstream CDN, the Upstream CDN can confirm the transmitted log record corresponds to a request redirection by the Upstream CDN.

SEC-5 {LOW} The CDNI solution may provide a mechanism allowing an Upstream CDN that has credentials to acquire content from the CSP origin server (or another CDN), to allow establishment of credentials authorizing the Downstream CDN to acquire the content from the CSP origin server (or the other CDN) (e.g. In case the content cannot be acquired from the Upstream CDN).

10. IANA Considerations

This document makes no request of IANA.

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

11. Security Considerations

This document discusses CDNI security requirements in Section 9.

12. Contributors

This document reflects the contributions from the following authors:

Francois Le Faucheur

Cisco Systems

flefauch@cisco.com

Mahesh Viveganandhan

Cisco Systems

mvittal@cisco.com

Grant Watson

Alcatel-Lucent (Velocix)

gwatson@velocix.com

13. Acknowledgements

This document leverages the earlier work of the IETF CDI working group in particular as documented in [I-D.cain-request-routing-req],

[I-D.amini-cdi-distribution-reqs] and [I-D.gilletti-cdn-aaa-reqs].

The authors would like to thank Gilles Bertrand, Christophe Caillet, Bruce Davie, Phil Eardley, Ben Niven-Jenkins, Agustin Schapira, Emile Stephan, Eric Burger, Susan He, Kevin Ma, Daryl Malas, Iuniana Oprescu, and Spencer Dawkins for their input. Serge Manning along with Robert Streijl, Vishwa Prasad, Percy Tarapore, Mike Geller, and Ramki Krishnan contributed to this document by addressing the requirements of the ATIS Cloud Services Forum.

Ray Brandenburg, Matt Caufield, and Gilles Bertrand provided valuable inputs for HTTP Adaptive Streaming, CDNI Metadata interface, and CDNI Logging interface, respectively.

Stephen Farrell, Adrian Farrel, Benoit Claise, Sean Turner, Christer Holmberg, and Carlos Pignataro provided review comments that helped improve the document.

14. References

14.1. Normative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-07 (work in progress), November 2013.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

14.2. Informative References

- [ATIS-0800042]
"ATIS IPTV Content on Demand Service,
<https://www.atis.org/docstore/product.aspx?id=25670>",
December 2010.
- [I-D.amini-cdi-distribution-reqs]
Amini, L., "Distribution Requirements for Content Internetworking", draft-amini-cdi-distribution-reqs-02 (work in progress), November 2001.
- [I-D.cain-request-routing-req]
Cain, B., "Request Routing Requirements for Content Internetworking", draft-cain-request-routing-req-03 (work in progress), November 2001.

- [I-D.gilletti-cdnp-aaa-reqs]
"CDI AAA Requirements,
draft-gilletti-cdnp-aaa-reqs-01.txt", June 2001.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2",
RFC 4949, August 2007.
- [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.
- [RTMP] "Adobe's Real Time Messaging Protocol, [http://
www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/
rtmp_specification_1.0.pdf](http://www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf)", December 2012.

Authors' Addresses

Kent Leung (editor)
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
U.S.A.

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

Yiu Lee (editor)
Comcast
One Comcast Center
Philadelphia, PA 19103
U.S.A.

Email: yiulee@cable.comcast.com

CDNI
Internet-Draft
Intended status: Standards Track
Expires: April 17, 2013

K. Leung
F. Le Faucheur
M. Caulfield
Cisco Systems
Oct 14, 2012

URI Signing for CDN Interconnection (CDNI)
draft-leung-cdni-uri-signing-01

Abstract

This document describes how the concept of URI signing supports the content access control requirements of CDNI and proposes a candidate URI signing scheme.

The proposed URI signing method specifies the information needed to be included in the URI and the algorithm used to authorize and to validate access request for the content referenced by the URI.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 17, 2013.

Copyright Notice

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

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

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. URI Signing Overview	4
2. Authorization Attributes in URI Signing	5
3. URI Signing and Validation	6
4. Considerations for CDNI Interfaces	9
4.1. CDNI Capabilities Advertisement	9
4.2. CDNI Metadata Interface	9
4.3. CDNI Logging Interface	10
5. URI Signing Operation	10
5.1. HTTP Redirection	10
5.2. DNS Redirection	13
6. HTTP Adaptive Bit Rate	16
7. IANA Considerations	16
8. Security Considerations	17
9. Acknowledgements	17
10. References	18
10.1. Normative References	18
10.2. Informative References	18
Authors' Addresses	18

1. Introduction

The overall problem space for CDN Interconnection is described in [RFC6707].

The CDNI Problem Statement [RFC6707], the CDN requirements document [I-D.ietf-cdni-requirements] and the CDNI Framework document [I-D.ietf-cdni-framework] discuss the need for the interconnected CDNs to be able to implement an access control mechanism that enforces the Content Service Provider (CSP) distribution policy.

Specifically, [I-D.ietf-cdni-framework] states:

"The CSP may also trust the CDN operator to perform actions such as ..., and to enforce per-request authorization performed by the CSP using techniques such as URI signing."

In particular, the following requirement is listed in [I-D.ietf-cdni-requirements]:

"META-17 [HIGH] The CDNI Metadata Distribution interface shall allow signaling of authorization checks and validation that are to be performed by the surrogate before delivery. For example, this could potentially include:

- * need to validate URI signed information (e.g. Expiry time, Client IP address)."

This document proposes a URI signing scheme that allows Surrogates in interconnected CDNs to enforce a per-request authorization performed by the CSP. Splitting the role of performing per-request authorization by CSP and the role of validation of this authorization by the CDN allows any arbitrary distribution policy to be enforced across CDNs without the need of CDNs to have any awareness of the actual CSP distribution policy.

1.1. Terminology

This document uses the terminology defined in [RFC6707].

This document also uses the terminology of [RFC2104] including the following terms (reproduced here for convenience):

- o MAC: message authentication code
- o HMAC: hash-based message authentication code (HMAC)

- o HMAC-SHA1: HMAC instantiation using SHA1 as the cryptographic hash function
- o HMAC-MD5: HMAC instantiation using MD5 as the cryptographic hash function

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

- o URI Signature: message digest that is computed with an algorithm that uses the key, the Original URI and request attributes as inputs to the hash function. This digest is conveyed inside the Signed URI.
- o Original URI: the URI before URI signing is applied.
- o Signed URI: the URI containing the Original URI, the attributes and the URI Signature.

1.2. URI Signing Overview

URI Signing is an authorization method for content delivery. This is based on embedding the URI with information that can be validated to ensure the request has legitimate access to the content. There are two parts: 1) attributes that convey authorization restrictions (e.g. source IP address and time period), and 2) message digest that confirms the integrity and authenticity of the URI provided by the URI creator. The authorization attributes can be anything agreed upon between the entity that creates the URI and the entity that validates the URI. A key is used by the HMAC algorithm of the URI signing function to generate the message digest (i.e. sign the URI). A key is also used by the HMAC algorithm of the URI signature validating function to validate the message digest (i.e. URI signature). The two functions may or may not use the same key.

Two types of keys can be used for URI Signing: asymmetric keys and symmetric key. Asymmetric keys always have a key pair made up of a public key and private key. The private key and public key are used for signing and validating the URI, respectively. A symmetric key is used for both functions. Regardless of the type of key, the entity that validates the URI has to obtain the key. There are very different requirements for key distribution with asymmetric keys and with symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent another party from getting access to the key, since it could then generate valid Signed URIs for unauthorized requests. Key distribution for asymmetric keys does not require confidentiality since public keys can typically be distributed openly (because they cannot be used for URI signing) and private keys are kept by the URI signing function.

URI Signing operates in the following way. After request authorisation, the CSP computed a Signed URO from the Original URI and provides the signed URI to the user out of band. The user request for the Signed URI is handled by the CDN which is responsible for validating the URI Signature before delivering the content.

2. Authorization Attributes in URI Signing

This section identifies the set of attributes that may be needed to enforce the CSP distribution policy. These attributes can therefore be covered by the URI Signature hash and can be embedded (by the signing function) in the as query component of the Signed URI (to enable subsequent signature validation by the signature validating function).

In order to provide flexibility in distribution policies to be enforced, the exact subset of attributes used for URI signature in a given request is a deployment decision. The defined keyword for each query string attribute is specified in parenthesis below.

- o Version (VER) - An integer used for identifying the version of URI signing method with its set of capabilities.
- o Expiry Time (ET) - Time in seconds when URI Signature expires since midnight 1/1/1970 UTC (i.e. UNIX epoch).
- o Client IP (CIP) - IP address of the client, in a dotted decimal format.
- o Key Owner (KO) - Identifier of the owner of the key used for URI signing, in an integer format.
- o Key ID (KN) - A number that is used as an index, within the set of keys of a given Key Owner, to the key used for URI signing, in an integer format.
- o Hash Function (HF) - A string used for identifying the hash function to compute the URI signature (e.g. "MD5", "SHA1").
- o Algorithm (ALG) - An integer used for identifying the algorithm to compute the URI signature.
- o Client ID (CID) - Identifier of the client such as IMSI, MSISDN, MEID, MAC address, etc.

The query string attributes are embedded within the Signed URI to be used for the content request in order to provide to the signature

validating function the information needed to enforce the distribution policy and to validate the URI Signature. Each of the attributes is further described below.

The Version attribute indicates which version of URI signing scheme is used (including which attributes and algorithms are supported). The present document specifies Version 0. More versions may be defined in the future.

The Expiry Time attribute ensures that the content authorization expires after a predetermined time. This limits the time window for content access and prevents replay of the request beyond the authorized time window.

The Client IP attribute is used to restrict content access to a particular End User, based on its client IP address for whom the content access was authorized.

The Key Owner and Key ID attributes are used to identify the key that is to be retrieved as input to the HMAC algorithm to compute the message digest for validating the signed URI.

The Hash function attribute indicates the HMAC hash function to be used for message digest computation.

The Algorithm indicates the specific algorithm for computation of the URI Signature. For example, this indicates whether the scheme component of the URI is to be covered by the signature computation or not.

The Client ID attribute is used to restrict content access to a particular user associated with this identifier. For example, it could be the information about the subscriber, device, or network access interface.

3. URI Signing and Validation

The keyword for embedding the actual URI Signature in the URI query string is "US".

The following steps are taken for signing a URI for the algorithms defined in this document. Note that some steps may be skipped if the attribute is not needed to enforce the distribution policy. The entire URI (i.e. scheme, authority, path, query, and fragment as defined in URI Generic Syntax [RFC3986]) is protected by the URI signature when the algorithm (i.e. "ALG") is set to 1. The scheme is removed from the URI when the algorithm is set to 2. This allows

the URI signature to be validated correctly in the case when a client performs a fallback to HTTP for a content referenced by an URI with RTSP scheme.

1. Check if the Original URI already contains a query string. If not, append a "?" character. If yes, append an "&" character.
2. Append the string "VER=0". This represents the version of URI Signing specified in this document.
3. Append the string "&ET=".
4. Get the current time in seconds since epoch (as an integer). Add the validity time in seconds as an integer.
5. Append this integer.
6. Append the string "&CIP=".
7. Append the client's IP address in dotted decimal format.
8. Append the string "&KO=".
9. Append the numeric value of the key owner corresponding to the key being used.
10. Append the string "&KN=".
11. Append the key ID number corresponding to the key being used.
12. Append the string "&HF=".
13. Append the string for the type of hash function.
14. Append the string "&ALG=".
15. Append the integer for the type of algorithm. If algorithm is "1", no additional logic needed by default. If algorithm is "2", remove the scheme part of the URI.
16. Append the string "&US=".
17. Store this as the message on which to compute the hash-based message authentication code (e.g. `http://example.com/content.mov?VER=0&ET=1209422976&CIP=171.71.50.123&KO=1&KN=2&HF=1&ALG=1&US=`).

18. For symmetric key, compute the message digest (i.e. URI signature) using the algorithm with key and message as inputs to the hash function. For asymmetric keys, after the message digest computation (as described previously only using the public key), use the public key again to encrypt the message digest.
19. Convert the message digest to its equivalent human readable hexadecimal value (e.g. f08b56f46075813e44b2d4888628a471).
20. Append this hexadecimal value to the previously created message. This is the complete Signed URI.

The following steps are taken for validating a Signed URI. Note that some steps are to be skipped if the corresponding attribute is not embedded in the Signed URI. The absence of a given attribute indicates enforcement of its purpose is not necessary in the distribution policy.

1. Check if the Signed URI contains a query string. If not, it is not a Signed URI. If the CDNI Metadata for the corresponding content indicate that access control is to be enforced via URI Signing, then the request is denied.
2. Extract the value from "US=" part of URI. This value is the URI signature.
3. Extract the values from "KO=" and "KN=" part of URI. Use these values to locate the key value and also key type (i.e. asymmetric or symmetric)
4. Extract the value from "HF=" part of URI. The value is the type of hash function.
5. Extract the value from "ALG=" part of URI. The value is the type of algorithm.
6. Store URI excluding the part after "US=" as the message on which to compute the hash-based message authentication code.
7. If the extracted algorithm value is "1", keep message without change. If algorithm value is "2", remove the scheme part of the URI in the message.
8. Compute the message digest (i.e. URI signature) using the algorithm with key and message as inputs to the hash function (based on the extracted hash function value).

9. For symmetric key, compare this computed digest with the received URI Signature. For asymmetric keys, decrypt the URI Signature with the public key. Then compare the computed digest with the decrypted URI Signature. If the comparison is not a match, the request is denied. Otherwise, continue with next step. Note that the request is denied if any of the following validations failed.
10. Validate that the request came from the same IP address as indicated in the "CIP=".
11. Validate that the request arrived before expiration time as indicated in the "ET=" based on the current time.

4. Considerations for CDNI Interfaces

The CDNI Interfaces need enhancements to support URI Signing. A Downstream CDN that supports URI Signing needs to be able to advertise this capability to the Upstream CDN. The Upstream CDN selects a Downstream CDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the Upstream CDN need to be able to distribute via the CDNI Metadata interface the information necessary to allow the Downstream CDN to validate a Signed URI . Events that pertain to URI Signing (e.g. request denial or delivery after access authorization) need to be included in the logs communicated through the CDNI Logging interface.

4.1. CDNI Capabilities Advertisement

The Downstream CDN advertises its capability to support URI Signing via the CDNI Request Routing/Footprint & Capabilities Advertisement interface. The supported version of URI Signing needs to be included. TBD: to be taken into account by Footprint & Capabilities design team working on this area.

- o URI Signing support and its version

4.2. CDNI Metadata Interface

The following CDNI metadata are specified for URI Signing. Note that the Key Owner and Key ID information are not needed if only one key is provided by CSP or Upstream CDN for the content or set of contents covered by the CDNI metadata. Also, the CDNI metadata for HMAC algorithm is not needed when the Algorithm attribute is embedded in the signed URI. TBD: CDNI Metadata Interface is work in progress.

- o Content access control indication.
- o Type of access control. Specifically, access to content is subject to URI Signing
- o Key value along with its key index (i.e. Key Owner and Key ID) and type (asymmetric or symmetric) used for validating URI signature
- o List of Downstream CDNs authorized for key distribution (i.e. trust relationship between CSP and CDNs) [Editor's Note: is this needed?]
- o Algorithm for HMAC to be used for validation.

4.3. CDNI Logging Interface

The Downstream CDN reports that enforcement of the access control was applied to the request for content delivery. TBD: CDNI Logging interface is work in progress.

- o URI signature validation events (e.g. invalid client IP address, expired signed URI, incorrect URI signature, successful validation)
- o Delivery log with confirmation of access control enforcement (i.e. Delivery CDN enforced URI Signing before content delivery)

5. URI Signing Operation

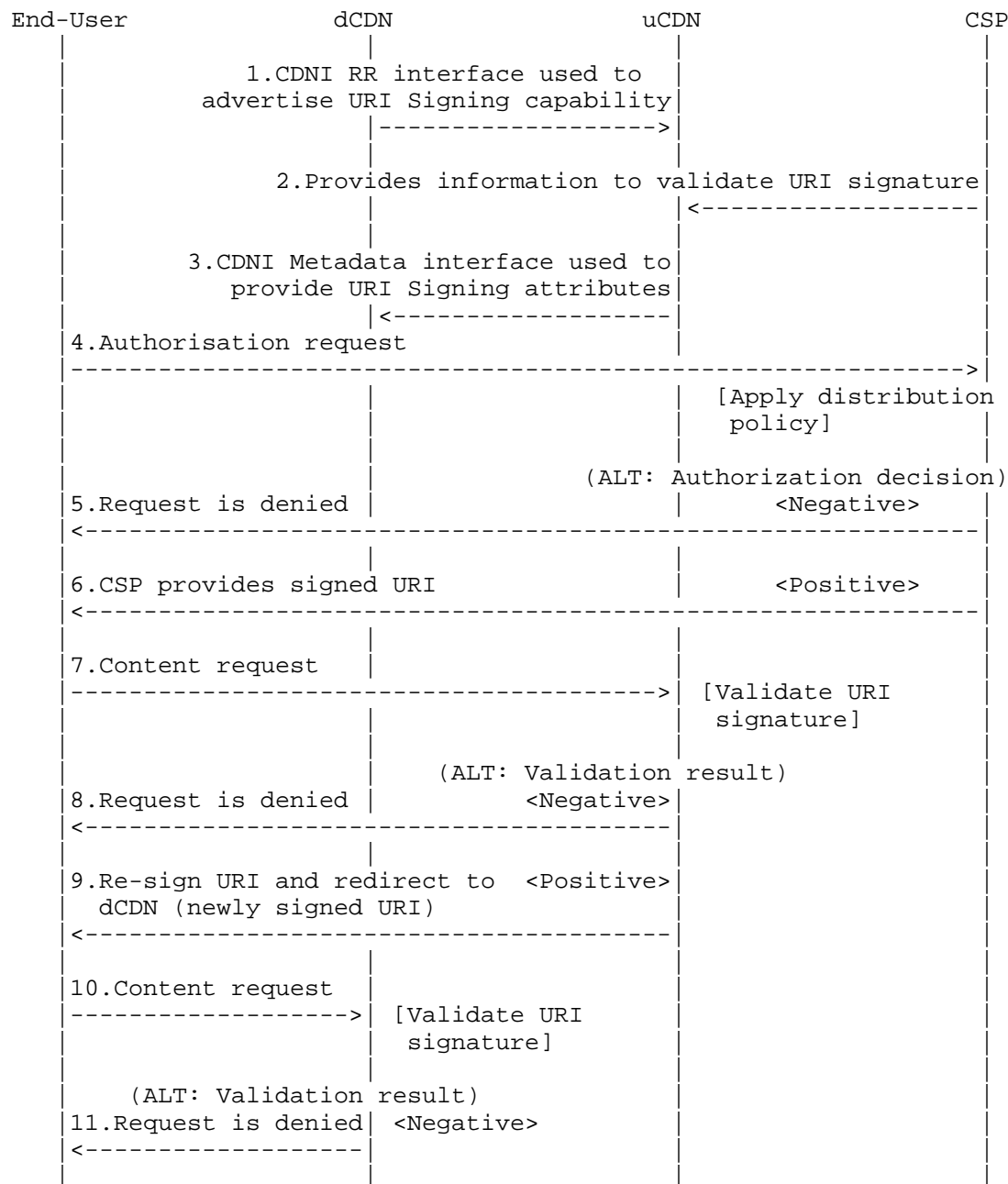
URI Signing supports both the HTTP-based and DNS-based request routing. HMAC [RFC2104] defines a hash-based message authentication code allowing two parties that share a symmetric key or asymmetric keys to establish the integrity and authenticity of a set of information (e.g. a message) through a cryptographic hash function.

5.1. HTTP Redirection

For HTTP-based request routing, HMAC is applied to a set of information that is unique to a given end user content request using key information that is specific to a pair of adjacent CDNI hops (e.g. between the CSP and the Authoritative CDN, between the Authoritative CDN and a Downstream CDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI signing scheme described below is based on the following

steps (assuming HTTP redirection, iterative request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



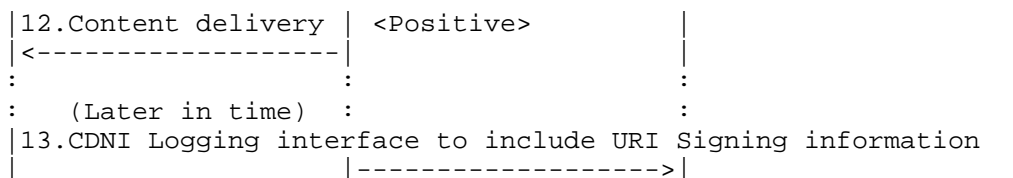


Figure 1: HTTP-based Request Routing with URI Signing

1. Using the CDNI Request Routing/Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.
2. CSP provides to the Authoritative CDN the information needed to validate URI signatures from that CSP. For example, this information may include a hashing function, algorithm, and a key value.
3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate URI signatures from the Authoritative CDN for the given CSP. For example, this information may include a hashing algorithm and private key corresponding to the trust relationship between the Authoritative CDN and the Downstream CDN.
4. On receipt of a given authorisation request on the CSP portal, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy.
5. If the authorization decision is negative, the CSP rejects the request.
6. If the authorization decision is positive, the CSP computes a Signed URI that is based on unique parameters of that request and conveys it to the end user as the URI to use to request the content.
7. On receipt of the corresponding content request, the authoritative CDN validates the URI Signature in the URI using the information provided by the CSP.
8. If the validation is negative, the authoritative CDN rejects the request
9. If the validation is positive, the authoritative CDN computes a Signed URI that is based on unique parameters of that request

and provides to the end user as the URI to use to further request the content from the Downstream CDN

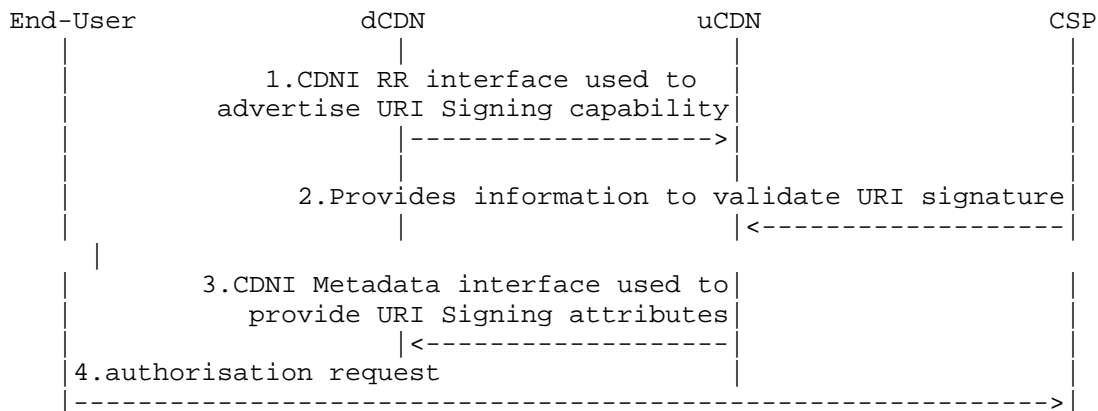
10. On receipt of the corresponding content request, the Downstream CDN validates the URI Signature in the Signed URI using the information provided by the Authoritative CDN in the CDNI Metadata
11. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.
12. If the validation is positive, the Downstream CDN serves the request and delivers the content.
13. At a later time, Downstream CDN reports logging events that includes URI signing information.

With HTTP-based request routing, URI Signing matches well the general chain of trust model of CDNI both with symmetric key and asymmetric keys because the key information only need to be specific to a pair of adjacent CDNI hops.

5.2. DNS Redirection

For DNS-based request routing, HMAC is applied to a set of information that is unique to a given end user content request using a secret key shared between CSP and the Delivery CDN. The Delivery CDN needs to obtain the key information to validate the Signed URL, which is computed by the CSP based on its distribution policy.

The URI signing scheme described below is based on the following steps (assuming iterative DNS request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



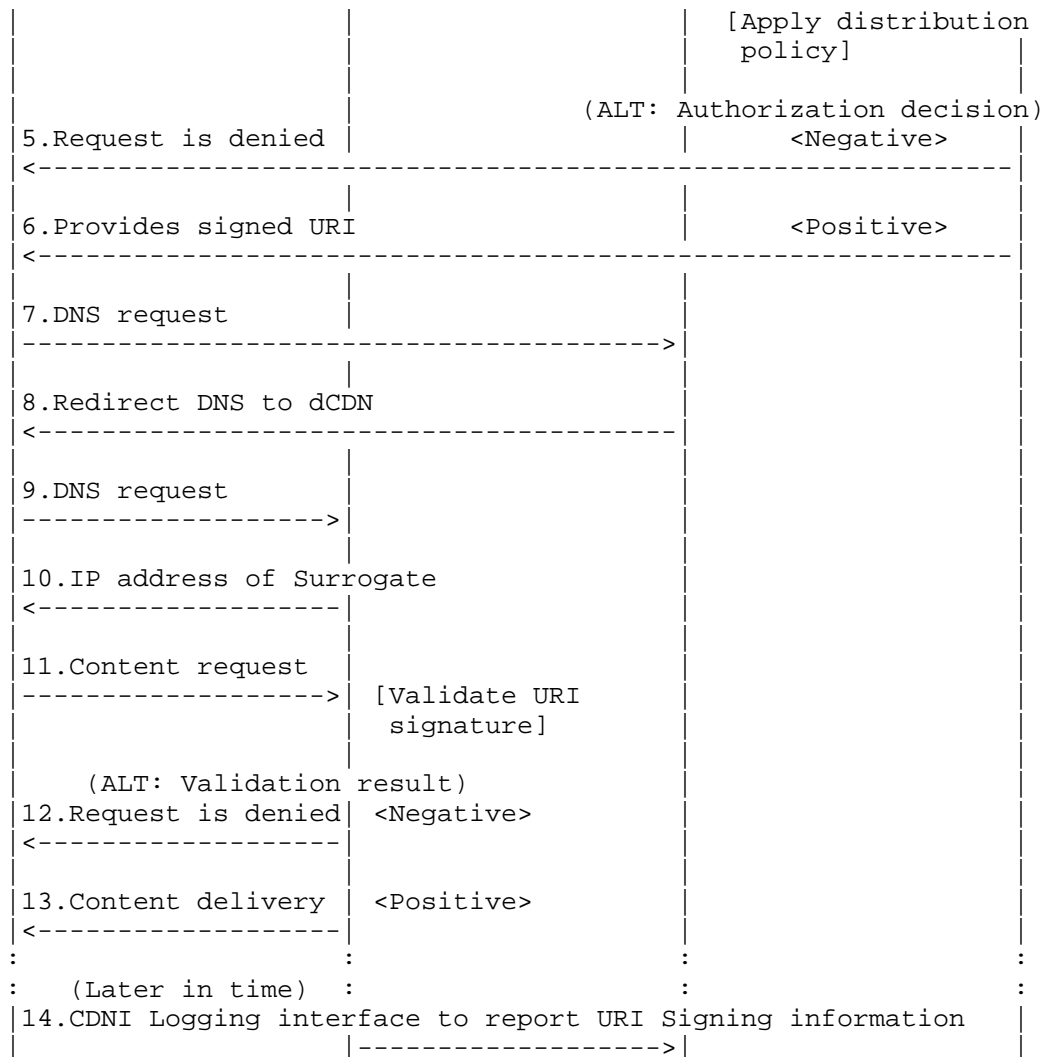


Figure 2: DNS-based Request Routing with URI Signing

1. Using the CDNI Request Routing interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.
2. CSP provides to the Authoritative CDN the information needed to validate cryptographic signatures from that CSP. For example, this information may include a hash function, algorithm, and a key.

3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate cryptographic signatures from the CSP (i.e. private key between CSP and participating CDNs). This requires a relationship between CSP and Downstream CDN. The CDNI metadata specifies CDNs with trust relationships according to the CSP. The set of Downstream CDNs is limited by this criteria.
4. On receipt of a given authorisation request on the CSP portal, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy.
5. If the authorization decision is negative, the CSP rejects the request
6. If the authorization decision is positive, the CSP computes a cryptographic signature that is based on unique parameters of that request and includes it in the URI provided to the end user to request the content.
7. End user sends DNS request to the authoritative CDN.
8. On receipt of the DNS request, the authoritative CDN redirects the request to the Downstream CDN.
9. End user sends DNS request to the Downstream CDN.
10. On receipt of the DNS request, the Downstream CDN responds with IP address of one of its Surrogates.
11. On receipt of the corresponding content request, the Downstream CDN validates the cryptographic signature in the URI using the information provided by the Authoritative CDN in the CDNI Metadata
12. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.
13. If the validation is positive, the Downstream CDN serves the request and delivers the content.
14. At a later time, Downstream CDN reports logging events that includes URI signing information.

With DNS-based request routing, URI Signing matches well the general chain of trust model of CDNI when used with asymmetric keys because the only key information that need to be distributed across multiple CDNI hops including non-adjacent hops is the public key, that is

generally not confidential.

With DNS-based request routing, URI Signing does match well the general chain of trust model of CDNI when used with symmetric keys because the symmetric key information needs to be distributed across multiple CDNI hops including non-adjacent hops. This raises a security concern for applicability of URI Signing with Symmetric keys in case of DNS-based inter-CDN request routing.

6. HTTP Adaptive Bit Rate

TBD - HTTP ABR calls for specific support by URI Signing ("flexible URI signing") as discussed in [I-D.brandenburg-cdni-has]. This will be added in a future version of this document.

7. IANA Considerations

This document requests IANA to create a new registry for CDNI URI Signing. The following query string attribute names (a.k.a. keywords) are assigned for the authorization attributes used in CDNI URI Signing. There is no intention to claim any query string attribute for URI beyond the CDNI URI Signing context. That means the entities that sign the URI or validate the URI signature comply to the keywords specified in the query string for the URI Signing function only when URI Signing is used and only in the context of CDNI.

- o US (URI signature)
- o VER (Version)
- o ET (Expiry time)
- o CIP (Client IP address)
- o KO (Key owner)
- o KN (Key ID)
- o HF (Hash Function)
- o ALG (Algorithm)
- o CID (Client ID)

This document requests IANA to create a registry for each of the

defined query string attribute and assign the following values for the authorization attribute:

VER: 0 (Base)

HF: "MD5", "SHA1", "SHA256"

ALG: 1 (Full URI), 2 (URI without scheme)

CID: "MAC:<value>", "IMSI:<value>", "MSISDN:<value>", "MEID:<value>", "NAI:<value>" (TBD)

8. Security Considerations

A symmetric key needs to be shared by the entity that produces the URI signature and the entity that validates the URI signature. In the case of DNS-based request routing, CSP that signed the URI may not have a relationship with the Downstream CDN that validates the signed URI. In this case, the Upstream CDN shall select only the Downstream CDN with a relationship with CSP. Otherwise, asymmetric keys should be used for DNS-based request routing. The Downstream CDN only needs to use the CSP's public key to validate the signed URI. Asymmetric keys method does not require a trust relationship between the two entities participating in URI Signing (i.e. signing function and signature validating function).

For HTTP-based request routing, the two entities participating in URI Signing are always the adjacent Upstream CDN and Downstream CDN because of the hop by hop nature of the redirection. Therefore, either symmetric key or asymmetric keys can be used because the adjacent Upstream CDN and Downstream CDN have a relationship.

The following security threats are identified (TBD):

- o Client IP address spoofing
- o Illegitimate client behind a NAT
- o Replay of request

9. Acknowledgements

TBD

10. References

10.1. Normative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-01 (work in progress), July 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-03 (work in progress), June 2012.
- [I-D.ietf-cdni-use-cases]
Bertrand, G., Emile, S., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", draft-ietf-cdni-use-cases-10 (work in progress), August 2012.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

10.2. Informative References

- [I-D.brandenburg-cdni-has]
Brandenburg, R., Deventer, O., Faucheur, F., and K. Leung, "Models for adaptive-streaming-aware CDN Interconnection", draft-brandenburg-cdni-has-03 (work in progress), July 2012.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

Authors' Addresses

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose 95134
USA

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

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

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

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

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

CDNI
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2014

K. Leung
F. Le Faucheur
Cisco Systems
B. Downey
Verizon Labs
R. van Brandenburg
TNO
S. Leibrand
Limelight Networks
March 4, 2014

URI Signing for CDN Interconnection (CDNI)
draft-leung-cdni-uri-signing-05

Abstract

This document describes how the concept of URI signing supports the content access control requirements of CDNI and proposes a URI signing scheme.

The proposed URI signing method specifies the information needed to be included in the URI and the algorithm used to authorize and to validate access requests for the content referenced by the URI. Some of the information may be accessed by the CDN via configuration or CDNI metadata.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 5, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Background on URI Signing	4
1.3. CDNI URI Signing Overview	6
1.4. URI Signing in a non-CDNI context	8
2. Signed URI Information Elements	8
2.1. Enforcement Information Elements	10
2.2. Signature Computation Information Elements	11
2.3. URI Signature Information Elements	12
2.4. URI Signing Package Attribute	13
3. Creating the Signed URI	14
3.1. Calculating the URI Signature	14
3.2. Packaging the URI Signature	17
4. Validating a URI Signature	18
4.1. Information element validation	18
4.2. Signature validation	19
5. Relationship with CDNI Interfaces	21
5.1. CDNI Control Interface	22
5.2. CDNI Footprint & Capabilities Advertisement Interface	22
5.3. CDNI Request Routing Redirection Interface	22
5.4. CDNI Metadata Interface	23
5.5. CDNI Logging Interface	24
6. URI Signing Message Flow	24
6.1. HTTP Redirection	25
6.2. DNS Redirection	27
7. HTTP Adaptive Streaming	30
8. IANA Considerations	30
9. Security Considerations	31
10. Privacy	33
11. Acknowledgements	33
12. References	33
12.1. Normative References	33
12.2. Informative References	33
Authors' Addresses	34

1. Introduction

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of interconnected CDNs (CDNI). The primary goal of URI Signing is to make sure that only authorized User Agents (UAs) are able to access the content, with a Content Service Provider (CSP) being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as DRM, are more appropriate.

The overall problem space for CDN Interconnection (CDNI) is described in CDNI Problem Statement [RFC6707]. In this document, along with the CDNI Requirements [I-D.ietf-cdni-requirements] document and the CDNI Framework [I-D.ietf-cdni-framework] the need for interconnected CDNs to be able to implement an access control mechanism that enforces the CSP's distribution policy is described.

Specifically, CDNI Framework [I-D.ietf-cdni-framework] states:

"The CSP may also trust the CDN operator to perform actions such as ..., and to enforce per-request authorization performed by the CSP using techniques such as URI signing."

In particular, the following requirement is listed in CDNI Requirements [I-D.ietf-cdni-requirements]:

"MI-16 [HIGH] The CDNI Metadata Distribution interface shall allow signaling of authorization checks and validation that are to be performed by the surrogate before delivery. For example, this could potentially include:

- * need to validate URI signed information (e.g. Expiry time, Client IP address)."

This document proposes a URI Signing scheme that allows Surrogates in interconnected CDNs to enforce a per-request authorization performed by the CSP. Splitting the role of performing per-request authorization by CSP and the role of validation of this authorization by the CDN allows any arbitrary distribution policy to be enforced across CDNs without the need of CDNs to have any awareness of the actual CSP distribution policy.

1.1. Terminology

This document uses the terminology defined in CDNI Problem Statement [RFC6707].

This document also uses the terminology of Keyed-Hashing for Message Authentication (HMAC) [RFC2104] including the following terms (reproduced here for convenience):

- o MAC: message authentication code.
- o HMAC: Hash-based message authentication code (HMAC) is a specific construction for calculating a MAC involving a cryptographic hash function in combination with a secret key.
- o HMAC-SHA1: HMAC instantiation using SHA-1 as the cryptographic hash function.
- o HMAC-MD5: HMAC instantiation using MD5 as the cryptographic hash function.

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

- o URI Signature: Message digest or digital signature that is computed with an algorithm for protecting the URI.
- o Original URI: The URI before URI Signing is applied.
- o Signed URI: Any URI that contains a URI Signature.
- o Target CDN URI: Embedded URI created by the CSP to direct UA towards the Upstream CDN. The Target CDN URI can be signed by the CSP and verified by the Upstream CDN.
- o Redirection URI: URI created by the Upstream CDN to redirect UA towards the Downstream CDN. The Redirection URI can be signed by the Upstream CDN and verified by the Downstream CDN. In a cascaded CDNI scenario, there can be more than one Redirection URI.

1.2. Background on URI Signing

The next section provides an overview of how URI Signing works in a CDNI environment. As background information, URI Signing is first explained in terms of a single CDN delivering content on behalf of a CSP.

A CSP and CDN are assumed to have a trust relationship that enables the CSP to authorize access to a content item by including a set of attributes in the URI before redirecting a UA to the CDN. Using these attributes, it is possible for a CDN to check an incoming content request to see whether it was authorized by the CSP (e.g. based on the UA's IP address or a time window). Of course, the

attributes need to be added to the URI in a way that prevents a UA from changing the attributes, thereby leaving the CDN to think that the request was authorized by the CSP when in fact it wasn't. For this reason, a URI Signing mechanism includes in the URI a message digest or digital signature that allows a CDN to check the authenticity of the URI. The message digest or digital signature can be calculated based on a shared secret between the CSP and CDN or using CSP's asymmetric public/private key pair, respectively.

Figure 1, shown below, presents an overview of the URI Signing mechanism in the case of a CSP with a single CDN. When the UA browses for content on CSP's website (#1), it receives HTML web pages with embedded content URIs. Upon requesting these URIs, the CSP redirects to a CDN, creating a Target CDN URI (#2) (alternatively, the Target CDN URI itself is embedded in the HTML). The Target CDN URI is the Signed URI which may include the IP address of the UA and/or a time window and always contains the URI Signature which is generated by the CSP using the shared secret or a private key. Once the UA receives the response with the embedded URI, it sends a new HTTP request using the embedded URI to the CDN (#3). Upon receiving the request, the CDN checks to see if the Signed URI is authentic by verifying the URI signature. In addition, it checks whether the IP address of the HTTP request matches that in the Signed URI and if the time window is still valid. After these values are confirmed to be valid, the CDN delivers the content (#4).

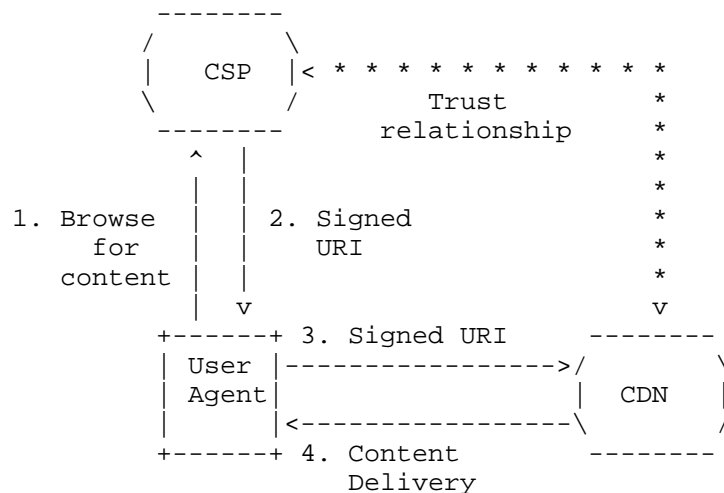


Figure 1: Figure 1: URI Signing in a CDN Environment

1.3. CDNI URI Signing Overview

In a CDNI environment, URI Signing operates the same way in the initial steps #1 and #2 but the later steps involve multiple CDNs in the process of delivering the content. The main difference from the single CDN case is a redirection step between the Upstream CDN and the Downstream CDN. In step #3, UA may send HTTP request or DNS request. Depending on whether HTTP-based or DNS-based request routing is used, the Upstream CDN responds by directing the UA towards the Downstream CDN using either a Redirection URI (which is a Signed URI generated by the Upstream CDN) or a DNS reply, respectively (#4). Once the UA receives the response, it sends the Redirection URI/Target CDN URI to the Downstream CDN (#5). The received URI is validated by the Downstream CDN before delivering the content (#6). This is depicted in the figure below. Note: The CDNI call flows are covered in Detailed URI Signing Operation (Section 6).

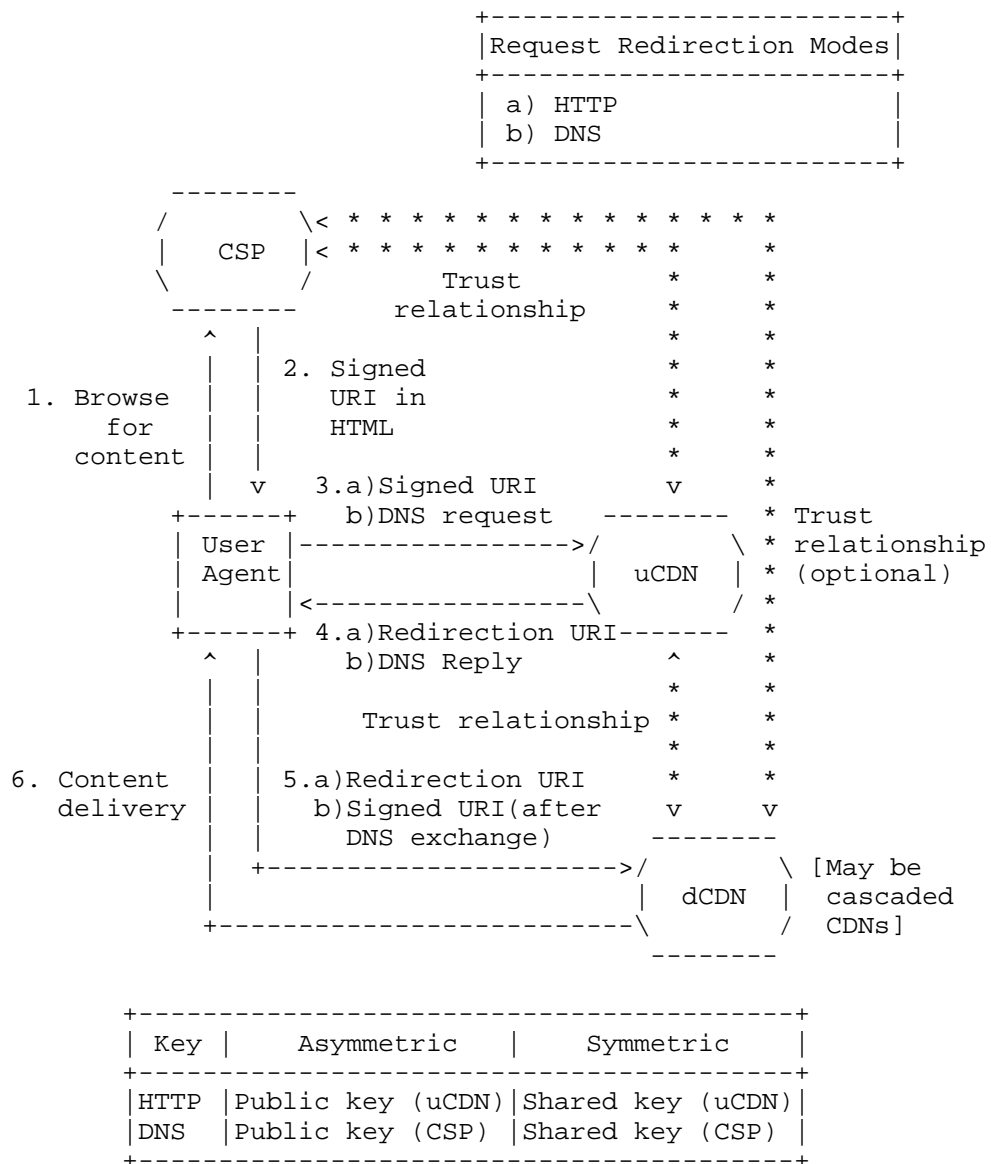


Figure 2: URI Signing in a CDNI Environment

The trust relationships between CSP, Upstream CDN, and Downstream CDN have direct implications for URI Signing. In the case shown in Figure 2, the CDN that the CSP has a trust relationship with is the Upstream CDN. The delivery of the content may be delegated to the Downstream CDN, which has a relationship with the Upstream CDN but

may have no relationship with the CSP.

In CDNI, there are two methods for request routing: DNS-based and HTTP-based. For DNS-based request routing, the Signed URI (i.e. Target CDN URI) provided by the CSP reaches the Downstream CDN directly. In the case where the Downstream CDN does not have a trust relationship with the CSP, this means that only an asymmetric public/private key method can be used for computing the URI Signature because the CSP and Downstream CDN are not able to exchange symmetric shared secret keys. Since the CSP is unlikely to have relationships with all the Downstream CDNs that are delegated to by the Upstream CDN, the CSP may choose to allow the Authoritative CDN to redistribute the shared key to a subset of their Downstream CDNs.

For HTTP-based request routing, the Signed URI (i.e. Target CDN URI) provided by the CSP reaches the Upstream CDN. After this URI has been verified to be correct by the Upstream CDN, the Upstream CDN creates and signs a new Redirection URI to redirect the UA to the Downstream CDN. Since this new URI also has a new URI Signature, this new signature can be based around the trust relationship between the Upstream CDN and Downstream CDN, and the relationship between the Downstream CDN and CSP is not relevant. Given the fact that such a relationship between Upstream CDN and Downstream CDN always exists, both asymmetric public/private keys and symmetric shared secret keys can be used for URI Signing. Note that the signed Redirection URI SHOULD maintain the same level of security as the original Signed URI.

1.4. URI Signing in a non-CDNI context

While the URI signing scheme defined in this document was primarily created for the purpose of allowing URI Signing in CDNI scenarios, e.g. between a uCDN and a dCDN or between a CSP and a dCDN, there is nothing in the defined URI Signing scheme that precludes it from being used in a non-CDNI context. As such, the described mechanism could be used in a single-CDN scenario such as shown in Figure 1 in Section 1.2, for example to allow a CSP that uses different CDNs to only have to implement a single URI Signing mechanism.

2. Signed URI Information Elements

The concept behind URI Signing is based on embedding in the Target CDN URI/Redirection URI a number of information elements that can be validated to ensure the UA has legitimate access to the content. These information elements are appended, in an encapsulated form, to the original URI.

For the purposes of the URI signing mechanism described in this document, three types of information elements may be embedded in the URI:

- o Enforcement Information Elements: Information Elements that are used to enforce a distribution policy defined by the CSP. Examples of enforcement attributes are IP address of the UA and time window.
- o Signature Computation Information Elements: Information Elements that are used by the CDN to verify the URI signature embedded in the received URI. In order to verify a URI Signature, the CDN requires some information elements that describe how the URI Signature was generated. Examples of Signature Computation Elements include the used HMACs hash function and/or the key identifier.
- o URI Signature Information Elements: The information elements that carry the actual message digest or digital signature representing the URI signature used for checking the integrity and authenticity of the URI. A typical Signed URI will only contain one embedded URI Signature Information Element.

In addition, the this document specifies the following URI attribute:

- o URI Signing Package Attribute: The URI attribute that encapsulates all the URI Signing information elements in an encoded format. Only this attribute is exposed in the Signed URI as a URI query parameter.

If the UA or another entity needs to add one or more attributes to the Signed URI for purposes other than URI Signing, those attributes MUST be appended after the URI Signing Packacke Attribute. Any attributes appended in such way after the URI Signature has been calculated are not validated for the purpose of content access authorization. Note that adding any such attributes to the Signed URI before the URI Signing Packacke Attribute will cause the URI Signing validation to fail.

Two types of keys can be used for URI Signing: asymmetric keys and symmetric keys. Asymmetric keys are based on a public/private key pair mechanism and always contain a private key only known to the entity signing the URI (either CSP or uCDN) and a public key for the verification of the Signed URI. With symmetric keys, the same key is used by both the signing entity for signing the URI as well as by the validating entity for validating the Signed URI. Regardless of the type of keys used, the validating entity has to obtain the key (either the public or the symmetric key). There are very different

requirements for key distribution (out of scope of this document) with asymmetric keys and with symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent another party from getting access to the key, since it could then generate valid Signed URIs for unauthorized requests. Key distribution for asymmetric keys does not require confidentiality since public keys can typically be distributed openly (because they cannot be used for URI signing) and private keys are kept by the URI signing function.

2.1. Enforcement Information Elements

This section identifies the set of information elements that may be needed to enforce the CSP distribution policy. New information elements may be introduced in the future to extend the capabilities of the distribution policy.

In order to provide flexibility in distribution policies to be enforced, the exact subset of information elements used in the URI Signature of a given request is a deployment decision. The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to enforce the distribution policy:

- o Expiry Time (ET) [optional] - Time when the Signed URI expires. This is represented as an integer denoting the number of seconds since midnight 1/1/1970 UTC (i.e. UNIX epoch). The request is rejected if the received time is later than this timestamp. Note: The time, including time zone, on the entities that generate and validate the signed URI need to be in sync (e.g. NTP is used).
- o Client IP (CIP) [optional] - IP address of the client for which this Signed URI is generated. This is represented in dotted decimal format for IPv4 or canonical text representation for IPv6 address [RFC5952] . The request is rejected if sourced from a client with a different IP address.

The Expiry Time Information Element ensures that the content authorization expires after a predetermined time. This limits the time window for content access and prevents replay of the request beyond the authorized time window.

The Client IP Information Element is used to restrict content access to a particular User Agent, based on its IP address for whom the content access was authorized.

Note: See the Security Considerations (Section 9) section on the

limitations of using an expiration time and client IP address for distribution policy enforcement.

2.2. Signature Computation Information Elements

This section identifies the set of information elements that may be needed to verify the URI (signature). New information elements may be introduced in the future if new URI signing algorithms are developed.

The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to validate the URI by recreating the URI Signature.

- o Version (VER) [optional] - An integer used for identifying the version of URI signing method. If this Information Element is not present in the URI Signing Package Attribute, the default version is 1.
- o Key ID (KID) [optional] - A string used for obtaining the key (e.g. database lookup, URI reference) which is needed to validate the URI signature.
- o Hash Function (HF) [optional] - A string used for identifying the hash function to compute the URI signature (e.g. "MD5", "SHA-1", "SHA-256", "SHA-3") with HMAC. If this Information Element is not present in the URI Signing Package Attribute, the default hash function is SHA-1.
- o Digital Signature Algorithm (DSA) [optional] - Algorithm used to calculate the Digital Signature (e.g. "RSA", "DSA", "EC-DSA"). If this Information Element is not present in the URI Signing Package Attribute, the default is EC-DSA.

The Version Information Element indicates which version of URI signing scheme is used (including which attributes and algorithms are supported). The present document specifies Version 1. If the Version attribute is not present in the Signed URI, then the version is obtained from the CDNI metadata, else it is considered to have been set to the default value. More versions may be defined in the future.

The Key ID Information Element is used to retrieve the key which is needed as input to the algorithm for validating the Signed URI. The method used for obtaining the actual key from the reference included in the Key ID Information Element is outside the scope of this

document.

The Hash Function Information Element indicates the hash function to be used for HMAC-based message digest computation. The Hash Function Information Element is used in combination with the Message Digest Information Element defined in section Section 2.3.

The Digital Signature Algorithm Information Element indicates the digital signature function to be in the case asymmetric keys are used. The Digital Signature Algorithm Information Element is used in combination with the Digital Signature Information Element defined in section Section 2.3.

2.3. URI Signature Information Elements

This section identifies the set of information elements that carry the URI Signature that is used for checking the integrity and authenticity of the URI.

The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to carry the actual URI Signature.

- o Message Digest (MD) [mandatory for symmetric key] - A string used for the message digest generated by the URI signing entity.
- o Digital Signature (DS) [mandatory for asymmetric keys] - A string used for the digital signature provided by the URI signing entity.

The Message Digest attribute contains the message digest used to validate the Signed URI when symmetric keys are used.

The Digital Signature attribute contains the digital signature used to verify the Signed URI when asymmetric keys are used.

In the case of symmetric key, HMAC algorithm is used for the following reasons: 1) Ability to use hash functions (i.e. no changes needed) with well understood cryptographic properties that perform well and for which code is freely and widely available, 2) Easy to replace the embedded hash function in case faster or more secure hash functions are found or required, 3) Original performance of the hash function is maintained without incurring a significant degradation, and 4) Simple way to use and handle keys.

In the case of asymmetric keys, Elliptic Curve Digital Signature Algorithm (EC DSA) - a variant of DSA - is used because of the

following reasons: 1) Key size is small while still offering good security, 2) Key is easy to store, and 3) Computation is faster than DSA or RSA.

2.4. URI Signing Package Attribute

The URI Signing Package Attribute is an encapsulation container for the URI Signing Information Elements defined in the previous sections. The URI Signing Information Elements are encoded and stored in this attribute. URI Signing Package Attribute is appended to the Original URI to create the Signed URI.

The primary advantage of the URI Signing Package Attribute is that it avoids having to expose the URI Signing Information Elements directly in the query string of the URI, thereby reducing the potential for a namespace collision space within the URI query string. A side benefit of the attribute is the obfuscation performed by the URI Signing Package Attribute hides the information (e.g. client IP address) from view of the common user, who is not aware of the encoding scheme. Obviously, this is not a security method since anyone who knows the encoding scheme is able to obtain the clear text. Note that any parameters appended to the query string after the URI Signing Package Attribute are not validated and hence do not affect URI Signing.

The following attribute is used to carry the encoded set of URI Signing attributes in the Signed URI.

- o URI Signing Package (URISigningPackage) - The encoded attribute containing all the CDNI URI Signing Information Elements used for URI Signing.

The URI Signing Package Attribute contains the URI Signing Information Elements in the Base-64 encoding with URL and Filename Safe Alphabet (a.k.a. "base64url") as specified in the Base-64 Data Encoding [RFC4648] document. The URI Signing Package Attribute is the only URI Signing attribute exposed in the Signed URI. The attribute MUST be the last parameter in the query string of the URI when the Signed URI is generated. However, a client or CDN may append other query parameters unrelated to URI Signing to the Signed URI. Such additional query parameters SHOULD NOT use the same name as the URI Signing Package Attribute to avoid namespace collision and potential failure of the URI Signing validation.

The parameter name of the URI Signing Package Attribute shall be defined in the CDNI Metadata interface. If the CDNI Metadata interface does not include a parameter name for the URI Signing Package Attribute, the parameter name is set by configuration ((out

of scope of this document).

3. Creating the Signed URI

The following procedure for signing a URI defines the algorithms in this version of URI Signing. Note that some steps may be skipped if the CSP does not enforce a distribution policy and the Enforcement Information Elements are therefore not necessary. A URI (as defined in URI Generic Syntax [RFC3986]) contains the following parts: scheme name, authority, path, query, and fragment. The entire URI except the "scheme name" part is protected by the URI signature. This allows the URI signature to be validated correctly in the case when a client performs a fallback to another scheme (e.g. HTTP) for a content item referenced by a URI with a specific scheme (e.g. RTSP). The benefit is that the content access is protected regardless of the type of transport used for delivery. If the CSP wants to ensure a specific protocol is used for content delivery, that information is passed by CDNI metadata. Note: Support for changing of the URL scheme requires that the default port is used, or that the protocols must both run on the same non-standard port.

The process of generating a Signed URI can be divided into two sets of steps: calculating the URI Signature and packaging the URI Signature and appending it to the Original URI. Note it is possible to use some other algorithm and implementation as long as the same result is achieved. An example for the Original URI, "http://example.com/content.mov", is used to clarify the steps.

3.1. Calculating the URI Signature

Calculate the URI Signature by following the procedure below.

1. Copy the Original URI, excluding the "scheme name" part, into a buffer to hold the message for performing the operations below.
2. Check if the URI already contains a query string. If not, append a "?" character. If yes, append an "&" character.
3. If the version is the default value, skip this step. Append the string "VER=". Append the string for the version number.
4. If time window enforcement is not needed, step 4 can be skipped.
 - A. If an attribute was added to the URI, append an "&" character. Append the string "ET=". Note in the case of re-signing a URI, the attribute is carried over from the received Signed URI.

- B. Get the current time in seconds since epoch (as an integer). Add the validity time in seconds as an integer. Note in the case of re-signing a URI, the value MUST remain the same as the received Signed URI.
 - C. Convert this integer to a string and append to the message.
5. If client IP enforcement is not needed, step 5 can be skipped.
- A. If an attribute was added to the URI, append an "&" character. Append the string "CIP=". Note in the case of re-signing a URI, the attribute is carried over from the received Signed URI.
 - B. Convert the client's IP address in dotted decimal notation format (i.e. for IPv4 address) or canonical text representation (for IPv6 address [RFC5952]) to a string and append to the message. Note in the case of re-signing an URI, the value MUST remain the same as the received Signed URI.
6. Depending on the type of key used to sign the URI, compute the message digest or digital signature for symmetric key or asymmetric keys, respectively.
- A. For symmetric key, HMAC is used.
 - 1. Obtain the shared key to be used for signing the URI.
 - 2. If the key identifier is provided by the CDNI metadata, skip this step. If an attribute was added to the URI, append an "&" character. Append the string "KID=". Append the key identifier (e.g. "example:keys:123") needed by the entity to locate the shared key for validating the URI signature.
 - 3. If the hash function for the HMAC uses the default value (SHA-1), skip this step. If an attribute was added to the URI, append an "&" character. Append the string "HF=". Append the string for the type of hash function. Note that re-signing a URI MUST use the same hash function as the received Signed URI or one of the allowable hash functions designated by the CDNI metadata.
 - 4. If an attribute was added to the URI, append an "&" character. Append the string "MD=". The message now contains the complete section of the URI that is protected (e.g. "://example.com/

```
content.mov?ET=1209422976&CIP=10.0.0.1&
KID=example:keys:123&MD=").
```

5. Compute the message digest using the HMAC algorithm with the shared key (e.g. "secretkey" and message as the two inputs to the hash function which is specified by the "HF" attribute.
6. Convert the message digest to its equivalent hexadecimal format.
7. Append the string for the message digest (e.g. "://example.com/content.mov?ET=1209422976&CIP=10.0.0.1&KID=example:keys:123&MD=da58513e8b309c1e8a9695baceba629d180b50b8").

B. For asymmetric keys, EC DSA is used.

1. Generate the EC private and public key pair (e.g. private key is "8b5b417336492707a83836b02ceee55b3847be5ec1521e4949977b224950e708", public key is "04840b1be11cfd1404c2fc588d30150a4103cadcc4172e786bcafl5d7feeb6d246f7d8a91fa055cb10efb2f52860d1dlb2f339244e9ad79a23e10ed9b720f6157f"). Store the EC public key in a location that's reachable for any entity that needs to validate the URI signature.
2. If the key identifier is provided by the CDNI metadata, skip this step. If an attribute was added to the URI, append an "&" character. Append the string "KID=". Append the key identifier (e.g. "http://example.com/public/keys/123") needed by the entity to locate the shared key for validating the URI signature. Note the Key ID URI contains only the "scheme name", "authority", and "path" parts (i.e. query string is not allowed).
3. If the digital signature algorithm uses the default value (EC-DSA), skip this step. If an attribute was added to the URI, append an "&" character. Append the string "DSA=". Append the string denoting the digital signature function used.
4. If an attribute was added to the URI, append an "&" character. Append the string "DS=". The message now contains the complete section of the URI that is protected. (e.g. "://example.com/content.mov?ET=1209422976&CIP=10.0.0.1&KID=http://

example.com/public/keys/123&DS=").

5. Compute the message digest using SHA-1 (without a key) for the message (e.g. message digest is "b95cb62f1d30ad03969619e9574a925fbfe9aeaf"). Note: The reason the digital signature calculated in the next step is calculated over the SHA-1 message digest, instead of over the cleartype message, is to reduce the length of the digital signature, and thereby the length of the URI Signing Package Attribute and the resulting Signed URI.
6. Compute the digital signature, using the EC-DSA algorithm by default or another algorithm if specified by the DSA Information Element, with the private EC key and message digest obtained in previous step as inputs.
7. Convert the digital signature to its equivalent hexadecimal format.
8. Append the string for the digital signature. In the case where EC-DSA algorithm is used, this string contains the values for the 'r' and 's' parameters, delimited by ':' (e.g. "://example.com/content.mov?ET=1209422976&CIP=10.0.0.1&KID=http://example.com/public/keys/123&DS=r:CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D:s:57ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E")

3.2. Packaging the URI Signature

Apply the URI Signing Package Attribute by following the procedure below to generate the Signed URI.

1. Remove the Original URI portion from the message to obtain all the URI Signing Information Elements, including the URI signature (e.g. "ET=1209422976&CIP=10.0.0.1&KID=example:keys:123&&MD=da58513e8b309c1e8a9695baceba629d180b50b8").
2. Compute the URI Signing Package Attribute using Base-64 Data Encoding [RFC4648] on the message (e.g. "RVQ9MTIwOTQyMjk3NiZDSVA9MTAuMCAwLjEmS0lEPWV4YWlwbGU6a2V5czoxMjMmJk1EPWRhNTglMTNlOGIzMdljMWU4YTlk2OTViYWNlYmE2MjlkMTgwYjUwYjg="). 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=RVQ9MTIwOTQyMjk3NiZDSVA9MTAuMC4wLjEmS0lEPWV4YW1wbGU6a2V5czoxMjMmJk1EPWRhNTg1MTNlOGIzMd1jMWU4YTk2OTViYWNlYmE2MjlkMTgwYjUwYjg="). Note: this is the completed Signed URI.

4. Validating a URI Signature

The process of validating a Signed URI can be divided into two sets of steps: validation of the information elements embedded in the Signed URI and validation of the URI Signature. Note it is possible to use some other algorithm and implementation as long as the same result is achieved.

4.1. Information element validation

Extract and validate the information elements embedded in the URI. Note that some steps are to be skipped if the corresponding URI Signing Information Element is not embedded in the Signed URI. The absence of a given Enforcement Information Element indicates enforcement of its purpose is not necessary in the CSP's distribution policy.

1. Extract the value from 'URISigningPackage' attribute. This value is the encoded URI Signing Package Attribute. If there are multiple instances of this attribute, the first one is used and the remaining ones are ignored. This ensures that the Signed URI can be validated despite a client appending another instance of the 'URISigningPackage' attribute.
2. Decode the string using Base-64 Data Encoding [RFC4648] (or another encoding method specified by configuration or CDNI metadata) to obtain all the URI Signing Information Elements (e.g. "ET=1209422976&CIP=10.0.0.1&KID=example:keys:123&&").

MD=da58513e8b309c1e8a9695baceba629d180b50b8").

3. Extract the value from "VER" if the information element exists in the query string. Determine the version of the URI Signing algorithm used to process the Signed URI. If the CDNI Metadata interface is used, check to see if the used version of the URI Signing algorithm is among the allowed set of URI Signing versions specified by the metadata. If this is not the case, the request is denied. If the attribute is not in the URI, then obtain the version number in another manner (e.g. configuration, CDNI metadata or default value).
4. Extract the value from "CIP" if the information element exists in the query string. Validate that the request came from the same IP address as indicated in the "CIP" attribute. If the IP address is incorrect, then the request is denied.
5. Extract the value from "ET" if the information element exists in the query string. Validate that the request arrived before expiration time based on the "ET" attribute. If the time expired, then the request is denied.
6. Extract the value from "MD" if the information element exists in the query string. The existence of this information element indicates a symmetric key is used.
7. Extract the value from "DS" if the information element exists in the query string. The existence of this information element indicates a asymmetric key is used.
8. If neither "MD" or "DS" attribute is in the URI, then no URI Signature exists and the request is denied. If both the "MD" and the "DS" information elements are present, the Signed URI is considered to be malformed and the request is denied.

4.2. Signature validation

Validate the URI Signature for the Signed URI.

1. Copy the Original URI, excluding the "scheme name" part, into a buffer to hold the message for performing the operations below.
2. Remove the "URISigningPackage" attribute from the message. Remove any subsequent part of the query string after the "URISigningPackage" attribute.
3. Append the decoded value from "URISigningPackage" attribute (which contains all the URI Signing Information Elements).

4. Depending on the type of key used to sign the URI, validate the message digest or digital signature for symmetric key or asymmetric keys, respectively.
 - A. For symmetric key, HMAC algorithm is used.
 - a. Extract the value from the "KID" information element, if it exists. Use the key identifier (e.g. "example:keys:123") to locate the shared key, which may be one of the keys available to use (i.e. set by configuration or CDNI metadata). If the information element is not in the URI Signing Package Attribute, then obtain the key in another manner (e.g. configuration or CDNI metadata). If the "KID" information element is present but its value is not in the allowable KID set as listed in the CDNI metadata, the request is denied.
 - b. Extract the value from the "HF" information element, if it exists. Determine the type of hash function (e.g. "MD5", "SHA-1", "SHA-256", "SHA-3") to use for HMAC. If the information element is not in the URI, the default hash function is SHA-1. If the "HF" information element is present but its value is not in the allowable "HF" set as listed in the CDNI metadata, the request is denied.
 - c. Extract the value from the "MD" information element. This is the received message digest.
 - d. Convert the message digest to binary format. This will be used to compare with the computed value later.
 - e. Remove the value part of the "MD" information element (but not the '=' character) from the message. The message is ready for validation of the message digest (e.g. "://example.com/content.mov?ET=1209422976&CIP=10.0.0.1&KID=example:keys:123&MD=").
 - f. Compute the message digest using the HMAC algorithm with the shared key and message as the two inputs to the hash function which is specified by the "HF" attribute.
 - g. Compare the result with the received message digest to validate the Signed URI.

- B. For asymmetric keys, a digital signature function is used.
- a. Extract the value from the "KID" information element, if it exists. Use the key identifier (e.g. "http://example.com/public/keys/123") to obtain the EC public key, which may be one of the keys available to use (i.e. set by configuration or CDNI metadata). If the information element is not in the URI, then obtain the key in another manner (e.g. configuration or CDNI metadata).
 - b. Extract the value from the "DSA" information element, if it exists. Determine the type of digital signature function (e.g. "RSA", "DSA", "EC-DSA") to use for calculating the Digital Signature. If the information element is not in the URI, the default digital signature function is EC-DSA. If the "DSA" information element is present but its value is not in the allowable "EC-DSA" set as listed in the CDNI metadata, the request is denied.
 - c. Extract the value from the "DS" information element. This is the digital signature.
 - d. Convert the digital signature to binary format. This will be used for verification later.
 - e. Remove the value part of the "DS" information element (but not the '=' character) from the message. The message is ready for validation of the digital signature (e.g. "://example.com/content.mov?ET=1209422976&CIP=10.0.0.1&KID=http://example.com/public/keys/123&DS=").
 - f. Compute the message digest using SHA-1 (without a key) for the message.
 - g. Verify the digital signature using the digital signature function (e.g. EC-DSA) with the public key, received digital signature, and message digest (obtained in previous step) as inputs. This validates the Signed URI.

5. Relationship with CDNI Interfaces

Some of the CDNI Interfaces need enhancements to support URI Signing. As an example: A Downstream CDN that supports URI Signing needs to be able to advertise this capability to the Upstream CDN. The Upstream

CDN needs to select a Downstream CDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the Upstream CDN needs to be able to distribute via the CDNI Metadata interface the information necessary to allow the Downstream CDN to validate a Signed URI . Events that pertain to URI Signing (e.g. request denial or delivery after access authorization) need to be included in the logs communicated through the CDNI Logging interface (Editor's Note: Is this within the scope of the CDNI Logging interface?).

5.1. CDNI Control Interface

URI Signing has no impact on this interface.

5.2. CDNI Footprint & Capabilities Advertisement Interface

The Downstream CDN advertises its capability to support URI Signing via the CDNI Footprint & Capabilities Advertisement interface (FCI). The supported version of URI Signing needs to be included to allow for future extensibility.

[Editor's Note: To be discussed with FCI authors]

5.3. CDNI Request Routing Redirection Interface

[Editor's Note: Debate the approach of dCDN providing the Signed URI vs. uCDN performing the signing function. List the pros/cons of each approach for the CDNI Request Routing Redirection interface (RI). Offer recommendation?]

The two approaches:

1. Downstream CDN provides the Signed URI
 - * Key distribution is not necessary
 - * Downstream CDN can use any scheme for Signed URI as long as the security level meets the CSP's expectation
2. Upstream CDN signs the URI
 - * Consistency with interative request routing method
 - * URI Signing works even when Downstream CDN does not have the signing function (which may be the case when the Downstream CDN operates only as a delivering CDN)

- * Upstream CDN can act as a conversion gateway for the requesting routing interface between Upstream CDN and CSP and request routing interface between Upstream CDN and Downstream CDN since these two interfaces may not be the same

5.4. CDNI Metadata Interface

The following CDNI Metadata objects are specified for URI Signing.

- o URI Signing enforcement flag. Specifically, this flag indicates if the access to content is subject to URI Signing. URI Signing requires the Downstream CDN to ensure that the URI must be signed and validated before content delivery. Otherwise, Downstream CDN does not perform validation regardless if URI is signed or not.
- o Designated key identifier used for URI Signing computation when the Signed URI does not contain the Key ID information element
- o Allowable Key ID set that the Signed URI's Key ID information element can reference
- o Designated hash function used for URI Signing computation when the Signed URI does not contain the Hash Function information element
- o Allowable Hash Function set that the Signed URI's Hash Function information element can reference
- o Designated digital signature function used for URI Signing computation when the Signed URI does not contain the Digital Signature Algorithm information element.
- o Allowable digital signature function set that the Signed URI's Digital Signature Algorithm information element can reference.
- o Designated version used for URI Signing computation when the Signed URI does not contain the VER attribute
- o Allowable version/algorithm set that the Signed URI's VER attribute can reference
- o Allowable set of Downstream CDNs that participate in URI Signing based on the symmetric key
- o Overwrite the default encoding method for URI Signing Attribute Set attribute? [Editor's Note: Do we need this?]
- o Overwrite the default name for the URL Signing Attribute Set attribute? [Editor's Note: Do we need this?]

Note that the Key ID information is not needed if only one key is provided by the CSP or the Upstream CDN for the content item or set of content items covered by the CDNI Metadata object. In the case of asymmetric keys, it's easy for any entity to sign the URI for content with a private key and provide the public key in the Signed URI. This just confirms that the URI Signer authorized the delivery. But it's necessary for the URI Signer to be the content owner. So, the CDNI Metadata interface MUST provide the public key for the content or information to authorize the received Key ID attribute.

5.5. CDNI Logging Interface

The Downstream CDN reports that enforcement of the access control was applied to the request for content delivery.

The following CDNI Logging field for URI Signing SHOULD be supported in the HTTP Request Logging Record as specified in CDNI Logging Interface [I-D.ietf-cdni-logging].

- o s-uri-signing:
 - * format: 1DIGIT
 - * field value: this characterises the uri signing validation performed by the Surrogate on the request. The allowed values are:
 - + "0" : no uri signature validation performed
 - + "1" : uri signature validation performed and validated
 - + "2" : uri signature validation performed and rejected
 - * occurrence: there MUST be zero or exactly one instance of this field.

[Editor's note: Need to log these URI signature validation events (e.g. invalid client IP address, expired signed URI, incorrect URI signature, successful validation)?]

TBD: CDNI Logging interface is work in progress.

6. URI Signing Message Flow

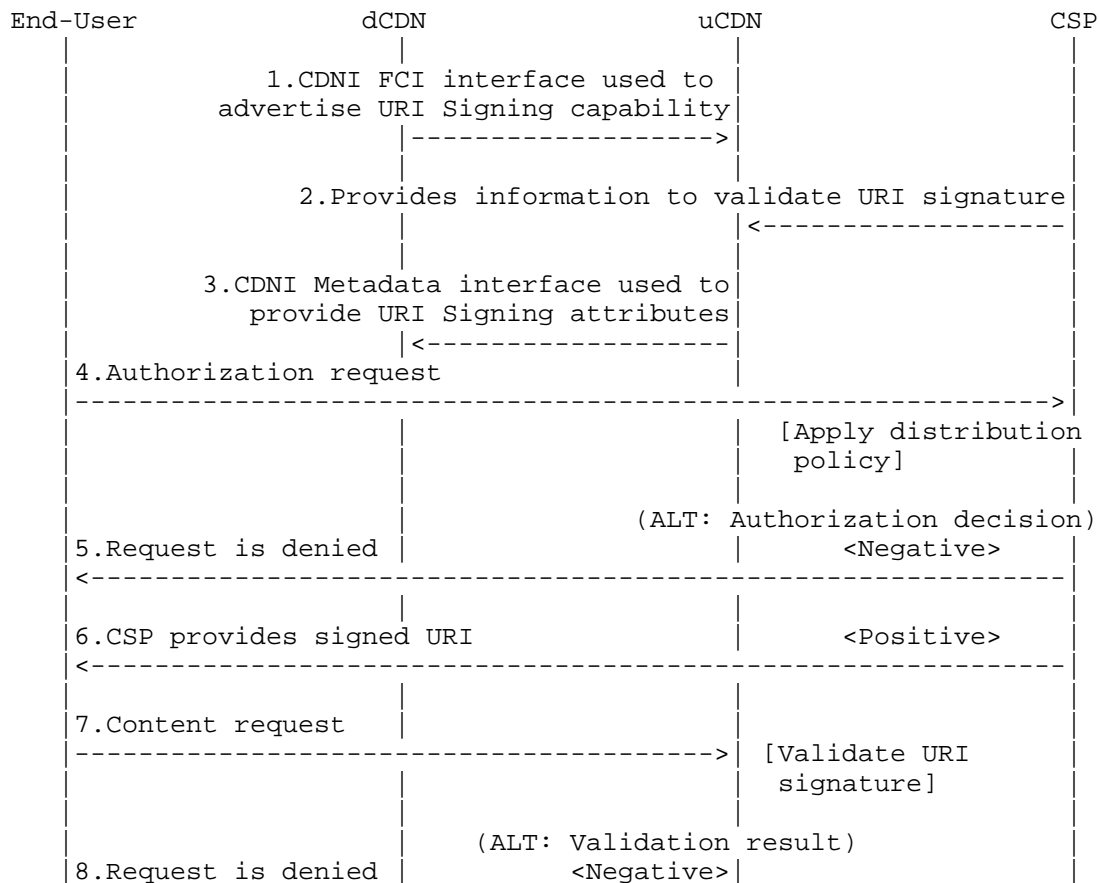
URI Signing supports both HTTP-based and DNS-based request routing. HMAC [RFC2104] defines a hash-based message authentication code allowing two parties that share a symmetric key or asymmetric keys to

establish the integrity and authenticity of a set of information (e.g. a message) through a cryptographic hash function.

6.1. HTTP Redirection

For HTTP-based request routing, HMAC is applied to a set of information that is unique to a given end user content request using key information that is specific to a pair of adjacent CDNI hops (e.g. between the CSP and the Authoritative CDN, between the Authoritative CDN and a Downstream CDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI signing scheme described below is based on the following steps (assuming HTTP redirection, iterative request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



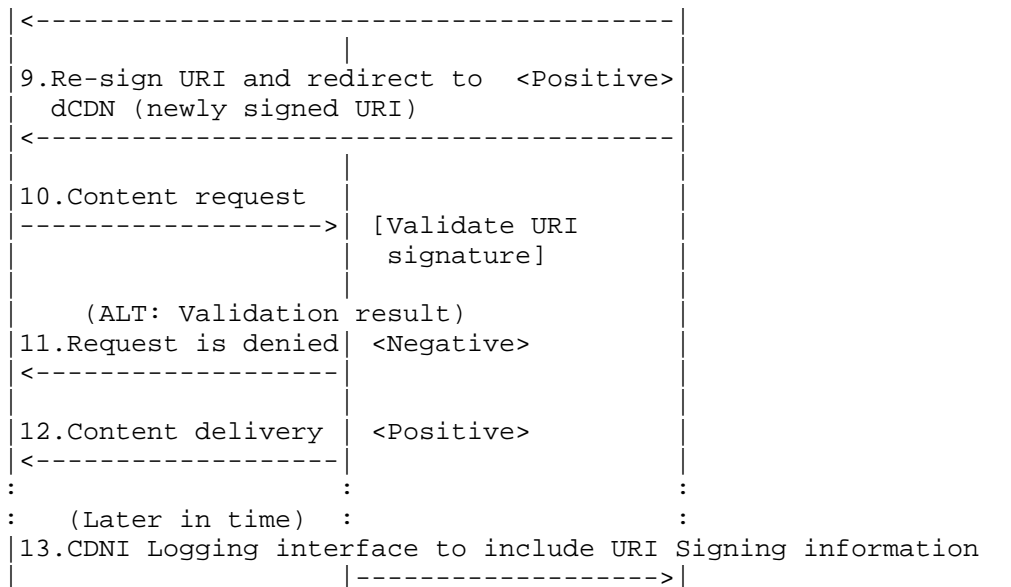


Figure 3: HTTP-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.
2. CSP provides to the Authoritative CDN the information needed to validate URI signatures from that CSP. For example, this information may include a hashing function, algorithm, and a key value.
3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate URI signatures from the Authoritative CDN for the given CSP. For example, this information may include the URI query string parameter name for the URI Signing Package Attribute, a hashing algorithm and/or a key corresponding to the trust relationship between the Authoritative CDN and the Downstream CDN.
4. When a UA requests a piece of protected content from the CSP, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy
5. If the authorization decision is negative, the CSP rejects the request.

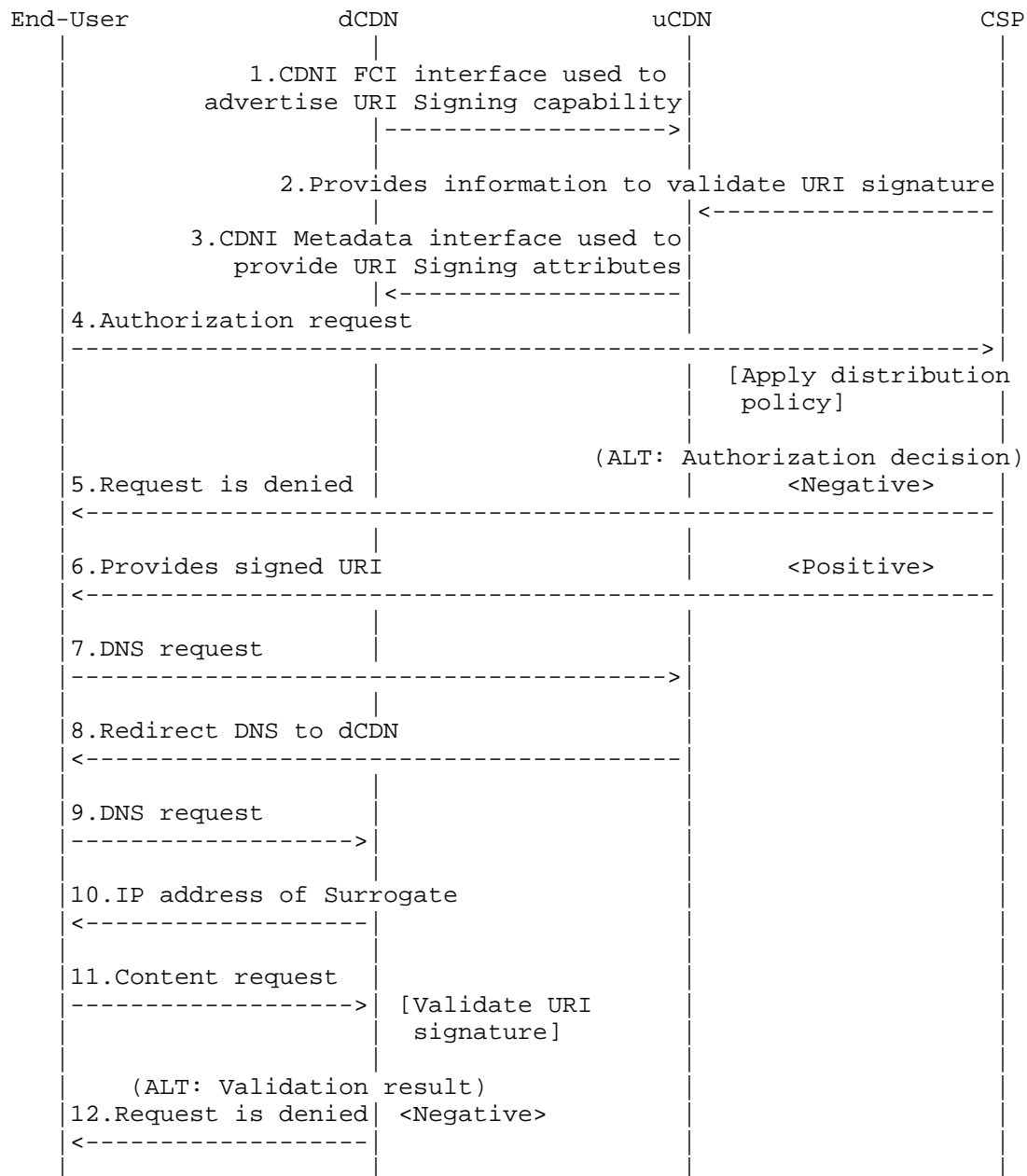
6. If the authorization decision is positive, the CSP computes a Signed URI that is based on unique parameters of that request and conveys it to the end user as the URI to use to request the content.
7. On receipt of the corresponding content request, the authoritative CDN validates the URI Signature in the URI using the information provided by the CSP.
8. If the validation is negative, the authoritative CDN rejects the request
9. If the validation is positive, the authoritative CDN computes a Signed URI that is based on unique parameters of that request and provides to the end user as the URI to use to further request the content from the Downstream CDN
10. On receipt of the corresponding content request, the Downstream CDN validates the URI Signature in the Signed URI using the information provided by the Authoritative CDN in the CDNI Metadata
11. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.
12. If the validation is positive, the Downstream CDN serves the request and delivers the content.
13. At a later time, Downstream CDN reports logging events that includes URI signing information.

With HTTP-based request routing, URI Signing matches well the general chain of trust model of CDNI both with symmetric key and asymmetric keys because the key information only need to be specific to a pair of adjacent CDNI hops.

6.2. DNS Redirection

For DNS-based request routing, the CSP and Authoritative CDN must agree on a trust model appropriate to the security requirements of the CSP's particular content. Use of asymmetric public/private keys allows for unlimited distribution of the public key to Downstream CDNs. However, if a shared secret key is preferred, then the CSP may want to restrict the distribution of the key to a (possibly empty) subset of trusted Downstream CDNs. Authorized Delivery CDNs need to obtain the key information to validate the Signed UR, which is computed by the CSP based on its distribution policy.

The URI signing scheme described below is based on the following steps (assuming iterative DNS request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



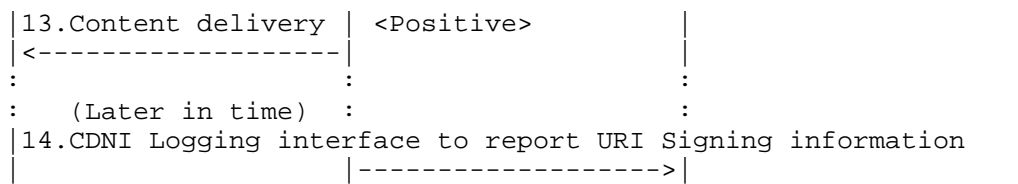


Figure 4: DNS-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.
2. CSP provides to the Authoritative CDN the information needed to validate cryptographic signatures from that CSP. For example, this information may include a hash function, algorithm, and a key.
3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate cryptographic signatures from the CSP (e.g. the URI query string parameter name for the URI Signing Package Attribute). In the case of symmetric key, the Authoritative CDN checks if the Downstream CDN is allowed by CSP to obtain the shared secret key.
4. When a UA requests a piece of protected content from the CSP, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy.
5. If the authorization decision is negative, the CSP rejects the request
6. If the authorization decision is positive, the CSP computes a cryptographic signature that is based on unique parameters of that request and includes it in the URI provided to the end user to request the content.
7. End user sends DNS request to the authoritative CDN.
8. On receipt of the DNS request, the authoritative CDN redirects the request to the Downstream CDN.
9. End user sends DNS request to the Downstream CDN.
10. On receipt of the DNS request, the Downstream CDN responds with IP address of one of its Surrogates.

11. On receipt of the corresponding content request, the Downstream CDN validates the cryptographic signature in the URI using the information provided by the Authoritative CDN in the CDNI Metadata
12. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.
13. If the validation is positive, the Downstream CDN serves the request and delivers the content.
14. At a later time, Downstream CDN reports logging events that includes URI signing information.

With DNS-based request routing, URI Signing matches well the general chain of trust model of CDNI when used with asymmetric keys because the only key information that need to be distributed across multiple CDNI hops including non-adjacent hops is the public key, that is generally not confidential.

With DNS-based request routing, URI Signing does not match well the general chain of trust model of CDNI when used with symmetric keys because the symmetric key information needs to be distributed across multiple CDNI hops including non-adjacent hops. This raises a security concern for applicability of URI Signing with symmetric keys in case of DNS-based inter-CDN request routing.

7. HTTP Adaptive Streaming

The authors note that in order to perform URI signing for individual content segments of HTTP Adaptive Bitrate content, specific URI signing mechanisms are needed. Such mechanisms are currently out-of-scope of this document. More details on this topic is covered in Models for HTTP-Adaptive-Streaming-Aware CDNI [RFC6983].

8. IANA Considerations

[Editor's note: (Is there a need to) register default value for URI Signing Package Attribute URI query string parameter name (i.e. URISigningPackage) to be used for URI Signing? Need anything from IANA?]

[Editor's note: To do: Convert to proper IANA Registry format]

This document requests IANA to create three new registries for the Information Elements and their defined values to be used for URI

Signing.

The following Enforcement Information Element names are allocated:

- o ET (Expiry time)
- o CIP (Client IP address)

The following Signature Computation Information Element names are allocated:

- o VER (Version): 1(Base)
- o KID (Key ID)
- o HF (Hash Function): "MD5", "SHA-1", "SHA-256", "SHA-3"
- o DSA (Digital Signature Algorithm): "RSA", "DSA", "EC-DSA"

The following URI Signature Information Element names are allocated:

- o MD (Message Digest)
- o DS (Digital Signature)

The IANA is requested to allocate a new entry to the CDNI Logging Field Names Registry as specified in CDNI Logging Interface [I-D.ietf-cdni-logging] in accordance to the "Specification Required" policy [RFC5226]

- o s-url-signing
- o [Editor's note: logging error codes are needed for URI Signing?]

The IANA is requested to allocate a new entry to the CDNI Metadata Field Names Registry as specified in CDNI Metadata Interface [I-D.ietf-cdni-metadata] in accordance to the "Specification Required" policy [RFC5226]

- o URI Signing Package URI query parameter name 1 Token
- o More metadata...

9. Security Considerations

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of interconnected

CDNs (CDNI). The primary goal of URI Signing is to make sure that only authorized UAs are able to access the content, with a Content Service Provider (CSP) being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as DRM, are more appropriate.

In general, it holds that the level of protection against illegitimate access can be increased by including more Enforcement Information Elements in the URI. The current version of this document includes elements for enforcing Client IP Address and Expiration Time, however this list can be extended with other, more complex, attributes that are able to provide some form of protection against some of the vulnerabilities highlighted below.

That said, there are a number of aspects that limit the level of security offered by URI signing and that anybody implementing URI signing should be aware of.

Replay attacks: Any (valid) Signed URI can be used to perform replay attacks. The vulnerability to replay attacks can be reduced by picking a relatively short window for the Expiration Time attribute, although this is limited by the fact that any HTTP-based request needs a window of at least a couple of seconds to prevent any sudden network issues from preventing legitimate UAs access to the content. One way to reduce exposure to replay attacks is to include in the URI a unique one-time access ID. Whenever the Downstream CDN receives a request with a given unique access ID, it adds that access ID to the list of 'used' IDs. In the case an illegitimate UA tries to use the same URI through a replay attack, the Downstream CDN can deny the request based on the already-used access ID.

Illegitimate client behind a NAT: In cases where there are multiple users behind the same NAT, all users will have the same IP address from the point of view of the Downstream CDN. This results in the Downstream CDN not being able to distinguish between the different users based on Client IP Address and illegitimate users being able to access the content. One way to reduce exposure to this kind of attack is to not only check for Client IP but also for other attributes that can be found in the HTTP headers.

TBD: ...

The shared key between CSP and Authoritative CDN may be distributed to Downstream CDNs - including cascaded CDNs. Since this key can be used to legitimately sign a URL for content access authorization,

it's important to know the implications of a compromised shared key.

[Editor's note: Threat model cover in the Security section - Prevent client from spoofing URI (Ray) - Security implications - The scope of protection by URI Signing - Protects against DoS (network bandwidth and other nodes besides the edge cache); limits the time window.]

10. Privacy

The privacy protection concerns described in CDNI Logging Interface [I-D.ietf-cdni-logging] apply when the client's IP address (CIP attribute) is embedded in the Signed URI. This means that, when anonymization is enabled, the URI Signing Package Attribute MUST be removed from the logging record.

11. Acknowledgements

The authors would like to thank the following people for their contributions in reviewing this document and providing feedback: Kevin Ma, Ben Niven-Jenkins, Thierry Magnien, Dan York, Bhaskar Bhupalam, Matt Caulfield, and Samuel Rajakumar .

12. References

12.1. Normative References

- [I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-10 (work in progress), March 2014.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

12.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L., Davie, B., and R. Brandenburg, "Framework for CDN Interconnection", draft-ietf-cdni-framework-10 (work in progress), March 2014.

- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M.,
Leung, K., and K. Ma, "CDN Interconnect Metadata",
draft-ietf-cdni-metadata-06 (work in progress),
February 2014.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network
Interconnection (CDNI) Requirements",
draft-ietf-cdni-requirements-17 (work in progress),
January 2014.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-
Hashing for Message Authentication", RFC 2104,
February 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, January 2005.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data
Encodings", RFC 4648, October 2006.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6
Address Text Representation", RFC 5952, August 2010.
- [RFC6770] Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma,
K., and G. Watson, "Use Cases for Content Delivery Network
Interconnection", RFC 6770, November 2012.
- [RFC6983] van Brandenburg, R., van Deventer, O., Le Faucheur, F.,
and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware
Content Distribution Network Interconnection (CDNI)",
RFC 6983, July 2013.

Authors' Addresses

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose 95134
USA

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

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

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

Bill Downey
Verizon Labs
60 Sylvan Road
Waltham, Massachusetts 02451
USA

Phone: +1 781 466 2475
Email: william.s.downey@verizon.com

Ray van Brandenburg
TNO
Brassersplein 2
Delft, 2612CT
the Netherlands

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

Scott Leibrand
Limelight Networks
222 S Mill Ave
Tempe, AZ 85281
USA

Phone: +1 360 419 5185
Email: sleibrand@llnw.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 8, 2013

K. Ma
Azuki Systems, Inc.
February 4, 2013

Content Distribution Network Interconnection (CDNI) Capabilities
Interface
draft-ma-cdni-capabilities-01

Abstract

The interconnection of Content Distribution Networks (CDNs) is predicated on the ability of downstream CDNs (dCDNs) to handle end-user requests in a functionally equivalent manner to the upstream CDN (uCDN). The uCDN must be able to assess the ability of the dCDN to handle individual requests. A CDN interconnection (CDNI) interface is needed to facilitate the advertisement of capabilities by the dCDN to the uCDN. This document describes an approach to implementing a CDNI capabilities advertisement interface.

Requirements Language

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

Status of this Memo

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

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

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

This Internet-Draft will expire on August 8, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. Capabilities	4
2.1. Delivery Protocol	6
2.2. Acquisition Protocol	7
2.3. Metadata Bundle	8
2.4. Redirection Mode	9
3. Capability Advertisement	11
3.1. Capability Update	11
3.2. Capability Removal	13
4. IANA Considerations	14
5. Security Considerations	14
6. Acknowledgements	15
7. References	15
7.1. Normative References	15
7.2. Informative References	15
Appendix A. Capability Aggregation	16
A.1. Downstream CDN Aggregation	16
A.2. Internal Request Router Aggregation	17
A.3. Internal Capability Aggregation	19
Author's Address	20

1. Introduction

The need for capabilities advertisement in CDNI is described in the CDNI requirements document [I-D.ietf-cdni-requirements]. Requirement REQ-2 describes the need to allow dCDNs to communicate capabilities to the uCDN. The uCDN aggregates the capabilities information for all dCDNs for use in making request routing decisions. Per requirement REQ-3, the uCDN should further use the aggregated capabilities in advertisements to CDNs further upstream. This concept of aggregation can apply to both organizationally different dCDNs (e.g., other CDN providers, or different business units within a larger organization) or logical entities within the same CDN (e.g., using multiple request routers for scalability reasons, to segregate surrogates based on specific protocol support, or to segregate surrogates based on software version or feature level, etc.).

Appendix A contains more detailed descriptions of the different capabilities management scenarios, but it is important to note that it is the ability of the dCDN to service each request in a functionally equivalent manner as the uCDN that is important, not the physical layout of resources through which it services the request. The aggregation of resource knowledge by the dCDN into a simple set of capabilities which can be advertised to the uCDN enables efficient decision making at each delegation point in the CDN interconnection hierarchy.

It is assumed that an authoritative request router in each CDN will be responsible for aggregating and advertising capabilities information in a dCDN, and receiving and aggregating capabilities information in the uCDN. As there is no centralized CDNI controller defined in the CDNI architecture, the request router seems the most logical place for capabilities aggregation to occur, as it is the request router that needs such information to make delegation decisions. The protocol defined herein may be implemented as part of an entity other than an authoritative request router, but for the purposes of this discussion, the authoritative request router is assumed to be the centralized capabilities aggregation point.

Though there is an obvious need for the ability to exchange and update capability information in real-time, it is assumed that capabilities do not change very often. There is also an assumption that the capabilities are not by themselves useful for making delegation decisions. Capability information is assumed to be input into business logic. It is the business logic which provides the algorithms for delegation decision making. The definition of business logic occurs outside the scope of CDNI and outside the timescale of capability advertisement. It may be the case that the business logic anticipates and reacts to changes in dCDN

capabilities. However, it may also be the case that business logic is tailored through offline processes as dCDN capabilities change. The capabilities interface is agnostic to the business processes employed by a given uCDN. The capabilities that are advertised over the capabilities interface may be used by the uCDN at its leisure to implement delegation rules. Setting proper defaults in the business logic should prevent any unwanted delegation from occurring when dCDN capabilities change, however, that is beyond the scope of this discussion.

1.1. Terminology

This document uses the terminology defined in section 1.1 of the CDNI Framework [I-D.ietf-cdni-framework] document.

2. Capabilities

There is a basic set of capabilities that must be advertised by the dCDN for the uCDN to be able to determine if the dCDN is functionally able to handle a given request. Mandatory capabilities include:

- o Delivery Protocol
- o Acquisition Protocol
- o Metadata Bundle
- o Redirection Mode

The following sections describe each of the capabilities in further detail, however, all of the capabilities can be described using the same general format. A capability object can be described as:

CapabilityObject:

 Name: Identifier for the capability

 Values: List of supported options for the capability

 Footprint: Optional list of footprint restrictions

The list of valid capability options for a given capability will be specific to the given capability type. Though the degenerate case may exist where the range of option values is a single value, it is anticipated that all capability types will have more than one capability option value. For consistency in the model, all capability types are implemented with lists of values. To optimize actions on the entire range of capability option values for a given capability type, the capability option value "ALL" is reserved and MUST be supported by all capability types. For completeness, the

capability option value "NONE" is also reserved and MUST be supported by all capability types. Reserved values SHOULD NOT be combined with any other valid capability option values for a given capability type. Reserved values MUST NOT be combined with each other for a given capability type. The reserved values MUST be given precedence over all other capability option values for a given capability type, when specified in the same capabilities advertisement message, with NONE superceding ALL if both are specified.

The footprint restriction list is optional. Footprint restriction lists override in their entirety all previously advertised footprint restriction lists for the specified capability option values for the given capability type. If no footprint restriction list is specified, it SHALL be understood that all of the capability option values specified have global reach, overriding any previous capability advertisements for the specified capability option values. The footprint restriction list supports either IP prefix or ASN values. IP prefix and ASN restrictions MUST NOT be mixed within a given footprint list. In the case of interconnecting private network CDNs, IP prefix or ASN-based footprint advertisement are the likely methods for describing coverage areas.

[Ed. note: Though other methods exist for defining footprints (e.g., GPS coordinates, country/zip/area codes, etc.), only IP prefix and ASN are considered here. Need to evaluate use cases for other methods of footprint definition.]

Note: Further optimization of the capabilities object to provide quality information for a given footprint is certainly possible, however, it is not critical to the basic interconnection of CDNs. The ability to transfer quality information in capabilities advertisements may be desirable and is noted here for completeness, however, the specifics of such mechanisms are outside the scope of this document.

Multiple capability objects of the same capability type are allowed within a given capability advertisement message as long as the capability option values do not overlap, i.e., a given capability option value MUST NOT show up in multiple capability objects within a single capability advertisement message. If multiple capability objects for a given capability type exist, those capability objects SHOULD have different footprint restrictions; capability objects of a given capability type with identical footprint restrictions SHOULD be combined.

Note: The examples shown below use an XML representation, however, other representations (e.g., JSON) may also be acceptable.

2.1. Delivery Protocol

The delivery protocol refers to the protocol over which an end user (EU) has requested content. If a dCDN does not support the protocol requested by the client, then the dCDN is not a viable candidate for delegation.

Though the delivery protocol is specified in the URI scheme (as defined in RFC3986 [RFC3986]) of the client request URL, protocol feature subsets or augmented protocol feature sets MAY be defined and SHOULD correspond with the protocols supported by the ProtocolACL defined in the CDNI Metadata Interface [I-D.ietf-cdni-metadata] document.

[Ed. note: A protocol identifier registry should be created to be able to consistently reference defined subsets of features for a given protocol (e.g., different HTTP methods, chunking, ranges, etc.) or augmented feature sets (e.g., from application layer protocols like HLS, DASH, etc.) which run over top of HTTP.]

The delivery protocol capability object MUST support a list of protocols for a given footprint. The delivery protocol capability SHOULD support optional footprint restriction information. The following XML-encoded example shows two lists of protocols with different footprints.

```
<Capabilities>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
      <Protocol>RTSP</Protocol>
      <Protocol>MMS</Protocol>
    </Protocols>
  </DeliveryProtocols>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>RTMP</Protocol>
      <Protocol>HTTPS</Protocol>
    </Protocols>
    <Footprint>
      <Prefixes>
        <Prefix>10.1.0.0/16</Prefix>
        <Prefix>10.10.10.0/24</Prefix>
      </Prefixes>
    </Footprint>
  </DeliveryProtocols>
</Capabilities>
```

In the above example, the three protocols HTTP, RTSP, and MMS are supported globally, while the protocols RTMP and HTTPS are only supported in a restricted footprint (in this case, specified by IP address prefix).

A protocol **MUST** not appear in multiple delivery protocol lists, within a given capability advertisement message. Redefinition of footprint restrictions in subsequent capability advertisement messages **MAY** occur and **SHALL** override all previous capability advertisements for the specified delivery protocol.

2.2. Acquisition Protocol

The acquisition protocol refers to the protocol over which the dCDN may acquire content from the uCDN. If a dCDN does not support any of the protocols offered by the uCDN, then the dCDN is not a viable candidate for delegation.

Though the acquisition protocol is disseminated to the dCDN in the URI scheme (as defined in RFC3986 [RFC3986]) of the URL provided by the uCDN via the CDNI Metadata Interface [I-D.ietf-cdni-metadata], protocol feature subsets or augmented protocol feature sets **MAY** be defined and **SHOULD** correspond with the protocols supported by the ProtocolACL defined in the CDNI Metadata Interface [I-D.ietf-cdni-metadata] document.

[Ed. note: A protocol identifier registry should be created to be able to consistently reference defined subsets of features for a given protocol (e.g., different HTTP methods, chunking, ranges, etc.) or augmented feature sets (e.g., from application layer protocols like HLS, DASH, etc.) which run over top of HTTP.]

The acquisition protocol capability object **MUST** support a list of protocols for a given footprint. The acquisition protocol capability **SHOULD** support optional footprint restriction information. The following XML-encoded example shows two lists of protocols with different footprints.

```
<Capabilities>
  <AcquisitionProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
      <Protocol>FTP</Protocol>
    </Protocols>
  </AcquisitionProtocols>
  <AcquisitionProtocols>
    <Protocols>
      <Protocol>SFTP</Protocol>
      <Protocol>HTTPS</Protocol>
    </Protocols>
    <Footprint>
      <ASNs>
        <ASN>0</ASN>
        <ASN>65535</ASN>
      </ASNs>
    </Footprint>
  </AcquisitionProtocols>
</Capabilities>
```

In the above example, the two protocols HTTP and FTP are supported globally, while the protocols SFTP and HTTPS are only supported in a restricted footprint (in this case, specified by ASN).

A protocol MUST not appear in multiple acquisition protocol lists, within a given capability advertisement message. Redefinition of footprint restrictions in subsequent capability advertisement messages MAY occur and SHALL override all previous capability advertisements for the specified acquisition protocol.

2.3. Metadata Bundle

Metadata bundles are groupings of one or more metadata definitions which the dCDN is capable of understanding. There will be a core set of metadata defined in the CDNI Metadata Interface [I-D.ietf-cdni-metadata] document, which all CDNI implementations will be required to support, however, future revisions of CDNI may include additional mandatory to implement metadata, and some operators may require the use of vendor specific metadata. If a dCDN is not capable of understanding a given piece of metadata which the uCDN feels is mandatory for delivery, then the dCDN is not a viable candidate for delegation.

[Ed. note: A metadata bundle identifier registry should be created to be able to consistently reference defined sets of metadata types and any individual options within a given piece of metadata within a given bundle which may need to be advertised.

The metadata bundle capability object MUST support a list of protocols for a given footprint. The metadata bundle capability SHOULD support optional footprint restriction information. The following XML-encoded example shows two lists of bundles with different footprints.

```
<Capabilities>
  <MetadataBundles>
    <Bundles>
      <Bundle>cdni.core.v1</Bundle>
    </Bundles>
  </MetadataBundles>
  <MetadataBundle>
    <Bundles>
      <Bundle>vendor.azuki.has.v1</Bundle>
      <Bundle>private.rw.color.v1</Bundle>
    </Bundles>
    <Footprint>
      <Prefixes>
        <Prefix>10.1.2.0/24</Prefix>
      </Prefixes>
    </Footprint>
  </MetadataBundle>
</Capabilities>
```

In the above example, core version 1 CDNI metadata `cdni.core.v1` is supported globally, while the vendor specific metadata bundles `vendor.azuki.has.v1` and `private.rw.color.v1` are only supported in a restricted footprint (in this case, specified by IP Prefix).

A bundle MUST not appear in multiple metadata bundle lists, within a given capability advertisement message. Redefinition of footprint restrictions in subsequent capability advertisement messages MAY occur and SHALL override all previous capability advertisements for the specified metadata bundle.

2.4. Redirection Mode

The redirection mode refers to the method(s) employed by request routers to perform request redirection. The CDNI framework [I-D.ietf-cdni-framework] document describes four possible request routing modes:

- o DNS iterative (DNS-I)
- o DNS recursive (DNS-R)

- o HTTP iterative (HTTP-I)
- o HTTP recursive (HTTP-R)

If a dCDN supports only a specific mode or subset of modes that does not overlap with the modes supported by the uCDN, then the dCDN is not a viable candidate for delegation.

[Ed. note: A request routing mode identifier registry should be created to be able to consistently reference an extensible list of supported CDNI request routing modes.

The redirection mode capability object MUST support a list of redirection modes for a given footprint. The redirection mode capability SHOULD support optional footprint restriction information. The following XML-encoded example shows two lists of modes with different footprints.

```
<Capabilities>
  <RedirectionModes>
    <Modes>
      <Mode>DNS-I</Mode>
      <Mode>HTTP-I</Mode>
    </Modes>
  </RedirectionModes>
  <RedirectionModes>
    <Modes>
      <Mode>DNS-R</Mode>
      <Mode>HTTP-R</Mode>
    </Modes>
    <Footprint>
      <ASNs>
        <ASN>64511</ASN>
      </ASNs>
    </Footprint>
  </RedirectionModes>
</Capabilities>
```

In the above example, iterative redirection is supported globally, while recursive redirection is only supported in a restricted footprint (in this case, specified by ASN).

A mode MUST not appear in multiple redirection mode lists, within a given capability advertisement message. Redefinition of footprint restrictions in subsequent capability advertisement messages MAY occur and SHALL override all previous capability advertisements for the specified redirection mode.

3. Capability Advertisement

The capabilities advertisement protocol is an HTTP-based protocol using the POST and DELETE methods. Capability removal is distinguished from capability advertisement by the HTTP request method. Capability advertisement uses the HTTP POST method, while capability removal uses the HTTP DELETE method. The dCDN request router asynchronously issues capability advertisement messages to the uCDN request router (see Appendix A for detailed descriptions of authoritative request router capabilities aggregation scenarios). It is assumed that the dCDN request router has been configured with the uCDN request router address either through the CDNI Control interface bootstrapping function, or through some other out-of-band configuration.

Note: The examples shown below use an XML representation, however, other representations (e.g., JSON) are also acceptable.

3.1. Capability Update

A capabilities advertisement message may combine objects from one or more capability types. On an initial advertisement, the dCDN SHOULD send a value for every capability type.

```
POST /CDNI/RI/capabilities HTTP/1.1
Host: rr0.u.example.com
Accept: */*
Content-Length: 530
Content-Type: application/x-www-form-urlencoded
```

```
<Capabilities>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
    </Protocols>
  </DeliveryProtocols>
  <AcquisitionProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
      <Protocol>FTP</Protocol>
    </Protocols>
  </AcquisitionProtocols>
  <MetadataBundles>
    <Bundles>
      <Bundle>cdni.core.v1</Bundle>
      <Bundle>private.rw.color.v1</Bundle>
    </Bundles>
  </MetadataBundles>
  <RedirectionModes>
    <Modes>
      <Mode>HTTP-I</Mode>
    </Modes>
  </RedirectionModes>
</Capabilities>
```

In the above example, the dCDN issues a post to the uCDN request router rr0.u.example.com with a capability advertisement message specifying support for HTTP-based delivery, HTTP or FTP-based content acquisition, iterative HTTP redirection, and support for both core and color metadata. A subsequent capabilities advertisement message may be used to update this information without respecifying all the fields.

In the following example, the dCDN updates its ability to deliver content via HTTPS while restricting its previously advertised support for delivering content via HTTP to only its internal network. All previous advertisements for acquisition protocol, metadata bundles, and redirection modes still are unaffected.

```
POST /CDNI/RI/capabilities HTTP/1.1
Host: rr0.u.example.com
Accept: */*
Content-Length: 356
Content-Type: application/x-www-form-urlencoded
```

```
<Capabilities>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>HTTPS</Protocol>
    </Protocols>
  </DeliveryProtocols>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>HTTP</Protocol>
    </Protocols>
    <Footprint>
      <Prefixes>
        <Prefix>10.0.0.0/8</Prefix>
      </Prefixes>
    </Footprint>
  </DeliveryProtocols>
</Capabilities>
```

3.2. Capability Removal

In the following example, the dCDN issues a DELETE command to the uCDN request router rr0.u.example.com with a capability advertisement message containing all four capability types specifying the reserved capability option value ALL. This removes capabilities information for all capability options for all capability types, effectively allowing the dCDN to remove itself from delegation consideration.


```
DELETE /CDNI/RI/capabilities HTTP/1.1
Host: rr0.u.example.com
Accept: */*
Content-Length: 442
Content-Type: application/x-www-form-urlencoded
```

```
<Capabilities>
  <DeliveryProtocols>
    <Protocols>
      <Protocol>ALL</Protocol>
    </Protocols>
  </DeliveryProtocols>
  <AcquisitionProtocols>
    <Protocols>
      <Protocol>ALL</Protocol>
    </Protocols>
  </AcquisitionProtocols>
  <MetadataBundles>
    <Bundles>
      <Bundle>ALL</Bundle>
    </Bundles>
  </MetadataBundles>
  <RedirectionModes>
    <Modes>
      <Mode>ALL</Mode>
    </Modes>
  </RedirectionModes>
</Capabilities>
```

4. IANA Considerations

This memo includes no request to IANA.

[Ed. note: Need to reference registry for protocol identifiers?]

[Ed. note: Need to reference registry for metadata bundle identifiers?]

[Ed. note: Need to reference registry for request routing mode identifiers?]

[Ed. note: Need to define a registry for capability names?]

5. Security Considerations

There are a number of security concerns associated with the

capabilities interface. As the capabilities interface essentially provides configuration information which the uCDN uses to make request routing decisions. Injection of fake capability advertisement messages, or interception and discard of real capability advertisement messages may be used for denial of service (e.g., by falsely advertising or deleting capabilities or preventing capability advertisements from reaching the uCDN). dCDN capability advertisements MUST be authenticated by the uCDN to prevent unauthorized capability advertisement message injection. uCDN request router capability interface servers MUST be authenticated by the dCDN to prevent unauthorized interception of capability advertisement messages. TLS with client authentication SHOULD be used for all capability interface implementations. Deployments in controlled environments where physical security and IP address white-listing is employed MAY choose not to use TLS.

6. Acknowledgements

The authors would like to thank Ray van Brandenburg, Gilles Bertrand, and Scott Wainner for their timely reviews and invaluable comments.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

7.2. Informative References

- [I-D.ietf-cdni-framework] Peterson, L., Ed. and B. Davie, Ed., "Framework for CDN Interconnection draft-ietf-cdni-framework-02", December 2012.
- [I-D.ietf-cdni-metadata] Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata draft-ietf-cdni-metadata-00", October 2012.
- [I-D.ietf-cdni-requirements] Leung, K. and Y. Lee, "Content Distribution Network

Interconnection (CDNI) Requirements
 draft-ietf-cdni-requirements-04", December 2012.

Appendix A. Capability Aggregation

The following sections show examples of three aggregation scenarios. In each case, CDN-U is the ultimate uCDN and CDN-P is the penultimate CDN which must perform capabilities aggregation.

A.1. Downstream CDN Aggregation

Figure A1 shows five organizationally different CDNs: CDN-U, CDN-P, and CDNs A, B, and C, the dCDNs of CDN-P which are being aggregated. Given the setup shown in Figure A1, we can construct a number of use cases, based on the coverage areas of each dCDN (i.e., CDNs P, A, B, and C). Note: In all cases, the reachability of the uCDN (i.e., CDN-U) is a don't care as it is assumed that the uCDN knows its own coverage area and is likely to favor itself in most situations, and if it has decided that it needs to delegate to a dCDN, then the only relevant question is if the dCDN can handle the request.

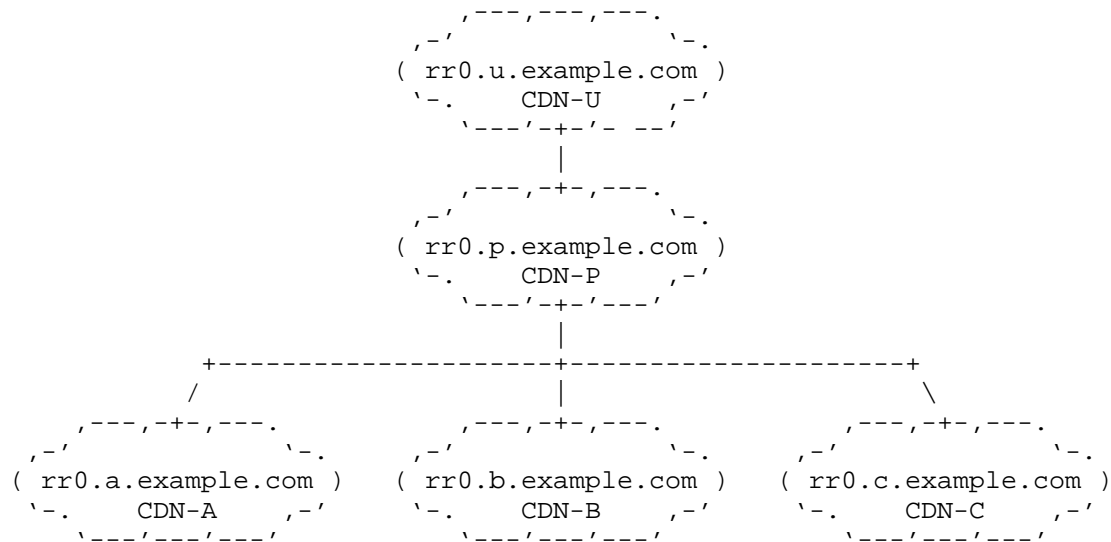


Figure A1: CDNI dCDN Request Router Aggregation

- o None of the four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, each CDN is likely to advertise footprint information with its capabilities, specifying its reachability. When CDN-P advertises capabilities to CDN-U, it may

advertise the aggregate footprint of itself and CDNs A, B, and C. Note: CDN-P MAY exclude any dCDN, and consequently its footprint, per its own internal aggregation decision criteria.

- o All four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, none of the CDNs is likely to advertise any footprint information as none have any footprint restrictions. When CDN-P advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.
- o Some of the four dCDNs (CDNs P, A, B, and C) have global reachability and some do not. In this case, even though some dCDNs do not have global reachability, the aggregate of some dCDNs having global reachability and some not should still be global reachability (for the given capability). When CDN-P advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one dCDN has global reach as being supported with global reachability. It is up to the CDN-P request router to properly select a dCDN to process individual client requests and not choose a dCDN whose restricted footprint makes it unsuitable for delivering the requested content.

A.2. Internal Request Router Aggregation

Figure A2 shows CDN-U and CDN-P where CDN-P internally has four request routers: the authoritative request router rr0, and three other request routers rr1, rr2, and rr3. The use of multiple request routers may be used to distribute request routing load across resources, possibly in different geographic regions covered by CDN-P. Similar to Figure A1, the setup shown in Figure A2 requires the authoritative request router rr0 in CDN-P to aggregate capabilities information from downstream request routers rr1, rr2, and rr3. The primary difference between the scenario is that the request routers in Figure A2 are logically within the same CDN-P organization. The same reachability scenarios apply to Figure A2 as with Figure A1.

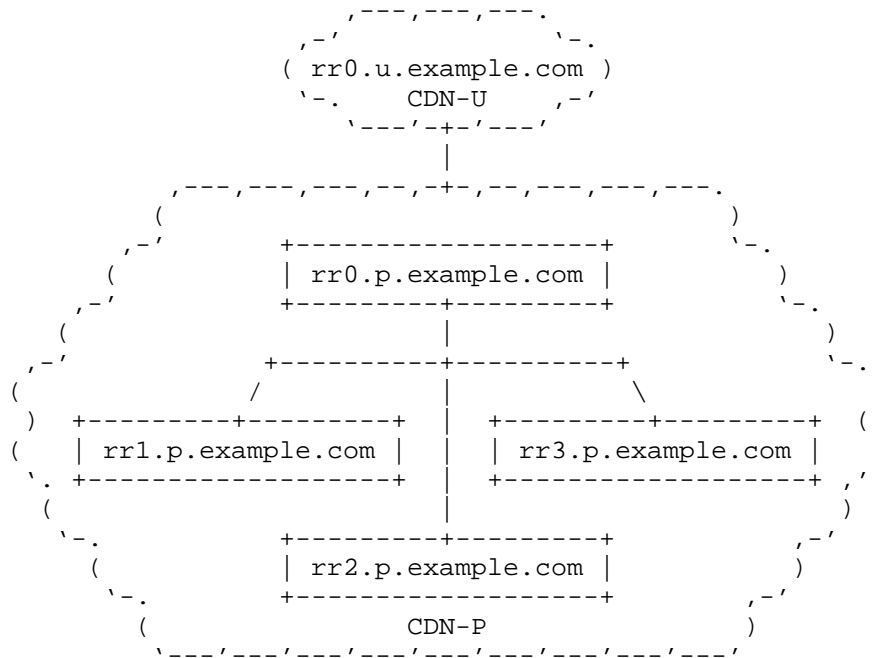


Figure A2: Local CDN Request Router Aggregation

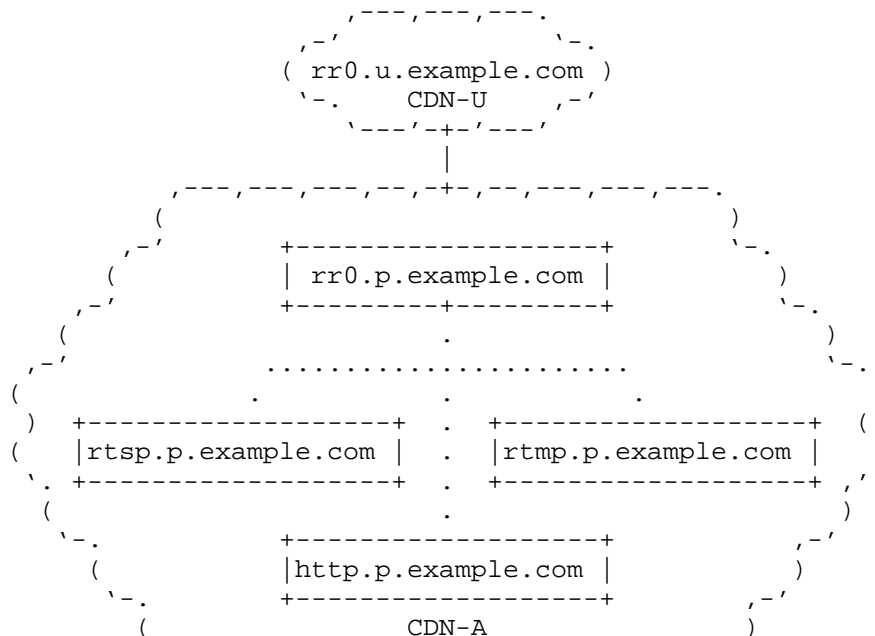
- o None of the four CDN-P request routers have global reachability. In this case, each request router is likely to advertise footprint information with its capabilities, specifying its reachability. When rr0 advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and rr1, rr2, and rr3.
- o All four CDN-P request routers have global reachability. In this case, none of the request routers is likely to advertise any footprint information as none has any footprint restrictions. When rr0 advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.
- o Some of the four CDN-P request routers have global reachability and some do not. In this case, even though some request routers do not have global reachability, the aggregate of some request routers having global reachability and some not should still be global reachability (for the given capability). When rr0 advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one request router has global reach as being supported with global reachability. It is up to the authoritative request router rr0 to properly select from the other request routers for any given request, and not choose a request router

whose restricted footprint makes it unsuitable for delivering the requested content.

A.3. Internal Capability Aggregation

Figure A3 shows CDN-U and CDN-P where the delivery network of CDN-P is segregated by delivery protocol (e.g., RTSP, HTTP, and RTMP). Figure A3 differs from Figures A1 and A2 in that request router rr0 of CDN-P is not aggregating the capabilities advertisements of multiple other downstream request routers, but rather it is managing the disparate capabilities across resources within its own local CDN. Though not every delivery node has the same protocol capabilities, the aggregate delivery protocol capabilities advertised by CDN-A may include all delivery protocols. Note, Figure A3 should not be construed to imply anything about the coverage areas for each delivery protocol. They may all support the same delivery footprint, or they may have different delivery footprints. It is the responsibility of the request router rr0 to properly assign protocol-appropriate delivery nodes to individual content requests. If certain protocols have limited reachability, CDN-P may advertise footprint restrictions for each protocol.

It should be noted that though the delivery protocol capability was selected for this example, the concept of internal capability aggregation applies to all capabilities as discussed below.



\\ _ _ _ _ / _ _ _ _ / _ _ _ _ / _ _ _ _ / _ _ _ _ / _ _ _ _ / _ _ _ _ / _ _ _ _ /

Figure A3: Local CDN Capability Segregation

Another situation in which physical footprint may not matter in an aggregated view has to do with feature support (e.g., new CDNI metadata features or new redirection modes). Situations often arise when phased roll-out of software upgrades, or staging network segregation result in only certain portions of a CDN's resources supporting the new feature set. The dCDN has a few options in this case:

- o Enforce atomic update: The dCDN does not advertise support for the new capability until all resources have been upgraded to support the new capability.
- o Transparent segregation: The dCDN advertises support for the new capability, and when requests are received that require the new capability, the dCDN request router properly selects a resource which supports that capability.
- o Advertised segregation: The dCDN advertises support for the new capability with a footprint restriction allowing the uCDN to make delegation decisions based on the dCDN's limit support.

The level of aggregation employed by the dCDN is likely to vary as business relationships dictate, however, the capability interface should support all possible modes of operation.

Author's Address

Kevin J. Ma
Azuki Systems, Inc.
43 Nagog Park
Acton, MA 01720
USA

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 24, 2016

K. Ma
Ericsson
J. Seedorf
NEC
April 22, 2016

CDNI Footprint & Capabilities Advertisement Interface
draft-ma-cdni-capabilities-09

Abstract

Content Distribution Network Interconnection (CDNI) is predicated on the ability of downstream CDNs (dCDNs) to handle end-user requests in a functionally equivalent manner to the upstream CDN (uCDN). The uCDN must be able to assess the ability of the dCDN to handle individual requests. The CDNI Footprint & Capabilities Advertisement interface (FCI) is provided for the advertisement of capabilities and the footprints to which they apply by the dCDN to the uCDN. This document describes an approach to implementing the CDNI FCI.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on October 24, 2016.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. CDNI FCI Capability Advertisement	4
2.1. CDNI FCI Capability Initialization	5
3. CDNI FCI Capabilities Service	5
3.1. CDNI FCI Map	5
3.1.1. Media Type	5
3.1.2. HTTP Method	5
3.1.3. Accept Input Parameters	6
3.1.4. Capabilities	6
3.1.5. Uses	6
3.1.6. Response	6
3.1.7. CDNI FCI Capabilities	7
3.1.7.1. Delivery Protocol	7
3.1.7.2. Acquisition Protocol	9
3.1.7.3. Redirection Mode	11
3.1.7.4. Logging Capabilities	13
3.1.7.5. Metadata Capabilities	14
4. CDNI FCI Capabilities Filtering Service	15
4.1. Filtered CDNI FCI Map	15
4.1.1. Media Type	15
4.1.2. HTTP Method	15
4.1.3. Accept Input Parameters	15
4.1.4. Capabilities	15
4.1.5. Uses	15
4.1.6. Response	16
4.1.7. Example	16
5. Footprint via ALTO Network Map	16
5.1. ALTO Network Maps	16
5.2. Example ALTO Network Map for CDNI FCI Footprint	16
5.3. Example of ALTO Network Map Footprint in FCI Map	17

6.	IANA Considerations	18
6.1.	ALTO Media Types	19
6.1.1.	ALTO CDNI FCI Map Media Type	19
6.1.2.	ALTO CDNI FCI Map Filter Media Type	20
6.2.	CDNI Footprint Types	22
7.	Security Considerations	22
7.1.	Securing the CDNI Footprint & Capabilities Advertisement interface	22
8.	Acknowledgements	23
9.	References	23
9.1.	Normative References	23
9.2.	Informative References	24
Appendix A.	Capability Aggregation	25
A.1.	Downstream CDN Aggregation	25
A.2.	Internal Request Router Aggregation	27
A.3.	Internal Capability Aggregation	28
Authors' Addresses	30

1. Introduction

The need for footprint and capabilities advertisement in Content Distribution Network Interconnection (CDNI) is described in the CDNI requirements document [RFC7337]. Requirements FCI-1 and FCI-2 describe the need to allow dCDNs to communicate capabilities to the uCDN. Requirement FCI-3 describes how a uCDN may aggregate the footprint and capabilities information for all cascaded dCDNs and use the aggregated information in advertisements to CDNs further upstream. This concept of aggregation can apply to both organizationally different dCDNs (e.g., other CDN providers, or different business units within a larger organization) or logical entities within the same CDN (e.g., using multiple request routers for scalability reasons, to segregate surrogates based on specific protocol support, or to segregate surrogates based on software version or feature level, etc.).

Appendix A contains more detailed descriptions of different footprint and capabilities management scenarios, but it is important to note that it is the ability of the dCDN to service each request in a functionally equivalent manner as the uCDN that is important, not the physical layout of resources through which it services the request. The aggregation of resource knowledge by the dCDN into a simple set of capabilities and their affective footprints, that is then advertised to the uCDN, enables efficient decision making at each delegation point in the CDN interconnection hierarchy.

It is assumed that an authoritative request router in each CDN will be responsible for aggregating and advertising capabilities information in a dCDN and/or receiving and aggregating capabilities

information in the uCDN. The CDNI Footprint & Capabilities Advertisement interface (FCI) along with the CDNI Request Routing Redirection interface (RI) [I-D.ietf-cdni-redirection] make up the CDNI Request Routing Interface. As there is no other centralized CDNI controller, the authoritative request router seems the most logical place for capabilities aggregation to occur, as it is the request router that needs such information to make delegation decisions. The protocol defined herein may be implemented as part of an entity other than an authoritative request router, but for the purposes of this discussion, the authoritative request router is assumed to be the centralized capabilities aggregation point.

Though there is an obvious need for the ability to exchange and update footprint and capability information in real-time, it is assumed that capabilities do not change very often. It is also assumed that the capabilities are not by themselves useful for making delegation decisions. Capability information is assumed to be input into business logic. It is the business logic which provides the algorithms for delegation decision making. The definition of business logic occurs outside the scope of CDNI and outside the timescale of footprint and capability advertisement [I-D.ietf-cdni-footprint-capabilities-antics]. It may be the case that the business logic anticipates and reacts to changes in dCDN capabilities, however, it may also be the case that business logic is tailored through offline processes as dCDN capabilities change. The FCI is agnostic to the business processes employed by any given uCDN. The footprints and capabilities that are advertised over the FCI may be used by the uCDN at its discretion, to implement delegation rules. Setting proper defaults in the business logic should prevent any unwanted delegation from occurring when dCDN capabilities change, however, that is beyond the scope of this discussion.

1.1. Terminology

This document uses the terminology defined in section 1.1 of the CDNI Framework [RFC7336] document.

2. CDNI FCI Capability Advertisement

The FCI is implemented as an ALTO [RFC7285] Service. The ALTO protocol defines an HTTP-based transport through which ALTO service information may be retrieved using either a GET or POST method. The uCDN request router may at any time query the dCDN ALTO FCI Service for the full set of dCDN capability information. The uCDN may use a separate FCI Filter Service to retrieve a subset of the dCDN capability information.

[Ed.: Need to update this with ALTO asynchronous update support.]

[Ed.: Need to update this with ALTO incremental update support.]

2.1. CDNI FCI Capability Initialization

In lieu of any out-of-band pre-configured capability information, when the FCI is first brought up between a uCDN and a dCDN, the uCDN SHOULD assume that the dCDN has no CDNI capabilities. If an out-of-band capability baseline has been exchanged, the uCDN MAY use that information to initialize its capabilities database. In either case, the uCDN SHOULD verify the initial state of the dCDN (as a temporary outage may affect availability in the dCDN).

The dCDN MUST support sending its entire set of capabilities to the uCDN through the ALTO service interface

3. CDNI FCI Capabilities Service

As described in Requirement FCI-2, there is a basic set of capabilities that must be supported by the FCI for the uCDN to be able to determine if the dCDN is functionally able to handle a given request. [I-D.ietf-cdni-footprint-capabilities-semantics] lists mandatory capabilities types:

- o Delivery Protocol
- o Acquisition Protocol
- o Redirection Mode
- o CDNI Logging Capabilities
- o CDNI Metadata Capabilities

To be consistent with the base ALTO service definitions, we use the JSON object definition notation as specified in the ALTO protocol [RFC7285].

3.1. CDNI FCI Map

3.1.1. Media Type

The media type of CDNI FCI Map is "application/alto-cdni-fcimap+json"

3.1.2. HTTP Method

A CDNI FCI Map resource is requested using the HTTP GET method.

3.1.3. Accept Input Parameters

None.

3.1.4. Capabilities

None.

3.1.5. Uses

None.

3.1.6. Response

The data component of a CDNI FCI Map resource is named "fcimap" which is a JSON object of type FCIMapData:

```
object {  
  FCIMapData fcimap<0..*>;  
} InfoResourceFCIMap : ResponseEntityBase;  
  
object {  
  FCICapability capabilities<1..*>;  
} FCIMapData;  
  
object {  
  JSONString capability-type;  
  JSONValue capability-value  
  FCIFootprint footprints<0..*>;  
} FCICapability;  
  
object {  
  JSONString footprint-type;  
  JSONString footprint-value<1..*>;  
} FCIFootprint;
```

The FCIMapData object contains a CDNI Payload Type [RFC7736] "ptype" which identifies the capability and a "value" object containing the associated Capability Advertisement Object (e.g., delivery-protocols, acquisition-protocols, or redirection-modes, as defined in [I-D.ietf-cdni-footprint-capabilities-semantics]). The FCIMapData object may also contain an optional list of FCIFootprint objects "footprints". The FCIFootprint object specifies a "footprint-type" (as defined by the CDNI Metadata Footprint Types registry, e.g., ipv4cidr, ipv6cidr, asn, or countrycode [I-D.ietf-cdni-metadata]) which identifies the contents and encoding of the individual footprint entries contained in the associated "footprint-value" array.

The "footprints" list MUST NOT contain multiple FCIFootprint objects of the same type. Footprint restriction information MAY be specified using multiple different footprint-types. If no footprint restriction list is specified (or an empty list is specified), it SHALL be understood that all footprint types MUST be reset to "global" coverage.

Note: Further optimization of the footprint object to provide quality information for a given footprint is certainly possible, however, it is not necessary for the basic interconnection of CDNs. The ability to transfer quality information in capabilities advertisements may be desirable and is noted here for completeness, however, the specifics of such mechanisms are outside the scope of this document.

Multiple FCIMapData objects with the same capability type are allowed within a given CDNI FCI Map response as long as the capability option footprint-value do not overlap, i.e., a given capability option value MUST NOT show up in multiple FCIMapData objects within a single CDNI FCI Map response. If multiple FCIMapData objects for a given capability type exist, those capability objects MUST have different footprint restrictions. Capability objects of a given capability type with identical footprint restrictions MUST be combined into a single capability object.

3.1.7. CDNI FCI Capabilities

3.1.7.1. Delivery Protocol

The delivery protocol refers to the protocol over which an end user (EU) has requested content. If a dCDN does not support the protocol requested by the client, then the dCDN is not a viable candidate for delegation.

Though the delivery protocol is specified in the URI scheme (as defined in [RFC3986]) of the client request URL, protocol feature subsets or augmented protocol feature sets MAY be defined and SHOULD correspond with the protocols listed in the CDNI Metadata Protocol Types registry, e.g., http1.1 or https1.1 [I-D.ietf-cdni-metadata].

The following example shows two lists of delivery protocols with different footprints.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 627
Content-Type: application/alto-fcimap+json
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.DeliveryProtocol"
        "capability-value": {
          "delivery-protocols": [
            "http1.1"
          ]
        }
      },
      { "capability-type": "FCI.DeliveryProtocol"
        "capability-value": {
          "delivery-protocols": [
            "https1.1"
          ]
        }
      },
      "footprints": [
        { "footprint-type": "ipv4cidr",
          "footprint-value": [
            "10.1.0.0/16",
            "10.10.10.0/24"
          ]
        }
      ]
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by IPv4 prefix).

A given protocol MUST NOT appear in multiple FCIMapData FCI.DeliveryProtocol object values.

3.1.7.2. Acquisition Protocol

The acquisition protocol refers to the protocol over which an end user (EU) has requested content. If a dCDN does not support the protocol requested by the client, then the dCDN is not a viable candidate for delegation.

Though the acquisition protocol is specified in the URI scheme (as defined in [RFC3986]) of the client request URL, protocol feature subsets or augmented protocol feature sets MAY be defined and SHOULD correspond with the protocols listed in the CDNI Metadata Protocol Types registry, e.g., http1.1 or https1.1 [I-D.ietf-cdni-metadata].

The following example shows two lists of acquisition protocols with different footprints.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 620
Content-Type: application/alto-fcimap+json
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.AcquisitionProtocol"
        "capability-value": {
          "acquisition-protocols": [
            "http1.1"
          ]
        }
      },
      { "capability-type": "FCI.AcquisitionProtocol"
        "capability-value": {
          "acquisition-protocols": [
            "https1.1"
          ]
        }
      },
      "footprints": [
        { "footprint-type": "asn",
          "footprint-value": [
            "AS0",
            "AS65535"
          ]
        }
      ]
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by Autonomous System number).

A given protocol MUST NOT appear in multiple FCIMapData FCI.AcquisitionProtocol value objects.

3.1.7.3. Redirection Mode

The redirection mode refers to the method(s) employed by request routers to perform request redirection. The CDNI framework [RFC7336] document describes four possible request routing modes:

- o DNS iterative (DNS-I)
- o DNS recursive (DNS-R)
- o HTTP iterative (HTTP-I)
- o HTTP recursive (HTTP-R)

[I-D.ietf-cdni-footprint-capabilities-semantics] defines the "CDNI Capabilities Redirection Modes" registry and the initial supported redirection mode values shown in parentheses above.

If a dCDN supports only a specific mode or subset of modes that does not overlap with the modes supported by the uCDN, then the dCDN might not be a viable candidate for delegation.

The following example shows two lists of redirection modes with different footprints.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json
```

```
HTTP/1.1 200 OK
Content-Length: 767
Content-Type: application/alto-fcimap+json
```

```
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.RedirectionMode",
        "capability-value": {
          "redirection-modes": [
            "DNS-I",
            "HTTP-I"
          ]
        }
      },
      { "capability-type": "FCI.RedirectionMode",
        "capability-value": {
          "redirection-modes": [
            "DNS-R",
            "HTTP-R"
          ]
        }
      },
      "footprints": [
        { "footprint-type": "countrycode",
          "footprint-value": [
            "SE"
          ]
        },
        { "footprint-type": "ipv6cidr",
          "footprint-value": [
            "9876:5432::1/36"
          ]
        }
      ]
    }
  }
}
```

In the above example, iterative redirection is supported globally, while recursive redirection is only supported in a restricted

footprint (in this case, specified by both Country Code and IPv6 prefix).

A given mode MUST NOT appear in multiple FCIMapData FCI.RedirectionMode object values.

3.1.7.4. Logging Capabilities

[I-D.ietf-cdni-logging] describes the "cdni_http_request_v1" logging record-types and optional vs. mandatory-to-implement logging fields for that record-type. It also allows new logging record-types and logging fields to be defined which would be optional for a dCDN to implement.

If a dCDN does not support certain logging parameters which may affect billing agreements or legal requirements of the uCDN, then the dCDN is not a viable candidate for delegation.

3.1.7.4.1. CDNI Logging Capability Object Serialization

The following shows an example of CDNI Logging Capability Object Serialization, for a CDN that supports the optional Content Collection ID logging field (but not the optional Session ID logging field) for the "cdni_http_request_v1" record type.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Logging",
      "capability-value": {
        "record-type": "cdni_http_request_v1",
        "fields": [ "s-ccid" ]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

The next example shows the CDNI Logging Capability Object Serialization, for a CDN that supports all optional fields for the "cdni_http_request_v1" record type.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Logging",
      "capability-value": {
        "record-type": "cdni_http_request_v1"
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

3.1.7.5. Metadata Capabilities

[I-D.ietf-cdni-metadata] describes GenericMetadata types which may be optional for a dCDN to implement, but which, if present, are mandatory-to-enforce. It also allows for new GenericMetadata to be defined which would be optional for a dCDN to implement.

If a dCDN does not support certain GenericMetadata types which are designated mandatory-to-enforce and may affect the correctness or security of the content being delivered, then the dCDN is not a viable candidate for delegation.

3.1.7.5.1. CDNI Metadata Capability Object Serialization

The following shows an example of CDNI Metadata Capability Object Serialization, for a CDN that supports only the SourceMetadata GenericMetadata type (i.e., it can acquire and deliver content, but cannot enforce and security policies, e.g., time, location, or protocol ACLs).

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": ["MI.SourceMetadata"]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

The next example shows the CDNI Metadata Capability Object Serialization, for a CDN that supports only structural metadata (i.e., it can parse metadata as a transit CDN, but cannot enforce security policies or deliver content).

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": []
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

4. CDNI FCI Capabilities Filtering Service

4.1. Filtered CDNI FCI Map

4.1.1. Media Type

Since a Filtered CDNI FCI Map is still a CDNI FCI Map, it uses the media type defined for CDNI FCI Map (see Section 3.1.1).

4.1.2. HTTP Method

A Filtered CDNI FCI Map is requested using the HTTP POST method.

4.1.3. Accept Input Parameters

TBD.

4.1.4. Capabilities

None.

4.1.5. Uses

TBD.

4.1.6. Response

The format is the same as unfiltered CDNI FCI Map (see Section 3.1.6).

4.1.7. Example

TBD.

5. Footprint via ALTO Network Map

5.1. ALTO Network Maps

The ALTO Protocol offers an information service "ALTO map service" that provides information to ALTO clients in the form of Network Map and Cost Map [RFC7285]. As an alternative to the explicit definition of a CDNI Footprint Type (e.g., ipv4cidr, ipv6cidr, as, countrycode), a reference to an ALTO network map can be used to define an FCI footprint. To enable such referencing to ALTO network maps, a new CDNI Footprint Type "altonetworkmap" is defined (see also Section 6.2).

The first altonetworkmap entry must be a URI for accessing the ALTO server that hosts the ALTO network map (as defined in the ALTO protocol specification [RFC7285]). All subsequent altonetworkmap entries must be of type PIDName (as defined in [RFC7285], where the PIDName corresponds to a PID in the ALTO network map referenced by the preceding URI. Parsing and processing of an ALTO network map follows the ALTO protocol specification [RFC7285].

5.2. Example ALTO Network Map for CDNI FCI Footprint

An ALTO client can retrieve a network map of media type 'application/alto-networkmap+json' under a given URI (e.g., 'http://alto.example.com/fcifootprint001') with a GET request for a network map as specified in the ALTO protocol [RFC7285]. The following network map would convey to a uCDN that the given dCDN (which would provide the map) has three footprints called "south-france" and "germany", and provides the corresponding IPv4 address ranges for these footprints.


```
GET /networkmap HTTP/1.1
Host: http://alto.example.com/fcifootprint001
Accept: application/alto-networkmap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 319
Content-Type: application/alto-networkmap+json

{
  "meta" : {
    "vtag": [
      { "resource-id": "my-eu-netmap",
        "tag": "1266506139"
      }
    ]
  },
  "network-map" : {
    "south-france" : {
      "ipv4" : [ "192.0.2.0/24", "198.51.100.0/25" ]
    },
    "germany" : {
      "ipv4" : [ "192.0.3.0/24" ]
    }
  }
}
```

5.3. Example of ALTO Network Map Footprint in FCI Map

To reference an ALTO network map as an FCI footprint, set the footprint-type to "altonetworkmap", and set the first entry in the footprint-value to the URI of the ALTO server hosting the network map, followed by a list of PID names contained in the network map.

The following example shows two lists of delivery protocols (see Section 3.1.7.1), with the second having an ALTO network map footprint.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json
```

```
HTTP/1.1 200 OK
Content-Length: 618
Content-Type: application/alto-fcimap+json
```

```
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [
          "http1.1"
        ]
      },
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [
          "values": [
            "https1.1"
          ],
          "footprints": [
            { "footprint-type": "altonetworkmap",
              "footprint-value": [
                "http://alto.example.com/fcifootprint001",
                "germany",
                "south-france"
              ]
            }
          ]
        ]
      }
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by an ALTO network map for Germany and Southern France).

6. IANA Considerations

6.1. ALTO Media Types

This document requests the registration of the following media types:

Type	Subtype
application	alto-cdni-fcimap+json
application	alto-cdni-fcimapfilter+json

6.1.1. ALTO CDNI FCI Map Media Type

Type name: application

Subtype name: alto-cdni-fcimap+json

Required parameters: none

Optional parameters: none

Encoding considerations:

Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations:

Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of [RFC7285]. Additional security considerations for the CDNI Footprint & Capabilities Advertisement interface are discussed in Section 7.

Interoperability considerations:

[RFC7285] and RFCthis specify the format of conforming messages and the interpretation thereof. [RFC Editor: Please replace RFCthis with the published RFC number for this document.]

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

Applications that use this media type:

ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Fragment identifier considerations: N/A

Additional information: N/A

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information:

Kevin Ma <kevin.j.ma@ericsson.com>

Intended usage: LIMITED USE

Restrictions on usage:

This media type is intended only for use in CDNI Footprint & Capabilities Advertisement interface protocol message exchanges.

Author: IETF CDNI working group

Change controller: IETF CDNI working group

Provisional registration: no

6.1.2. ALTO CDNI FCI Map Filter Media Type

Type name: application

Subtype name: alto-cdni-fcimapfilter+json

Required parameters: none

Optional parameters: none

Encoding considerations:

Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations:

Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of

[RFC7285]. Additional security considerations for the CDNI Footprint & Capabilities Advertisement interface are discussed in Section 7.

Interoperability considerations:

[RFC7285] and RFCthis specify the format of conforming messages and the interpretation thereof. [RFC Editor: Please replace RFCthis with the published RFC number for this document.]

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

Applications that use this media type:

ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Fragment identifier considerations: N/A

Additional information: N/A

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information:

Kevin Ma <kevin.j.ma@ericsson.com>

Intended usage: LIMITED USE

Restrictions on usage:

This media type is intended only for use in CDNI Footprint & Capabilities Advertisement interface protocol message exchanges.

Author: IETF CDNI working group

Change controller: IETF CDNI working group

Provisional registration: no

6.2. CDNI Footprint Types

This document requests the following addition to the "CDNI Metadata Footprint Types" registry:

Footprint Type	Description	Specification
altonetworkmap	URI of an ALTO Server hosting an ALTO network map, followed by a list of PID-names	RFCthis

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

7. Security Considerations

There are a number of security concerns associated with the FCI. The FCI essentially provides configuration information which the uCDN uses to make request routing decisions. Injection of fake capability advertisement messages or the interception and discard of real capability advertisement messages may be used for denial of service (e.g., by falsely advertising or deleting capabilities or preventing capability advertisements from reaching the uCDN). FCI messages may also be monitored to detect when CDN performance degrades or outages occur. Such information may be considered private by the dCDN.

dCDN capability advertisements MUST be authenticated by the uCDN to prevent unauthorized capability injection. uCDN FCI servers MUST be authenticated by the dCDN to prevent unauthorized interception of ALTO messages. Encryption MUST be used to ensure confidentiality of the dCDN's private messages.

7.1. Securing the CDNI Footprint & Capabilities Advertisement interface

An implementation of the CDNI Footprint & Capabilities Advertisement interface MUST support TLS transport as per [RFC2818] and [RFC7230]. The use of TLS for transport of the CDNI metadata interface messages allows:

- o The dCDN and uCDN to authenticate each other.

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

- o The dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI FCI messages from an authorized CDN);

- o CDNI FCI messages to be transmitted with confidentiality; and
- o The integrity of the CDNI FCI messages to be protected during the exchange.

In an environment where any such protection is required, TLS MUST be used (including authentication of the remote end) by the server-side (uCDN) and the client-side (dCDN) of the CDNI Footprint & Capabilities Advertisement interface unless alternate methods are used for ensuring the confidentiality of the information in the CDNI FCI messages (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

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

8. Acknowledgements

The authors would like to thank Jon Peterson, Ray van Brandenburg, Gilles Bertrand, and Scott Wainner for their timely reviews and invaluable comments.

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

9. References

9.1. Normative References

- [I-D.ietf-cdni-footprint-capabilities-semantic]
Seedorf, J., Peterson, J., Previdi, S., Brandenburg, R., and K. Ma, "CDNI Request Routing: Footprint and Capabilities Semantics", draft-ietf-cdni-footprint-capabilities-semantic-16 (work in progress), April 2016.
- [I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-25 (work in progress), April 2016.

- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma,
"CDN Interconnection Metadata", draft-ietf-cdni-
metadata-15 (work in progress), April 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
Protocol (HTTP/1.1): Message Syntax and Routing",
RFC 7230, DOI 10.17487/RFC7230, June 2014,
<<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S.,
Previdi, S., Roome, W., Shalunov, S., and R. Woundy,
"Application-Layer Traffic Optimization (ALTO) Protocol",
RFC 7285, DOI 10.17487/RFC7285, September 2014,
<<http://www.rfc-editor.org/info/rfc7285>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre,
"Recommendations for Secure Use of Transport Layer
Security (TLS) and Datagram Transport Layer Security
(DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May
2015, <<http://www.rfc-editor.org/info/rfc7525>>.

9.2. Informative References

- [I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing
Redirection interface for CDN Interconnection", draft-
ietf-cdni-redirection-18 (work in progress), April 2016.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818,
DOI 10.17487/RFC2818, May 2000,
<<http://www.rfc-editor.org/info/rfc2818>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March
2014, <<http://www.rfc-editor.org/info/rfc7159>>.

- [RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <<http://www.rfc-editor.org/info/rfc7336>>.
- [RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <<http://www.rfc-editor.org/info/rfc7337>>.
- [RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <<http://www.rfc-editor.org/info/rfc7736>>.

Appendix A. Capability Aggregation

The following sections show examples of three aggregation scenarios. In each case, CDN-U is the ultimate uCDN and CDN-P is the penultimate CDN which must perform capabilities aggregation.

A.1. Downstream CDN Aggregation

Figure A1 shows five organizationally different CDNs: CDN-U, CDN-P, and CDNS A, B, and C, the dCDNs of CDN-P which are being aggregated. Given the setup shown in Figure A1, we can construct a number of use cases, based on the coverage areas of each dCDN (i.e., CDNs P, A, B, and C). Note: In all cases, the reachability of the uCDN (i.e., CDN-U) is a don't care as it is assumed that the uCDN knows its own coverage area and is likely to favor itself in most situations, and if it has decided that it needs to delegate to a dCDN, then the only relevant question is if the dCDN can handle the request.

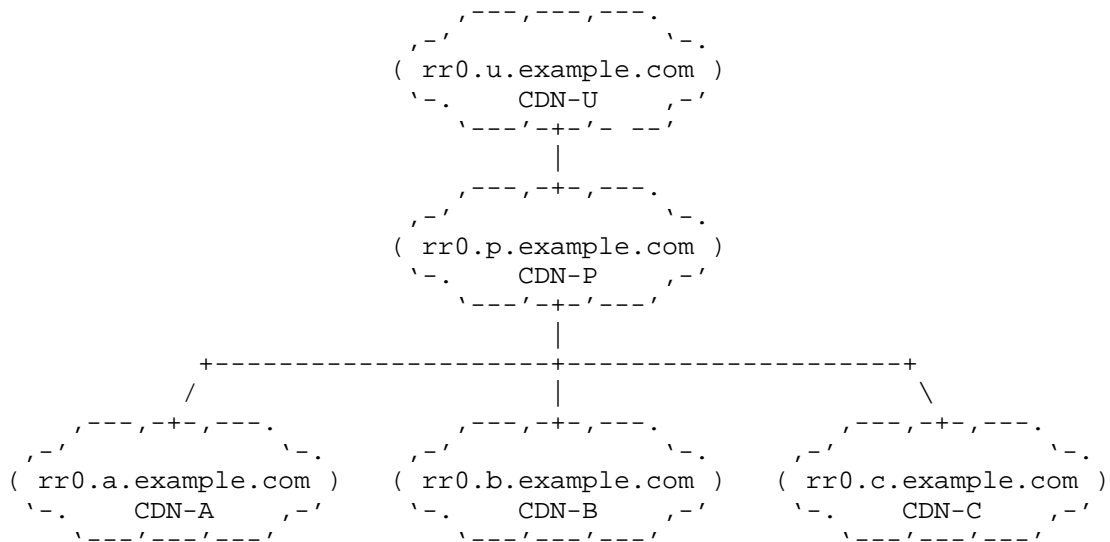


Figure A1: CDNI dCDN Request Router Aggregation

- o None of the four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, each CDN is likely to advertise footprint information with its capabilities, specifying its reachability. When CDN-P advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and CDNs A, B, and C. Note: CDN-P MAY exclude any dCDN, and consequently its footprint, per its own internal aggregation decision criteria.
- o All four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, none of the CDNs is likely to advertise any footprint information as none have any footprint restrictions. When CDN-P advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.
- o Some of the four dCDNs (CDNs P, A, B, and C) have global reachability and some do not. In this case, even though some dCDNs do not have global reachability, the aggregate of some dCDNs having global reachability and some not should still be global reachability (for the given capability). When CDN-P advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one dCDN has global reach as being supported with global reachability. It is up to the CDN-P request router to properly select a dCDN to process individual client requests and not choose a dCDN whose restricted footprint makes it unsuitable for delivering the requested content.

A.2. Internal Request Router Aggregation

Figure A2 shows CDN-U and CDN-P where CDN-P internally has four request routers: the authoritative request router rr0, and three other request routers rr1, rr2, and rr3. The use of multiple request routers may be used to distribute request routing load across resources, possibly in different geographic regions covered by CDN-P. Similar to Figure A1, the setup shown in Figure A2 requires the authoritative request router rr0 in CDN-P to aggregate capabilities information from downstream request routers rr1, rr2, and rr3. The primary difference between the scenario is that the request routers in Figure A2 are logically within the same CDN-P organization. The same reachability scenarios apply to Figure A2 as with Figure A1.

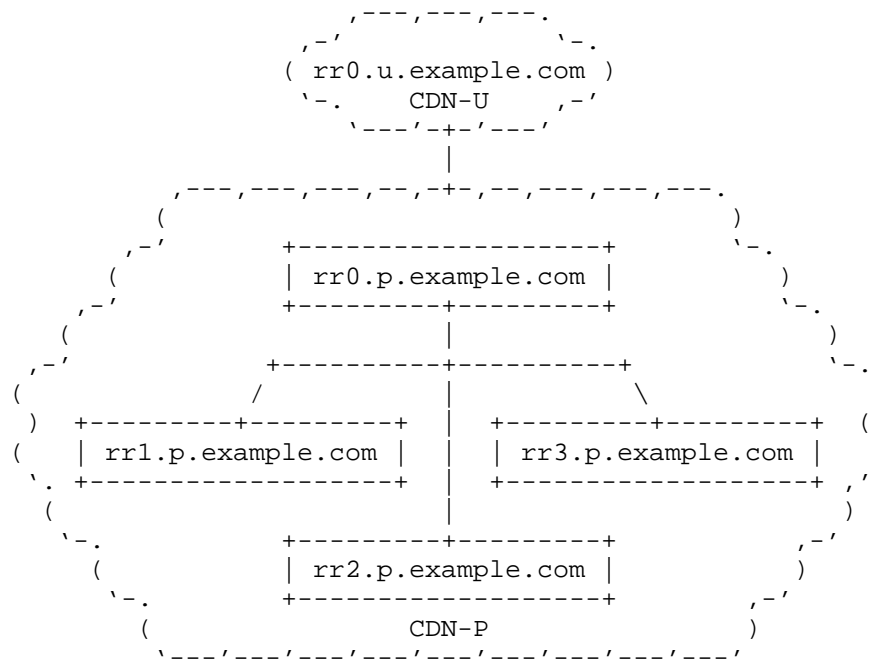


Figure A2: Local CDN Request Router Aggregation

- o None of the four CDN-P request routers have global reachability. In this case, each request router is likely to advertise footprint information with its capabilities, specifying its reachability. When rr0 advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and rr1, rr2, and rr3.
- o All four CDN-P request routers have global reachability. In this case, none of the request routers is likely to advertise any

footprint information as none has any footprint restrictions. When rr0 advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.

- o Some of the four CDN-P request routers have global reachability and some do not. In this case, even though some request routers do not have global reachability, the aggregate of some request routers having global reachability and some not should still be global reachability (for the given capability). When rr0 advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one request router has global reach as being supported with global reachability. It is up to the authoritative request router rr0 to properly select from the other request routers for any given request, and not choose a request router whose restricted footprint makes it unsuitable for delivering the requested content.

A.3. Internal Capability Aggregation

Figure A3 shows CDN-U and CDN-P where the delivery network of CDN-P is segregated by delivery protocol (e.g., RTSP, HTTP, and RTMP). Figure A3 differs from Figures A1 and A2 in that request router rr0 of CDN-P is not aggregating the capabilities advertisements of multiple other downstream request routers, but rather it is managing the disparate capabilities across resources within its own local CDN. Though not every delivery node has the same protocol capabilities, the aggregate delivery protocol capabilities advertised by CDN-A may include all delivery protocols. Note, Figure A3 should not be construed to imply anything about the coverage areas for each delivery protocol. They may all support the same delivery footprint, or they may have different delivery footprints. It is the responsibility of the request router rr0 to properly assign protocol-appropriate delivery nodes to individual content requests. If certain protocols have limited reachability, CDN-P may advertise footprint restrictions for each protocol.

It should be noted that though the delivery protocol capability was selected for this example, the concept of internal capability aggregation applies to all capabilities as discussed below.

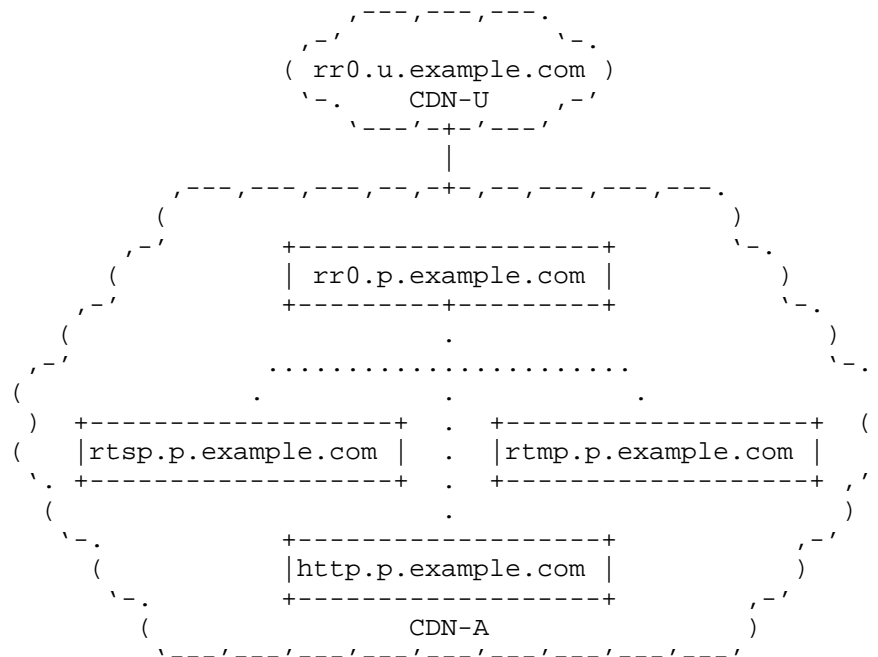


Figure A3: Local CDN Capability Segregation

Another situation in which physical footprint may not matter in an aggregated view has to do with feature support (e.g., new CDNI metadata features or new redirection modes). Situations often arise when phased roll-out of software upgrades, or staging network segregation result in only certain portions of a CDN's resources supporting the new feature set. The dCDN has a few options in this case:

- o Enforce atomic update: The dCDN does not advertise support for the new capability until all resources have been upgraded to support the new capability.
- o Transparent segregation: The dCDN advertises support for the new capability, and when requests are received that require the new capability, the dCDN request router properly selects a resource which supports that capability.
- o Advertised segregation: The dCDN advertises support for the new capability with a footprint restriction allowing the uCDN to make delegation decisions based on the dCDN's limit support.

The level of aggregation employed by the dCDN is likely to vary as business relationships dictate, however, the FCI should support all possible modes of operation.

Authors' Addresses

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
USA

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

Jan Seedorf
NEC
Kurfuerstenanlage 36
Heidelberg 69115
Germany

Phone: +49 6221 4342 221
Fax: +49 6221 4342 155
Email: seedorf@neclab.eu

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 28, 2013

R. Murray
B. Niven-Jenkins
Velocix (Alcatel-Lucent)
February 24, 2013

CDN Interconnect Triggers
draft-murray-cdni-triggers-02

Abstract

This document proposes a mechanism for a CDN to trigger activity in an interconnected CDN that is configured to deliver content on its behalf. The upstream CDN can use this mechanism to request that the downstream CDN pre-positions metadata or content, or that it re-validate or purge metadata or content. The upstream CDN can monitor the status of activity that it has triggered in the downstream CDN.

Requirements Language

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

Status of this Memo

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

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

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

This Internet-Draft will expire on August 28, 2013.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	4
1.1. Terminology	5
2. Model for CDNI Triggers	5
2.1. Timing of Triggered Activity	7
2.2. Trigger Results	7
3. Collections of Trigger Status Resources	7
4. CDNI Trigger interface	8
4.1. Creating Triggers	9
4.2. Checking Status	10
4.2.1. Polling Trigger Status Resource collections	11
4.2.2. Polling Trigger Status Resources	11
4.3. Deleting Triggers	11
4.4. Expiry of Trigger Status Resources	12
4.5. Error Handling	12
5. Properties of Triggers	13
5.1. Properties of Trigger Requests	13
5.1.1. Content URLs	14
5.2. Properties of Trigger Status Resources	14
5.3. Properties of ErrorDesc	15
5.4. Properties of Trigger Collections	15
5.5. Trigger Resource Simple Data Type Descriptions	16
5.5.1. TriggerType	16
5.5.2. TriggerStatus	16
5.5.3. URLs	16
5.5.4. AbsoluteTime	17
5.5.5. ErrorCode	17
6. JSON Encoding of Objects	17
6.1. JSON Encoding of Embedded Types	18
6.1.1. TriggerType	18
6.1.2. TriggerStatus	18
6.1.3. PatternMatch	18
6.1.4. ErrorDesc	19
6.1.5. ErrorCode	19
6.1.6. Relationship	19
6.2. MIME Media Types	20
7. Examples	21
7.1. Creating Triggers	21

7.1.1.	Preposition	21
7.1.2.	Invalidate	22
7.2.	Examining Trigger Status	24
7.2.1.	Collection of All Triggers	24
7.2.2.	Filtered Collections of Triggers	25
7.2.3.	Trigger Status Resources	27
7.2.4.	Polling for Change	29
7.2.5.	Cancelling or Removing a Trigger	32
7.2.6.	Error Reporting	34
8.	IANA Considerations	35
9.	Security Considerations	35
10.	Acknowledgements	35
11.	References	35
11.1.	Normative References	35
11.2.	Informative References	35
	Authors' Addresses	36

1. Introduction

[RFC6707] introduces the Problem scope for CDN Interconnection (CDNI) and lists the four categories of interfaces that may be used to compose a CDNI solution (Control, Metadata, Request Routing, Logging).

[I-D.ietf-cdni-framework] expands on the information provided in [RFC6707] and describes each of the interfaces and the relationships between them in more detail.

This draft concentrates on the "High" and "Medium" priority requirements for the CDNI Control Interface identified in section 4 of [I-D.ietf-cdni-requirements], reproduced here for convenience:

CNTL-1 [HIGH] The CDNI Control interface shall allow the Upstream CDN to request that the Downstream CDN (and, if cascaded CDNs are supported by the solution, that the potential cascaded Downstream CDNs) perform the following actions on an object or object set:

- * Mark an object(s) and/or its CDNI metadata as "stale" and revalidate them before they are delivered again.
- * Delete an object(s) and/or its CDNI metadata from the CDN surrogates and any storage. Only the object(s) and CDNI metadata that pertain to the requesting Upstream CDN are allowed to be purged.

CNTL-2 [HIGH] The CDNI Control interface shall allow the downstream CDN to report on the completion of these actions (by itself, and if cascaded CDNs are supported by the solution, by potential cascaded Downstream CDNs), in a manner appropriate for the action (e.g. synchronously or asynchronously).

CNTL-3 [HIGH] The CDNI Control interface shall support initiation and control by the Upstream CDN of pre-positioned CDNI metadata acquisition by the Downstream CDN.

CNTL-4 [MED] The CDNI Control interface should support initiation and control by the Upstream CDN of pre-positioned content acquisition by the Downstream CDN.

This document does not consider those parts of the control interface that relate to configuration, bootstrapping or authentication of CDN Interconnect interfaces.

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

- o Section 5 lists properties of Trigger Requests and Status Resources.
- o Section 6 defines a JSON encoding for Trigger Requests and Status Resources.
- o Section 7 contains example messages.

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

2. Model for CDNI Triggers

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

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

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

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

Requests to invalidate and purge metadata or content apply to all variants of that data with a given URI.

The dCDN maintains a collection of Trigger Status Resources for each uCDN, each uCDN only has access to its own collection and the location of that collection is shared when CDN interconnection is established.

To trigger activity in dCDN, uCDN will POST to the collection of Trigger Status Resources. If dCDN accepts the trigger, it creates a new Trigger Status Resource and returns its location to uCDN. To monitor progress, uCDN may GET the Trigger Status Resource. To cancel a trigger, or remove a trigger from the collection once its activity has been completed, uCDN may DELETE the Trigger Status Resource.

In addition to the collection of all Trigger Status Resources for uCDN, uCDN shall have access to filtered views of that collection. These filtered views are defined in Section 3 and include collections of active and completed triggers. These collections provide a mechanism for polling the status of multiple jobs.

Figure 1 is an example showing the basic message flow used by the uCDN to trigger activity in dCDN, and for uCDN to discover the status of that activity. Only successful triggering is shown. Examples of the messages are given in Section 7.

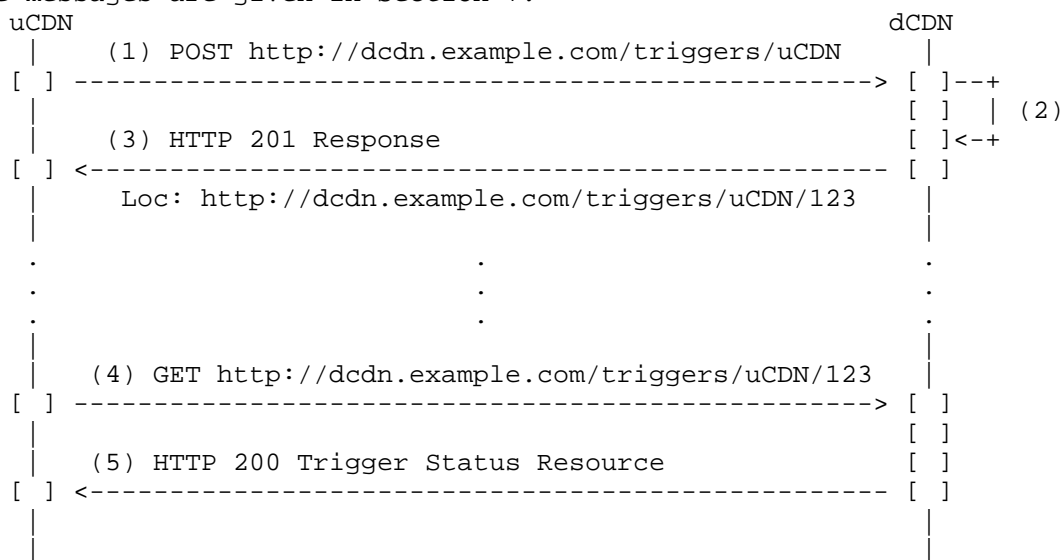


Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are:

1. uCDN triggers action in dCDN by posting to a collection of Trigger Status Resources, "http://dcdn.example.com/triggers/uCDN". The URL of this was given to uCDN when the trigger interface was established.
2. dCDN authenticates the request, validates the trigger and if it accepts the request, creates a new Trigger Status Resource.
3. dCDN responds to uCDN with an HTTP 201 response status, and the location of the Trigger Status Resource.
4. uCDN may repeatedly poll the Trigger Status Resource in dCDN.
5. dCDN responds with the Trigger Status Resource, describing progress or results of the triggered activity.

The remainder of this document describes the messages, Trigger Status Resources, and collections of Trigger Status Resources in more

detail.

2.1. Timing of Triggered Activity

Timing of triggered activity is under dCDN control, including its start-time and pacing of the activity in the network.

Invalidate and purge triggers MUST be applied to all data acquired before the trigger was created in dCDN. The dCDN MAY apply the triggers to data acquired after trigger creation.

If uCDN wishes to invalidate or purge content, then immediately preposition replacement content at the same URLs, it must ensure the dCDN has completed the invalidate/purge before initiating the prepositioning. If it fails to do that and the requests overlap, and dCDN passes the triggers on to a further dCDN in a cascade, that CDN may preposition content that has not yet been invalidated/purged in its uCDN.

2.2. Trigger Results

Each Trigger Request may operate on multiple data items. The trigger shall be reported as "complete" only if all actions can be completed successfully, otherwise it shall be reported as "failed". The reasons for failure and URLs or Patterns affected shall be enumerated in the Trigger Status Resource. For more detail, see section Section 4.5.

If a dCDN is also acting as uCDN in a cascade, it MUST forward triggers to any downstream CDNs that may have data affected by the trigger. The trigger MUST NOT be reported as complete in a CDN until it is complete in all of its downstream CDNs. A trigger MAY be reported as failed as soon as it fails in a CDN or in any of its downstream CDNs.

3. Collections of Trigger Status Resources

As described in Section 2, Trigger Status Resources exist in dCDN to report the status of activity triggered by each uCDN.

A collection of Trigger Status Resources is a resource that contains a reference to each Trigger Status Resource in that collection.

To trigger activity in dCDN, uCDN creates a new Trigger Status Resource by posting to dCDN's collection of uCDN's Trigger Status Resources. The URL of each Trigger Status Resource is generated by the dCDN when it accepts the trigger, and returned to uCDN. This

immediately enables uCDN to check the status of that trigger.

The dCDN must present a different set of Trigger Status Resources to each interconnected uCDN, only Trigger Status Resources belonging to a uCDN shall be visible to it. The dCDN may, for example, achieve this by offering different collection URLs to uCDNs, or by filtering the response based on the client uCDN.

The dCDN resource representing the collection of all uCDN's Trigger Status Resources is accessible to uCDN. This collection lists all uCDN triggers that have been accepted by dCDN, and have not yet been deleted by uCDN or expired and removed by dCDN.

In order to allow uCDN to check status of multiple jobs in a single request, dCDN shall also maintain collections representing filtered views of the collection of all Trigger Status Resources. The filtered collections are:

- o Pending - Trigger Status Resources for triggers that have been accepted, but not yet acted upon.
- o Active - Trigger Status Resources for triggered activity that is currently being processed in dCDN.
- o Complete - Trigger Status Resources representing activity that completed successfully.
- o Failed - Trigger Status Resources representing activity that failed.

4. CDNI Trigger interface

This section describes an interface to enable an upstream CDN to trigger defined activities in a downstream CDN. The interface is intended to be independent of the set of activities defined now, or that may be defined in future.

The CDNI Trigger interface is built on the principles of RESTful web services. Requests are made over HTTP, and the HTTP Method defines the operation the request would like to perform. The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses from servers implementing the CDNI Triggers interface that contain a response body SHOULD include an ETag to enable validation of cached versions of returned resources.

Servers implementing the CDNI Trigger interface MUST support the HTTP GET, HEAD, POST and DELETE methods. The only representation specified in this document is JSON.

Trigger Requests are POSTed to a URI in dCDN. If the trigger is accepted by dCDN, it creates a Trigger Status Resource and returns its URI to dCDN in an HTTP 201 response. The triggered activity can then be monitored by uCDN using that resource and the collections described in Section 3.

The URI that Trigger Requests should be POSTed to needs to be either discovered by or configured in the upstream CDN. Performing a GET on that URI retrieves a collection of the URIs of all Trigger Status Resources. The URI of each Trigger Status Resource is also returned to uCDN when it is created. This means all Trigger Status Resources can be discovered, so CDNI Trigger servers are free to assign whatever structure they desire to the URIs for CDNI Trigger resources. CDNI Trigger clients MUST NOT make any assumptions regarding the structure of CDNI Trigger URIs or the mapping between CDNI Trigger objects and their associated URIs. Therefore any URIs present in the examples below are purely illustrative and are not intended to impose a definitive structure on CDNI Trigger interface implementations.

The CDNI Trigger interface builds on top of HTTP, so CDNI Trigger servers may make use of any HTTP feature when implementing the CDNI Trigger interface. For example, a CDNI Trigger server may make use of HTTP's caching mechanisms to indicate that the returned response/representation has not been modified since it was last returned, reducing the processing needed to determine whether the status of triggered activity has changed.

This specification is neutral with regard to the transport below the HTTP layer.

[Editor's note: It is anticipated that decisions on use of HTTPS for other CDNI interfaces will be adopted for Triggers.]

Discovery of the CDNI Triggers Interface is outside the scope of this document. It is anticipated that a common mechanism for discovery of all CDNI interfaces will be defined.

The dCDN must ensure that activity triggered by uCDN only affects metadata or content originating from that uCDN. Since only one CDN can be authoritative for a given item of metadata or content, this requirement means there cannot be any "loops" in trigger requests between CDNs.

4.1. Creating Triggers

To create a new trigger, uCDN makes an HTTP POST to the unfiltered collection of its triggers. The request body of that POST is a

Trigger Request.

dCDN validates and authenticates that request, if it is malformed or uCDN does not have sufficient access rights it MAY reject the request immediately. In this case, it SHALL respond with an appropriate 4xx HTTP error code and no resource shall be created on dCDN.

If the request is accepted, uCDN SHALL create a new Trigger Status Resource. The HTTP response to dCDN SHALL have status code 201 and the URI of the Trigger Status Resource in the Location header field. The HTTP response MAY include the content of the newly created Trigger Status Resource, this is recommended particularly in cases where the trigger has completed immediately.

Once a Trigger Status Resource has been created dCDN MUST NOT re-use its location, even after that resource has been removed through deletion or expiry.

The "request" property of the Trigger Status Resource SHALL contain the information posted in the body of the Trigger Request. Note that this need not be a byte-for-byte copy. For example, in the JSON representation the dCDN may re-serialise the information differently.

If the trigger is queued by dCDN for later action, the "status" property of the Trigger Status Resource SHALL be "pending". Once trigger processing has started the "status" SHALL be "active".

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

If dCDN is not able to track triggered activity, it MAY indicate that it has undertaken to complete the activity but will not report completion or any further errors. To do this, it must set the trigger status to "complete", with an estimated completion time in the future ("etime" greater than "mtime").

Once created, Trigger Status Resources may be deleted by uCDN but not modified. The dCDN MUST reject PUT and POST requests from uCDN to Trigger Status Resources using HTTP status code 403.

4.2. Checking Status

The uCDN has two ways to check progress of activity it has triggered in dCDN, described in the following sections.

Because the triggers protocol is based on HTTP, Entity Tags may be used by the uCDN as cache validators, as defined in section 3.11 of [RFC2616], to check for change in status of a resource or collection of resources without re-fetching the whole resource or collection.

The dCDN should use the cache control headers for responses to GETs for Trigger Status Resources and Collections to indicate the frequency at which it recommends uCDN should poll for change.

4.2.1. Polling Trigger Status Resource collections

uCDN can fetch the collection of its Trigger Status Resources, or filtered views of that collection.

This makes it possible to poll status of all triggered activity in a single request. If dCDN moves a Trigger Status Resource from the Active to the Completed collection, uCDN may chose to fetch the result of that activity.

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

4.2.2. Polling Trigger Status Resources

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

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

4.3. Deleting Triggers

The uCDN MAY delete Trigger Status Resources at any time, using the HTTP DELETE method.

Once deleted, the references to a Trigger Status Resource MUST be removed from all Trigger Status Resource collections. Subsequent requests for the resource shall be handled as required by HTTP, and so will receive responses with status 404 or 410.

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

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

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

4.4. Expiry of Trigger Status Resources

The dCDN MAY choose to automatically delete Trigger Status Resources some time after they become completed or failed. In this case, dCDN will remove the resource and respond to subsequent requests for it with HTTP status 404 or 410.

If dCDN performs this housekeeping, it MUST have reported the length of time after which completed Trigger Status Resources become stale via a property of the collection of all Trigger Status Resources. It is recommended that Trigger Status Resources are automatically deleted 24 hours after they become completed or failed.

To ensure it has access to the status of its completed and failed triggers, it is recommended that uCDN's polling interval is half the time after which records for completed activity will be considered stale.

4.5. Error Handling

A CDNI Triggers server may reject a trigger request using HTTP status codes, for example 400 if the request is malformed or 401 if the client does not have permission to create triggers or it is trying to act on another CDN's data.

If any part of the trigger request fails the trigger shall be reported as "failed" once its activity is complete, or if no further errors will be reported. The "errors" property in the Trigger Status Resource will be used to enumerate which actions failed and the reasons for failure, and may be present while the trigger is still "pending" or "active" if the trigger is still running for some URLs or Patterns in the trigger request.

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

If a surrogate affected by a trigger is offline in dCDN, or dCDN is unable to pass a trigger request on to any of its affected dCDNs; dCDN should report an error if the request is abandoned, otherwise it

must keep the trigger in state "pending" or "active" until it's acted upon or uCDN chooses to cancel it. Or, if the request is queued and dCDN will not report further status, dCDN may report the trigger as "complete" with an "etime" in the future.

Note that an "invalidate" trigger may be reported as "complete" when surrogates that may have the data are offline, if those surrogates will not use the affected data without first revalidating it when they are back online. This does not apply to "preposition" or "purge" triggers.

5. Properties of Triggers

5.1. Properties of Trigger Requests

Properties of Trigger Requests are defined in the following subsections.

Property: type

Description: This property defines the type of the trigger:

Type: TriggerType

Mandatory: Yes

Property: metadata.urls

Description: The uCDN URL for the metadata the trigger applies to.

Type: URLs

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

Property: content.urls

Description: URLs of content data the trigger applies to, see Section 5.1.1.

Type: URLs

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

Property: metadata.patterns

Description: The metadata the trigger applies to.

Type: List of PatternMatch

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty, and metadata.patterns MUST NOT be present if the TriggerType is Preposition.

Property: content.patterns

Description: The content data the trigger applies to.

Type: List of PatternMatch

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty, and content.patterns MUST NOT be present if the TriggerType is Preposition.

5.1.1. Content URLs

To refer to content in dCDN, uCDN must present URLs in the same form clients will use to access content in that dCDN, after transformation to remove any surrogate-specific parts of a 302-redirect URL form. By definition, it is always possible to locate content based on URLs in this form.

If content URLs are transformed by an intermediate CDN in a cascade, that intermediate CDN must transform URLs in trigger requests it passes to its dCDN.

[Editor's note: Design for CDNI Metadata transformation, including discussion of URL transformation, is being undertaken as part of the work on the metadata interface. The intention is to align with that document or make reference to it when it's complete.]

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

5.2. Properties of Trigger Status Resources

Property: trigger

Description: The properties of trigger request that created this record.

Type: TriggerRequest

Mandatory: Yes

Property: ctime

Description: Time at which the request was received by dCDN. Time is local to dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Type: AbsoluteTime

Mandatory: Yes

Property: mtime

Description: Time at which the resource was last modified. Time is local to dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Type: AbsoluteTime

Mandatory: Yes

Property: etime

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

Type: AbsoluteTime
Mandatory: No

Property: status
Description: Current status of the triggered activity.
Type: TriggerStatus
Mandatory: Yes

Property: errors
Description: List of ErrorDesc.
Mandatory: No.

5.3. Properties of ErrorDesc

An ErrorDesc object is used to report failure for URLs and patterns in a trigger request.

Property: error
Type: ErrorCode.
Mandatory: Yes.

Description: List of metadata.urls, content.urls,
metadata.patterns, content.patterns

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

Mandatory: At least one of these properties is mandatory in each ErrorDesc.

Property: description
Description: A String containing a human-readable description of the error.
Mandatory: No.

5.4. Properties of Trigger Collections

Property: links
Description: References to Trigger Status Resources in the collection.
Type: List of Relationships.
Mandatory: Yes
Property: staleresourcetime

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

Type: Integer, time in seconds.

Mandatory: Yes, in the collection of all Trigger Status Resources if dCDN deletes stale entries. If the property is present in the filtered collections, it MUST have the same value as in the collection of all Trigger Status Resources.

5.5. Trigger Resource Simple Data Type Descriptions

This section describes the simpler data types that are used for properties of Trigger Status resources.

5.5.1. TriggerType

This type defines the type of action being triggered, permitted actions are:

- o Preposition - a request for dCDN to acquire metadata or content.
- o Invalidate - a request for dCDN to invalidate metadata or content. After servicing this request the dCDN will not use the specified data without first re-validating it using, for example, an "If-None-Match" HTTP request. The dCDN need not erase the associated data.
- o Purge - a request for dCDN to erase metadata or content. After servicing the request, the specified data must not be held on dCDN.

5.5.2. TriggerStatus

This type describes the current status of a Trigger, possible values are:

- o Pending - the trigger has not yet been acted upon.
- o Active - the trigger is currently being acted upon.
- o Complete - the triggered activity completed successfully, or the trigger has been accepted and no further status update will be made.
- o Failed - the triggered activity could not be completed.

5.5.3. URLs

This type describes a set of references to metadata or content, it is simply a list of absolute URLs.

5.5.4. AbsoluteTime

Times are expressed in seconds since the UNIX epoch.

5.5.5. ErrorCode

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

- o EMETA - dCDN was unable to acquire metadata required to fulfil the request.
- o ECONTENT - dCDN was unable to acquire content (preposition triggers only).
- o EPERM - uCDN does not have permission to trigger the requested activity (for example, the data is owned by another CDN).
- o EREJECT - dCDN is not willing to fulfil the request (for example, a preposition request for content at a time when dCDN would not accept Request Routing requests from uCDN).
- o ECDN - An internal error in dCDN or one of its downstream CDNs.

6. JSON Encoding of Objects

This encoding is based on that described in [I-D.ietf-cdni-metadata], but has been reproduced here while metadata work is in progress. Once that work is complete, the authors would look to align with the structure of the metadata draft and make reference to common definitions as appropriate.

The encoding for a CDNI Trigger object is a JSON object containing a dictionary of (key,value) pairs where the keys are the property names, and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource). Likewise, the values associated with each key are dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource).

The "trigger" property of the top level JSON object lists the requested action.

Key: trigger

Description: An object specifying the trigger type and a set of data to act upon.

Type: A JSON object.

Mandatory: Yes.

Object keys in JSON are case sensitive and therefore any dictionary key defined by this document (for example the names of CDNI Triggers object properties) MUST always be represented in lowercase.

In addition to the properties of an object, the following additional keys may be present.

Key: base

Description: Provides a prefix for any relative URLs in the object. This is similar to the XML base tag [XML-BASE]. If absent, all URLs in the remainder of the document must be absolute URLs.

Type: URI

Mandatory: No

Key: links

Description: The relationships of this object to other addressable objects.

Type: Array of Relationships.

Mandatory: Yes

6.1. JSON Encoding of Embedded Types

6.1.1. TriggerType

Key: type

Description: One of "preposition", "invalidate" or "purge".

Type: string

6.1.2. TriggerStatus

Key: status

Description: One of "pending", "active", "failed", "complete"

Type: string

6.1.3. PatternMatch

A PatternMatch is encoded as a JSON Object containing a string to match and flags describing the type of match.

Key: pattern

Description: A pattern for string matching. The pattern may contain the wildcards * and ?, where * matches any sequence of characters (including the empty string) and ? matches exactly one character. The three literals \ , * and ? should be escaped as \\, * and \?

Type: String
Mandatory: Yes
Key: case-sensitive
Description: Flag indicating whether or not case-sensitive matching should be used.
Type: Boolean
Mandatory: No, default is case-insensitive match.
Key: match-query-string
Description: Flag indicating whether or not the query string should be included in the pattern match.
Type: Boolean
Mandatory: No, default is not to include query.
Example of case-sensitive prefix match against "http://www.example.com/trailers/":
{
 "pattern": "http://www.example.com/trailers/*",
 "case-sensitive": true
}

6.1.4. ErrorDesc

ErrorDesc shall be encoded as a JSON object with the following keys:

Key: error
Type: ErrorCode
Mandatory: Yes
Keys: metadata.urls, content.urls
Type: Array of strings
Mandatory: At least one of metadata.* or content.* must be present.
Keys: metadata.patterns, content.patterns
Type: Array of PatternMatch
Mandatory: At least one of metadata.* or content.* must be present.
Key: description
Type: String
Mandatory: No.

6.1.5. ErrorCode

One of the strings "EMETA", "ECONTENT", "EPERM", "EREJECT" or "ECDN".

6.1.6. Relationship

JSON: A dictionary with the following keys:

- o href - The URI of the of the addressable object being referenced.
- o rel - The Relationship between the referring object and the object it is referencing.
- o type - The MIME Media Type of the referenced object. See Section 6.2 for the MIME Media Types of objects specified in this document.
- o title - An optional title for the Relationship/link.

Note: The above structure follows the pattern of atom:link in [RFC4287].

Example Relationship to a CDNI Trigger Resource within a CDNI Trigger Collection:

```
{
  "href": "http://triggers.cdni.example.com/trigger/12345",
  "rel": "Trigger",
  "type": "application/vnd.cdni.control.trigger.status+json"
}
```

The format of relationship values is expected to align with other CDNI interfaces. For example, rather than use simple names (like "Trigger" in this case), there may be a namespace that allows well-known and proprietary values to co-exist.

6.2. MIME Media Types

Table 1 lists the MIME Media Type for each trigger object (resource) that is retrievable through the CDNI Trigger interface.

Note: A prefix of "vnd.cdni" is used, however it is expected that a more appropriate prefix will be used if the CDNI WG accepts this document.

Data Object	MIME Media Type
TriggerStatus	application/ vnd.cdni.control.trigger.status+json
TriggerCollection	application/ vnd.cdni.control.trigger.collection+json

Table 1: MIME Media Types for CDNI Trigger resources

7. Examples

The following sections provide examples of different CDNI Trigger objects encoded as JSON.

No authentication is shown in the following illustrative examples, it is anticipated that authentication mechanisms will be aligned with other CDNI Interfaces as and when those mechanisms are defined.

Discovery of the triggers interface is out of scope of this document. In an implementation, all URLs are under control of dCDN and the uCDN must not attempt to ascribe any meaning to individual elements of the path. In examples in this section, the following URLs are used as the location of the collections of triggers:

- o Collection of all Triggers belonging to one uCDN:
http://dcdn.example.com/triggers
- o Filtered collections:
 - Pending: http://dcdn.example.com/triggers/pending
 - Active: http://dcdn.example.com/triggers/active
 - Complete: http://dcdn.example.com/triggers/complete
 - Failed: http://dcdn.example.com/triggers/failed

7.1. Creating Triggers

Examples of uCDN triggering activity in dCDN:

7.1.1. Preposition

An example of a preposition request, a POST to the "AllTriggers" collection.

Note that "metadata.patterns" and "content.patterns" are not allowed in a preposition trigger request.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: 315
```

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

    "metadata.urls" : [ "http://metadata.example.com/a/b/c" ],
```

```
    "content.urls" : [  
      "http://www.example.com/a/b/c/1",  
      "http://www.example.com/a/b/c/2",  
      "http://www.example.com/a/b/c/3",  
      "http://www.example.com/a/b/c/4"  
    ]  
  }  
}
```

RESPONSE:

```
HTTP/1.1 201 Created  
Date: Sat, 23 Feb 2013 14:20:06 GMT  
Content-Length: 472  
Content-Type: application/vnd.cdni.control.trigger.status+json  
Location: http://dcdn.example.com/triggers/0  
Server: example-server/0.1
```

```
{  
  "ctime": 1361629206,  
  "etime": 1361629214,  
  "mtime": 1361629206,  
  "status": "pending",  
  "trigger": {  
    "content.urls": [  
      "http://www.example.com/a/b/c/1",  
      "http://www.example.com/a/b/c/2",  
      "http://www.example.com/a/b/c/3",  
      "http://www.example.com/a/b/c/4"  
    ],  
    "metadata.urls": [  
      "http://metadata.example.com/a/b/c"  
    ],  
    "type": "preposition"  
  }  
}
```

7.1.2. Invalidate

An example of an invalidate request, another POST to the "AllTriggers" collection. This instructs dCDN to re-validate the content at "http://www.example.com/a/index.html", as well as any metadata and content whose URLs are prefixed by "http://metadata.example.com/a/b/" and "http://www.example.com/a/b/" respectively, using case-insensitive matching.

REQUEST:

```
POST /triggers HTTP/1.1
```

```
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: 352
```

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

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

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

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Length: 551
Content-Type: application/vnd.cdni.control.trigger.status+json
Location: http://dcdn.example.com/triggers/1
Server: example-server/0.1
```

```
{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
  "trigger": {
    "content.patterns": [
      {
        "case-sensitive": true,
        "pattern": "http://www.example.com/a/b/*"
      }
    ],
    "content.urls": [
      "http://www.example.com/a/index.html"
    ],
    "metadata.patterns": [
```

```
        {
            "pattern": "http://metadata.example.com/a/b/*"
        },
        {
            "type": "invalidate"
        }
    ]
}
```

7.2. Examining Trigger Status

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

7.2.1. Collection of All Triggers

The uCDN can fetch the set of all the triggers it has created and which have not yet been deleted or removed as expired. After creation of the "preposition" and "invalidate" triggers shown above, this collection might look as follows:

REQUEST:

```
GET /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "1484827667515030767"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.2. Filtered Collections of Triggers

The filtered collections are also available to uCDN. Before dCDN starts processing the two triggers shown above, both will appear in the collection of Pending Triggers, for example:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-970375801048973013"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

At this point, if no other triggers had been created, the other filtered views of the triggers would be empty. For example:

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 53
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-654105208640281650"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

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

7.2.3. Trigger Status Resources

The Trigger Status Resources can also be examined for detail about individual triggers. For example, for the "preposition" and "invalidate" triggers from previous examples:

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 472
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "-7651038857765989381"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629206,
  "etime": 1361629214,
  "mtime": 1361629206,
  "status": "pending",
  "trigger": {
    "content.urls": [
      "http://www.example.com/a/b/c/1",
      "http://www.example.com/a/b/c/2",
      "http://www.example.com/a/b/c/3",
      "http://www.example.com/a/b/c/4"
    ],
    "metadata.urls": [
      "http://metadata.example.com/a/b/c"
    ],
    "type": "preposition"
  }
}
```

REQUEST:

```
GET /triggers/1 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 551
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-1103964172288811711"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
  "trigger": {
    "content.patterns": [
      {
        "case-sensitive": true,
        "pattern": "http://www.example.com/a/b/*"
      }
    ],
    "content.urls": [
      "http://www.example.com/a/index.html"
    ],
    "metadata.patterns": [
      {
        "pattern": "http://metadata.example.com/a/b/*"
      }
    ],
    "type": "invalidate"
  }
}
```

7.2.4. Polling for Change

The uCDN may use the Entity Tags of collections or resources when polling for change in status, as shown in the following examples:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-970375801048973013"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-970375801048973013"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-7651038857765989381"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "-7651038857765989381"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

When the triggered activity is complete, the contents of the filtered collections will be updated, along with their Entity Tags. For example, when the two example triggers are complete, the collections of pending and complete triggers may look like:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-970375801048973013"
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 53
Expires: Sat, 23 Feb 2013 14:21:13 GMT
Server: example-server/0.1
ETag: "-7056231826368088123"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:13 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

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

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:20 GMT
Server: example-server/0.1
ETag: "2013095476705515794"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.5. Cancelling or Removing a Trigger

To request dCDN to cancel a Trigger, uCDN may delete the Trigger Resource. It may also delete completed and failed triggers to reduce the size of the collections. For example, to remove the "preposition" request from earlier examples:

REQUEST:

```
DELETE /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 204 No Content
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: example-server/0.1
```

This would, for example, cause the collection of completed triggers shown in the example above to be updated to:

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 239
Expires: Sat, 23 Feb 2013 14:21:20 GMT
Server: example-server/0.1
ETag: "4257416552489354137"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.6. Error Reporting

In this example uCDN has requested prepositioning of "http://newsite.example.com/index.html", but dCDN was unable to locate metadata for that site:

REQUEST:

```
GET /triggers/2 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 505
Expires: Sat, 23 Feb 2013 14:21:28 GMT
Server: example-server/0.1
ETag: "2621489144226897896"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:28 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629220,
  "errors": [
    {
      "content.urls": [
        "http://newsite.example.com/index.html"
      ],
      "description":
        "No HostIndex entry found for newsite.example.com",
      "error": "EMETA"
    }
  ],
  "etime": 1361629228,
  "mtime": 1361629224,
  "status": "active",
  "trigger": {
    "content.urls": [
      "http://newsite.example.com/index.html"
    ],
    "type": "preposition"
  }
}
```


8. IANA Considerations

TBD.

9. Security Considerations

The dCDN must ensure that each uCDN only has access to its own Trigger Status Resources.

It is anticipated that a common authentication mechanism will be used by this and other CDNI Interconnect interfaces, the mechanism must exist but is not identified in this document.

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

10. Acknowledgements

TBD.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

11.2. Informative References

- [I-D.ietf-cdni-framework] Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-03 (work in progress), February 2013.
- [I-D.ietf-cdni-metadata] Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata",

draft-ietf-cdni-metadata-00 (work in progress),
October 2012.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network
Interconnection (CDNI) Requirements",
draft-ietf-cdni-requirements-05 (work in progress),
February 2013.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom
Syndication Format", RFC 4287, December 2005.

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

[XML-BASE]

Marsh, J., Ed. and R. Tobin, Ed., "XML Base (Second
Edition) - <http://www.w3.org/TR/xmlbase/>", January 2009.

Authors' Addresses

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: rmurray@velocix.com

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

Email: ben@velocix.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 5, 2013

R. Murray
B. Niven-Jenkins
Velocix (Alcatel-Lucent)
April 3, 2013

CDNI Control Interface / Triggers
draft-murray-cdni-triggers-03

Abstract

This document describes the part of the CDN Interconnect Control Interface that allows a CDN to trigger activity in an interconnected CDN that is configured to deliver content on its behalf. The upstream CDN can use this mechanism to request that the downstream CDN pre-positions metadata or content, or that it re-validate or purge metadata or content. The upstream CDN can monitor the status of activity that it has triggered in the downstream CDN.

Requirements Language

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

Status of this Memo

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

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

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

This Internet-Draft will expire on October 5, 2013.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	4
1.1. Terminology	5
2. Model for CDNI Triggers	5
2.1. Timing of Triggered Activity	7
2.2. Trigger Results	7
3. Collections of Trigger Status Resources	7
4. CDNI Trigger interface	8
4.1. Creating Triggers	10
4.2. Checking Status	11
4.2.1. Polling Trigger Status Resource collections	11
4.2.2. Polling Trigger Status Resources	11
4.3. Deleting Triggers	11
4.4. Expiry of Trigger Status Resources	12
4.5. Error Handling	12
5. Properties of Triggers	13
5.1. Properties of Trigger Requests	13
5.1.1. Content URLs	14
5.2. Properties of Trigger Status Resources	14
5.3. Properties of ErrorDesc	15
5.4. Properties of Trigger Collections	16
5.5. Trigger Resource Simple Data Type Descriptions	16
5.5.1. TriggerType	16
5.5.2. TriggerStatus	16
5.5.3. URLs	17
5.5.4. AbsoluteTime	17
5.5.5. ErrorCode	17
6. JSON Encoding of Objects	17
6.1. JSON Encoding of Embedded Types	18
6.1.1. TriggerType	18
6.1.2. TriggerStatus	18
6.1.3. PatternMatch	19
6.1.4. ErrorDesc	19
6.1.5. ErrorCode	20
6.1.6. Relationship	20
6.2. MIME Media Types	20
7. Examples	21

7.1. Creating Triggers	21
7.1.1. Preposition	21
7.1.2. Invalidate	23
7.2. Examining Trigger Status	24
7.2.1. Collection of All Triggers	24
7.2.2. Filtered Collections of Triggers	25
7.2.3. Trigger Status Resources	27
7.2.4. Polling for Change	29
7.2.5. Cancelling or Removing a Trigger	32
7.2.6. Error Reporting	34
8. IANA Considerations	35
9. Security Considerations	35
10. Acknowledgements	35
11. References	35
11.1. Normative References	35
11.2. Informative References	35
Authors' Addresses	36

1. Introduction

[RFC6707] introduces the Problem scope for CDN Interconnection (CDNI) and lists the four categories of interfaces that may be used to compose a CDNI solution (Control, Metadata, Request Routing, Logging).

[I-D.ietf-cdni-framework] expands on the information provided in [RFC6707] and describes each of the interfaces and the relationships between them in more detail.

This draft concentrates on the "High" and "Medium" priority requirements for the CDNI Control Interface identified in section 4 of [I-D.ietf-cdni-requirements], reproduced here for convenience:

CNTL-1 [HIGH] The CDNI Control interface shall allow the Upstream CDN to request that the Downstream CDN (and, if cascaded CDNs are supported by the solution, that the potential cascaded Downstream CDNs) perform the following actions on an object or object set:

- * Mark an object or set of objects and/or its CDNI metadata as "stale" and revalidate them before they are delivered again
- * Delete an object or set of objects and/or its CDNI metadata from the CDN surrogates and any storage. Only the object(s) and CDNI metadata that pertain to the requesting Upstream CDN are allowed to be purged.

CNTL-2 [HIGH] The CDNI Control interface shall allow the Downstream CDN to report on the completion of these actions (by itself, and if cascaded CDNs are supported by the solution, by potential cascaded Downstream CDNs), in a manner appropriate for the action (e.g. synchronously or asynchronously). The confirmation receipt should include a success or failure indication. The failure indication is used if the Downstream CDN cannot delete the content in its storage.

CNTL-3 [HIGH] The CDNI Control interface shall support initiation and control by the Upstream CDN of pre-positioned CDNI metadata acquisition by the Downstream CDN.

CNTL-4 [MED] The CDNI Control interface should support initiation and control by the Upstream CDN of pre-positioned content acquisition by the Downstream CDN.

CNTL-12 [MED] The CDNI Control interface should allow for multiple content items identified by a Content Collection ID to be purged using a single Content Purge action.

This document describes the CI/T interface, Control Interface / Triggers. It does not consider those parts of the control interface that relate to configuration, bootstrapping or authentication of CDN Interconnect interfaces.

- o Section 2 outlines the model for the Trigger Interface at a high level.
- o Section 3 describes collections of Trigger Resources.
- o Section 4 defines the RESTful web service provided by dCDN.
- o Section 5 lists properties of Trigger Requests and Status Resources.
- o Section 6 defines a JSON encoding for Trigger Requests and Status Resources.
- o Section 7 contains example messages.

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

2. Model for CDNI Triggers

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

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

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

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

Requests to invalidate and purge metadata or content apply to all variants of that data with a given URI.

The dCDN maintains a collection of Trigger Status Resources for each uCDN, each uCDN only has access to its own collection and the location of that collection is shared when CDN interconnection is established.

To trigger activity in dCDN, uCDN will POST to the collection of Trigger Status Resources. If dCDN accepts the trigger, it creates a new Trigger Status Resource and returns its location to uCDN. To

monitor progress, uCDN may GET the Trigger Status Resource. To cancel a trigger, or remove a trigger from the collection once its activity has been completed, uCDN may DELETE the Trigger Status Resource.

In addition to the collection of all Trigger Status Resources for uCDN, uCDN shall have access to filtered views of that collection. These filtered views are defined in Section 3 and include collections of active and completed triggers. These collections provide a mechanism for polling the status of multiple jobs.

Figure 1 is an example showing the basic message flow used by the uCDN to trigger activity in dCDN, and for uCDN to discover the status of that activity. Only successful triggering is shown. Examples of the messages are given in Section 7.

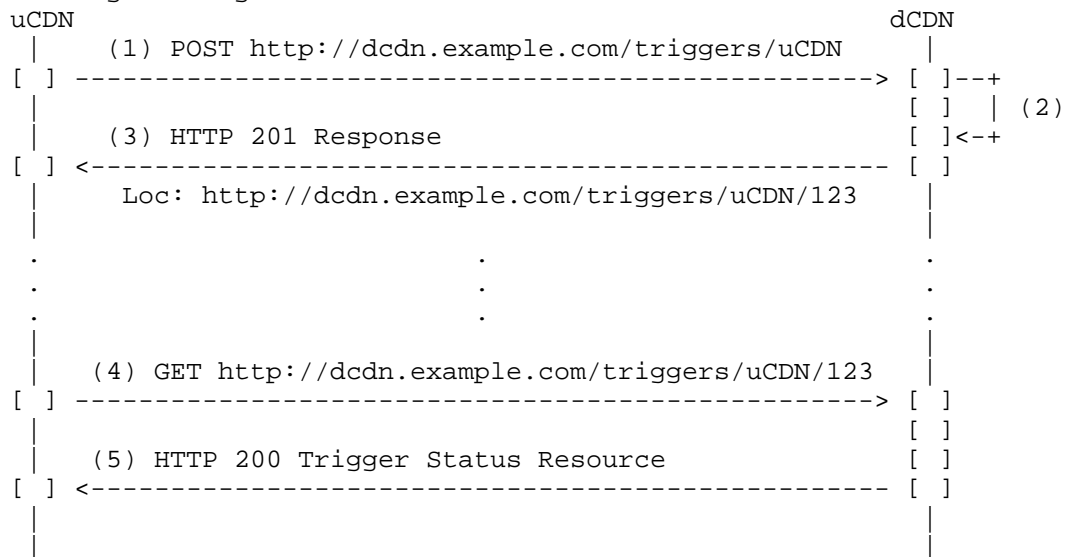


Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are:

1. uCDN triggers action in dCDN by posting to a collection of Trigger Status Resources, "http://dcdn.example.com/triggers/uCDN". The URL of this was given to uCDN when the trigger interface was established.
2. dCDN authenticates the request, validates the trigger and if it accepts the request, creates a new Trigger Status Resource.
3. dCDN responds to uCDN with an HTTP 201 response status, and the location of the Trigger Status Resource.

4. uCDN may repeatedly poll the Trigger Status Resource in dCDN.
5. dCDN responds with the Trigger Status Resource, describing progress or results of the triggered activity.

The remainder of this document describes the messages, Trigger Status Resources, and collections of Trigger Status Resources in more detail.

2.1. Timing of Triggered Activity

Timing of triggered activity is under dCDN control, including its start-time and pacing of the activity in the network.

Invalidate and purge triggers MUST be applied to all data acquired before the trigger was created in dCDN. The dCDN MAY apply the triggers to data acquired after trigger creation.

If uCDN wishes to invalidate or purge content, then immediately preposition replacement content at the same URLs, it must ensure the dCDN has completed the invalidate/purge before initiating the prepositioning. If it fails to do that and the requests overlap, and dCDN passes the triggers on to a further dCDN in a cascade, that CDN may preposition content that has not yet been invalidated/purged in its uCDN.

2.2. Trigger Results

Each Trigger Request may operate on multiple data items. The trigger shall be reported as "complete" only if all actions can be completed successfully, otherwise it shall be reported as "failed". The reasons for failure and URLs or Patterns affected shall be enumerated in the Trigger Status Resource. For more detail, see section Section 4.5.

If a dCDN is also acting as uCDN in a cascade, it MUST forward triggers to any downstream CDNs that may have data affected by the trigger. The trigger MUST NOT be reported as complete in a CDN until it is complete in all of its downstream CDNs. A trigger MAY be reported as failed as soon as it fails in a CDN or in any of its downstream CDNs.

3. Collections of Trigger Status Resources

As described in Section 2, Trigger Status Resources exist in dCDN to report the status of activity triggered by each uCDN.

A collection of Trigger Status Resources is a resource that contains

a reference to each Trigger Status Resource in that collection.

To trigger activity in dCDN, uCDN creates a new Trigger Status Resource by posting to dCDN's collection of uCDN's Trigger Status Resources. The URL of each Trigger Status Resource is generated by the dCDN when it accepts the trigger, and returned to uCDN. This immediately enables uCDN to check the status of that trigger.

The dCDN must present a different set of Trigger Status Resources to each interconnected uCDN, only Trigger Status Resources belonging to a uCDN shall be visible to it. The dCDN may, for example, achieve this by offering different collection URLs to uCDNs, or by filtering the response based on the client uCDN.

The dCDN resource representing the collection of all uCDN's Trigger Status Resources is accessible to uCDN. This collection lists all uCDN triggers that have been accepted by dCDN, and have not yet been deleted by uCDN or expired and removed by dCDN.

In order to allow uCDN to check status of multiple jobs in a single request, dCDN shall also maintain collections representing filtered views of the collection of all Trigger Status Resources. The filtered collections are:

- o Pending - Trigger Status Resources for triggers that have been accepted, but not yet acted upon.
- o Active - Trigger Status Resources for triggered activity that is currently being processed in dCDN.
- o Complete - Trigger Status Resources representing activity that completed successfully.
- o Failed - Trigger Status Resources representing activity that failed.

4. CDNI Trigger interface

This section describes an interface to enable an upstream CDN to trigger defined activities in a downstream CDN. The interface is intended to be independent of the set of activities defined now, or that may be defined in future.

The CI/T interface is built on the principles of RESTful web services. Requests are made over HTTP, and the HTTP Method defines the operation the request would like to perform. The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses from servers implementing the CI/T interface that contain a response body SHOULD include an ETag to enable validation of cached versions of returned

resources.

Servers implementing the CI/T interface MUST support the HTTP GET, HEAD, POST and DELETE methods. The only representation specified in this document is JSON.

Trigger Requests are POSTed to a URI in dCDN. If the trigger is accepted by dCDN, it creates a Trigger Status Resource and returns its URI to dCDN in an HTTP 201 response. The triggered activity can then be monitored by uCDN using that resource and the collections described in Section 3.

The URI that Trigger Requests should be POSTed to needs to be either discovered by or configured in the upstream CDN. Performing a GET on that URI retrieves a collection of the URIs of all Trigger Status Resources. The URI of each Trigger Status Resource is also returned to uCDN when it is created. This means all Trigger Status Resources can be discovered, so CI/T servers are free to assign whatever structure they desire to the URIs for CI/T resources. CI/T clients MUST NOT make any assumptions regarding the structure of CI/T URIs or the mapping between CI/T objects and their associated URIs. Therefore any URIs present in the examples below are purely illustrative and are not intended to impose a definitive structure on CI/T interface implementations.

The CI/T interface builds on top of HTTP, so CI/T servers may make use of any HTTP feature when implementing the CI/T interface. For example, a CI/T server may make use of HTTP's caching mechanisms to indicate that the returned response/representation has not been modified since it was last returned, reducing the processing needed to determine whether the status of triggered activity has changed.

This specification is neutral with regard to the transport below the HTTP layer.

[Editor's note: It is anticipated that decisions on use of HTTPS for other CDNI interfaces will be adopted for Triggers.]

Discovery of the CI/T Interface is outside the scope of this document. It is anticipated that a common mechanism for discovery of all CDNI interfaces will be defined.

The dCDN must ensure that activity triggered by uCDN only affects metadata or content originating from that uCDN. Since only one CDN can be authoritative for a given item of metadata or content, this requirement means there cannot be any "loops" in trigger requests between CDNs.

4.1. Creating Triggers

To create a new trigger, uCDN makes an HTTP POST to the unfiltered collection of its triggers. The request body of that POST is a Trigger Request.

dCDN validates and authenticates that request, if it is malformed or uCDN does not have sufficient access rights it MAY reject the request immediately. In this case, it SHALL respond with an appropriate 4xx HTTP error code and no resource shall be created on dCDN.

If the request is accepted, uCDN SHALL create a new Trigger Status Resource. The HTTP response to dCDN SHALL have status code 201 and the URI of the Trigger Status Resource in the Location header field. The HTTP response MAY include the content of the newly created Trigger Status Resource, this is recommended particularly in cases where the trigger has completed immediately.

Once a Trigger Status Resource has been created dCDN MUST NOT re-use its location, even after that resource has been removed through deletion or expiry.

The "request" property of the Trigger Status Resource SHALL contain the information posted in the body of the Trigger Request. Note that this need not be a byte-for-byte copy. For example, in the JSON representation the dCDN may re-serialise the information differently.

If the trigger is queued by dCDN for later action, the "status" property of the Trigger Status Resource SHALL be "pending". Once trigger processing has started the "status" SHALL be "active".

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

If dCDN is not able to track triggered activity, it MAY indicate that it has undertaken to complete the activity but will not report completion or any further errors. To do this, it must set the trigger status to "complete", with an estimated completion time in the future ("etime" greater than "mtime").

Once created, Trigger Status Resources may be deleted by uCDN but not modified. The dCDN MUST reject PUT and POST requests from uCDN to Trigger Status Resources using HTTP status code 403.

4.2. Checking Status

The uCDN has two ways to check progress of activity it has triggered in dCDN, described in the following sections.

Because the triggers protocol is based on HTTP, Entity Tags may be used by the uCDN as cache validators, as defined in section 3.11 of [RFC2616], to check for change in status of a resource or collection of resources without re-fetching the whole resource or collection.

The dCDN should use the cache control headers for responses to GETs for Trigger Status Resources and Collections to indicate the frequency at which it recommends uCDN should poll for change.

4.2.1. Polling Trigger Status Resource collections

uCDN can fetch the collection of its Trigger Status Resources, or filtered views of that collection.

This makes it possible to poll status of all triggered activity in a single request. If dCDN moves a Trigger Status Resource from the Active to the Completed collection, uCDN may chose to fetch the result of that activity.

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

4.2.2. Polling Trigger Status Resources

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

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

4.3. Deleting Triggers

The uCDN MAY delete Trigger Status Resources at any time, using the HTTP DELETE method.

Once deleted, the references to a Trigger Status Resource MUST be removed from all Trigger Status Resource collections. Subsequent requests for the resource shall be handled as required by HTTP, and so will receive responses with status 404 or 410.

If a "pending" Trigger Status Resource is deleted, dCDN SHOULD NOT

start processing of that activity. Deleting a "pending" trigger does not however guarantee that it is not started because, once it has triggered activity, uCDN cannot control the timing of that activity. Processing may, for example, start after the DELETE is sent by uCDN and before the DELETE is processed by dCDN.

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

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

4.4. Expiry of Trigger Status Resources

The dCDN MAY choose to automatically delete Trigger Status Resources some time after they become completed or failed. In this case, dCDN will remove the resource and respond to subsequent requests for it with HTTP status 404 or 410.

If dCDN performs this housekeeping, it MUST have reported the length of time after which completed Trigger Status Resources become stale via a property of the collection of all Trigger Status Resources. It is recommended that Trigger Status Resources are automatically deleted 24 hours after they become completed or failed.

To ensure it has access to the status of its completed and failed triggers, it is recommended that uCDN's polling interval is half the time after which records for completed activity will be considered stale.

4.5. Error Handling

A CI/T server may reject a trigger request using HTTP status codes, for example 400 if the request is malformed or 401 if the client does not have permission to create triggers or it is trying to act on another CDN's data.

If any part of the trigger request fails the trigger shall be reported as "failed" once its activity is complete, or if no further errors will be reported. The "errors" property in the Trigger Status Resource will be used to enumerate which actions failed and the reasons for failure, and may be present while the trigger is still "pending" or "active" if the trigger is still running for some URLs or Patterns in the trigger request.

Once a request has been accepted, processing errors are reported in the Trigger Status Resource using a list of "ErrorDesc". Each

ErrorDesc is used to report errors against one or more of the URLs or Patterns in the trigger request.

If a surrogate affected by a trigger is offline in dCDN, or dCDN is unable to pass a trigger request on to any of its affected dCDNs; dCDN should report an error if the request is abandoned, otherwise it must keep the trigger in state "pending" or "active" until it's acted upon or uCDN chooses to cancel it. Or, if the request is queued and dCDN will not report further status, dCDN may report the trigger as "complete" with an "etime" in the future.

Note that an "invalidate" trigger may be reported as "complete" when surrogates that may have the data are offline, if those surrogates will not use the affected data without first revalidating it when they are back online. This does not apply to "preposition" or "purge" triggers.

5. Properties of Triggers

5.1. Properties of Trigger Requests

Properties of Trigger Requests are defined in the following subsections.

Property: type

Description: This property defines the type of the trigger:

Type: TriggerType

Mandatory: Yes

Property: metadata.urls

Description: The uCDN URL for the metadata the trigger applies to.

Type: URLs

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

Property: content.urls

Description: URLs of content data the trigger applies to, see Section 5.1.1.

Type: URLs

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

Property: content.ccid

Description: The Content Collection IDentifier of data the trigger applies to.

Type: List of strings

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

Property: metadata.patterns

Description: The metadata the trigger applies to.

Type: List of PatternMatch

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty, and metadata.patterns MUST NOT be present if the TriggerType is Preposition.

Property: content.patterns

Description: The content data the trigger applies to.

Type: List of PatternMatch

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty, and content.patterns MUST NOT be present if the TriggerType is Preposition.

5.1.1. Content URLs

To refer to content in dCDN, uCDN must present URLs in the same form clients will use to access content in that dCDN, after transformation to remove any surrogate-specific parts of a 302-redirect URL form. By definition, it is always possible to locate content based on URLs in this form.

If content URLs are transformed by an intermediate CDN in a cascade, that intermediate CDN must transform URLs in trigger requests it passes to its dCDN.

[Editor's note: Design for CDNI Metadata transformation, including discussion of URL transformation, is being undertaken as part of the work on the metadata interface. The intention is to align with that document or make reference to it when it's complete.]

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

5.2. Properties of Trigger Status Resources

Property: trigger

Description: The properties of trigger request that created this record.

Type: TriggerRequest

Mandatory: Yes

Property: ctime

Description: Time at which the request was received by dCDN. Time is local to dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Type: AbsoluteTime

Mandatory: Yes

Property: mtime

Description: Time at which the resource was last modified.

Time is local to dCDN, there is no requirement to synchronise clocks between interconnected CDNs.

Type: AbsoluteTime

Mandatory: Yes

Property: etime

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

Type: AbsoluteTime

Mandatory: No

Property: status

Description: Current status of the triggered activity.

Type: TriggerStatus

Mandatory: Yes

Property: errors

Description: List of ErrorDesc.

Mandatory: No.

5.3. Properties of ErrorDesc

An ErrorDesc object is used to report failure for URLs and patterns in a trigger request.

Property: error

Type: ErrorCode.

Mandatory: Yes.

Description: List of metadata.urls, content.urls, metadata.patterns, content.patterns

Description: Metadata and content references copied from the trigger request. Only those URLs and patterns to which the error applies shall be included in each property, but those URLs and patterns shall be exactly as they appear in the request, dCDN must not generalise the URLs. (For example, if uCDN requests prepositioning of URLs

"http://ucdn.example.com/a" and "http://ucdn.example.com/b", dCDN may not generalise its error report to Pattern

"http://ucdn.example.com/*").

Mandatory: At least one of these properties is mandatory in each ErrorDesc.
Property: description
Description: A String containing a human-readable description of the error.
Mandatory: No.

5.4. Properties of Trigger Collections

Property: links
Description: References to Trigger Status Resources in the collection.
Type: List of Relationships.
Mandatory: Yes
Property: staleresourcetime
Description: The length of time for which dCDN guarantees to keep a completed Trigger Status Resource. After this time, dCDN MAY delete the resource and all references to it from collections.
Type: Integer, time in seconds.
Mandatory: Yes, in the collection of all Trigger Status Resources if dCDN deletes stale entries. If the property is present in the filtered collections, it MUST have the same value as in the collection of all Trigger Status Resources.

5.5. Trigger Resource Simple Data Type Descriptions

This section describes the simpler data types that are used for properties of Trigger Status resources.

5.5.1. TriggerType

This type defines the type of action being triggered, permitted actions are:

- o Preposition - a request for dCDN to acquire metadata or content.
- o Invalidate - a request for dCDN to invalidate metadata or content. After servicing this request the dCDN will not use the specified data without first re-validating it using, for example, an "If-None-Match" HTTP request. The dCDN need not erase the associated data.
- o Purge - a request for dCDN to erase metadata or content. After servicing the request, the specified data must not be held on dCDN.

5.5.2. TriggerStatus

This type describes the current status of a Trigger, possible values are:

- o Pending - the trigger has not yet been acted upon.
- o Active - the trigger is currently being acted upon.
- o Complete - the triggered activity completed successfully, or the trigger has been accepted and no further status update will be made.
- o Failed - the triggered activity could not be completed.

5.5.3. URLs

This type describes a set of references to metadata or content, it is simply a list of absolute URLs.

5.5.4. AbsoluteTime

Times are expressed in seconds since the UNIX epoch.

5.5.5. ErrorCode

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

- o EMETA - dCDN was unable to acquire metadata required to fulfil the request.
- o ECONTENT - dCDN was unable to acquire content (preposition triggers only).
- o EPERM - uCDN does not have permission to trigger the requested activity (for example, the data is owned by another CDN).
- o EREJECT - dCDN is not willing to fulfil the request (for example, a preposition request for content at a time when dCDN would not accept Request Routing requests from uCDN).
- o ECDN - An internal error in dCDN or one of its downstream CDNs.

6. JSON Encoding of Objects

This encoding is based on that described in [I-D.ietf-cdni-metadata], but has been reproduced here while metadata work is in progress. Once that work is complete, the authors would look to align with the structure of the metadata draft and make reference to common definitions as appropriate.

The encoding for a CI/T object is a JSON object containing a dictionary of (key,value) pairs where the keys are the property names, and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource). Likewise, the values associated with each key are

dependent on the specific object being encoded (i.e. dependent on the MIME Media Type of the returned resource).

The "trigger" property of the top level JSON object lists the requested action.

Key: trigger

Description: An object specifying the trigger type and a set of data to act upon.

Type: A JSON object.

Mandatory: Yes.

Object keys in JSON are case sensitive and therefore any dictionary key defined by this document (for example the names of CI/T object properties) MUST always be represented in lowercase.

In addition to the properties of an object, the following additional keys may be present.

Key: base

Description: Provides a prefix for any relative URLs in the object. This is similar to the XML base tag [XML-BASE]. If absent, all URLs in the remainder of the document must be absolute URLs.

Type: URI

Mandatory: No

Key: links

Description: The relationships of this object to other addressable objects.

Type: Array of Relationships.

Mandatory: Yes

6.1. JSON Encoding of Embedded Types

6.1.1. TriggerType

Key: type

Description: One of "preposition", "invalidate" or "purge".

Type: string

6.1.2. TriggerStatus

Key: status

Description: One of "pending", "active", "failed", "complete"

Type: string

6.1.3. PatternMatch

A PatternMatch is encoded as a JSON Object containing a string to match and flags describing the type of match.

Key: pattern

Description: A pattern for string matching. The pattern may contain the wildcards * and ?, where * matches any sequence of characters (including the empty string) and ? matches exactly one character. The three literals \, * and ? should be escaped as \\, * and \?

Type: String

Mandatory: Yes

Key: case-sensitive

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

Type: Boolean

Mandatory: No, default is case-insensitive match.

Key: match-query-string

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

Type: Boolean

Mandatory: No, default is not to include query.

Example of case-sensitive prefix match against

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

6.1.4. ErrorDesc

ErrorDesc shall be encoded as a JSON object with the following keys:

Key: error

Type: ErrorCode

Mandatory: Yes

Keys: metadata.urls, content.urls

Type: Array of strings

Mandatory: At least one of metadata.* or content.* must be present.

Keys: metadata.patterns, content.patterns

Type: Array of PatternMatch

Mandatory: At least one of metadata.* or content.* must be present.

Key: description

Type: String
Mandatory: No.

6.1.5. ErrorCode

One of the strings "EMETA", "ECONTENT", "EPERM", "EREJECT" or "ECDN".

6.1.6. Relationship

JSON: A dictionary with the following keys:

- o href - The URI of the of the addressable object being referenced.
- o rel - The Relationship between the referring object and the object it is referencing.
- o type - The MIME Media Type of the referenced object. See Section 6.2 for the MIME Media Types of objects specified in this document.
- o title - An optional title for the Relationship/link.

Note: The above structure follows the pattern of atom:link in [RFC4287].

Example Relationship to a CI/T Resource within a CI/T Collection:

```
{
  "href": "http://triggers.cdni.example.com/trigger/12345",
  "rel": "Trigger",
  "type": "application/vnd.cdni.control.trigger.status+json"
}
```

The format of relationship values is expected to align with other CDNI interfaces. For example, rather than use simple names (like "Trigger" in this case), there may be a namespace that allows well-known and proprietary values to co-exist.

6.2. MIME Media Types

Table 1 lists the MIME Media Type for each trigger object (resource) that is retrievable through the CI/T interface.

Note: A prefix of "vnd.cdni" is used, however it is expected that a more appropriate prefix will be used if the CDNI WG accepts this document.

Data Object	MIME Media Type
TriggerStatus	application/ vnd.cdni.control.trigger.status+json
TriggerCollection	application/ vnd.cdni.control.trigger.collection+json

Table 1: MIME Media Types for CDNI Trigger resources

7. Examples

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

No authentication is shown in the following illustrative examples, it is anticipated that authentication mechanisms will be aligned with other CDNI Interfaces as and when those mechanisms are defined.

Discovery of the triggers interface is out of scope of this document. In an implementation, all URLs are under control of dCDN and the uCDN must not attempt to ascribe any meaning to individual elements of the path. In examples in this section, the following URLs are used as the location of the collections of triggers:

- o Collection of all Triggers belonging to one uCDN:
http://dcdn.example.com/triggers
- o Filtered collections:
 - Pending: http://dcdn.example.com/triggers/pending
 - Active: http://dcdn.example.com/triggers/active
 - Complete: http://dcdn.example.com/triggers/complete
 - Failed: http://dcdn.example.com/triggers/failed

7.1. Creating Triggers

Examples of uCDN triggering activity in dCDN:

7.1.1. Preposition

An example of a preposition request, a POST to the "AllTriggers" collection.

Note that "metadata.patterns" and "content.patterns" are not allowed in a preposition trigger request.

REQUEST:


```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: 315
```

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

    "metadata.urls" : [ "http://metadata.example.com/a/b/c" ],
    "content.urls" : [
      "http://www.example.com/a/b/c/1",
      "http://www.example.com/a/b/c/2",
      "http://www.example.com/a/b/c/3",
      "http://www.example.com/a/b/c/4"
    ]
  }
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sat, 23 Feb 2013 14:20:06 GMT
Content-Length: 472
Content-Type: application/vnd.cdni.control.trigger.status+json
Location: http://dcdn.example.com/triggers/0
Server: example-server/0.1
```

```
{
  "ctime": 1361629206,
  "etime": 1361629214,
  "mtime": 1361629206,
  "status": "pending",
  "trigger": {
    "content.urls": [
      "http://www.example.com/a/b/c/1",
      "http://www.example.com/a/b/c/2",
      "http://www.example.com/a/b/c/3",
      "http://www.example.com/a/b/c/4"
    ],
    "metadata.urls": [
      "http://metadata.example.com/a/b/c"
    ],
    "type": "preposition"
  }
}
```

7.1.2. Invalidate

An example of an invalidate request, another POST to the "AllTriggers" collection. This instructs dCDN to re-validate the content at "http://www.example.com/a/index.html", as well as any metadata and content whose URLs are prefixed by "http://metadata.example.com/a/b/" and "http://www.example.com/a/b/" respectively, using case-insensitive matching.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/vnd.cdni.control.trigger.request+json
Content-Length: 352

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

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

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

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Length: 551
Content-Type: application/vnd.cdni.control.trigger.status+json
Location: http://dcdn.example.com/triggers/1
Server: example-server/0.1

{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
```

```
"trigger": {
  "content.patterns": [
    {
      "case-sensitive": true,
      "pattern": "http://www.example.com/a/b/*"
    }
  ],
  "content.urls": [
    "http://www.example.com/a/index.html"
  ],
  "metadata.patterns": [
    {
      "pattern": "http://metadata.example.com/a/b/*"
    }
  ],
  "type": "invalidate"
}
```

7.2. Examining Trigger Status

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

7.2.1. Collection of All Triggers

The uCDN can fetch the set of all the triggers it has created and which have not yet been deleted or removed as expired. After creation of the "preposition" and "invalidate" triggers shown above, this collection might look as follows:

REQUEST:

```
GET /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "1484827667515030767"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.2. Filtered Collections of Triggers

The filtered collections are also available to uCDN. Before dCDN starts processing the two triggers shown above, both will appear in the collection of Pending Triggers, for example:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-970375801048973013"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

At this point, if no other triggers had been created, the other filtered views of the triggers would be empty. For example:

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 53
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-654105208640281650"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

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

7.2.3. Trigger Status Resources

The Trigger Status Resources can also be examined for detail about individual triggers. For example, for the "preposition" and "invalidate" triggers from previous examples:

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 472
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "-7651038857765989381"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629206,
  "etime": 1361629214,
  "mtime": 1361629206,
  "status": "pending",
  "trigger": {
    "content.urls": [
      "http://www.example.com/a/b/c/1",
      "http://www.example.com/a/b/c/2",
      "http://www.example.com/a/b/c/3",
      "http://www.example.com/a/b/c/4"
    ],
    "metadata.urls": [
      "http://metadata.example.com/a/b/c"
    ],
    "type": "preposition"
  }
}
```

REQUEST:

```
GET /triggers/1 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 551
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-1103964172288811711"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629208,
  "etime": 1361629216,
  "mtime": 1361629208,
  "status": "pending",
  "trigger": {
    "content.patterns": [
      {
        "case-sensitive": true,
        "pattern": "http://www.example.com/a/b/*"
      }
    ],
    "content.urls": [
      "http://www.example.com/a/index.html"
    ],
    "metadata.patterns": [
      {
        "pattern": "http://metadata.example.com/a/b/*"
      }
    ],
    "type": "invalidate"
  }
}
```

7.2.4. Polling for Change

The uCDN may use the Entity Tags of collections or resources when polling for change in status, as shown in the following examples:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-970375801048973013"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sat, 23 Feb 2013 14:21:09 GMT
Server: example-server/0.1
ETag: "-970375801048973013"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:09 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-7651038857765989381"
```

RESPONSE:

```
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sat, 23 Feb 2013 14:21:08 GMT
Server: example-server/0.1
ETag: "-7651038857765989381"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:08 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

When the triggered activity is complete, the contents of the filtered collections will be updated, along with their Entity Tags. For example, when the two example triggers are complete, the collections of pending and complete triggers may look like:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "-970375801048973013"
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 53
Expires: Sat, 23 Feb 2013 14:21:13 GMT
Server: example-server/0.1
ETag: "-7056231826368088123"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:13 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

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

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 422
Expires: Sat, 23 Feb 2013 14:21:20 GMT
Server: example-server/0.1
ETag: "2013095476705515794"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json

{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/0",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    },
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.5. Cancelling or Removing a Trigger

To request dCDN to cancel a Trigger, uCDN may delete the Trigger Resource. It may also delete completed and failed triggers to reduce the size of the collections. For example, to remove the "preposition" request from earlier examples:

REQUEST:

```
DELETE /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 204 No Content
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: example-server/0.1
```

This would, for example, cause the collection of completed triggers shown in the example above to be updated to:

REQUEST:

```
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 239
Expires: Sat, 23 Feb 2013 14:21:20 GMT
Server: example-server/0.1
ETag: "4257416552489354137"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:20 GMT
Content-Type: application/vnd.cdni.control.trigger.collection+json
```

```
{
  "links": [
    {
      "href": "http://dcdn.example.com/triggers/1",
      "rel": "Trigger",
      "type": "application/vnd.cdni.control.trigger.status+json"
    }
  ],
  "staleresourcetime": 86400
}
```

7.2.6. Error Reporting

In this example uCDN has requested prepositioning of "http://newsite.example.com/index.html", but dCDN was unable to locate metadata for that site:

REQUEST:

```
GET /triggers/2 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 505
Expires: Sat, 23 Feb 2013 14:21:28 GMT
Server: example-server/0.1
ETag: "2621489144226897896"
Cache-Control: max-age=60
Date: Sat, 23 Feb 2013 14:20:28 GMT
Content-Type: application/vnd.cdni.control.trigger.status+json
```

```
{
  "ctime": 1361629220,
  "errors": [
    {
      "content.urls": [
        "http://newsite.example.com/index.html"
      ],
      "description":
        "No HostIndex entry found for newsite.example.com",
      "error": "EMETA"
    }
  ],
  "etime": 1361629228,
  "mtime": 1361629224,
  "status": "active",
  "trigger": {
    "content.urls": [
      "http://newsite.example.com/index.html"
    ],
    "type": "preposition"
  }
}
```

8. IANA Considerations

TBD.

9. Security Considerations

The dCDN must ensure that each uCDN only has access to its own Trigger Status Resources.

It is anticipated that a common authentication mechanism will be used by this and other CDNI Interconnect interfaces, the mechanism must exist but is not identified in this document.

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

10. Acknowledgements

TBD.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

11.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-03 (work in progress), February 2013.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata",

draft-ietf-cdni-metadata-01 (work in progress),
February 2013.

[I-D.ietf-cdni-requirements]

Leung, K. and Y. Lee, "Content Distribution Network
Interconnection (CDNI) Requirements",
draft-ietf-cdni-requirements-05 (work in progress),
February 2013.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom
Syndication Format", RFC 4287, December 2005.

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

[XML-BASE]

Marsh, J., Ed. and R. Tobin, Ed., "XML Base (Second
Edition) - <http://www.w3.org/TR/xmlbase/>", January 2009.

Authors' Addresses

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: rmurray@velocix.com

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

Email: ben@velocix.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: October 2013

M-K. Shin
S. Lee
ETRI
D. Chang
T. Kwon
SNU
February 15, 2013

CDNI Request Routing with SDN
draft-shin-cdni-request-routing-sdn-01

Abstract

Software-defined networking (SDN) is emerging and intensively discussed as one of the most promising technologies to provide centralized, programmable control planes for network service providers (NSPs). In this sense, SDN could be also considered as one of candidates to facilitate CDNI Request Routing. This document discusses how SDN can be used for downstream CDN selection 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 to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on January 1, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. SDN Overview and Assumption	3
3. SDN within CDNI Request Routing	4
4. Selection of a Downstream CDN with SDN	5
5. Example of Content Request Redirection and Path Setup for Content Delivery with SDN	6
6. Advantages of using SDN	7
7. Further Considerations	7
8. Security Considerations	7
9. Acknowledgements	7
10. Informative References	7
Author's Address	9

1. Introduction

The CDNI PS and framework documents [RFC6707][I-D.ietf-cdni-framework] define the following four interfaces for CDNI; Request Routing Interface, Metadata Interface, Logging Interface, and CDNI Control Interface. As for the Request Routing - Redirection, HTTP and DNS are being discussed as one of candidate protocols. Recently, Software-defined networking (SDN) is emerging and intensively discussed as one of the most promising technologies to provide centralized, programmable control planes for network service providers (NSPs). In this sense, SDN could be also considered as one of candidates to facilitate CDNI Request Routing. This document discusses how OF/SDN technology can be integrated in CDNI request routing.

1.1. Terminology

This document draws freely on the terminology defined in [RFC3466] and [RFC6706].

We also introduce the following terms, tentatively:

CDN Frontend Server (CFS): CDN Frontend Server (CFS): It is a server which has SDN capability. A Network Service Providers (NSP) registers the address of its own CFS to DNS. In this way, a CFS can redirect "request" messages to its SDN controller.

2. SDN Overview and Assumption

SDN is a new networking technology which allows centralized, programmable control planes, so that NSPs can control and manage directly their own virtualized resources and networks without recognizing detailed hardware technologies. To achieve this, with SDN, control and data planes are separated which allows control to be directly programmable and manageable in a centralized manner and data plane to be simplified and abstracted rather than specialized hardware. Since work on SDN architecture and framework is still being discussed and is not standardized yet, in this document, it is assumed that OpenFlow is as one of architectural components for SDN framework to facilitate CDNI Request Routing, as an example, but any other existing and/or possible solutions for SDN, such as I2RS and I2AEX could be also integrated without any big modifications.

Most modern Ethernet switches and routers contain flowtables that run at line-rate to implement firewalls, NAT, QoS, and to collect statistics. While each vendor's flowtable is different, OpenFlow's

basic idea composed an interesting common set of functions that run in many switches and routers. OpenFlow exploits this common set of functions. OpenFlow provides an open protocol to program the flowtable in different switches and routers. In an OpenFlow network, a central controller manages switches that support the concept of a flow, a stream of related packets that are processed in the same way. Every switch maintains a flow table containing a set of rules, where each rule includes a pattern that is the set of packets belonging to the flow, a priority that disambiguates overlapping rules, an expiration time, a list of actions to apply to the packets, and counters to measure the traffic. To process an incoming packet, the switch identifies the matching rule with the highest priority, updates the counters of the rule, and applies the actions. If no matching rule is found, the switch forwards the packet to the controllers and awaits further instructions.

3. SDN within CDNI Request Routing

The scope of the CDNI Request Routing Interface SHOULD contain two functionalities [I-D.ietf-cdni-framework] :

- o Request Routing Interface - Footprint and Capabilities Advertisement;
the asynchronous advertisement of footprint and capabilities by a dCDN that allows a uCDN to decide whether to redirect particular user requests to that dCDN;
- o Request Routing Interface - Redirection;
the synchronous operation of actually redirecting a user request

First of all, it is assumed that ALTO is used for Request Routing Interface - Footprint and Capabilities Advertisement. Details and examples on how a downstream CDN can advertise its footprint and other information by means of ALTO are being discussed in [I-D.seedorf-cdni-request-routing-alto]. Application Layer Traffic Optimization (ALTO) is an approach for guiding the resource provider selection process in distributed applications that can choose among several candidate resources providers to retrieve a given resource. By conveying network layer (topology) information, an ALTO server can provide important information to guide the resource provider selection process in distributed applications.

As for the Request Routing - Redirection, HTTP and DNS are being discussed as one of candidate protocols. Recently, SDN is emerging and intensively discussed as one of the most promising technologies to provide centralized, programmable control planes for network service providers (NSPs). In this sense, SDN could be also considered

as one of candidates to facilitate CDNI Request Routing as well as HTTP and DNS. This document discusses how SDN technology can be integrated in CDNI request routing.

4. Selection of a downstream CDN with SDN

SDN can help the upstream CDN provider to select a proper downstream CDN provider for a given end user request as follows. It is assumed that each downstream CDN provider hosts SDN controller.

An example of operation is as follows :

0) dCDN advertises information relevant to its delivery capabilities (e.g. content availability, geographic footprint, etc.) using ALTO extension (e.g., I2AEX) provisioning prior to any content requests being redirected.

1) A content request from a user agent arrives in the CFS of uCDN.

2) The CFS at uCDN relays the message to the its SDN controller by a "Packet-In" message.

3) ALTO client at the OF controller requests the best dCDN information to ALTO server (ALTO cost map and/or other information may be used).

4) ALTO server responses and then OF controller in uCDN knows which is the best dCDN.

5) The SDN controller sends a query to the SDN controller of the best dCDN

6) (uCDN redirects the request to the best dCDN).

5. Example of Content Request Redirection and Path Setup for Content Delivery with SDN

SDN can help the upstream CDN provider to redirect a content request message to a downstream CDN provider for a given end user request as follows. It is assumed that the upstream and the downstream CDN providers have SDN controllers. And OF/SDN sets up the path for the content delivery.

An example of operation is as follows :

0) Content distribution metadata is pre-positioned between CDNs prior

to any content requests being redirected; that is, a controller in uCDN knows its own surrogates in other CDNs and information relevant to its delivery capabilities (e.g. geo-blocking information, availability windows, desired distribution policy, etc.)

1) An end user issues an HTTP GET message to get content. By contacting DNS, this message is forwarded to the CDN frontend server (CFS) of uCDN which has SDN capability.

2) The CFS of uCDN relays this message to its own SDN controller by "Packet-In" message in SDN protocol.

3) The controller of uCDN checks content distribution metadata, and sends a query to the controller of dCDN whether it can deliver the content. In the query, the source address of the request packets (i.e. the host address of the user agent) is included. Optionally, a QoS requirement for the content delivery may be specified in the query as well. And there can be multiple candidate dCDNs for a given user request.

4) If the SDN controller of dCDN decides to provide content, it checks the location of the content object and network traffic status. And it assigns an IP address for the content delivery. The assigned IP address will be used as the content identifier, which is the source address of the data packets. After that, it sends a reply to the controller of uCDN with these data and sets up the path for content delivery from the surrogate in dCDN to the end user.

5) The SDN controller in uCDN informs the end user of the URL of the surrogate in dCDN by sending HTTP Redirection.

6) The end user sends HTTP GET message to the surrogate in dCDN.

6. Advantages of using SDN

The following reasons make SDN a suitable candidate protocol for downstream CDN selection as part of CDNI request routing:

- o Synchronous CDNI operations
- o Integrated with SDN framework and architecture (e.g., OpenFlow)
- o Traffic isolated with desired QoS/QoE, security, etc.
- o More extensible (suitable for i2aex proposal)

- o More centralized, programmable (e.g., using SDN Apps for CDNI)
- o Mobility support

7. Further Considerations

The following further issues should be also discussed on SDN architecture and framework for downstream CDN selection as part of CDNI request routing:

- o ALTO extension (i2aex)
- o Northbound interfaces of SDN controllers
- o Multi-controllers
- o East-west bound interfaces of SDN controllers

8. Security Considerations

TBD

9. Acknowledgements

TBD

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.

10.2. Informative References

- [RFC6707] Niven-Jenkins, B., Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6706, September 2012.
- [I-D.ietf-cdni-requirements] Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-02 (work in progress), December 2011.

- [I-D.ietf-cdni-framework] Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-00 (work in progress), April 2012.
- [I-D.seedorf-cdni-request-routing-alto] Seedorf, J., "CDNI Request Routing with ALTO", draft-seedorf-cdni-request-routing-alto-01 (work in progress), March 2012.
- [RFC3466] Day, M., Cain, B., Tomlinson, G., and P. Rzewski, "A Model for Content Internetworking (CDI)", RFC 3466, February 2003.
- [RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, October 2009.
- [b-OpenFlow] OpenFlow Switch Specification 1.3,
<http://www.opennetworking.org/>.

Authors' Addresses

Myung-Ki Shin
ETRI
161 Gajeong-dong Yuseng-gu
Daejeon, 305-700
Korea

Phone: +82 42 860 4847
Email: mkshin@etri.re.kr

Seungik Lee
ETRI
161 Gajeong-dong Yuseng-gu
Daejeon, 305-700
Korea

Phone: +82 42 860 1483
Email: seungiklee@etri.re.kr

Dukhyun Chang
Seoul National University
Seoul, Korea

Email: dhchang@mmlab.snu.ac.kr

Ted Taekyoung Kwon
Seoul National University
Seoul, Korea

Email: tkkwon98@gmail.com

CDNI
Internet-Draft
Intended status: Informational
Expires: August 29, 2013

J. Seedorf
NEC
J. Peterson
Neustar
S. Previdi
Cisco
R. van Brandenburg
TNO
K. Ma
Azuki Systems, Inc.
February 25, 2013

CDNI Request Routing: Footprint and Capabilities Semantics
draft-spp-cdni-rr-foot-cap-semantics-04

Abstract

This document tries to capture the semantics of the "Footprint and Capabilities Advertisement" part of the CDNI Request Routing interface, i.e. the desired meaning and what "Footprint and Capabilities Advertisement" is expected to offer within CDNI. The discussion in this document has the goal to facilitate the choosing of one or more suitable protocols for "Footprint and Capabilities Advertisement" within CDNI Request Routing.

Status of this Memo

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

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

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

This Internet-Draft will expire on August 29, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction and scope	3
2. CDNI FCI in existing CDNI Documents	4
3. Design Decisions for Footprint and Capabilities	7
3.1. Advertising Limited Coverage	7
3.2. Capabilities and Dynamic Data	8
3.3. Advertisement versus Queries	9
3.4. Avoiding or Handling 'cheating' dCDNs	9
3.5. Focus on Main Use Cases may Simplify Things	10
4. Main Use Case to foster the Clarification of Semantics	11
5. Towards Semantics for Footprint Advertisement	12
6. Towards Semantics for Capabilities Advertisement	14
7. Open Issues and Questions	17
8. Security Considerations	18
9. References	19
9.1. Normative References	19
9.2. Informative References	19
Appendix A. Acknowledgment	20
Authors' Addresses	21

1. Introduction and scope

The CDNI working group is working on a set of protocols to enable the interconnection of multiple CDNs to a CDN federation. This CDN-federation should serve multiple purposes, as discussed in [RFC6770], for instance, to extend the reach of a given CDN to areas in the network which are not covered by this particular CDN.

The goal of this document is to achieve a clear understanding in the CDNI WG about the semantics associated with the CDNI Request Routing Footprint & Capabilities Advertisement Interface (from now on referred to as FCI), in particular the type of information a downstream CDN 'advertises' regarding its footprint and capabilities. To narrow down undecided aspects of these semantics, this document tries to establish a common understanding of what the FCI should offer and accomplish in the context of CDN Interconnection.

It is explicitly outside the scope of this document to decide on specific protocols to use for the FCI.

General assumptions in this document:

- o The CDNs participating in the CDN federation have already performed a boot strap process, i.e., they have connected to each other, either directly or indirectly, and can exchange information amongst each other.
- o The uCDN has received footprint and/or capability advertisements from a set of dCDNs. Footprint advertisement and capability advertisement need not use the same underlying protocol.
- o The upstream CDN (uCDN) receives the initial request-routing request from the endpoint requesting the resource.

This document is organized as follows. First, a recap of the definition of "footprint and capabilities advertisement" in existing documents is given, attempting to distill the apparent common understanding of what the terms 'footprint' and 'capabilities' mean in the context of CDNI. Then, the detailed semantics of the footprint advertisement mechanism and the capability advertisement mechanism will be discussed. Finally, open issues and questions to be discussed in the CDNI WG will be listed.

2. CDNI FCI in existing CDNI Documents

In the following section, the existing descriptions of the CDNI FCI interface in existing documents will be examined. After this, the apparent common understanding of what this interface is intended to offer and accomplish will be carved out.

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

The CDNI Use Cases document [RFC6770] describes capabilities as "... supported range of devices and User Agents or the supported range of delivery technologies". Examples for such capabilities given are specific delivery protocols, technology migration, and meeting a certain QoS.

The CDNI requirements draft [I-D.ietf-cdni-requirements] lists several requirements relevant for the "footprint and capabilities advertisement" part of the CDNI request routing interface. In summary, the following requirements for the CDNI Request Routing Interface and general requirements are relevant for the understanding of the semantics of "footprint and capabilities advertisement":

- o GEN-4 [HIGH], "The CDNI solution shall not require intra-CDN information to be exposed to other CDNs for effective and efficient delivery of the content. Examples of intra-CDN information include surrogate topology, surrogate status, cached content, etc."
- o GEN-9 [MED], "The CDNI solution should support cascaded CDN redirection (CDN1 redirects to CDN2 that redirects to CDN3) to an arbitrary number of levels beyond the first level."
- o GEN-10 [MED], "The CDNI solution should support an arbitrary topology of interconnected CDNs (i.e. the CDN topology cannot be restricted to a tree, a loop-free topology, etc.)."

- o GEN-11 [HIGH], "The CDNI solution shall prevent looping of any CDNI information exchange."
- o REQ-1 [HIGH], allowing the downstream CDN "to communicate to the Upstream CDN coarse information about the Downstream CDN ability and/or willingness to handle requests from the Upstream CDN. For example, this could potentially include a binary signal ("Downstream CDN ready/not-ready to take additional requests from Upstream CDN") to be used in case of excessive load or failure condition in the Downstream CDN."
- o REQ-2 [MED], allowing the downstream CDN to communicate capabilities such as supported content types and delivery protocols, a set of metrics/attributes (e.g. Streaming bandwidth, storage resources, distribution and delivery priority), a set of affinities (e.g. Preferences, indication of distribution/delivery fees), information to facilitate request redirection, as well as footprint information (e.g. "layer-3 coverage").
- o REQ-3 [MED], "In the case of cascaded redirection, the CDNI Request-Routing interface shall allow the Downstream CDN to also include in the information communicated to the Upstream CDN, information on the capabilities, resources and affinities of CDNs to which the Downstream CDN may (in turn) redirect requests received by the Upstream CDN. In that case, the CDNI Request-Routing interface shall prevent looping of such information exchange."
- o REQ-4 [LOW], allowing the downstream CDN to communicate "aggregate information on CDNI administrative limits and policy" (e.g. the maximum number of requests redirected by the Upstream CDN to be served simultaneously by the Downstream CDN or maximum aggregate volume of content (e.g. in Terabytes) to be delivered by the Downstream CDN over a time period).
- o REQ-11 [LOW], "The CDNI Request-Routing protocol may support a mechanism allowing an Upstream CDN to avoid redirecting a request to a Downstream CDN if that is likely to result in the total redirection time exceeding some limit."

Note that in REQ-2 [MED] "Layer-3 coverage" is given as an example of what "footprint" information might convey in the CDNI requirements draft [I-D.ietf-cdni-requirements]. Also, note that REQ-3 [MED] addresses cascaded (transitive) downstream CDNs. In such a case, a downstream CDN needs to include (in its advertisement information that it conveys to an upstream CDN) aggregate footprint and capabilities information for any further transitive downstream CDNs. Such information may be included implicitly (i.e. the cascaded dCDN

is oblivious to the uCDN), or explicitly (i.e. the cascaded dCDN of the fact that there is a cascaded dCDN is visible to the uCDN). In either case, logic is needed to process incoming footprint information from a cascaded dCDN and decide if/how it is to be re-advertised/aggregated when advertising footprint to an upstream CDN.

The CDNI framework draft [I-D.ietf-cdni-framework] describes a "footprint" as in [I-D.previdi-cdni-footprint-advertisement], consisting of two parts: 1) "a class of end user requests (represented, for example, by a set of IP prefixes, or a geographic region) that the dCDN is willing and able to serve directly, without use of another dCDN", and 2) "the connectivity of the dCDN to other CDNs that may be able to serve content to users on behalf of dCDN". The term "connectivity" has recently been replaced with "reachability" in [I-D.previdi-cdni-footprint-advertisement], and as discussed above, "without use of another dCDN" may include aggregated transitive dCDNs. Further examples for capabilities are "the ability to handle certain types of content (e.g. specific streaming formats) or quality of service (QoS)." Content handling capabilities discussed in [I-D.ma-cdni-capabilities] include delivery and acquisition protocols, redirection modes, and metadata related capabilities (e.g., authorization algorithm).

From reading the various draft listed above, it is safe to conclude that neither the term 'footprint' nor 'capabilities' has been clearly and unambiguously defined in these documents and a very broad range of potential capabilities is listed.

3. Design Decisions for Footprint and Capabilities

A large part of the difficulty in discussing the FCI lies in understanding what exactly is meant when trying to define footprint in terms of "coverage" or "reachability." While the operators of CDNs pick strategic locations to situate caches, a cache with a public IPv4 address is reachable by any endpoint on the Internet unless some policy enforcement precludes the use of the cache.

Some CDNs aspire to cover the entire world, which we will henceforth call global CDNs. The footprint advertised by such a CDN in the CDNI environment would, from a coverage or reachability perspective, presumably cover all prefixes. Potentially more interesting for CDNI use cases, however, are CDNs that claim a more limited coverage, but seek to federate with other CDNs in order to create a single CDN fabric which shares resources.

Futhermore, not all capabilities need be footprint restricted. Depending upon the use case, the optimal semantics of "footprints with capability attributes" vs. "capabilities with footprint restrictions" are not clear.

The key to understanding the semantics of footprint and capability advertisement lies in understand why a dCDN would advertise a limited coverage area, and how a uCDN would use such advertisements to decide among one of several dCDNs. The following section will discuss some of the trade-offs and design decisions that need to be decided upon for the CDNI FCI.

3.1. Advertising Limited Coverage

The basic use case that would motivate a dCDN to advertise a limited coverage is that the CDN was built to cover only a particular portion of the Internet. For example, an ISP could purpose-build a CDN to serve only their own customers by situating caches in close topological proximity to high concentrations of their subscribers. The ISP knows the prefixes it has allocated to end users and thus can easily construct a list of prefixes that its caches were positioned to serve.

When such a purpose-built CDN joins a federation, however, and advertises its footprint to a uCDN, the original intended coverage of the CDN might not represent its actual value to the federation of CDNs. Consider an ISP-A and ISP-B that both field their own CDNs, which they federate through CDNI. A given user E, who is customer of ISP-B, might happen to be topologically closest to a cache fielded by ISP-A, if E happens to live in a region where ISP-B has few customers and ISP-A has many. In this case, should ISP-A's CDN "cover" E? If

ISP-B's CDN has a failure condition, should the uCDN understand that ISP-A's caches are potentially available back-ups - and if so, how does ISP-A advertise itself as a "standby" for E? What about the case where CDNs advertising to the same uCDN express overlapping coverage (for example, a federation mixing global and limited CDNs)?

The answers to these questions greatly depend on how much information we want the uCDN to use to make a selection of a dCDN. If a uCDN has three dCDNs to choose from that "cover" the IP address of user E, obviously the uCDN might be interested to know how optimal the coverage is from each of the dCDNs - coverage need not be binary, either provided or not provided. dCDNs could advertise a coverage "score," for example, and provided that they all reported scores fairly on the same scale, uCDNs could use that to make their topological optimality decision. Alternatively, dCDNs could for their footprint advertise the IP addresses of their caches rather than prefix "coverage," and let the uCDN decide for itself (based on its own topological intelligence) which dCDN has better resources to serve a given user.

In summary, the semantics of advertising footprint depend on whether such qualitative metrics for expressing footprint (such as the coverage 'score' mentioned above) should be part of the CDNI FCI, or if it should focus just on 'binary' footprint.

3.2. Capabilities and Dynamic Data

In cases where the apparent footprint of dCDNs overlaps, uCDNs might also want to rely on a host of other factors to evaluate the respective merits of dCDNs. These include facts related to the caches themselves, to the network where the cache is deployed, to the nature of the resource sought and to the administrative policies of the respective networks.

In the absence of network-layer impediments to reaching caches, the choice to limit coverage is necessarily an administrative policy. Much policy must be agreed upon before CDNs can merge into federations, including questions of membership, compensation, volumes and so on. A uCDN certainly will factor these sorts of considerations into its decision to select a dCDN, but there is probably little need for dCDNs to actually advertise them through an interface - they will be settled out of band as a precondition for federating.

Other facts about the dCDN would be expressed through the interface to the uCDN. Some capabilities of a dCDN are static, and some are highly dynamic. Expressing the total storage built into its caches, for example, changes relatively rarely, whereas the amount storage in

use at any given moment is highly volatile. Network bandwidth similarly could be expressed as either total bandwidth available to a cache, or based on the current state of the network. A cache may at one moment lack a particular resource in storage, but have it the next.

The semantics of the capabilities interface will depend on how much of the dCDN state needs to be pushed to the uCDN and qualitatively how often that information should be updated.

3.3. Advertisement versus Queries

In a federated CDN environment, each dCDN shares some of its state with the uCDN, which the uCDN uses to build a unified picture of all of the dCDNs available to it. In architectures that share detailed capability information, the uCDN could basically perform the entire request-routing intelligence down to selecting a particular cache before sending the request to the dCDN (note that within the current CDNI WG scope, such direct selection of specific caches by the uCDN is out of scope). However, when the uCDN must deal with many potential dCDNs, this approach does not scale. Especially as CDNs scale up from dozens or hundreds of caches to thousands or tens of thousands, the volume of updates to footprint and capability may become onerous.

Were the volume of updates to exceed the volumes of requests to the uCDN, it might make more sense for the uCDN to query dCDNs upon receiving requests (as is the case in the recursive redirection mode described in [I-D.ietf-cdni-framework]), instead of receiving advertisements and tracking the state of dCDNs itself. The advantage of querying dCDNs would be that much of the dynamic data that dCDNs cannot share with the uCDN would now be factored into the uCDN's decision. dCDNs need not replicate any state to the uCDN - uCDNs could effectively operate in a stateless mode.

The semantics of both footprint and capability advertisement depend on the service model here: are there cases where a synchronous query/response model would work better for the uCDN decision than a state replication model?

3.4. Avoiding or Handling 'cheating' dCDNs

In a situation where more than one dCDN is willing to serve a given end user request, it might be attractive for a dCDN to 'cheat' in the sense that the dCDN provides inaccurate information to the uCDN in order to convince the uCDN to select it opposed to 'competing' dCDNs. It could therefore be desirable to take away the incentive for dCDNs to cheat (in information advertised) as much as possible. One option

here is to make the information the dCDN advertises somehow verifiable for the uCDN. On the other hand, a cheating dCDN might be avoided or handled by the fact that there will be strong contractual agreements between a uCDN and a dCDN, so that a dCDN would risk severe penalties or legal consequences when caught cheating.

Overall, it seems that information a dCDN advertises should (in the long run) be somehow qualitatively verifiable by the uCDN, though possibly through non-real-time out-of-band audits. It is probably an overly strict requirement to mandate that such verification be possible "immediately", i.e. during the request routing process itself. If the uCDN can detect a cheating dCDN at a later stage, it should suffice for the uCDN to "de-incentivize" cheating because it would negatively affect the long-term business relationship with a particular dCDN.

3.5. Focus on Main Use Cases may Simplify Things

To narrow down semantics for "footprint" and "capabilities" in the CDNI context, it can be useful to initially focus on key use cases to be addressed by the CDNI WG that are to be envisioned the main deployments in the foreseeable future. In this regard, a main realistic use case is the existence of ISP-owned CDNs, which essentially cover a certain operator's network. At the same time, however, the possibility of overlapping footprints should not be excluded, i.e. the scenario where more than one dCDN claims it can serve a given end user request. The ISPs may also choose to federate with a fallback global CDN.

It seems reasonable to assume that in most use cases it is the uCDN that makes the decision on selecting a certain dCDN for request routing based on information the uCDN has received from this particular dCDN. It may be assumed that 'cheating' CDNs will be dealt with via means outside the scope of CDNI and that the information advertised between CDNs is accurate. In addition, excluding the use of qualitative information (e.g., cache proximity, delivery latency, cache load) to predict the quality of delivery would further simplify the use case allowing it to better focus on the basic functionality of the FCI.

4. Main Use Case to foster the Clarification of Semantics

Focusing on a main use case that contains a simple (yet somewhat challenging), realistic, and generally imaginable scenario can help in narrowing down the requirements for the CDNI FCI. To this end, the following (simplified) use case can help in clarifying the semantics of footprint and capabilities for CDNI. In particular, the intention of the use case is to clarify what information needs to be exchanged on the CDNI FCI, what types of information need to be supported in a mandatory fashion (and which should be considered optional), and what types of information need to be updated with respect to a priori established CDNI contracts.

In short, one can imagine the following use case: A given uCDN has several dCDNs. It selects one dCDN for delivery protocol A and footprint 1 and another dCDN for delivery protocol B and footprint 1. The dCDN that serves delivery protocol B has a further, transitive (level-2) dCDN, that serves delivery protocol B in a subset of footprint 1 where the first-level dCDN cannot serve delivery protocol B itself. What happens if capabilities change in the transitive level-2 dCDN that might affect how the uCDN selects a level-1 dCDN (e.g. in case the level-2 dCDN cannot serve delivery protocol B anymore)? How will these changes be conveyed to the uCDN? In particular, what information does the uCDN need to be able to select a new first-level dCDN, either for all of footprint 1 or only for the subset of footprint 1 that the transitive level-2 dCDN served on behalf of the first-level dCDN?

5. Towards Semantics for Footprint Advertisement

Roughly speaking, "footprint" can be defined as "ability and willingness to serve" by a downstream CDN. However, in addition to simple "ability and willingness to serve", the uCDN may wish to have additional information to make a dCDN selection decision, e.g., "how well" a given dCDN can actually serve a given end user request. The "ability and willingness" to serve should be distinguished from the subjective qualitative measurement of "how well" it was served. One can imagine that such additional information is implicitly associated with a given footprint, e.g. due to contractual agreements (e.g. SLAs), business relationships, or perceived dCDN quality in the past. As an alternative, such additional information could also be explicitly tagged along with the footprint.

It is reasonable to assume that a significant part of the actual footprint advertisement will happen in contractual agreements between participating CDNs, i.e. prior to the advertisement phase using the CDNI FCI. The reason for this assumption is that any contractual agreement is likely to contain specifics about the dCDN coverage (i.e. the dCDN footprint) the contractual agreement applies to. In particular, additional information to judge the delivery quality associated with a given dCDN footprint might be defined in contractual agreements (i.e. outside of the CDNI FCI). Further, one can assume that dCDN contractual agreements about the delivery quality associated with a given footprint will probably be based on high-level aggregated statistics (i.e. not too detailed).

Given that a large part of footprint advertisement will actually happen in contractual agreements, the semantics of CDNI footprint advertisement refer to answering the following question: what exactly still needs to be advertised by the CDNI FCI? For instance, updates about temporal failures of part of a footprint can be useful information to convey via the CDNI request routing interface. Such information would provide updates on information previously agreed in contracts between the participating CDNs. In other words, the CDNI FCI is a means for a dCDN to provide changes/updates regarding a footprint it has prior agreed to serve in a contract with a uCDN.

Generally speaking, one can imagine two categories of footprint to be advertised by a dCDN:

- o Footprint could be defined based on (layer-3) "coverage/reachability", where coverage/reachability refers to a set of prefixes, a geographic region, or similar boundary. The dCDN claims that it can cover/reach 'end user requests coming from this footprint'.

- o Footprint could be defined based on "resources", where resources refers to surrogates/caches a dCDN claims to have (e.g., the location of surrogates/resources). The dCDN claims that 'from this footprint' it can serve incoming end user requests.

For each of these footprint types, there are capabilities associated with a given footprint, i.e. the capabilities (e.g., delivery protocol, redirection mode, metadata) supported in the coverage area for a "coverage/reachability" defined footprint, or the capabilities of resources (e.g., delivery protocol, redirection mode, metadata support) for a "resources" defined footprint. It seems clear that "coverage/reachability" types of footprint must be supported within CDNI. It needs to be decided, whether - in addition to "coverage/reachability" types - also "resource" types of footprint should be supported within CDNI.

Different concrete candidates for a "coverage/reachability" footprint are imaginable. In particular, the following concrete types of footprint seem useful in the CDNI context: 'set of IP-prefixes', 'AS number list', and 'geographic region'. Among these, 'set of IP-prefixes' must be able to contain full IP addresses (i.e., a /32 for IPv4 and a /128 for IPv6) and also IP prefixes with an arbitrary prefix length. There must also be support for multiple IP address versions, i.e., IPv4 and IPv6, in the footprint.

Independent of the exact type of a footprint, a footprint might also include the connectivity of a given dCDN to other CDNs that may be able to serve content to users on behalf of that dCDN, to cover cases where there is a transitive CDN interconnection. Further, the downstream CDN must be able to express its footprint to an interested upstream CDN (uCDN) in a comprehensive form, e.g., as a complete data set containing the complete footprint. Making incremental updates, however, to express dynamic changes in state is also desirable.

6. Towards Semantics for Capabilities Advertisement

In general, the dCDN must be able to express its general capabilities to the uCDN. These general capabilities could express if the dCDN supports a given service, for instance, HTTP delivery, RTP/RTSP delivery or RTMP. Furthermore, the dCDN must be able to express particular capabilities for the delivery in a particular footprint area. For example, the dCDN might in general offer RTMP but not in some specific areas, either for maintenance reasons or because the caches covering this particular area cannot deliver this type of service. Hence, in certain cases footprint and capabilities are tied together and cannot be interpreted independently from each other. In such cases, i.e. where capabilities must be expressed on a per footprint basis, it may be beneficial to combine footprint and capabilities advertisement.

A high-level and very rough semantic for capabilities is thus the following: Capabilities are types of information that allow a uCDN to determine if a downstream CDN is able (and willing) to accept (and properly handle) a delegated content request. In addition, Capabilities are characterized by the fact that this information may possibly change over time based on the state of the network or caches.

At a first glance, several broad categories of capabilities seem useful to convey via an advertisement interface (and indeed many such candidate capabilities have been suggested in CDNI drafts, see Section 2). However, advertising capabilities that change highly dynamically (e.g. real-time delivery performance metrics, CDN resource load, or other highly dynamically changing QoS information) should probably not be in scope for the CDNI FCI. First, out of the multitude of possible metrics and capabilities, it is hard to agree on a subset and the precise metrics to be used. Second, and perhaps more importantly, it seems not feasible to specify such highly dynamically changing capabilities and the corresponding metrics within the CDNI charter time-frame.

Useful capabilities refer to information that does not change highly dynamically and which in many cases is absolutely necessary to decide on a particular dCDN for a given end user request. For instance, if an end user request concerns the delivery of a video file with a certain protocol (e.g. RTMP), the uCDN needs to know if a given dCDN has the capability of supporting this delivery protocol.

Similar to footprint advertisement, it is reasonable to assume that a significant part of the actual (resource) capabilities advertisement will happen in contractual agreements between participating CDNs, i.e. prior to the advertisement phase using the CDNI FCI. The role

of capability advertisement is hence rather to enable the dCDN to update a uCDN on changes since a contract has been set up (e.g. in case a new delivery protocol is suddenly being added to the list of supported delivery protocols of a given dCDN, or in case a certain delivery protocol is suddenly not being supported anymore due to failures). Capabilities advertisement thus refers to conveying information to a uCDN about changes/updates of certain capabilities with respect to a given contract.

Given these semantics, it needs to be decided what exact capabilities are useful and how these can be expressed. Since the details of CDNI contracts are not known at the time of this writing (and the CDNI interface should probably be agnostic to these contracts anyway), it remains to be seen what capabilities will be used to define agreements between CDNs in practice. One implication for standardization may be to initially only specify a very limited set of mandatory capabilities for advertisement and have on top of that a flexible data model that allows exchanging additional capabilities when needed. Still, agreement needs to be found on which capabilities (if any) should be mandatory among CDNs. As discussed in Section 3.5, finding the concrete answers to these questions can benefit from focusing on a small number of key use cases that are highly relevant and contain enough complexity to help in understanding what concrete capabilities are needed to facilitate CDN Interconnection.

Under the above considerations, the following capabilities seem useful as 'base' capabilities, i.e. ones that are needed in any case and therefore constitute mandatory capabilities to be supported by the CDNI FCI:

- o Delivery Protocol (e.g., HTTP vs. RTMP)
- o Acquisition Protocol (for acquiring content from a uCDN)
- o Redirection Mode (e.g., DNS Redirection vs. HTTP Redirection as discussed in [I-D.ietf-cdni-framework])
- o Capabilities related to CDNI Logging (e.g., supported logging mechanisms)
- o Capabilities related to CDNI Metadata (e.g., authorization algorithms or support for proprietary vendor metadata)

It is not feasible to enumerate all the possible options for the mandatory capabilities listed above (e.g., all the potential delivery protocols or metadata options) or anticipate all the future needs for additional capabilities. It would be unreasonable to burden the CDNI

FCI specification with defining each supported capability. The CDNI FCI specification should define a generic protocol for conveying any capability information. It would be reasonable to define a registry which initially contains the mandatory capabilities listed above, but may be extended as needs dictate. The CDNI FCI specification SHOULD define the registry (and the rules for adding new entries to the registry) for the different capability types. Each capability type MAY further have a list of valid values. The individual CDNI interface specifications which define a given capability SHOULD define any necessary registries (and the rules for adding new entries to the registry) for the values advertised for a given capability type.

The mandatory capabilities listed above generally relate to information that is configured on a content asset or group of assets basis via CDNI metadata. The capability requirements for acquisition and delivery protocol, redirection mode, and other mandatory metadata capabilities (e.g. authorization algorithms) are defined in [I-D.ietf-cdni-metadata].

Note: CDNI interface support for logging configuration (i.e., control interface vs. metadata interface) has not yet been decided. Once it has been decided, the corresponding CDNI interface specification should define the associated capability requirements.

7. Open Issues and Questions

The following open issues deserve further discussion in the CDNI WG:

- o What is the service model of this interface: Does the uCDN always query the dCDNs? Or does the dCDN always push information to the uCDNs?
- o In addition to "reachability" types of footprint, should also "resource" types of footprint be considered by CDNI (e.g., the location of surrogates/resources a dCDN has)?
- o Does a footprint need to explicitly include the "transitive reachability" of a dCDN to further dCDNs that may be able to serve content to users on behalf of dCDN?
- o What is the assumed business relationship between the uCDN and the dCDN? Is the uCDN always the "authoritative" CDN provider which transitively has itself contracted several downstream CDN providers?
- o How exactly can a given dCDN derive its footprint?
- o Should the footprint/capabilities advertisement interface only signal the delta with respect to a given contract (between a uCDN and a dCDN) or send the whole dCDN state each time?

8. Security Considerations

Security considerations will be discussed in a future version of this document.

9. References

9.1. Normative References

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

9.2. Informative References

- [I-D.ietf-cdni-framework]
Peterson, L. and B. Davie, "Framework for CDN Interconnection", draft-ietf-cdni-framework-03 (work in progress), February 2013.
- [I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Watson, G., Caulfield, M., Leung, K., and K. Ma, "CDN Interconnect Metadata", draft-ietf-cdni-metadata-00 (work in progress), October 2012.
- [I-D.ietf-cdni-requirements]
Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", draft-ietf-cdni-requirements-05 (work in progress), February 2013.
- [I-D.ma-cdni-capabilities]
Ma, K., "Content Distribution Network Interconnection (CDNI) Capabilities Interface", draft-ma-cdni-capabilities-01 (work in progress), February 2013.
- [I-D.previdi-cdni-footprint-advertisement]
Previdi, S., Faucheur, F., Faucheur, F., Medved, J., and L. Faucheur, "CDNI Footprint Advertisement", draft-previdi-cdni-footprint-advertisement-02 (work in progress), September 2012.
- [RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.
- [RFC6770] Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, November 2012.

Appendix A. Acknowledgment

Jan Seedorf is partially supported by the CHANGE project (CHANGE: Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions, <http://www.change-project.eu/>), a research project supported by the European Commission under its 7th Framework Program (contract no. 257422). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the CHANGE project or the European Commission.

Jan Seedorf has been partially supported by the COAST project (Content Aware Searching, retrieval and sTreaming, <http://www.coast-fp7.eu/>), a research project supported by the European Commission under its 7th Framework Program (contract no. 248036). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the COAST project or the European Commission.

Martin Stiernerling provided initial input to this document and valuable comments to the ongoing discussions among the authors of this document.

Authors' Addresses

Jan Seedorf
NEC
Kurfuerstenanlage 36
Heidelberg 69115
Germany

Phone: +49 6221 4342 221
Fax: +49 6221 4342 155
Email: seedorf@neclab.eu

Jon Peterson
NeuStar
1800 Sutter St Suite 570
Concord CA 94520
USA

Phone:
Fax:
Email: jon.peterson@neustar.biz

Stefano Previdi
Cisco Systems
Via Del Serafico 200
Rome 0144
Italy

Phone:
Fax:
Email: sprevidi@cisco.com

Ray van Brandenburg
TNO
Brassersplein 2
Delft 2612CT
The Netherlands

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

Kevin J. Ma
Azuki Systems, Inc.
43 Nagog Park
Acton MA 01720
USA

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

