Abstract

This document illustrates some use cases and associated problems related to delivery of HTTPS secured traffic in the context of interconnected CDNs. A common scenario is when a uCDN is delegating the delivery of encrypted traffic to a dCDN using HTTP or DNS redirection. Some potential solutions are considered.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Copyright (c) 2015 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.
1. Introduction

CDNI deals with contents stored on hierarchical CDNs. An end user browser might request a particular piece of content from a uCDN, and if the uCDN cannot serve the content, it may redirect the request to a dCDN. Because the uCDN trusts the dCDN to deliver the content, this delegation should ideally not trigger a security warning in the user’s browser.

This draft shall review some methods of redirection used in the field today, their impacts on the security of delegation, in the Interconnected CDNs use case, and the implications of redirection in a web environment that has transitioned entirely to HTTPS.

This document currently addresses three HTTP redirection techniques as well as one DNS redirection method employed in CDNs today:

HTTP 3xx redirect

URL rewriting

API mode, or scripted redirection (e.g. via AJAX requests, JSONP, etc.)

DNS-layer redirection

2. HTTP Request Redirection

In this use case, the browser asks for a piece of content on a server, and the HTTP server replies with a "302 temporarily moved" message with a Location header field value containing the chosen content URL.
In the CDNI case, we have to distinguish two sub-cases when dealing with redirections and HTTPS:

- **HTTP -> HTTPS**: In this case, the uCDN domain A redirects end-users’ HTTP requests to the dCDN domain B, and upgrades the security scheme to HTTPS. In this case, the initial delegation will not be secured; the end user will have no cryptographic assurance that the uCDN is delegating to that dCDN, even though the user may subsequently form a secure connection to the dCDN.

- **HTTPS -> HTTPS**: In this case, the uCDN domain A redirects the browsers’ request to dCDN domain B. The browser forms an initial TLS connection to domain A, receives a secure delegation, and then forms a new security association with domain B.

In the first case above, the HTTPS upgrade should always be accepted automatically by the browser, on the condition that the certificate is valid and trusted (e.g., not self-signed).

In the second case, the delegation is secure. Browser implementation determines how transparent the delegation may be to an end user. If a browser generates a warning message notifying a of secure domain change, that would be problematic. However, mainstream browser implementations support seamless secure redirection via HTTP 3xx responses. Ultimately, the secure delegation from a uCDN to a dCDN is entirely tractable in the HTTPS environment provided that application layer redirection such as HTTP 3xx responses is used.

3. URL Rewriting

In some cases, when a web page is rendered on the browser side, embedded URLs in the page are modified in order to point at new web locations. This modification is typically caused by a script embedded in the page. Alternatively, a server-side script of some kind could modify embedded URLs before the page is retrieved by a browser, including certain classes of web intermediaries. URL rewriting can therefore serve as a form of application-layer redirection, effectively.

As far as CDNI is concerned, provided that the page and the script are served over TLS, then this will prevent intermediaries from modifying URLs without the consent of the user or the uCDN, but client-side scripts pushed by the uCDN will still be secure, and then the redirection to any dCDNs via rewriting this case would be secure as well.
3.1. URL Rewriting for Video Contents

A specific case concerns the HTTP Adaptive Streaming (HAS) video contents. In this case, contents are chunked in order to be played with multiple available video qualities. To do that, a manifest file is required to describe the way the contents was prepared/encoded, e.g. how many qualities, chunks size, and their network location. The manifest file shall requested by the player prior to any chunks.

Regarding CDNI, in one case the manifest could be available on the uCDN domain A, while video chunks are available on the dCDN domain B, thus making a transparent redirection. In a second, more complex case, both the uCDN and the dCDN may deliver some of the video chunks.

When using HTTPS, concerns arise in both cases: if the manifest is rewritten insecurely, then

4. API Mode

In the API redirection mode, the web page requested by the browser contains a script that "transparently" (not strictly from the browser’s perspective) requests contents on another Web page.

As far as CDNI is concerned, the initial web page would be located on domain A, whereas contents requested by the script would be located on a secondary domain B.

Apart from "cross-domain" (CORS) issues that can be easily fixed with an "Access-Control-Allow-Origin" header, this use case raises also the HTTPS certificate issues likewise the first two use cases.

5. DNS Request Redirection

The CDNI Redirection draft ([I-D.ietf-cdni-redirection]) specifies that in addition to HTTP, DNS redirection can be used as a means of delegation from a uCDN to a dCDN. In this case, the DNS resolver, when it queries for the hostname associated with the uCDN URL, will be served a DNS response (such as CNAME) that will direct the client to the dCDN. However, in an HTTPS environment, this will result in the client containing a domain other than the one originally specified by the URL inputted by the end user. Consequently, this will almost certainly result in a security failure when the browser attempts to negotiate TLS with the web server it contacts, as the change in domain name will be indistinguishable from a malicious attacker.
This poses serious concerns about the use of DNS as a redirection method when HTTPS is expected for CDNI use cases.

6. Topology hiding

A further security concern associated with redirection is the question of how much information a uCDN imparts to the browser, and consequently to the end user, about its policy decisions in delegating to a dCDN. This may tempt us to develop solutions that do not rely on a uCDN delegating via a redirection mechanism, either at the HTTP or the DNS layer. However, in order to preserve crucial security properties, it is likely unavoidable that a certain amount of information will be divulged to any browser or client of a CDN system. For example, consider that eventually, content will be downloaded from a dCDN cache at a particular IP address, and that consequently, information about a responsible network will always be revealed to an end user. The guidance in [I-D.ietf-cdni-redirection] Section 5 considers the possibility of using "probes" of this form, and the potential topology leakage of any redirection interface.

7. Existing solutions

The problem CDNI faces with secure redirection in a pure HTTPS world is very similar to a set of known problems with secure redirection in the application layer. For example, [I-D.barnes-hard-problem] covers this question based on the experience of implementing comparable secure redirection systems in personal messaging. Any case where the client specifies an original domain name, and expects to forge a security association with a server holding a credential corresponding to that original domain name, will face certain difficulties when redirection is performed outside of the application layer, for example, in the DNS. There are several known ways to address this problem, though none is perfect.

DNSSEC makes it possible to secure DNS redirections. Were CDNI to use DNSSEC for DNS based redirection, the client's resolver would have a strong assurance that the uCDN had in fact designated the dCDN as its delegate. However, DNSSEC adoption remains patchy, and consequently this may not be a practicable solution in the immediate future. While technologies like DANE which build on DNSSEC could help, they remain dependent on DNSSEC adoption.

Attribute certificates are another potential solution, as a dCDN might be able to present a certificate containing a cryptographic assertion that it is an authorized delegate of the uCDN. However, attribute certificate adoption among browser vendors does not make this a viable market adoption today.
It is also possible that a uCDN could give its private credentials to one or more dCDNs. Some virtual hosting providers in the application space have previously required this sort of credential sharing. However, this is usually very undesirable for the assignee of the credentials, and especially if they must share them with multiple parties, it significantly degrades the security of the entire architecture.

Consequently, DNS redirection introduces a serious of difficulties that are more tractable at the application layer. Barring any progress in the market adoption of other solutions described above, HTTPS redirection should be preferred to DNS redirection.

8. Acknowledgments

The authors would like to thank Jon Peterson and Jan Seedorf for their help in putting this draft together.

9. References

9.1. Normative References


9.2. Informative References


Frederic Fieau (editor)
Orange
38-40 rue du General Leclerc
Issy les Moulineaux 92130
FR

Email: frederic.fieau@orange.com

Iuniana Oprescu
Orange
38-40 rue du General Leclerc
Issy les Moulineaux 92130
FR

Email: iuniana.oprescu@orange.com
HTTPS and delegation of encrypted traffic between interconnected CDNs
draft-fieau-https-delivery-delegation-01

Abstract

This document provides a basic non exhaustive background and
discusses potential approaches to the issue of correct delegation of
the encrypted TLS-based traffic requests between CDNs in inter CDN
networks and during interactions between client User Agents (UA), and
Content Delivery Networks (CDNs).

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 21, 2016.

Copyright Notice

Copyright (c) 2015 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
1. Introduction

In the interconnected CDNs context where CDNs are organized into a
hierarchy, an upstream CDN (uCDN) that is not able to deliver the
requested content, may need to delegate the delivery to a downstream
CDN (dCDN). When end-users’ connections are transported over TLS,
this delivery delegation involves security requirements such as:

- Ensuring the trusted delegation of the delivery of contents hosted
  by a uCDN to the downstream CDN (dCDN), i.e., that uCDN has
  wilfully designated the uCDN to deliver the content, thus ensuring
  the delegation is legitimate.

- Ensuring a seamless security scheme when redirecting users’
  requests from a uCDN to a dCDN, i.e., the certificates received by
  the browser are valid and are tied to the domain for which the
delegation occurs. For instance, an User Agent (UA) might request
a piece of content from an upstream CDN (uCDN), and if the uCDN
cannot serve the content, it may redirect the request to a
downstream CDN (dCDN). Because the uCDN shall trust in that case
the dCDN to deliver the content, this delegation should ideally
not trigger a security warning in the user’s browser.
Guaranteeing the confidentiality of CDNs configuration and topology information, i.e. keep this information hidden from each CDNs, and from the end-user browser. As an example, the end-user shall not be aware of the CDNI topology.

A brief survey indicates that there is a multitude of ad hoc approaches for handling TLS-based traffic in CDN environment. However, many of the currently adopted practices seem to have problems of various nature, inhibiting and compromising security, scalability, and ease of operation and maintenance (see e.g. [HTTPS-CDN] and [SSL-Challenges]).

This document is intended as a starting point for discussion. It shall review redirection methods introduced in [RFC3568] used in the Interconnected CDNs use case, their impacts on the security of delegation, and the implications of redirection in a secured web environment. It eventually identifies some workarounds, or solutions to the raised issues.

HTTP-based redirection methods: 3xx, URL rewriting, API based

DNS-layer redirection

2. Problem Scope

In a secured CDN interconnection model where aCDN and a dCDN have two distinct domains, the User Agent (UA) tries first to resolve the uCDN domain when it contacts the uCDN for a piece of content. At this step, two types of redirection methods can be considered, both delegating the content delivery to the dCDN:

- using DNS routing, the UA is redirected to the dCDN, e.g., with a CNAME sent by the uCDN authoritative DNS server.

- the UA is redirected by the uCDN to the dCDN using a HTTP-based mechanism: Once the uCDN domain is resolved, the UA negotiates a secured connection to the uCDN for that content, and receives the uCDN certificate. Then the uCDN subsequently uses an HTTP redirection method to redirect the UA to a dCDN surrogate content server.

Then, the UA resolves dCDN domain name, and then negotiates a new secured connection with the dCDN, retrieving the dCDN certificate. If the certificate is valid, then the UA will be able to connect to the dCDN surrogate, and the dCDN will deliver the requested piece of content to the UA.

Both cases raises security issues in a CDNI environment:
o How to ensuring that the delegation is legitimate? In DNS, this cannot be ensured without additional cryptographic measures (see DNS redirection related section) contrary to HTTPS, refer to HTTP-based redirection methods section for more details.

o How to ensure that certificates received by the browser are valid and are tied to the domain for which the delegation occurs? This is may not be the case if uCDN and dCDN don’t share the same domain name, see related [redirection methods] section and [TLS API] section for more details.

o How to guarantee the confidentiality of CDNs configuration and topology information? The UA may be able to discover the CDNI topology which is not secure. Please refer to [Topology hiding] section for more details.

3. HTTP-based redirection methods

This section deals with HTTP-based redirection methods for secured TLS connections in CDNI.

3.1. 3xx directives

When dealing with redirection over HTTP/S, two sub-cases should be considered:

o HTTP -> HTTPS: In this case, the CSP / uCDN from domain A redirects end-users’ HTTP requests to the dCDN from domain B, and upgrades the security scheme to HTTPS.

o HTTPS -> HTTPS: In this case, the CSP / uCDN domain A redirects the browsers’ request to dCDN domain B.

Regardless of DNS resolution aspects, in the first sub-case, the initial delegation will not be secured, or trusted: the end user will have no cryptographic assurance that the uCDN is delegating to that dCDN, even though the user may subsequently form a secure connection to the dCDN. The HTTPS upgrade should always be accepted automatically by the browser, on the condition that the certificate is valid and trusted (e.g., not self-signed).

In the second sub-case, TLS set-up happens before the first HTTP request is sent, therefore, the subsequent traffic including request URI and query parameters will be encrypted. First, the TLS session is established between the UA and the origin uCDN, authenticating the uCDN. Then, the uCDN uses HTTP mechanisms for redirection, using 3xx messages like 302 Found, embedding the new target URI aiming at the CDN surrogate in the Location header. UA then sets up a separate TLS
session with the dCDN surrogate, validating the dCDN certificate. Trust relationship is implied since the redirection message has been received over the authenticated TLS tunnel with the origin uCDN.

The delegation is trusted and legitimate even if two independent TLS sessions will have to be set up in sequence by the UA. The trust delegation endorsement, tying two sessions together is realized by the fact that the target URI has been communicated in the redirection message by the authenticated origin uCDN over the encrypted tunnel.

However the issue here is when both uCDN and dCDN domains are distinct. Indeed, the browser has received two certificates linked to two distinct domains. Therefore, the browser can’t seamlessly assure the end-user that delegation has occurred on the same domain. While browser implementation determines how transparent the delegation may be to an end user, the browser may in most of the cases generate a warning message notifying the user of a secure domain change, which is not a seamless delegation.

However, mainstream browser implementations support seamless secure redirection via HTTP 3xx responses. Ultimately, the secure delegation from a uCDN to a dCDN is entirely tractable in the HTTPS environment provided that application layer redirection such as HTTP 3xx responses is used.

3.2. URL Rewriting

URL rewriting is a method to compute in a web page destination URLs to point at new web location while this page is rendered on the browser. This modification could be typically caused by a script embedded in the page. Alternatively, a server-side code could modify embedded URLs before the page is retrieved by a browser, including certain classes of web intermediaries. URL rewriting can therefore serves as a form of application-layer redirection.

Regarding CDNI, a page served over TLS by an uCDN will prevent intermediaries from modifying URLs without the consent of the user or the uCDN. Client-side scripts pushed by the uCDN will still be secure, and then the redirection to any dCDNs via rewriting would be secure as well.

3.2.1. URL Rewriting for Video Contents

In the case of HTTP Adaptive Streaming (HAS) techniques where contents are chunked in order to be played with multiple video qualities, a manifest file will describe the way the content was prepared/encoded, e.g. how many qualities, chunks size, and their
network location. This manifest is requested by the player prior to any chunks.

Regarding CDNI, if the manifest is available on the uCDN domain A, while video chunks are available on the dCDN domain B, the player requesting for chunks will be redirected by the uCDN to the dCDN using 3xx redirection methods (see the previous section), thus ensuring a trusted delegation but not seamless for the end-user.

In another more complex case, both the uCDN and the dCDN may deliver some of the video chunks.

3.3. API Mode, or scripted redirection (e.g. via AJAX requests, JSONP, etc.)

In the API mode, a web page requested by the browser contains a script that "transparently" - from the user’s perspective - requests contents on another web page.

As far as CDNI is concerned, the initial web page and scripts would be located on domain A, whereas contents requested by the script would be located on a secondary domain B.

Apart from "cross-domain" (CORS) issues that can be fixed with an "Access-Control-Allow-Origin" header, this use case raises also the HTTPS certificate issues likewise in other HTTP-based redirection cases.

4. DNS redirection

In the case of DNS-based redirection, prior to any HTTP delivery requests, the UA tries to resolve the uCDN domain, then uCDN DNS server answers with the dCDN domain name (e.g., using a CNAME) instead of the uCDN domain, thus realizing the DNS redirection to the dCDN.

In that case, the actual redirect happens before the establishment of the TLS tunnel. The issue here is that the user UA expects the uCDN’s certificate, but instead obtains the dCDN surrogate’s certificate during the TLS handshake. Mismatch between the expected origin uCDN URI and the received dCDN URI designated in the obtained certificate causes certificate validation warnings at the UA. Eventually a client UA displays a warning to the end user requiring additional steps, which compromises the seamless delegation.

The CDNI Redirection draft ([I-D.ietf-cdni-redirection]) specifies that in addition to HTTP, DNS redirection can be used as a means of delegation from a uCDN to a dCDN. In this case, the DNS resolver,
when it queries for the hostname associated with the uCDN URL, will be served a DNS response (such as CNAME) that will direct the client to the dCDN. However, in an HTTPS environment, this will result in the client containing a domain other than the one originally specified by the URL input by the end user. Consequently, this will result in a security failure when the browser attempts to negotiate TLS with the web server it contacts, as the change in domain name will be indistinguishable from a malicious attacker.

Another security issue wider than CDNI scope, related to the "HARD problem" draft [I-D.barnes-hard-problem] ("High Assurance Re-Direction") is where a malicious DNS resolver could return DNS responses (IP addresses/CNAMEs) that steer the User Agent to a malicious server. DNS response hijacking could be used to mount a DoS attack against the CDN/Content Provider as the User Agent won’t be able to receive the content that it wants because it is being told to retrieve it from a server that it can’t establish a TLS session to.

DNSSEC would prevent that because responses would need to be signed and a malicious DNS resolver would therefore not be able to return malicious responses as it would not be able to generate properly signed DNS response.

4.1. a DNSSEC-based approach

DNSSEC makes it possible to secure DNS redirections. Were CDNI to use DNSSEC for DNS based redirection, the client’s resolver would have a strong assurance that the uCDN had in fact designated the dCDN as its delegate. However, DNSSEC adoption remains patchy, and consequently this may not be a practicable solution in the immediate future. While technologies like DANE which build on DNSSEC could help, they remain dependent on DNSSEC adoption.

Such an approach is proposed in [HTTPS-CDN] for the DANE-based [RFC6698] front-end authentication using the DNS-based redirection. We reproduce a brief overview of the proposal from [HTTPS-CDN] here as an example of a possible approach.

Using DANE, an origin CSP binds target CDN’s certificate with the CSP’s own certificate and domain name (see section VI. B. of [HTTPS-CDN]) by adding both certificates to the CSP’s TLSA record [RFC6698]. After initiating a TLS connection to target CDN surrogate, and having received CDN’s certificate, the UA further issues a DNS query to request origin CSP’s TLSA records. UA then is capable of validating both URIs and Certificates with those received in the TLSA record, explicitly ensuring the delegation of trust.
5. Enforcing trust delegation: CDNI URI Signing

CDNI URI Signing [I-D.leung-cdni-uri-signing] draft specifies a detailed mechanism to ensure validation of parameters communicated in the redirection URI.
Considering CDNI and HTTPS delegation, this URI signing mechanism could be used as means to enforce trust delegation.

While this later draft focuses on the validation by the target CDN of the authenticity of the parameters communicated in the redirect URI generated by the origin CSP, CDNI URI Signing could be extended or used to include the certificate information or hashes either in the provided URI Signing Package Attribute, or in an additional Package Attribute (e.g. Redirect Authentication Attribute), reusing much of the mechanisms detailed in the draft.

6. Topology hiding

A further security concern associated with redirection is the question of how much information a uCDN imparts to the browser, and consequently to the end user, about its policy decisions in delegating to a dCDN. However, in order to preserve crucial security properties, it is likely unavoidable that a certain amount of information will be divulged to any browser or client of a CDN system. For example, consider that eventually, content will be downloaded from a dCDN cache at a particular IP address, and that consequently, information about a responsible network will always be revealed to an end user.

The guidance in [I-D.ietf-cdni-redirection] Section 5 considers the possibility of using "probes" of this form, and the potential topology leakage of any redirection interface.

7. TLS API for third-party

Delegating delivery of HTTPS traffic to a third party without triggering any warnings on the client’s browser can be achieved if the dCDN surrogate is able to present the security credentials for the domain name in the user’s initial request. One of the common practices so far has been for the content provider to directly delegate the storage of the private keys to the CDN that is delivering the content on its behalf. This solution could persist in the CDNI context, but in a scenario where delivery is done through multiple cascaded dCDN, it becomes unlikely that the content provider would be willing to share its private keys with all the parties involved.

Another solution is the introduction of a Private Key Server in the TLS handshake. Such solutions are commercially deployed and several examples are made publicly available. Such a setup allows the private keys to remain under the authority of the content owner (or the uCDN) while the actual content can be served from a dCDN surrogate that is closer to the end user. Indeed, the architecture
introduces a split in the setup of the secure tunnel between the client’s browser and the surrogate delivering the content. Since the dCDN does not possess the private keys for the requested domain name, during the setup of the TLS tunnel between the client and the dCDN surrogate, the latter forwards the challenge to the Private Key Server which is under the control of the content owner (or the uCDN). With the response it receives, the surrogate is able to successfully establish a secure connection to the end user and serve the requested content without triggering any warnings in the client’s browser.

- The user agent connects to the CDN surrogate. The client sends a secret encrypted with the site’s public key for the surrogate to decrypt.

- The surrogate contacts the key server, authenticating itself with a certificate. The surrogate sends the encrypted secret to the key server to decrypt it. The key server returns the decrypted secret to the dCDN surrogate over a secure tunnel.

- Both client and surrogate use the shared secret to establish a secure connection. The user agent issues its request for content over HTTPS.

- The surrogate then processes the original request.

Below is an example of the handshake establishment:
Figure 2: Overview of Keyless SSL DH handshake

8. IANA Considerations

This document has no IANA considerations.

9. Security Considerations

The entire document is about security.

10. Acknowledgments

The authors would like to thank Jon Peterson, Jan Seedorf, and Ben Nivens-Jenkins for their help in putting this draft together.

11. References

11.1. Normative References


11.2. Informative References

[HTTPS-CDN]
J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS Meets CDN: A Case of Authentication in Delegated Service", in 2014 IEEE Symposium on Security and Privacy (SP), 2014, pp. 67-82..

[I-D.barnes-hard-problem]

[I-D.ietf-cdni-redirection]

[I-D.leung-cdni-uri-signing]

[SSL-Challenges]

Authors' Addresses

Frederic Fieau
Orange
38-40 rue du General Leclerc
Issy les Moulineaux  92130
FR

Email: frederic.fieau@orange.com
Iuniana Oprescu
Orange
38-40 rue du General Leclerc
Issy les Moulineaux  92130
FR
Email: iuniana.oprescu@orange.com

Sergey Slovetskiy
Email: sergey.slovetskiy@fastmail.com
Abstract

This document describes the part of the CDN Interconnection Control Interface that allows a CDN to trigger activity in an interconnected CDN that is configured to deliver content on its behalf. The upstream CDN can use this mechanism to request that the downstream CDN pre-positions metadata or content, or that it invalidates or purges metadata or content. The upstream CDN can monitor the status of activity that it has triggered in the downstream CDN.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 18, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ........................................ 3
   1.1. Terminology .................................. 4
2. Model for CDNI Triggers ............................ 4
   2.1. Timing of Triggered Activity ................. 6
   2.2. Scope of Triggered Activity .................. 6
   2.3. Trigger Results ................................ 6
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 .... 10
      4.2.2. Polling Trigger Status Resources ........... 11
   4.3. Cancelling Triggers ............................ 11
   4.4. Deleting Triggers ................................ 12
   4.5. Expiry of Trigger Status Resources ............ 12
   4.6. Loop Detection and Prevention ................. 13
   4.7. Error Handling ............................... 13
   4.8. Content URLs .................................. 14
5. CI/T Object Properties and Encoding ................ 14
   5.1. CI/T Objects .................................. 15
      5.1.1. CI/T Commands ............................ 15
      5.1.2. Trigger Status Resource ................... 16
      5.1.3. Trigger Collection ........................ 17
   5.2. Properties of CI/T Objects .................... 18
      5.2.1. Trigger Specification ...................... 19
      5.2.2. Trigger Type ................................ 20
      5.2.3. Trigger Status ............................. 21
      5.2.4. PatternMatch ................................ 21
      5.2.5. Absolute Time ................................ 22
      5.2.6. Error Description ......................... 22
      5.2.7. Error Code ................................ 23
   5.3. Formalization of the JSON Data ................ 23
6. Examples ........................................... 25
   6.1. Creating Triggers ................................ 25
      6.1.1. Preposition ............................... 25
      6.1.2. Invalidate ................................ 27
1. Introduction

[RFC6707] introduces the problem scope for CDN Interconnection (CDNI) and lists the four categories of interfaces that may be used to compose a CDNI solution (Control, Metadata, Request Routing, Logging).

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

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

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

This document reuses the terminology defined in [RFC6707].

2. Model for CDNI Triggers

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

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

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

The CI/T interface is a web service offered by the dCDN. It allows
CI/T commands to be issued, and triggered activity to be tracked.
When the dCDN accepts a CI/T Command it creates a resource describing
status of the triggered activity, a Trigger Status Resource. The
uCDN can poll Trigger Status Resources to monitor progress.

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

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

In addition to the collection of all Trigger Status Resources for the
uCDN, the dCDN can maintain filtered views of that collection. These
filtered views are defined in Section 3 and include collections of
Trigger Status Resources corresponding to active and completed CI/T
Trigger Commands. These collections provide a mechanism for polling the status of multiple jobs.

Figure 1 is an example showing the basic message flow used by the uCDN to trigger activity in the dCDN, and for the uCDN to discover the status of that activity. Only successful triggering is shown. Examples of the messages are given in Section 6.

```
<table>
<thead>
<tr>
<th>uCDN</th>
<th>dCDN</th>
</tr>
</thead>
</table>
| (1) POST http://dcdn.example.com/triggers/uCDN | []--------------------------------------------------> []--+
| [ ]-----------------------------------------------| [ ] | (2) |
| (3) HTTP 201 Response | [ ]<--+
| [ ]-----------------------------------------------| [ ] |
| Loc: http://dcdn.example.com/triggers/uCDN/123 | |
| | . | |
| | . | |
| | . | |
| (4) GET http://dcdn.example.com/triggers/uCDN/123 | []--------------------------------------------------> []
| [ ]-----------------------------------------------| [ ] |
| (5) HTTP 200 Trigger Status Resource | [ ]
| [ ]-----------------------------------------------| [ ] |
```

Figure 1: Basic CDNI Message Flow for Triggers

The steps in Figure 1 are:

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

2. The dCDN authenticates the request, validates the CI/T Command and, if it accepts the request, creates a new Trigger Status Resource.

3. The dCDN responds to the uCDN with an HTTP 201 response status, and the location of the Trigger Status Resource.

4. The uCDN can poll, possibly repeatedly, the Trigger Status Resource in the dCDN.
5. The dCDN responds with the Trigger Status Resource, describing progress or results of the CI/T Trigger Command.

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

2.1. Timing of Triggered Activity

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

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

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

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

2.2. Scope of Triggered Activity

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

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

The dCDN MUST reject CI/T Commands from a uCDN that act on another uCDN’s data. Security considerations are discussed further in section Section 8.

2.3. Trigger Results

Possible states for a Trigger Status Resource are defined in section Section 5.2.3.
The CI/T Trigger Command MUST NOT be reported as 'complete' until all actions have been completed successfully. The reasons for failure, and URLs or Patterns affected, SHOULD be enumerated in the Trigger Status Resource. For more detail, see section Section 4.7.

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

3. Collections of Trigger Status Resources

As described in Section 2, Trigger Status Resources exist in the dCDN to report the status of activity triggered by each uCDN.

A collection of Trigger Status Resources is a resource that contains a reference to each Trigger Status Resource in that collection.

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

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

In order to allow the uCDN to check the status of multiple jobs in a single request, the dCDN SHOULD also maintain collections representing filtered views of the collection of all Trigger Status Resources. These filtered collections are optional-to-implement but, if implemented, the dCDN MUST include links to them in the collection of all Trigger Status Resources. The filtered collections are:
Pending - Trigger Status Resources for CI/T Trigger Commands that have been accepted, but not yet acted upon.

Active - Trigger Status Resources for CI/T Trigger Commands that are currently being processed in the dCDN.

Complete - Trigger Status Resources representing activity that completed successfully, and ‘processed’ CI/T Trigger Commands for which no further status updates will be made by the dCDN.

Failed - Trigger Status Resources representing CI/T Commands that failed or were cancelled by the uCDN.

4. CDNI Trigger Interface

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

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

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

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

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

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

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

4.1. Creating Triggers

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

The dCDN validates the CI/T Command, if it is malformed or the uCDN does not have sufficient access rights it MUST either respond with an appropriate 4xx HTTP error code and a Trigger Status Resource MUST NOT be created on the dCDN, or create a ‘failed’ Trigger Status Resource containing an appropriate error description.

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

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

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

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

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

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

4.2. Checking Status

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

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

The dCDN SHOULD use the cache control headers for responses to GETs for Trigger Status Resources and Collections to indicate the frequency at which it recommends the uCDN should poll for change.

4.2.1. Polling Trigger Status Resource collections

The uCDN can fetch the collection of its Trigger Status Resources, or filtered views of that collection.

This makes it possible to poll status of all CI/T Trigger Commands in a single request. If the dCDN moves a Trigger Status Resource from the Active to the Completed collection, the uCDN can fetch the result of that activity.

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

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

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

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

4.3. Cancelling Triggers

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

Cancellation of a CI/T Trigger Command is optional-to-implement in the dCDN.

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

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

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

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

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

4.4. Deleting Triggers

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

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

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

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

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

4.5. Expiry of Trigger Status Resources

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

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

Murray & Niven-Jenkins Expires April 18, 2016
To ensure it is able to get the status of its Trigger Status Resources for completed and failed CI/T Commands, it is RECOMMENDED that the uCDN polling interval is less than the time after which records for completed activity will be deleted.

4.6. Loop Detection and Prevention

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

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

The CDN Provider Id consists of the two characters "AS" followed by the CDN Provider’s Autonomous System number, then a colon (‘:’) and an additional qualifier that is used to guarantee uniqueness in case a particular AS has multiple independent CDNs deployed. For example "AS64496:0".

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

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

4.7. Error Handling

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

If any part of the CI/T Trigger Command fails, the trigger SHOULD be reported as "failed" once its activity is complete or if no further errors will be reported. The "errors" property in the Trigger Status Resource will be used to enumerate which actions failed and the reasons for failure, and can be present while the Trigger Status
Resource is still "pending" or "active", if the CI/T Trigger Command is still running for some URLs or Patterns in the Trigger Specification.

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

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

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

4.8. Content URLs

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

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

5. CI/T Object Properties and Encoding

CI/T Commands, Trigger Status Resources and Trigger Collections and their properties are encoded using JSON, as defined in Section 5.1.1, Section 5.2.1, and Section 5.2.2. They MUST use the MIME Media Type 'application/cdni', with parameter 'ptype' values as defined below and in Section 7.1.
Names in JSON are case sensitive. The names and literal values specified in the present document MUST always use lower-case.

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

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

5.1. CI/T Objects

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

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

5.1.1. CI/T Commands

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

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

Name: trigger

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

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

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

Name: cancel

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

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

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

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

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

Mandatory: Yes.

5.1.2. Trigger Status Resource

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

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

Name: trigger

Description: The Trigger Specification posted in the body of the CI/T Command. Note that this need not be a byte-for-byte copy. For example, in the JSON representation the dCDN may re-serialise the information differently.

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

Mandatory: Yes

Name: ctime

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

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: Yes

Name: mtime

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

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: Yes
Name: etime

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

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: No

Name: status

Description: Current status of the triggered activity.

Value: Trigger Status, as defined in Section 5.2.3.

Mandatory: Yes

Name: errors

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

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

Mandatory: No

5.1.3. Trigger Collection

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

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

Name: triggers

Description: Links to Trigger Status Resources in the collection.

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

Mandatory: Yes
Name: staleresourcetime

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

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

Mandatory: Yes, in the collection of all Trigger Status Resources if the dCDN deletes stale entries. If the property is present in the filtered collections, it MUST have the same value as in the collection of all Trigger Status Resources.

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

Description: Link to a Trigger Collection.

Value: A URL represented as a JSON string.

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

Name: cdn-id

Description: The CDN Provider Identifier of the dCDN.

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

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

5.2. Properties of CI/T Objects

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

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

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

Name: type
   Description: This property defines the type of the CI/T Trigger Command.
   Value: Trigger Type, as defined in Section 5.2.2.
   Mandatory: Yes

Name: metadata.urls
   Description: The uCDN URLs of the metadata the CI/T Trigger Command applies to.
   Value: A JSON array of URLs represented as JSON strings.
   Mandatory: No, but at least one of ‘metadata.*’ or ‘content.*’ MUST be present and non-empty.

Name: content.urls
   Description: URLs of content the CI/T Trigger Command applies to, see Section 4.8.
   Value: A JSON array of URLs represented as JSON strings.
   Mandatory: No, but at least one of ‘metadata.*’ or ‘content.*’ MUST be present and non-empty.

Name: content.ccid
   Description: The Content Collection Identifier of content the trigger applies to. The ‘ccid’ is a grouping of content, as defined by [I-D.ietf-cdni-metadata].
   Value: A JSON array of strings, where each string is a Content Collection Identifier.
Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty.

Name: metadata.patterns

Description: The metadata the trigger applies to.

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

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty, and metadata.patterns MUST NOT be present if the TriggerType is Preposition.

Name: content.patterns

Description: The content data the trigger applies to.

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

Mandatory: No, but at least one of 'metadata.*' or 'content.*' MUST be present and non-empty, and content.patterns MUST NOT be present if the TriggerType is Preposition.

5.2.2. Trigger Type

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

<table>
<thead>
<tr>
<th>JSON String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preposition</td>
<td>A request for the dCDN to acquire metadata or content.</td>
</tr>
<tr>
<td>invalidate</td>
<td>A request for the dCDN to invalidate metadata or content. After servicing this request the dCDN will not use the specified data without first re-validating it using, for example, an &quot;If-None-Match&quot; HTTP request. The dCDN need not erase the associated data.</td>
</tr>
<tr>
<td>purge</td>
<td>A request for the dCDN to erase metadata or content. After servicing the request, the specified data MUST NOT be held on the dCDN (the dCDN should re-acquire the metadata or content from uCDN if it needs it).</td>
</tr>
</tbody>
</table>
5.2.3. Trigger Status

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

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

5.2.4. PatternMatch

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

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

Name: pattern

Description: A pattern for string matching.

Value: A JSON string representing the pattern. The pattern may contain the wildcards * and ?, where * matches any sequence of characters (including the empty string) and ? matches exactly one character. The three literals "\", "*" and "?" MUST be escaped as "\\", "\*" and "\?".

Mandatory: Yes.

Name: case-sensitive

Description: Flag indicating whether or not case-sensitive matching should be used.
Value: One of the JSON values ‘true’ or ‘false’.

Mandatory: No, default is case-insensitive match.

Name: match-query-string

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

Value: One of the JSON values ‘true’ or ‘false’.

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

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

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

5.2.5. Absolute Time

A JSON number, seconds since the UNIX epoch.

5.2.6. Error Description

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

Name: error

Value: Error Code, as defined in Section 5.2.7.

Mandatory: Yes.

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

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

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

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

Name: description

Description: A human-readable description of the error.

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

Mandatory: No.

5.2.7. Error Code

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emeta</td>
<td>The dCDN was unable to acquire metadata required to fulfill the request.</td>
</tr>
<tr>
<td>econtent</td>
<td>The dCDN was unable to acquire content (CT/T preposition commands only).</td>
</tr>
<tr>
<td>eperm</td>
<td>The uCDN does not have permission to issue the CI/T Command (for example, the data is owned by another CDN).</td>
</tr>
<tr>
<td>ereject</td>
<td>The dCDN is not willing to fulfill the CI/T Command (for example, a preposition request for content at a time when the dCDN would not accept Request Routing requests from the uCDN).</td>
</tr>
<tr>
<td>ecdn</td>
<td>An internal error in the dCDN or one of its downstream CDNs.</td>
</tr>
<tr>
<td>ecancelled</td>
<td>The uCDN cancelled the request.</td>
</tr>
</tbody>
</table>

5.3. Formalization of the JSON Data

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

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

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

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

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

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

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

Pattern-Match = { ; 5.2.4
  pattern: tstr
? case-sensitive: bool
? match-query-string: bool
}
Absolute-Time = number ; seconds since UNIX epoch, 5.2.5

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

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

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

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

URI = tstr

6. Examples

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

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

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

6.1. Creating Triggers

Examples of the uCDN triggering activity in the dCDN:

6.1.1. Preposition

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

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

REQUEST:
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger-command
Content-Length: 347

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

    "metadata.urls": [ "http://metadata.example.com/a/b/c" ],
    "content.urls": [ "http://www.example.com/a/b/c/1",
                     "http://www.example.com/a/b/c/2",
                     "http://www.example.com/a/b/c/3",
                     "http://www.example.com/a/b/c/4"
                     ]
  }
}

RESPONSE:

HTTP/1.1 201 Created
Date: Sun, 31 Aug 2014 09:53:18 GMT
Content-Length: 472
Content-Type: application/cdni; ptype=ci-trigger-status
Location: http://dcdn.example.com/triggers/0
Server: example-server/0.1

{
  "ctime": 1409478798,
  "etime": 1409478806,
  "mtime": 1409478798,
  "status": "pending",
  "trigger": {
    "content.urls": [ "http://www.example.com/a/b/c/1",
                     "http://www.example.com/a/b/c/2",
                     "http://www.example.com/a/b/c/3",
                     "http://www.example.com/a/b/c/4"
                     ],
    "metadata.urls": [ "http://metadata.example.com/a/b/c"
                     ],
    "type": "preposition"
  }
}
6.1.2. Invalidate

An example of a CI/T invalidate command, another POST to the collection of all Trigger Status Resources. This instructs the dCDN to re-validate the content at "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/" using case-insensitive matching, and "http://www.example.com/a/b/" respectively, using case-sensitive matching.

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger-command
Content-Length: 384

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

RESPONSE:

```
HTTP/1.1 201 Created
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Length: 551
Content-Type: application/cdni; ptype=ci-trigger-status
Location: http://dcdn.example.com/triggers/1
Server: example-server/0.1
```
{  
  "ctime": 1409478799,  
  "etime": 1409478807,  
  "mtime": 1409478799,  
  "status": "pending",  
  "trigger": {  
    "content.patterns": [  
      {  
        "case-sensitive": true,  
        "pattern": "http://www.example.com/a/b/*"  
      }  
    ],  
    "content.urls": [  
      "http://www.example.com/a/index.html"  
    ],  
    "metadata.patterns": [  
      {  
        "pattern": "http://metadata.example.com/a/b/*"  
      }  
    ],  
    "type": "invalidate"  
  }  
}

6.2. Examining Trigger Status

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

6.2.1. Collection of All Triggers

The uCDN can fetch the collection of all Trigger Status Resources it has created that have not yet been deleted or removed as expired. After creation of the "preposition" and "invalidate" triggers shown above, this collection might look as follows:
REQUEST:

GET /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*

RESPONSE:

HTTP/1.1 200 OK
Content-Length: 347
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "-6516741166528256414"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

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

6.2.2. Filtered Collections of Trigger Status Resources

The filtered collections are also available to the uCDN. Before the dCDN starts processing the two CI/T Trigger Commands shown above, both will appear in the collection of Pending Triggers, for example:
REQUEST:

GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*

RESPONSE:

HTTP/1.1 200 OK
Content-Length: 153
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "5012053611544832286"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

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

At this point, if no other Trigger Status Resources had been created, the other filtered views would be empty. For example:
REQUEST:

GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*

RESPONSE:

HTTP/1.1 200 OK
Content-Length: 56
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "29bb340333785000363"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

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

6.2.3. Individual Trigger Status Resources

The Trigger Status Resources can also be examined for detail about individual CI/T Trigger Commands. For example, for the CI/T "preposition" and "invalidate" commands from previous examples:
REQUEST:
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*

RESPONSE:
HTTP/1.1 200 OK
Content-Length: 472
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
ETag: "-4765587034697674779"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni; ptype=ci-trigger-status

{
   "ctime": 1409478798,
   "etime": 1409478806,
   "mtime": 1409478798,
   "status": "pending",
   "trigger": {
      "content.urls": [
         "http://www.example.com/a/b/c/1",
         "http://www.example.com/a/b/c/2",
         "http://www.example.com/a/b/c/3",
         "http://www.example.com/a/b/c/4"
      ],
      "metadata.urls": [
         "http://metadata.example.com/a/b/c"
      ],
      "type": "preposition"
   }
}
REQUEST:

GET /triggers/1 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*

RESPONSE:

HTTP/1.1 200 OK
Content-Length: 551
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "-7657333837290433420"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni; ptype=ci-trigger-status

{
  "ctime": 1409478799,
  "etime": 1409478807,
  "mtime": 1409478799,
  "status": "pending",
  "trigger": {
    "content.patterns": [
      {
        "case-sensitive": true,
        "pattern": "http://www.example.com/a/b/**"
      }
    ],
    "content.urls": [
      "http://www.example.com/a/index.html"
    ],
    "metadata.patterns": [
      {
        "pattern": "http://metadata.example.com/a/b/**"
      }
    ],
    "type": "invalidate"
  }
}

6.2.4. Polling for Change

The uCDN SHOULD use the Entity Tags of collections or Trigger Status Resources when polling for change in status, as shown in the following examples:
REQUEST:
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "5012053611544832286"

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

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

RESPONSE:
HTTP/1.1 304 Not Modified
Content-Length: 0
Expires: Sun, 31 Aug 2014 09:54:19 GMT
Server: example-server/0.1
Etag: "-4765587034697674779"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:19 GMT
Content-Type: application/cdni; ptype=ci-trigger-status

When the CI/T Trigger Command is complete, the contents of the filtered collections will be updated along with their Entity Tags. For example, when the two example CI/T Trigger Commands are complete, the collections of pending and complete Trigger Status Resources might look like:
REQUEST:

GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "5012053611544832286"

RESPONSE:

HTTP/1.1 200 OK
Content-Length: 56
Expires: Sun, 31 Aug 2014 09:54:29 GMT
Server: example-server/0.1
Etag: "-4471185573414616962"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:29 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

{
  "staleresourcetime": 86400,
  "triggers": []
}
REQUEST:
GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "2986340333785000363"

RESPONSE:
HTTP/1.1 200 OK
Content-Length: 153
Expires: Sun, 31 Aug 2014 09:54:30 GMT
Server: example-server/0.1
Etag: "-1508172875796647067"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:30 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

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

6.2.5. Deleting Trigger Status Resources

The dCDN can delete completed and failed Trigger Status Resources to reduce the size of the collections. For example, to delete the "preposition" request from earlier examples:
REQUEST:

DELETE /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*

RESPONSE:

HTTP/1.1 204 No Content
Date: Sun, 31 Aug 2014 09:53:30 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: example-server/0.1

This would, for example, cause the collection of completed Trigger Status Resources shown in the example above to be updated to:

REQUEST:

GET /triggers/complete HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*

RESPONSE:

HTTP/1.1 200 OK
Content-Length: 106
Expires: Sun, 31 Aug 2014 09:54:30 GMT
Server: example-server/0.1
Etag: "-1842390246836476263"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:30 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

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

In this example the uCDN has requested prepositioning of "http://newsite.example.com/index.html", but the dCDN was unable to locate metadata for that site:

REQUEST:

GET /triggers/2 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*

RESPONSE:

HTTP/1.1 200 OK
Content-Length: 505
Expires: Sun, 31 Aug 2014 09:54:38 GMT
Server: example-server/0.1
Etag: "-3893590191073700822"
Cache-Control: max-age=60
Date: Sun, 31 Aug 2014 09:53:38 GMT
Content-Type: application/cdni; ptype=ci-trigger-status

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

7.1. CDNI Payload Type Parameter Registrations

The IANA is requested to register the following new Payload Types in the CDNI Payload Type Parameter registry defined by [I-D.ietf-cdni-media-type], for use with the ‘application/cdni’ MIME media type.

RFC Editor Note: Please replace references to [RFCthis] below with this document’s RFC number before publication.

<table>
<thead>
<tr>
<th>Payload Type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ci-trigger-command</td>
<td>[RFCthis]</td>
</tr>
<tr>
<td>ci-trigger-status</td>
<td>[RFCthis]</td>
</tr>
<tr>
<td>ci-trigger-collection</td>
<td>[RFCthis]</td>
</tr>
</tbody>
</table>

8. Security Considerations

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

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

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

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

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

The use of TLS for transport of the CI/T interface allows:

- The dCDN and the uCDN to authenticate each other.

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

- The dCDN and the uCDN to authorize each other (to ensure they are receiving CI/T Commands from, or reporting status to, an authorized CDN).

- CDNI commands and responses to be transmitted with confidentiality.

- Protection of the integrity of CDNI commands and responses.

In an environment where any such protection is required, mutually authenticated encrypted transport MUST be used to ensure confidentiality of the CI/T information. To that end, TLS MUST be used by CI/T, including authentication of the remote end.

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

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

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

8.2. Denial of Service

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

Depending on the implementation, triggered activity may consume significant processing and bandwidth in the dCDN. A malicious or faulty uCDN could use this to generate unnecessary load in the dCDN. The dCDN should consider mechanisms to avoid overload, for example by rate-limiting acceptance or processing of CI/T Commands, or batching up its processing.

8.3. Privacy

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

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

9. Acknowledgements

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

10. References

10.1. Normative References


10.2. Informative References

[I-D.greevenbosch-appsawg-cbor-cddl]

[I-D.ietf-cdni-media-type]
Ma, K., "CDNI Media Type Registration", draft-ietf-cdni-media-type-06 (work in progress), October 2015.

[I-D.ietf-cdni-metadata]

[I-D.ietf-cdni-redirection]


Internet-Draft          CDN Interconnect Triggers           October 2015


Authors' Addresses

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge  CB24 6DD
UK

Email: rob.murray@alcatel-lucent.com

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

Email: ben.niven-jenkins@alcatel-lucent.com
CDNI Request Routing: Footprint and Capabilities Semantics

draft-ietf-cdni-footprint-capabilities-semantics-09

Abstract

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

Requirements Language

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

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
This Internet-Draft will expire on May 6, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction and Scope ..................................... 3
2. Design Decisions for Footprint and Capabilities ............. 4
  2.1. Advertising Limited Coverage .......................... 4
  2.2. Capabilities and Dynamic Data .......................... 5
  2.3. Advertisement versus Queries ............................ 6
  2.4. Avoiding or Handling 'cheating' dCDNs .................. 7
  2.5. Focusing on Main Use Cases ............................. 7
3. Main Use Case to Consider .................................. 8
4. Semantics for Footprint Advertisement ........................ 8
5. Semantics for Capabilities Advertisement .................. 11
6. Negotiation of Support for Optional Types of Footprint/Capabilities .................................. 13
7. Capability Advertisement Object ............................. 14
  7.1. Base Advertisement Object ............................... 14
  7.2. Delivery Protocol Capability Object .................... 14
  7.3. Acquisition Protocol Capability Object ................. 15
  7.4. Redirection Mode Capability Object ..................... 15
  7.5. Capability Advertisement Object Serialization ......... 15
8. IANA Considerations ........................................ 16
  8.1. CDNI Payload Types ...................................... 16
    8.1.1. CDNI FCI DeliveryProtocol Payload Type ............ 17
    8.1.2. CDNI FCI AcquisitionProtocol Payload Type ......... 17
    8.1.3. CDNI FCI RedirectionMode Payload Type .............. 17
  8.2. Redirection Mode Registry ................................ 17
9. Security Considerations ..................................... 18
10. References ................................................ 19
  10.1. Normative References ................................... 19
  10.2. Informative References .................................. 19
Appendix A. Acknowledgment ..................................... 20
1. Introduction and Scope

The CDNI working group is working on a set of protocols to enable the interconnection of multiple CDNs to a CDN federation. This CDN-federation should serve multiple purposes, as discussed in [RFC6770], for instance, to extend the reach of a given CDN to areas in the network which are not covered by this particular CDN.

The goal of this document is to achieve a clear understanding 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. However, guidelines for such FCI protocols are provided.

General assumptions in this document:

- 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.
- 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.
- The upstream CDN (uCDN) receives the initial request-routing request from the endpoint requesting the resource.

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

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

2. Design Decisions for Footprint and Capabilities

A large part of the difficulty in discussing the FCI lies in understanding what exactly is meant when trying to define footprint in terms of "coverage" or "reachability." While the operators of CDNs pick strategic locations to situate caches, a cache with a public IPv4 address is reachable by any endpoint on the Internet unless some policy enforcement precludes the use of the cache.

Some CDNs aspire to cover the entire world, which we will henceforth call global CDNs. The footprint advertised by such a CDN in the CDNI environment would, from a coverage or reachability perspective, presumably cover all prefixes. Potentially more interesting for CDNI use cases, however, are CDNs that claim a more limited coverage, but seek to federate with other CDNs in order to create a single CDN fabric which shares resources.

Furthermore, not all capabilities need be footprint restricted. Depending upon the use case, the optimal semantics of "footprints with capability attributes" vs. "capabilities with footprint restrictions" are not clear.

The key to understanding the semantics of footprint and capability advertisement lies in understand why a dCDN would advertise a limited coverage area, and how a uCDN would use such advertisements to decide among one of several dCDNs. The following section will discuss some of the trade-offs and design decisions that need to be decided upon for the CDNI FCI.

2.1. Advertising Limited Coverage

The basic use case that would motivate a dCDN to advertise a limited coverage is that the CDN was built to cover only a particular portion of the Internet. For example, an ISP could purpose-build a CDN to
serve only their own customers by situating caches in close
topological proximity to high concentrations of their subscribers.
The ISP knows the prefixes it has allocated to end users and thus can
easily construct a list of prefixes that its caches were positioned
to serve.

When such a purpose-built CDN joins a federation, and advertises its
footprint to a uCDN, however, 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
the uCDN wants to use to make a selection of a dCDN. If a uCDN has
three dCDNs to choose from that "cover" the IP address of user E,
obviously the uCDN might be interested to know how optimal the
coverage is from each of the dCDNs - coverage need not be binary,
either provided or not provided. dCDNs could advertise a coverage
"score," for example, and provided that they all reported scores
fairly on the same scale, uCDNs could use that to make their
topological optimality decision. Alternately, dCDNs could advertise
the IP addresses of their caches rather than prefix "coverage," and
let the uCDN decide for itself (based on its own topological
intelligence) which dCDN has better resources to serve a given user.

In summary, the semantics of advertising footprint depend on whether
such qualitative metrics for expressing footprint (such as the
coverage 'score' mentioned above) should be part of the CDNI FCI, or
if it should focus just on 'binary' footprint.

2.2. Capabilities and Dynamic Data

In cases where the apparent footprints of dCDNs overlap, uCDNs might
also want to rely on other factors to evaluate the respective merits
of dCDNs. These include facts related to the caches themselves, to
the network where the cache is deployed, to the nature of the
resource sought, and to the administrative policies of the respective
networks.
In the absence of network-layer impediments to reaching caches, the choice to limit coverage is necessarily an administrative policy. Much policy must be agreed upon before CDNs can merge into federations, including questions of membership, compensation, volumes, and so on. A uCDN certainly will factor these sorts of considerations into its decision to select a dCDN, but there is probably little need for dCDNs to actually advertise them through an interface - they will be settled out-of-band as a precondition for federating.

Other facts about the dCDN would be expressed through the interface to the uCDN. Some capabilities of a dCDN are static, and some are highly dynamic. Expressing the total storage built into its caches, for example, changes relatively rarely, whereas the amount of storage in use at any given moment is highly volatile. Network bandwidth similarly could be expressed as either total bandwidth available to a cache, or based on the current state of the network. A cache may at one moment lack a particular resource in storage, but have it the next.

The semantics of the capabilities interface will depend on how much of the dCDN state needs to be pushed to the uCDN and qualitatively how often that information should be updated.

2.3. Advertisement versus Queries

In a federated CDN environment, each dCDN shares some of its state with the uCDN. The uCDN uses this information to build a unified picture of all of the dCDNs available to it. In architectures that share detailed capability information, the uCDN could perform the entire request-routing operation down to selecting a particular cache in the dCDN (note: within the current CDNI WG charter, 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 for dCDNs with thousands or tens of thousands of caches; the volume of updates to footprint and capability becomes onerous.

Were the volume of FCI updates from dCDNs to exceed the volume of requests to the uCDN, it might make more sense for the uCDN to query dCDNs upon receiving requests (as is the case in the recursive redirection mode described in [RFC7336]), instead of receiving advertisements and tracking the state of dCDNs. The advantage of querying dCDNs would be that much of the dynamic data that dCDNs cannot share with the uCDN would now be factored into the uCDN’s decision. dCDNs need not replicate any state to the uCDN - uCDNs could effectively operate in a stateless mode.
The semantics of both footprint and capability advertisement depend on the service model here: are there cases where a synchronous query/response model would work better for the uCDN decision than a state replication model?

2.4. Avoiding or Handling ‘cheating’ dCDNs

In a situation where more than one dCDN is willing to serve a given end user request, it might be attractive for a dCDN to ‘cheat’ in the sense that the dCDN provides inaccurate information to the uCDN in order to convince the uCDN to select it over ‘competing’ dCDNs. It could therefore be desirable to take away the incentive for dCDNs to cheat (in information advertised) as much as possible. One option is to make the information the dCDN advertises somehow verifiable for the uCDN. One the other hand, a cheating dCDN might be avoided or handled by the fact that there will be strong contractual agreements between a uCDN and a dCDN, so that a dCDN would risk severe penalties or legal consequences when caught cheating.

Overall, the information a dCDN advertises should (in the long run) be somehow qualitatively verifiable by the uCDN, though possibly through non-real-time out-of-band audits. It is probably an overly strict requirement to mandate that such verification be possible "immediately", i.e., during the request routing process itself. If the uCDN can detect a cheating dCDN at a later stage, it should suffice for the uCDN to "de-incentivize" cheating because it would negatively affect the long-term business relationship with a particular dCDN.

2.5. Focusing on Main Use Cases

To narrow down semantics for "footprint" and "capabilities" in the CDNI context, it can be useful to initially focus on key use cases to be addressed by the CDNI WG that are to be envisioned the main deployments in the foreseeable future. In this regard, a main realistic use case is the existence of ISP-owned CDNs, which essentially cover a certain operator’s network. At the same time, however, the possibility of overlapping footprints should not be excluded, i.e., the scenario where more than one dCDN claims it can serve a given end user request. The ISPs may also choose to federate with a fallback global CDN.

It seems reasonable to assume that in most use cases it is the uCDN that makes the decision on selecting a certain dCDN for request routing based on information the uCDN has received from this particular dCDN. It may be assumed that ‘cheating’ CDNs will be dealt with via means outside the scope of CDNI and that the information advertised between CDNs is accurate. In addition,
excluding the use of qualitative information (e.g., cache proximity, delivery latency, cache load) to predict the quality of delivery would further simplify the use case allowing it to better focus on the basic functionality of the FCI.

3. Main Use Case to Consider

Focusing on a main use case that contains a simple (yet somewhat challenging), realistic, and generally imaginable scenario can help in narrowing down the requirements for the CDNI FCI. To this end, the following (simplified) use case can help in clarifying the semantics of footprint and capabilities for CDNI. In particular, the intention of the use case is to clarify what information needs to be exchanged on the CDNI FCI, what types of information need to be supported in a mandatory fashion (and which should be considered optional), and what types of information need to be updated with respect to a priori established CDNI contracts.

Use case: A given uCDN has several dCDNs. It selects one dCDN for delivery protocol A and footprint 1 and another dCDN for delivery protocol B and footprint 1. The dCDN that serves delivery protocol B has a further, transitive (level-2) dCDN, that serves delivery protocol B in a subset of footprint 1 where the first-level dCDN cannot serve delivery protocol B itself. What happens if capabilities change in the transitive level-2 dCDN that might affect how the uCDN selects a level-1 dCDN (e.g., in case the level-2 dCDN cannot serve delivery protocol B anymore)? How will these changes be conveyed to the uCDN? In particular, what information does the uCDN need to be able to select a new first-level dCDN, either for all of footprint 1 or only for the subset of footprint 1 that the transitive level-2 dCDN served on behalf of the first-level dCDN?

4. Semantics for Footprint Advertisement

Roughly speaking, "footprint" can be defined as "ability and willingness to serve" by a downstream CDN. However, in addition to simple "ability and willingness to serve", the uCDN may wish to have additional information to make a dCDN selection decision, e.g., "how well" a given dCDN can actually serve a given end user request. The "ability and willingness" to serve should be distinguished from the subjective qualitative measurement of "how well" it was served. One can imagine that such additional information is implicitly associated with a given footprint, e.g., due to contractual agreements (e.g., SLAs), business relationships, or perceived dCDN quality in the past. As an alternative, such additional information could also be explicitly tagged along with the footprint.
It is reasonable to assume that a significant part of the actual footprint advertisement will happen in contractual agreements between participating CDNs, i.e., prior to the advertisement phase using the CDNI FCI. The reason for this assumption is that any contractual agreement is likely to contain specifics about the dCDN coverage (i.e., the dCDN footprint) to which the contractual agreement applies. In particular, additional information to judge the delivery quality associated with a given dCDN footprint might be defined in contractual agreements (i.e., outside of the CDNI FCI). Further, one can assume that dCDN contractual agreements about the delivery quality associated with a given footprint will probably be based on high-level aggregated statistics (i.e., not too detailed).

Given that a large part of footprint advertisement will actually happen in contractual agreements, the semantics of CDNI footprint advertisement refer to answering the following question: what exactly still needs to be advertised by the CDNI FCI? For instance, updates about temporal failures of part of a footprint can be useful information to convey via the CDNI request routing interface. Such information would provide updates on information previously agreed in contracts between the participating CDNs. In other words, the CDNI FCI is a means for a dCDN to provide changes/updates regarding a footprint it has prior agreed to serve in a contract with a uCDN.

Generally speaking, one can imagine two categories of footprint to be advertised by a dCDN:

- Footprint could be defined based on "coverage/reachability", where coverage/reachability refers to a set of prefixes, a geographic region, or similar boundary. The dCDN claims that it can cover/reach ‘end user requests coming from this footprint’.

- 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 "resource" defined footprint.

It seems clear that "coverage/reachability" types of footprint MUST be supported within CDNI. The following such types of footprint are mandatory and MUST be supported by the CDNI FCI:
A ‘set of IP-prefixes’ must be able to contain full IP addresses, i.e., a /32 for IPv4 and a /128 for IPv6, as well as IP prefixes with an arbitrary prefix length. There must also be support for multiple IP address versions, i.e., IPv4 and IPv6, in such a footprint.

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

For all of these mandatory-to-implement footprint types, the footprints can be viewed as constraints for delegating requests to a dCDN: A dCDN footprint advertisement tells the uCDN the limitations for delegating a request to the dCDN. For IP prefixes or ASN(s), the footprint signals to the uCDN that it should consider the dCDN a candidate only if the IP address of the request routing source falls within the prefix set (or ASN, respectively). The CDNI specifications do not define how a given uCDN determines what address ranges are in a particular ASN. Similarly, for country codes a uCDN should only consider the dCDN a candidate if it covers the country of the request routing source. The CDNI specifications do not define how a given uCDN determines the country of the request routing source. Multiple footprint constraints are additive, i.e., the advertisement of different types of footprint narrows the dCDN candidacy cumulatively.

In addition to these mandatory "coverage/reachability" types of footprint, other optional "coverage/reachability" types of footprint or "resource" types of footprint may defined by future specifications. To facilitate this, a clear process for specifying optional footprint types in an IANA registry is specified in the CDNI Metadata Footprint Types registry (defined in the CDNI Metadata Interface document [I-D.ietf-cdni-metadata].
Independent of the exact type of a footprint, a footprint might also include the connectivity of a given dCDN to other CDNs that may be able to serve content to users on behalf of that dCDN, to cover cases where there is a transitive CDN interconnection. Further, the downstream CDN must be able to express its footprint to an interested upstream CDN (uCDN) in a comprehensive form, e.g., as a data set containing the complete footprint. Making incremental updates, however, to express dynamic changes in state is also desirable.

5. Semantics for Capabilities Advertisement

In general, the dCDN must be able to express its general capabilities to the uCDN. These general capabilities could express if the dCDN supports a given service, for instance, HTTP delivery, RTP/RTSP delivery or RTMP. Furthermore, the dCDN must be able to express particular capabilities for the delivery in a particular footprint area. For example, the dCDN might in general offer RTMP but not in some specific areas, either for maintenance reasons or because the caches covering this particular area cannot deliver this type of service. Hence, in certain cases footprint and capabilities are tied together and cannot be interpreted independently from each other. In such cases, i.e., where capabilities must be expressed on a per footprint basis, it may be beneficial to combine footprint and capabilities advertisement.

A high-level and very rough semantic for capabilities is thus the following: Capabilities are types of information that allow a uCDN to determine if a downstream CDN is able (and willing) to accept (and properly handle) a delegated content request. In addition, Capabilities are characterized by the fact that this information may possibly change over time based on the state of the network or caches.

At a first glance, several broad categories of capabilities seem useful to convey via an advertisement interface, however, advertising capabilities that change highly dynamically (e.g., real-time delivery performance metrics, CDN resource load, or other highly dynamically changing QoS information) should probably not be in scope for the CDNI FCI. First, out of the multitude of possible metrics and capabilities, it is hard to agree on a subset and the precise metrics to be used. Second, and perhaps more importantly, it seems not feasible to specify such highly dynamically changing capabilities and the corresponding metrics within the CDNI charter time-frame.

Useful capabilities refer to information that does not change highly dynamically and which in many cases is absolutely necessary to decide on a particular dCDN for a given end user request. For instance, if an end user request concerns the delivery of a video file with a
certain protocol (e.g., RTMP), the uCDN needs to know if a given dCDN has the capability of supporting this delivery protocol.

Similar to footprint advertisement, it is reasonable to assume that a significant part of the actual (resource) capabilities advertisement will happen in contractual agreements between participating CDNs, i.e., prior to the advertisement phase using the CDNI FCI. The role of capability advertisement is hence rather to enable the dCDN to update a uCDN on changes since a contract has been set up (e.g., in case a new delivery protocol is suddenly being added to the list of supported delivery protocols of a given dCDN, or in case a certain delivery protocol is suddenly not being supported anymore due to failures). Capabilities advertisement thus refers to conveying information to a uCDN about changes/updates of certain capabilities with respect to a given contract.

Given these semantics, it needs to be decided what exact capabilities are useful and how these can be expressed. Since the details of CDNI contracts are not known at the time of this writing (and the CDNI interface should probably be agnostic to these contracts anyway), it remains to be seen what capabilities will be used to define agreements between CDNs in practice. One implication for standardization may be to initially only specify a very limited set of mandatory capabilities for advertisement and have on top of that a flexible data model that allows exchanging additional capabilities when needed. Still, agreement needs to be found on which capabilities (if any) should be mandatory among CDNs. As discussed in Section 2.5, finding the concrete answers to these questions can benefit from focusing on a small number of key use cases that are highly relevant and contain enough complexity to help in understanding what concrete capabilities are needed to facilitate CDN Interconnection.

Under the above considerations, the following capabilities seem useful as 'base' capabilities, i.e., ones that are needed in any case and therefore constitute mandatory capabilities that MUST be supported by the CDNI FCI:

- Delivery Protocol (e.g., HTTP vs. RTMP)
- Acquisition Protocol (for acquiring content from a uCDN)
- Redirection Mode (e.g., DNS Redirection vs. HTTP Redirection as discussed in [RFC7336])
- CDNI Logging (i.e., supported logging fields)
- CDNI Metadata (i.e., supported Generic Metadata types)
It is not feasible to enumerate all the possible options for the mandatory capabilities listed above (e.g., all the potential delivery protocols or metadata options) or anticipate all the future needs for additional capabilities. It would be unreasonable to burden the CDNI FCI specification with defining each supported capability. Instead, the CDNI FCI specification should define a generic protocol for conveying any capability information (e.g. with common encoding, error handling, and security mechanism; further requirements for the CDNI FCI Advertisement Interface are listed in [RFC7337]). In this respect, it seems reasonable to define a registry which initially contains the mandatory capabilities listed above, but may be extended as needs dictate. This document defines the registry (and the rules for adding new entries to the registry) for the different capability types (see Section 8). Each capability type MAY have a list of valid values. Future specifications which define a given capability SHOULD define any necessary registries (and the rules for adding new entries to the registry) for the values advertised for a given capability type.

The "CDNI Logging Fields Names" registry defines all supported logging fields, including mandatory-to-implement logging fields. Advertising support for mandatory-to-implement logging fields SHOULD be supported but would be redundant. CDNs SHOULD NOT advertise support for mandatory-to-implement logging fields. The following logging fields are defined as optional in the CDNI Logging Interface document [I-D.ietf-cdni-logging]:

- s-ccid
- s-sid

The CDNI Metadata Interface document [I-D.ietf-cdni-metadata] does not define any optional GenericMetadata types. Advertising support for mandatory-to-implement GenericMetadata types SHOULD be supported but would be redundant. CDNs SHOULD NOT advertise support for mandatory-to-implement GenericMetadata types.

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

The notion of optional types of footprint and capabilities implies that certain implementations may not support all kinds of footprint and capabilities. Therefore, any FCI solution protocol must define how the support for optional types of footprint/capabilities will be negotiated between a uCDN and a dCDN that use the particular FCI protocol. In particular, any FCI solution protocol needs to specify how to handle failure cases or non-supported types of footprint/capabilities.
In general, a uCDN may ignore capabilities or types of footprints it does not understand; in this case it only selects a suitable downstream CDN based on the types of capabilities and footprint it understands. Similarly, if a dCDN does not use an optional capability or footprint which is, however, supported by a uCDN, this causes no problem for the FCI functionality because the uCDN decides on the remaining capabilities/footprint information that is being conveyed by the dCDN.

7. Capability Advertisement Object

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

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

7.1. Base Advertisement Object

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

Property: capability-type

Description: CDNI Capability object type.

Type: FCI specific CDNI Payload type (from the CDNI Payload Types registry [I-D.ietf-cdni-media-type])

Mandatory-to-Specify: Yes.

Property: capability-value

Description: CDNI Capability object.

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

Mandatory-to-Specify: Yes.

7.2. Delivery Protocol Capability Object

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

Property: delivery-protocols
Description: List of supported CDNI Delivery Protocols.
Type: List of Protocol Types (from the CDNI Metadata Protocol Types registry [I-D.ietf-cdni-metadata])
Mandatory-to-Specify: Yes.

7.3. Acquisition Protocol Capability Object

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

Property: acquisition-protocols
Description: List of supported CDNI Acquisition Protocols.
Type: List of Protocol Types (from the CDNI Metadata Protocol Types registry [I-D.ietf-cdni-metadata])
Mandatory-to-Specify: Yes.

7.4. Redirection Mode Capability Object

The Redirection Mode capability object is used to indicate support for one or more of the modes listed in the CDNI Capabilities Redirection Modes registry (see Section 8.2).

Property: redirection-modes
Description: List of supported CDNI Redirection Modes.
Type: List of Redirection Modes (from Section 8.2)
Mandatory-to-Specify: Yes.

7.5. Capability Advertisement Object Serialization

The following shows an example of CDNI FCI Capability Advertisement Object Serialization.
8. IANA Considerations

8.1. CDNI Payload Types

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

<table>
<thead>
<tr>
<th>Payload Type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCI.DeliveryProtocol</td>
<td>RFCthis</td>
</tr>
<tr>
<td>FCI.AcquisitionProtocol</td>
<td>RFCthis</td>
</tr>
<tr>
<td>FCI.RedirectionMode</td>
<td>RFCthis</td>
</tr>
</tbody>
</table>

8.1.1. CDNI FCI DeliveryProtocol Payload Type

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

Interface: FCI

Encoding: see Section 7

8.1.2. CDNI FCI AcquisitionProtocol Payload Type

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

Interface: FCI

Encoding: see Section 7

8.1.3. CDNI FCI RedirectionMode Payload Type

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

Interface: FCI

Encoding: see Section 7

8.2. Redirection Mode Registry

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

The following table defines the initial Redirection Modes:
9. Security Considerations

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

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

It is not believed that there are any serious privacy risks in sharing footprint or capability information: it will represent highly aggregated data about networks and, at best, policy-related information about media, rather than any personally identifying information. However, particular dCDNs may wish to share information about their footprint with a uCDN but not with other, competing dCDNs. For example, if a dCDN incurs an outage that reduces footprint coverage temporarily, that may be information the dCDN would want to share confidentially with the uCDN. Protocols implementing these semantics SHOULD provide confidentiality services.
As specified in this document, the security requirements of the FCI could be met by hop-by-hop transport-layer security mechanisms coupled with domain certificates as credentials. There is no apparent need for further object-level security in this framework, as the trust relationships it defines are bilateral relationships between uCDNs and dCDNs rather than transitive relationships.

10. References

10.1. Normative References


10.2. Informative References


[I-D.ietf-cdni-media-type] Ma, K., "CDNI Media Type Registration", draft-ietf-cdni-media-type-06 (work in progress), October 2015.


Appendix A. Acknowledgment

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

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

Authors’ Addresses

Jan Seedorf
NEC
Kurfuerstenanlage 36
Heidelberg 69115
Germany
Phone: +49 6221 4342 221
Fax: +49 6221 4342 155
Email: seedorf@neclab.eu

Jon Peterson
NeuStar
1800 Sutter St Suite 570
Concord CA 94520
USA
Email: jon.peterson@neustar.biz
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 May 5, 2016.

Copyright Notice

Copyright (c) 2015 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. Requirements Language ............................. 4
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 Example ............................ 35
   3.6. Cascaded CDNI Logging Files Example ................. 37
4. Protocol for Exchange of CDNI Logging File After Full Collection ............................................. 40
   4.1. CDNI Logging Feed .................................. 41
      4.1.1. Atom Formatting ................................ 41
      4.1.2. Updates to Log Files and the Feed ................ 41
      4.1.3. Redundant Feeds ................................ 42
      4.1.4. Example CDNI Logging Feed ........................ 42
   4.2. CDNI Logging File Pull ............................... 44
5. Protocol for Exchange of CDNI Logging File During Collection ............................................. 45
6. IANA Considerations ..................................... 46
   6.1. CDNI Logging Directive Names Registry ............... 46
   6.2. CDNI Logging File version Registry .................. 47
   6.3. CDNI Logging record-types Registry ................... 47
   6.4. CDNI Logging Field Names Registry ................... 48
   6.5. CDNI Logging MIME Media Type ........................ 49
7. 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:

- CDNI problem statement [RFC6707] and framework [RFC7336] identify a Logging interface,
- Section 8 of [RFC7337] specifies a set of requirements for Logging,
- [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:

- The CDNI Logging reference model (Section 2),
- The CDNI Logging File format (Section 3),
- 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:

- 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.

- 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 CDN interfaces.

- 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.

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 3.6.

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.
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:

- 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)
- Percentage of delivery successes/failures among the aforementioned requests
- 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].

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 ["." *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 = 2DIGIT ":" 2DIGIT ":" 2DIGIT ["." *DIGIT]

; Times are encoded as "partial-time" specified in [RFC3339].

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.
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."

\[DIRLINE = "#" directive CRLF\]
\[DIRGROUP = 1*DIRLINE\]
\[RECLINE = any subset of record values that match what is expected according to the fields directive within the immediately preceding DIRGROUP.\]
All directive names and field names defined in the logging file are case-insensitive as per the basic ABNF ([RFC5234]).

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. 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 record-type "cdni_http_request_v1."

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:

- 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.
  * 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".

** 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".

** 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".

** 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.

  * 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).

  * 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)".

* 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 reject 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 reject 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 reject 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:

- "c-" refers to the User Agent that issues the request (corresponds to the "client" of W3C Extended Log Format)
"d-" refers to the dCDN (relative to a given CDN acting as an uCDN)

"s-" refers to the dCDN Surrogate that serves the request (corresponds to the "server" of W3C Extended Log Format)

"u-" refers to the uCDN (relative to a given CDN acting as a dCDN)

"cs-" refers to communication from the User Agent towards the dCDN Surrogate

"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:

- 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.

- time:
  * format: TIME
  * field value: the time at which the processing of request completed on the Surrogate.
  * occurrence: there MUST be one and only one instance of this field.

- 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.

- 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 (e.g., US/TN/MEM/38138/43a5bddd6-95c4-4d62-be65-7410df0021e2) where some elements identify aggregates and one element identifies the client.

+ the address with a shared secret that is pre-synchronised and rotated at a predefined time interval. It is RECOMMENDED that the mapping varies at least once every 24 hours. The mapping from IP address to the element identifying the client (effectively an encryption of the address) SHOULD be done using a symmetric key that is known only to both the uCDN and dCDN. One method of doing this is to use an analog of how key derivation is via HKDF ([RFC5869]), as will be used in TLS 1.3 ([I-D.ietf-tls-rfc5246-bis]). When the two CDNs set up the relationship, they agree out-of-band on a mapping between IPv4 and IPv6 addresses and aggregates and on the algorithmic mapping from IPv4/IPv6 addresses and the element identifying the client; they agree on the salt and input key material, as described in [RFC5869], Section 2.2, the hash mechanism to use (SHA-2 or SHA-3 SHOULD be used), and the key lifetime which SHOULD NOT exceed 25 hours. They also agree on an initial "info" parameter, which can be something such as the business names of the two organizations in UTF-8, concatenated. The encryption algorithm also needs to be defined, and SHOULD be 128-bit AES-GCM or better. The PRK should be chosen by both parties contributing alternate random bytes until sufficient length exists. After this initial setup, client-information is encrypted using the key generated by the "expand" step, Section 2.3 of [RFC5869], and hex-encoded or base64-encoded. At the agreed-upon lifetime, a new key is used, indicated by prefixing the key with a special character such as exclamation point. In this way, shorter lifetimes can be used as needed.
+ 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); one example way to achieve this is to hash.

+ The algorithmic mapping and variation over time MAY allow the uCDN (with the knowledge of the algorithm and time variation and associated attributes and keys) to reconstruct the actual enduser IPv4/IPv6 addresses where that is required (e.g., to allow a-posteriori analysis of individual delivery for example in situations of performance-based penalties). However, these enduser IPv4/IPv6 addresses SHOULD only be reconstructed on-demand and the CDNI Logging File SHOULD only be stored with the anonymised c-groupid value.

+ Using the c-groupid field in this way 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.

* 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.

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.

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:

- date
- time
- time-taken
- c-groupid
- s-ip
- s-hostname
- s-port
- cs-method
- cs-uri
- u-uri
- protocol
- sc-status
- sc-total-bytes
- sc-entity-bytes
- cs(insert_HTTP_header_name_here)
- sc(insert_HTTP_header_name_here)
- 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":

- s-ccid
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 reject 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 Example

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.

3.6. 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 5. 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 5 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> <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 5: 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-2 on behalf of dCDN-3 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 6.
#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 6: 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 6, we observe that:

- 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).

- 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:

- 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.

- 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 [I-D.ietf-cdni-media-type]) 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 [I-D.ietf-cdni-media-type]).

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
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 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 7 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 7: 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:

- 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.

- 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;

- MUST support exchange of CDNI Logging Files with no content encoding applied to the representation;

- 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:

- MUST implement HTTP/1.1 to handle the client-side request and MAY also support other HTTP versions (e.g., HTTP/2);

- MUST include the CDNI Logging File identified by the request URI inside the body of the HTTP response;

- MUST support exchange of CDNI Logging Files with no content encoding applied to the representation;

- MUST support exchange of CDNI Logging Files with "gzip" content encoding (as defined in [RFC7230]) 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

When IANA allocates new extensions to CDNI Logging Directive Names Registry, CDNI Logging File version Registry, CDNI Logging record-type Registry or CDNI Logging fields Registry, IANA MUST take into account that the directive names are case-insensitive as per the basic ABNF ([RFC5234]).

6.1. CDNI Logging Directive Names Registry

The IANA is requested to create a new "CDNI Logging Directive Names" registry under the "Content Delivery Networks Interconnection (CDNI) Parameters" category.

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:

<table>
<thead>
<tr>
<th>Directive Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>UUID</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>claimed-origin</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>established-origin</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>remark</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>record-type</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>fields</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>SHA256-hash</td>
<td>RFC xxxx</td>
</tr>
</tbody>
</table>

Figure 8

[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 and field 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" registry under the "Content Delivery Networks Interconnection (CDNI) Parameters" category.

The initial contents of the CDNI Logging File version registry comprise the value "CDNI/1.0" specified in Section 3.3 of the present document, and are as follows:

<table>
<thead>
<tr>
<th>version</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdni/1.0</td>
<td>RFC xxxx</td>
<td>CDNI Logging File version 1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>as specified in RFC xxxx</td>
</tr>
</tbody>
</table>

Figure 9

[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).

6.3. CDNI Logging record-types Registry

The IANA is requested to create a new "CDNI Logging record-types" under the "Content Delivery Networks Interconnection (CDNI) Parameters" category.

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:

<table>
<thead>
<tr>
<th>record-types</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdni_http_request_v1</td>
<td>RFC xxxx</td>
<td>CDNI Logging Record version 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for content delivery using HTTP</td>
</tr>
</tbody>
</table>

Figure 10

[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).

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:

- a list of all the CDNI Logging fields that can appear in a CDNI Logging Record of the new record-type
- for all these fields: a specification of the occurrence for each Field in the new record-type
- 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" under the "Content Delivery Networks Interconnection (CDNI) Parameters" category.

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:
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>time</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>time-taken</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>c-groupid</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>s-ip</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>s-hostname</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>s-port</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>cs-method</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>cs-uri</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>u-uri</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>protocol</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>sc-status</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>sc-total-bytes</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>sc-entity-bytes</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>cs(insert_HTTP_header_name_here)</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>sc(insert_HTTP_header_name_here)</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>s-ccid</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>s-sid</td>
<td>RFC xxxx</td>
</tr>
<tr>
<td>s-cached</td>
<td>RFC xxxx</td>
</tr>
</tbody>
</table>

Figure 11

[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).

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.

[RFCEditor Note: Please replace the references to [RFCthis] below with this document’s RFC number before publication.]
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].

The use of TLS for transport of the CDNI Logging feed and CDNI Logging File pull allows:

- the dCDN and uCDN to authenticate each other

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

- the dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI Logging File to/from an authorized CDN)

- the CDNI Logging information to be transmitted with confidentiality

- the integrity of the CDNI Logging information to be protected during the exchange.

In an environment where any such protection is required, mutually authenticated encrypted transport MUST be used to ensure confidentiality of the logging information. To that end, TLS MUST be used (including authentication of the remote end) 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.
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 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. 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 - at least when HTTP redirection is used between uCDN and dCDN). 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).

However, one 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


9.2. Informative References


[I-D.ietf-cdni-media-type] Ma, K., "CDNI Media Type Registration", draft-ietf-cdni-media-type-06 (work in progress), October 2015.


Authors’ Addresses

Francois Le Faucheur (editor)
Cisco Systems
E.Space Park - Batiment D
6254 Allee des Ormes - BP 1200
Mougins cedex  06254
FR

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

Gilles Bertrand (editor)
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)
Orange
38-40 rue du General Leclerc
Issy les Moulineaux 92130
FR
Phone: +33 6 89 06 92 72
Email: iuniana.oprescu@orange.com

Roy Peterkofsky
Skytide, Inc.
One Kaiser Plaza, Suite 785
Oakland CA 94612
USA
Phone: +01 510 250 4284
Email: roy@skytide.com
CDNI Media Type Registration
draft-ietf-cdni-media-type-06

Abstract

This document defines the standard media type used by the Content Delivery Network Interconnection (CDNI) protocol suite, including the registration procedure and recommended usage of the required payload-type parameter.

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 15, 2016.

Copyright Notice

Copyright (c) 2015 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.
1. Introduction and Scope

The Content Delivery Network Interconnection (CDNI) working group is developing a set of protocols to enable the interconnection of multiple content delivery networks (CDNs), as discussed in [RFC6770]. The CDNI protocol suite consists of multiple HTTP-based interfaces, many of which transfer various JSON (JavaScript Object Notation) encoded payloads [RFC7159]. The main interfaces (i.e., CDNI Control interface, CDNI Footprint & Capabilities Advertisement interface, CDNI Request Routing Redirection interface, CDNI Metadata interface, and CDNI Logging interface) are described in [RFC7336]. It is desirable to be able to indicate the type of object carried in the HTTP entity-body without having to register separate media types for each CDNI object. To accomplish this aim, this document defines a single new media type for CDNI that includes a required payload-type parameter. A separate registry of CDNI payload-type parameters is also defined. CDNI protocol specifications may register interface-specific payload-types, specifying the payload encoding and parsing semantics for that message (e.g., JSON serialization for a CDNI metadata object). The same payload-type parameter may also be used as references for other purposes (e.g., referencing CDNI metadata objects from CDNI capability advertisement objects).

2. IANA Considerations

This section contains the CDNI media type registration request for IANA, as well as the payload-type parameter registry definition for IANA.

2.1. CDNI Media Type

Type name: application

Subtype name: cdni

Required parameters:
ptype

The required parameter "ptype" describes the type of CDNI message contained in the message payload, as registered in the CDNI Payload Type Parameter Registry (Section 2.2) defined below.

Optional parameters: none

Encoding considerations:

The CDNI protocol suite includes interfaces with encoded messages which may be 8bit or binary, as well as generic logging information which may be 7bit or binary.

Security considerations:

CDNI interfaces that return encoded data may be (mis)interpreted if parsed by non-CDNI or non-compliant CDNI implementations. In addition, CDNI logging information is likely to transfer large amounts of data which may overload unexpecting clients. The individual CDNI interface specifications provide more detailed analysis of security and privacy concerns, and define the requirements for authentication, authorization, confidentiality, integrity, and privacy for each interface.

The application/cdni media type is a generic media type to be used by multiple CDNI interfaces for transporting different types of control and logging information. Proper validation of message data requires parsing and understanding the ptype parameter and the associated data encoding. Failure to properly validate payloads may allow data extrusion under the auspices of the application/cdni media type.

Interoperability considerations:

The required ptype field is intended to fully describe the structure and parsing of CDNI messages, as enforced by the ptype registry designated expert.

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

Applications that use this media type:

CDNI is intended for use between interconnected CDNs for sharing configuration and logging data, as well as for issuing content management and redirection requests.
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 protocol message exchanges.

Author: IETF CDNI working group

Change controller: IETF CDNI working group

Provisional registration: no

2.2. CDNI Payload Type Parameter Registry

The IANA is requested to create a new "CDNI Payload Type" registry, in a new category called "Content Delivery Networks Interconnection (CDNI) Parameters". The "CDNI Payload Type" namespace defines the valid values for the required "ptype" parameter of the "application/cdni" media type. The CDNI Payload Type is an ASCII string value, consisting of only visible (printing) characters, but excluding equal signs (=), double quotes ("), and semicolons (;), and not exceeding 256 characters in length.

\[
\text{ptype} = 1\times256(\text{ptype-char})
\]

\[
\text{ptype-char} = %x21 / %x23-3A / %x3C / %x3E-7E
\]

; Includes ALPHA, DIGIT, and other printables

; Excludes equal signs (=), double quotes ("), semicolons (;)

Additions to the CDNI Payload Type namespace will 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 (in name or functionality),
prevent gratuitous additions to the namespace, and prevent any
additions to the namespace which would impair the interoperability of
CDNI implementations. The designated expert will review the
specification, even if it is a Standards Track RFC, to verify that it
contains the following information:

- The review will verify that the specification contains a
  reasonably defined purpose for the new payload type, where the
  purpose is related to an existing or proposed CDNI interface and
does not duplicate the functionality of any existing CDNI protocol
feature without specifying a rational reason (e.g., updating an
obsolete feature), a method for detecting and handling conflicts
(e.g., a versioning system with prioritization matrix), and a
suggested migration path (e.g., deprecation of the overlapped
feature, or justification for co-existence).

- The review will verify that the specification contains information
  as to which CDNI interface the new payload type pertains/affects.
The payload type may be applicable to multiple CDNI interfaces,
but the justification for the new payload type will include a
reasonable relationship to at least one standards track CDNI
interface.

- The review will verify that the specification contains sufficient
detail about the data encoding (e.g., JSON serialization for new
CDNI metadata or capability advertisement objects, or ABNF and
description for new CDNI logging file formats) to allow senders
and receivers of the new payload type to implement compliant and
interoperable payload parsers.

The registry contains the Payload Type, and the specification
describing the Payload Type. The registry will initially be
unpopulated.

+-----------------+-----------------setup
| Payload Type    | Specification  |
+-----------------+-----------------+-----------------+

3. References

3.1. Normative References

[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,

Ma                       Expires April 15, 2016                 [Page 5]
3.2. Informative References


Appendix A. Acknowledgment

This document is the culmination of the efforts of many in the CDNI working group, including (in alphabetical order): Francois Le Faucheur, Daryl Malas, Rob Murray, Ben Niven-Jenkins, Iuniana Oprescu, Jon Peterson, and Jan Seedorf.

Author’s Address

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
USA

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

Ma Expires April 15, 2016 [Page 6]
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 April 21, 2016.
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 ....................... 11
  3.3. Metadata Inheritance and Override ................... 14
4. CDNI Metadata objects .................................... 15
  4.1. Definitions of the CDNI structural metadata objects ... 16
    4.1.1. HostIndex ......................................... 16
    4.1.2. HostMatch ......................................... 17
    4.1.3. HostMetadata ....................................... 18
    4.1.4. PathMatch ......................................... 19
    4.1.5. PatternMatch ...................................... 20
    4.1.6. PathMetadata ....................................... 21
    4.1.7. GenericMetadata .................................... 22
  4.2. Definitions of the initial set of CDNI Generic Metadata objects .................................... 24
    4.2.1. SourceMetadata ..................................... 24
      4.2.1.1. Source .......................................... 25
    4.2.2. LocationACL Metadata .............................. 26
      4.2.2.1. LocationRule ................................... 28
      4.2.2.2. Footprint ....................................... 28
    4.2.3. TimeWindowACL .................................... 29
      4.2.3.1. TimeWindowRule ................................ 30
      4.2.3.2. TimeWindow ..................................... 31
    4.2.4. ProtocolACL Metadata ............................. 32
      4.2.4.1. ProtocolRule .................................... 33
    4.2.5. DeliveryAuthorization Metadata .................. 34
4.2.6. Cache .................................................. 34
4.2.7. Auth .................................................... 35
4.2.8. Grouping ................................................. 36
4.3. CDNI Metadata Simple Data Type Descriptions ............. 37
  4.3.1. Link .................................................. 37
  4.3.2. Protocol ............................................... 38
  4.3.3. Endpoint .............................................. 38
  4.3.4. Time .................................................. 38
  4.3.5. IPv4CIDR ............................................. 39
  4.3.6. IPv6CIDR ............................................. 39
  4.3.7. ASN .................................................... 39
  4.3.8. CountryCode ......................................... 39
5. CDNI Metadata Capabilities .................................... 40
6. CDNI Metadata interface ........................................ 40
  6.1. Transport ............................................... 41
  6.2. Retrieval of CDNI Metadata resources .................... 42
  6.3. Bootstrapping .......................................... 43
  6.4. Encoding ................................................ 43
  6.5. Extensibility .......................................... 44
  6.6. Metadata Enforcement ................................... 44
  6.7. Metadata Conflicts ..................................... 45
  6.8. Versioning .............................................. 45
  6.9. Media Types ............................................ 46
  6.10. Complete CDNI Metadata Example ......................... 47
7. IANA Considerations ........................................... 50
  7.1. CDNI Payload Types ....................................... 50
    7.1.1. CDNI MI HostIndex Payload Type .................... 51
    7.1.2. CDNI MI HostMatch Payload Type .................... 51
    7.1.3. CDNI MI HostMetadata Payload Type ................ 52
    7.1.4. CDNI MI PathMatch Payload Type .................... 52
    7.1.5. CDNI MI PatternMatch Payload Type ................ 52
    7.1.6. CDNI MI PathMetadata Payload Type ................ 52
    7.1.7. CDNI MI SourceMetadata Payload Type ............... 52
    7.1.8. CDNI MI Source Payload Type ....................... 53
    7.1.9. CDNI MI LocationACL Payload Type ................. 53
    7.1.10. CDNI MI LocationRule Payload Type ............... 53
    7.1.11. CDNI MI Footprint Payload Type .................. 53
    7.1.12. CDNI MI TimeWindowACL Payload Type .............. 53
    7.1.13. CDNI MI TimeWindowRule Payload Type ............ 54
    7.1.14. CDNI MI TimeWindow Payload Type ................ 54
    7.1.15. CDNI MI ProtocolACL Payload Type ............... 54
    7.1.16. CDNI MI ProtocolRule Payload Type ............... 54
    7.1.17. CDNI MI DeliveryAuthorization Payload Type ...... 54
    7.1.18. CDNI MI Cache Payload Type ....................... 55
    7.1.19. CDNI MI Auth Payload Type ......................... 55
    7.1.20. CDNI MI Grouping Payload Type .................... 55
  7.2. CDNI Metadata Footprint Types Registry ................. 55
  7.3. CDNI Metadata Protocol Types Registry .................. 56
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 may be used to compose a CDNI solution (CDNI Control interface, CDNI Request Routing Redirection interface, CDNI Footprint & Capabilities Advertisement interface and CDNI Logging interface). [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 focuses on 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:

- Redirection requests received over the CDNI Request Routing Redirection interface [I-D.ietf-cdni-redirection].
- Content requests received directly from User Agents.

Specifically, this document specifies:

- A data structure for mapping content requests and redirection requests to CDNI metadata objects (Section 3 and Section 4.1).
- An initial set of CDNI Generic metadata objects (Section 4.2).
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:

- **Object** - a collection of properties.
- **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:

- **Acquisition Source**: Metadata for allowing a dCDN to fetch content from a uCDN.
- **Delivery Access Control**: Metadata for restricting (or permitting) access to content based on any of the following factors:
  - Location
  - Time Window
  - Delivery Protocol
- **Delivery Authorization**: Metadata for authorizing dCDN user agent requests.
- **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, covering the initial capabilities above, is only able to support CDN interconnection for the delivery of content by a dCDN using HTTPv1.1 [RFC7230] and for a dCDN to be able to acquire content from a uCDN using either HTTPv1.1 or HTTPv1.1 over TLS [RFC2818].

Supporting CDN interconnection for the delivery of content using unencrypted HTTPv2.0 [RFC7540] (as well as for a dCDN to acquire content using unencrypted HTTPv2.0 or HTTPv2.0 over TLS) requires the registration of these protocol names in the CDNI Metadata Protocol Registry.

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

2. Design Principles

The CDNI metadata interface was designed to achieve the following objectives:

1. Cacheability of CDNI metadata objects.
2. Deterministic mapping from redirection requests and content requests to CDNI metadata properties.
3. Support for DNS redirection as well as application-specific redirection (for example HTTP redirection).
4. Minimal duplication of CDNI metadata.
5. Leveraging of existing protocols.

Cacheability improves the latency of acquiring metadata while maintaining its freshness, and therefore improves the latency of serving content requests and redirection requests, without sacrificing accuracy. The CDNI metadata interface uses HTTP and its existing caching mechanisms to achieve CDNI metadata cacheability.

Deterministic mappings from content to metadata properties eliminates ambiguity and ensures that policies are applied consistently by all dCDNs.

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 (by leveraging I-JSON [RFC7493]) and data transport (by leveraging HTTP [RFC7230]).

3. CDNI Metadata object model

The CDNI metadata object model describes a data structure for mapping redirection requests and content requests to metadata properties. Metadata properties describe how to acquire content from an uCDN, authorize access to content, and deliver content from a dCDN. The object model relies on the assumption that these metadata properties may be aggregated 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 or URI path. As a result of this structure, significant commonality may exist between the CDNI metadata properties specified for different Hostnames, different URI paths within a Hostname and different URI paths on different Hostnames. For example the definition of which User Agent IP addresses should be treated as being grouped together into a single network or geographic location is likely to be common for a number of different Hostnames. Another example is that although a uCDN is likely to have several different policies configured to express geo-blocking rules, it is likely that a single geo-blocking policy would be applied to multiple Hostnames delivered through the CDN.

In order to enable the CDNI metadata for a given Hostname or URI Path to be decomposed into sets of CDNI metadata properties that can be reused by multiple Hostnames and URI Paths, the CDNI metadata interface specified in this document splits the CDNI metadata into a number of objects. Efficiency is improved by enabling a single CDNI metadata object (that is shared across Hostname and/or URI paths) to
be retrieved and stored by a dCDN once, even if it is referenced by
the CDNI metadata of multiple Hostnames or of multiple URI paths.

Important Note: Any CDNI metadata object A that contains another CDNI
metadata object B may, instead of including the second object B
embedded within object A, include a Link object that contains a URI
that can be dereferenced to retrieve the complete serialized
representation of the second metadata object B. 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 and which CDNI metadata objects to embed and which to make
available as 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 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 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.
A HostIndex object (see Section 4.1.1) contains a list of HostMatch objects (see Section 4.1.2) that contain Hostnames (and/or IP addresses) for which content requests may 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, 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 (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 CDNI metadata within 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 properties for each host as well as PathMetadata objects (see Section 4.1.6), via PathMatch objects (see Section 4.1.4), which may override those properties for given URI paths within the host. The CDNI metadata contained in HostMetadata objects is applied to content requests for which there is not more specific metadata, i.e. for content requests that do not match any of the PathMatch objects contained by that HostMetadata object and its child PathMetadata objects.
HostMetadata can also contain PathMatch objects. PathMatch objects define patterns, contained inside PatternMatch objects (see Section 4.1.5), to match against the requested URI path, and contain PathMetadata objects which contain the GenericMetadata objects to be applied when a content request matches against the defined URI path pattern. PatternMatch objects contain the pattern strings and flags that describe the URI path that a PathMatch applies to.

PathMetadata objects override the CDNI metadata in the HostMetadata object or one or more parent PathMetadata objects with more specific CDNI metadata that applies to content requests matching the URI pattern defined in the PatternMatch object of that 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.

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 may describe the source from which a CDN may acquire a piece of content. The GenericMetadata object is an atomic unit that may be referenced by HostMetadata and/or PathMetadata objects.

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. 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 CDNI metadata objects for those more specific URI paths. The PathMetadata object for "example.com/movies/**" describes CDNI metadata which apply to that URI path and could 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.
Internet-Draft        CDN Interconnection Metadata          October 2015

<table>
<thead>
<tr>
<th>Data Object</th>
<th>Objects it contains or references</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostIndex</td>
<td>0 or more HostMatch objects.</td>
</tr>
<tr>
<td>HostMatch</td>
<td>1 HostMetadata object.</td>
</tr>
<tr>
<td>HostMetadata</td>
<td>0 or more PathMatch objects. 0 or more GenericMetadata objects.</td>
</tr>
<tr>
<td>PathMatch</td>
<td>1 PatternMatch object. 1 PathMetadata object.</td>
</tr>
<tr>
<td>PatternMatch</td>
<td>Does not contain or reference any other objects.</td>
</tr>
<tr>
<td>PathMetadata</td>
<td>0 or more PathMatch objects. 0 or more GenericMetadata objects.</td>
</tr>
</tbody>
</table>

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 (e.g., property semantics, enforcement options, etc.).

The GenericMetadata object defines the type of properties contained within it as well as whether or not the properties are "mandatory-to-enforce". If the dCDN does not understand or support the property type and the property type is "mandatory-to-enforce", the dCDN MUST NOT serve the content to the User Agent. If the dCDN does not understand or support the property type and the property type is not "mandatory-to-enforce", then that GenericMetadata object may be safely ignored and the dCDN MUST process the content request in accordance with the rest of the CDNI metadata.

Although a CDN MUST NOT serve content to a User Agent if a "mandatory-to-enforce" property cannot be enforced, it may be "safe-to-redistribute" that metadata to another CDN without modification. For example, in the cascaded CDN case, a transit CDN (tCDN) may 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 natively 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 may be safely, though possibly not optimally, redistributed unmodified. For example, source acquisition address may not be optimal if transparently redistributed, but might still work.

Redistribution safety MUST be specified for each GenericMetadata. If a CDN does not understand or support a given GenericMetadata property type and the property type is not "safe-to-redistribute", before redistributing the metadata, the CDN MUST set the "incomprehensible" flag for the GenericMetadata object that it did not understand and was marked as not "safe-to-redistribute". The "incomprehensible" flag signals to a dCDN that the metadata was not properly transformed by the transit CDN. A CDN MUST NOT attempt to use metadata that has been marked as "incomprehensible" by a uCDN.

Transit CDNs MUST NOT change the value of "mandatory-to-enforce" or "safe-to-redistribute" when propagating metadata to a dCDN. Although a transit CDN may set the value of "incomprehensible" to true, a transit CDN MUST NOT change the value of "incomprehensible" from true to false.

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:
<table>
<thead>
<tr>
<th>MtE</th>
<th>StR</th>
<th>Metadata Understood by tCDN</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>Can serve and redistribute.</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>Can serve and redistribute.</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>Can serve. MUST set &quot;incomprehensible&quot; to True when redistributing.</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>Can serve. Can redistribute either by transforming not StR metadata (if the CDN knows how to do so safely), otherwise MUST set &quot;incomprehensible&quot; to True when redistributing.</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>Can serve and redistribute.</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>MUST NOT serve but can redistribute.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>Can serve. Can redistribute either by transforming not StR metadata (if the CDN knows how to do so safely), otherwise MUST set &quot;incomprehensible&quot; to True when redistributing.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>MUST NOT serve. MUST set &quot;incomprehensible&quot; to True when redistributing.</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>MtE</th>
<th>Incomp</th>
<th>Meta Data</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>Can serve.</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>Can serve.</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>Can serve.</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>MUST NOT serve.</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>MUST NOT serve.</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>MUST NOT serve.</td>
</tr>
</tbody>
</table>

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 may contain multiple PathMetadata objects (via PathMatch objects). Each PathMetadata object may 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:

- 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
- 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 a list 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 the inheritance tree and identify which specific 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 sub-objects (i.e., GenericMetadata sub-objects contained within the GenericMetadata object that refers to that GenericMetadata sub-object). 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 may 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 the metadata object, though that may restrict the content that a given dCDN can serve. uCDNs as generators of CDNI metadata only need to support generating the CDNI metadata that they need in order to express the policies and treatment required by the content they are describing.

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. See Section 6.4 for more details of 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" by this document 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 a list of HostMatch objects. An incoming content request is checked against the Hostname (or IP address) specified by each of the listed HostMatch objects to find the HostMatch object which applies to the request.

Property: hosts

Description: List of HostMatch objects. 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: List 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:

```json
{
    "hosts": [
        {
            <Properties of embedded HostMatch object>
        },
        {
            "type": "MI.HostMatch.v1",
            "href": "http://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: String (Hostname or IP address) to match against the requested host. 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 when converted to lowercase in the content request MUST be identical to the value of the host property when converted to lowercase. IPv4 addresses MUST be 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].

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.

Example HostMatch object with an embedded HostMetadata object:

```json
{
  "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:
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: List of host related metadata.
Type: List of GenericMetadata objects
Mandatory-to-Specify: Yes.

Property: paths
Description: Path specific rules. Path patterns (PathMatch objects) MUST be evaluated in the order they appear and the first PathMatch object that matches the content request being processed MUST be used.
Type: List 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 a single embedded PathMatch object that will describe the CDNI metadata for that path which overrides the default metadata for the host:
4.1.4.  PathMatch

A PathMatch object contains a pattern within a PatternMatch object to match against a resource’s URI path and contains a PathMetadata object to apply if the resource’s URI path matches the pattern within the PatternMatch object.

Property: path-pattern

Description: Pattern to match against the requested resource’s URI 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 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):

```json
{
   "path-pattern": {
      "pattern": "/movies/\*",
      "case-sensitive": true
   },
   "path-metadata": {
      "type": "MI.PathMetadata.v1",
      "href": "http://metadata.ucdn.example/host1234/pathDCE"
   }
}
```

4.1.5. PatternMatch

A PatternMatch object contains the pattern string and flags that describe the PathMatch expression.

**Property: pattern**

Description: A pattern for string matching. The pattern may contain the wildcards * and ?, where * matches any sequence of characters (including the empty string) and ? matches exactly one character. The three literals $, * and ? should be escaped as $$, $* and $?.$ All other characters are treated as literals.

Type: String

Mandatory-to-Specify: Yes.

**Property: case-sensitive**

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

Type: Boolean

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

**Property: ignore-query-string**

Description: List of query parameters which should be ignored when searching for a pattern match. Matching against query parameters to ignore MUST be case-insensitive. If all query parameters should be ignored then the list MUST be empty.
Type: List of String

Mandatory-to-Specify: No. Default is to include query strings when matching.

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,
    "ignore-query-string": []
}
```

Example PatternMatch object that matches the case-sensitive URI path pattern "/movies/**". The query parameter "sessionid" will be ignored when matching URIs requested from surrogates by content clients against this path pattern:

```
{
    "pattern": "/movies/**",
    "case-sensitive": true,
    "ignore-query-string": ["sessionid"]
}
```

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) and possibly child PathMatch objects.

Note that if DNS-based redirection is employed, then a dCDN will be unable to evaluate any metadata at the PathMetadata level or below against the content redirection request at request routing time because only the hostname of the content request is available at request routing time. dCDNs SHOULD still process any metadata at the PathMetadata level or below before responding to the redirection request in order to detect if any unsupported metadata is specified. If any metadata is included and marked as "mandatory-to-enforce" which is not supported by the dCDN then the dCDN SHOULD NOT redirect the the content redirection request to itself in order to avoid receiving content requests that it is not able to satisfy/serve.

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.

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.

```
{
    "metadata": [
        {  
            <Properties of 1st embedded GenericMetadata object>
        },
        {  
            <Properties of 2nd embedded GenericMetadata object>
        },
        ...
        {  
            <Properties of Nth embedded GenericMetadata 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 of the object contained in the generic-metadata-value property.

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.

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 may 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):
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: List 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.v1"  "generic-metadata-value":  {  "sources": [  {  "endpoints": [  "a.service123.ucdn.example",  "b.service123.ucdn.example"  ],  "protocol": "http1.1"  },  {  "endpoints": ["origin.service123.example"],  "protocol": "http1.1"  }  ]  }  }

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 so a uCDN can specify a list of sources in preference order and for each source/preference rank a uCDN can specify a list 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, for example a pool of servers behind a load balancer.

Type: List 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.service123.ucdn.example",
    "b.service123.ucdn.example"
  ],
  "protocol": "http1.1"
}
```

4.2.2. 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, meaning that delivery can be performed regardless of the User Agent’s location. The action from the first footprint to match against the User Agent’s location is the action a CDN MUST take. If two or more footprints overlap, the first footprint that matches against the User Agent’s location determines the action a CDN MUST take. If the locations property is included but is empty, or if none of the listed footprints matches the User Agent’s location, then the result is an action of deny.

Although the LocationACL, TimeWindowACL (see Section 4.2.3), and ProtocolACL (see Section 4.2.4) are independent GenericMetadata objects, they may provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the LocationACL and denied based on the TimeWindowACL. The dCDN MUST use
the logical AND of all ACLs (where 'allow' is true and 'deny' is false) to determine whether or not a request should be allowed.

Property: locations

Description: Access control list which allows or denies (blocks) delivery based on the User Agent’s location.

Type: List 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.v1"
"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.v1"
"generic-metadata-value":
{
"locations": [
{
"action": "allow",
"footprints": [
{
"footprint-type": "countrycode",
"footprint-value": ["us"]
}
]
}
]
}
```
4.2.2.1. LocationRule

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

Property: footprints

Description: List of footprints to which the rule applies.

Type: List of Footprint objects (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:

```json
{
   "action": "allow",
   "footprints": [
   {
      "footprint-type": "countrycode",
      "footprint-value": ["us"]
   }
   ]
}
```

4.2.2.2. Footprint

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

Property: footprint-type

Description: Registered footprint type. 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: List of footprint values conforming to the specification associated with the registered footprint type. Footprint values may be simple strings (e.g., IPv4CIDR, IPv5CIDR, ASN, and CountryCode), however, other Footprint objects may be defined in the future, along with a more complex encoding (e.g., GPS coordinate tuples).

Type: List of footprints

Mandatory-to-Specify: Yes.

Example Footprint object describing a footprint covering the USA:

```json
{
    "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:

```json
{
    "footprint-type": "ipv4cidr",
    "footprint-value": ["192.0.2.0/24", "198.51.100.0/24"]
}
```

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, meaning that delivery can be performed regardless of the time of the User Agent’s request. The action from the first window to match against the current time is the action a CDN MUST take. If two or more windows overlap, the first window that matches against the current time determines the action a CDN MUST take. If the times property is included but is empty, or if none of the listed windows matches the current time, then the result is an action of deny.
Although the LocationACL, TimeWindowACL, and ProtocolACL are independent GenericMetadata objects, they may provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the LocationACL and denied based on the TimeWindowACL. The dCDN MUST use the logical AND of all ACLs (where ‘allow’ is true and ‘deny’ is false) to determine whether or not a request should be allowed.

Property: times

Description: Access control list which allows or denies (blocks) delivery based on the time of a User Agent’s request.

Type: List 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 09:00AM 01/01/2000 UTC and 17:00AM 01/01/2000 UTC:

```json
{
  "generic-metadata-type": "MI.TimeWindowACL.v1",
  "generic-metadata-value": {
    "times": [
      {
        "action": "allow",
        "windows": [
          {
            "start": 946717200,
            "end": 946746000
          }
        ]
      }
    ]
  }
}
```

4.2.3.1. TimeWindowRule

A TimeWindowRule contains or references a list of TimeWindow objects and the corresponding action.

Property: windows

Description: List of time windows to which the rule applies.
Type: List of TimeWindow objects (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 09:00AM 01/01/2000 UTC and 17:00AM 01/01/2000 UTC:

```json
{
    "action": "allow",
    "windows": [
        {
            "start": 946717200,
            "end": 946746000
        }
    ]
}
```

4.2.3.2. TimeWindow  

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

Property: start  
Description: The start time of the window.  
Type: Time (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 09:00AM 01/01/2000 UTC to 17:00AM 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, meaning that delivery can be performed regardless of the protocol of the User Agent’s request. The action from the first protocol to match against the request protocol is the action a CDN MUST take. If two or more request protocols overlap, the first protocol that matches the request protocol determines the action a CDN MUST take. If the protocol-acl property is included but is empty, or if none of the listed protocol matches the request protocol, then the result is an action of deny.

Although the LocationACL, TimeWindowACL, and ProtocolACL are independent GenericMetadata objects, they may provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the 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: Access control list which allows or denies (blocks) delivery based on delivery protocol.
- Type: List 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:
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 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 includes the protocol HTTP/1.1:

```json
{
    "action": "allow",
    "protocols": ["http1.1"]
}
```
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: List 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.v1"
    "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.

Property: ignore-query-string

Description: Allows a Surrogate to ignore URI query string parameters when comparing the requested URI against the URIs in its cache for equivalence. Matching against query parameters to ignore MUST be case-insensitive. Each query parameter to ignore is specified in the list. If all query parameters
should be ignored, then the list MUST be specified and MUST be empty.

Type: List of String

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

Example Cache object that instructs the dCDN to ignore all query parameters:

```
{
  "generic-metadata-type": "MI.Cache.v1"
  "generic-metadata-value": {
    "ignore-query-string": []
  }
}
```

Example Cache object that instructs the dCDN to ignore the (case-insensitive) query parameters named "sessionid" and "random":

```
{
  "generic-metadata-type": "MI.Cache.v1"
  "generic-metadata-value": {
    "ignore-query-string": ["sessionid", "random"]
  }
}
```

4.2.7. Auth

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

Property: auth-type

Description: Registered Auth type (Section 7.4).

Type: String

Mandatory-to-Specify: Yes.

Property: auth-value
Description: An object conforming to the specification associated with the Registered Auth type.

Type: GenericMetadata Object

Mandatory-to-Specify: Yes.

Example Auth object:

```json
{
    "generic-metadata-type": "MI.Auth.v1",
    "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 large group of content to which a given asset belongs.

Property: ccid

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

Type: String

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

Example Grouping object that specifies a Content Collection Identifier for the content associated with the Grouping object’s parent HostMetadata or PathMetadata:

```json
{
    "generic-metadata-type": "MI.Grouping.v1",
    "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 may 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 instead of another metadata 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 type of the object being referenced.

Type: String

Mandatory-to-Specify: No

Example Link object referencing a HostMetadata object:

```json
{
    "type": "MI.HostMetadata.v1",
    "href": "http://metadata.ucdn.example/host1234"
}
```
4.3.2. Protocol

Protocol objects are used to specify registered protocols for content acquisition or delivery (see Section 7.3).

Type: String

Example:
"http1.1"

4.3.3. Endpoint

A Hostname (with optional port) or an IP address (with optional port).

Note: All implementations MUST support IPv4 addresses encoded as specified by the ‘IPv4address’ rule in Section 3.2.2 of [RFC3986]. 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].

Type: String

Example Hostname:
"http://metadata.ucdn.example/host1234"

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 Unix epoch in the UTC timezone.

Type: Integer

Example Time representing 09:00AM 01/01/2000 UTC:
946717200
4.3.5. IPv4CIDR

An IPv4 address 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 IPv6 address 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.

Type: String

Example ASN:

"as64496"

4.3.8. CountryCode


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 may 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 may be assigned optional values, it may be useful for the uCDN to know which values a dCDN supports, prior to delegating any content requests to that dCDN. If the optional value assigned to a given piece of content’s metadata is not supported by the dCDN, any delegated requests for that content may fail, so again the uCDN is likely to want to avoid delegating those requests to that dCDN.

The CDNI Footprint and Capabilities Interface (FCI) [RFC7336] provides a means of advertising capabilities from dCDN to uCDN. Support for optional metadata and support for optional metadata values may be advertised using the FCI.

6. CDNI Metadata interface

This section specifies an interface to enable a dCDN to retrieve CDNI metadata objects from a uCDN.

The interface can be used by a dCDN to retrieve CDNI metadata objects either:

- 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;

- In advance of being required. For example in the case of pre-positioned CDNI metadata acquisition.

The CDNI metadata interface is built on the principles of 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 a list of Hostnames for which the uCDN may 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.

The HTTP Method in the request defines the operation the request would like to perform. A server implementation of the CDNI metadata interface MUST support the HTTP GET and HEAD methods.

The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses from servers implementing the CDNI metadata interface that contain a response body SHOULD include an ETag to enable validation of cached versions of returned resources.

The CDNI metadata interface specified in this document is a read-only interface. Therefore support for other HTTP methods such as PUT, POST and DELETE etc. is not specified. A server implementation of the CDNI metadata interface SHOULD reject all methods other than GET and HEAD.

As the CDNI metadata interface builds on top of HTTP, CDNI metadata server implementations MAY make use of any HTTP feature when implementing the CDNI metadata interface, for example a CDNI metadata server MAY make use of HTTP’s caching mechanisms to indicate that the returned response/representation can be reused without re-contacting the CDNI metadata server.
6.2. Retrieval of CDNI Metadata resources

In the general case a CDNI metadata server makes CDNI metadata objects available via unique URIs 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 uCDN may 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 the PathMatch). 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 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 a redirection request from an uCDN or a 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, for example because the uCDN is uncontactable or returns an 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 unless the dCDN has stale versions of all the required metadata and the stale-if-error Cache-Control extension [RFC5861] was included in all previous responses that are required but cannot currently be retrieved. The dCDN can continue to serve other content for which it can retrieve (or for which it has fresh responses cached) all the required metadata even if some non-applicable part of the metadata tree is missing.

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 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 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 may 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 either configured in, or discovered by, the dCDN. All other objects/resources are then discoverable from the HostIndex object by following any links in the HostIndex object and the referenced HostMetadata and PathMetadata objects and their GenericMetadata sub-objects.

If the URI for the HostIndex object is not manually configured in the dCDN then the HostIndex URI could be discovered. A mechanism allowing the dCDN to discover the URI of the HostIndex 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 represented in lowercase.
6.5. Extensibility

The set of GenericMetadata objects may 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 lists.

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 SHOULD:

1. Specify the CDNI Payload Type used to identify the new GenericMetadata type being specified.

2. Define the set of properties associated with the new type contained within the GenericMetadata object. 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.

3. For each property, 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.

Note: In the case of vendor specific extensions, identification within the type name defined for a GenericMetadata object, of the organization that defined the new GenericMetadata object decreases the possibility of GenericMetadata type collisions.

6.6. Metadata Enforcement

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

In the cases where a uCDN sends metadata containing a GenericMetadata type that a dCDN does not support, the dCDN MUST enforce the semantics of the "mandatory-to-enforce" property. If a dCDN does not understand or is unable to perform the functions associated with any "mandatory-to-enforce" metadata, the dCDN MUST NOT service any requests for the corresponding content.
Note: Ideally, uCDNs would not delegate content requests to a dCDN which does not support the "mandatory-to-enforce" metadata associated with the content being requested. However, even if the uCDN has a priori knowledge of the metadata supported by the dCDN (e.g., via the CDNI capabilities interface or through out-of-band negotiation between CDN operators) metadata support may fluctuate or be inconsistent (e.g., due to mis-communication, mis-configuration, or temporary outage). Thus, the dCDN MUST always evaluate all metadata associated with redirection requests 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 may obsolete or conflict with existing GenericMetadata (e.g., a future revision of the CDNI metadata interface may redefine the Auth GenericMetadata object or a custom vendor extension may 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 (e.g., MI.Auth.v1) and all are marked as "mandatory-to-enforce", it may be ambiguous which metadata should be applied, especially if the functionality of the metadata overlap.

As described in Section 3.3, metadata override only applies to metadata objects of the same exact type, found in HostMetadata and nested PathMetadata structures. The CDNI metadata interface does not support enforcement of dependencies between different metadata types. It is the responsibility of the CSP and the CDN operators to ensure that metadata assigned to a given content do not conflict.

Note: Because metadata is inherently ordered in GenericMetadata lists, as well as in the PathMetadata hierarchy and PathMatch lists, multiple conflicting metadata types MAY be used, however, metadata hierarchies MUST ensure that independent PathMatch root objects are used to prevent ambiguous or conflicting metadata definitions.

6.8. Versioning

The version of CDNI metadata objects is conveyed inside the CDNI Payload Type that is included in the HTTP Content-Type header. Upon responding to a request for an object, a CDNI metadata server MUST include a Content-Type header with the CDNI Payload Type containing the version number of the object. HTTP requests sent to a metadata server SHOULD include an Accept header with the CDNI Payload Type (which includes the version) 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 may decide to include both in the Accept header. The version of each object defined by this document is version 1. For example: "Content-Type: application/cdni; ptype=MI.HostIndex.v1".

GenericMetadata objects include a "type" property which specifies the CDNI Payload Type of the GenericMetadata value. This CDNI Payload Type should also include a version. Any document which defines a new GenericMetadata type MUST specify the version number which it describes. For example: "MI.Location.v1".

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) that are specified in this document.

<table>
<thead>
<tr>
<th>Data Object</th>
<th>CDNI Payload Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostIndex</td>
<td>MI.HostIndex.v1</td>
</tr>
<tr>
<td>HostMatch</td>
<td>MI.HostMatch.v1</td>
</tr>
<tr>
<td>HostMetadata</td>
<td>MI.HostMetadata.v1</td>
</tr>
<tr>
<td>PathMatch</td>
<td>MI.PathMatch.v1</td>
</tr>
<tr>
<td>PatternMatch</td>
<td>MI.PatternMatch.v1</td>
</tr>
<tr>
<td>PathMetadata</td>
<td>MI.PathMetadata.v1</td>
</tr>
<tr>
<td>GenericMetadata</td>
<td>MI.SourceMetadata.v1</td>
</tr>
<tr>
<td>Source</td>
<td>MI.Source.v1</td>
</tr>
<tr>
<td>LocationACL</td>
<td>MI.LocationACL.v1</td>
</tr>
<tr>
<td>LocationRule</td>
<td>MI.LocationRule.v1</td>
</tr>
<tr>
<td>Footprint</td>
<td>MI.Footprint.v1</td>
</tr>
<tr>
<td>TimeWindowACL</td>
<td>MI.TimeWindowACL.v1</td>
</tr>
<tr>
<td>TimeWindowRule</td>
<td>MI.TimeWindowRule.v1</td>
</tr>
<tr>
<td>TimeWindow</td>
<td>MI.TimeWindow.v1</td>
</tr>
<tr>
<td>ProtocolACL</td>
<td>MI.ProtocolACL.v1</td>
</tr>
<tr>
<td>ProtocolRule</td>
<td>MI.ProtocolRule.v1</td>
</tr>
<tr>
<td>DeliveryAuthorization</td>
<td>MI.DeliveryAuthorization.v1</td>
</tr>
<tr>
<td>Cache</td>
<td>MI.Cache.v1</td>
</tr>
<tr>
<td>Auth</td>
<td>MI.Auth.v1</td>
</tr>
<tr>
<td>Grouping</td>
<td>MI.Grouping.v1</td>
</tr>
</tbody>
</table>

Table 4: CDNI Payload Types for CDNI Metadata objects
6.10. Complete CDNI Metadata Example

A dCDN may request the HostIndex and receive the following object with a CDNI payload type of "MI.HostIndex.v1":

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

If the incoming request has a Host header with "video.example.com" then the dCDN would fetch the next metadata object from "http://metadata.ucdn.example/host1234" expecting a CDNI payload type of "MI.HostMetadata.v1":

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.SourceMetadata.v1",
      "generic-metadata-value": {
        "sources": [
          {
            "endpoint": "acq1.ucdn.example",
            "protocol": "http1.1"
          },
          {
            "endpoint": "acq2.ucdn.example",
            "protocol": "http1.1"
          }
        ]
      }
    },
    {
    }
  ]
}
```
"generic-metadata-type": "MI.LocationACL.v1",
"generic-metadata-value": {
  "locations": [
    {
      "footprints": [
        {
          "footprint-type": "IPv4CIDR",
          "footprint-value": "192.0.2.0/24"
        }
      ],
      "action": "deny"  
    }
  ]
},

"generic-metadata-type": "MI.ProtocolACL.v1",
"generic-metadata-value": {
  "protocol-acl": [
    {
      "protocols": [
        "http1.1"
      ],
      "action": "allow"
    }
  ]
},

"paths": [
  {
    "path-pattern": {
      "pattern": "*/video/trailers/**"
    },
    "path-metadata": {
      "type": "MI.PathMetadata.v1",
      "href": "http://metadata.ucdn.example/host1234/pathABC"
    }
  },
  {
    "path-pattern": {
      "pattern": "*/video/movies/**"
    },
    "path-metadata": {
      "type": "MI.PathMetadata.v1",
      "href": "http://metadata.ucdn.example/host1234/pathDCE"
    }
  }
]
Suppose the path of the requested resource matches the "/video/movies/*" pattern, the next metadata requested would be for "http://metadata.ucdn.example/host1234/movies" with an expected CDNI payload type of "MI.PathMetadata.v1":

```json
{
    "metadata": [],
    "paths": [
        {
            "path-pattern": {
                "pattern": "videos/movies/hd/*"
            },
            "path-metadata": {
                "type": "MI.PathMetadata.v1",
                "href": "http://metadata.ucdn.example/host1234/pathABC/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 "http://metadata.ucdn.example/host1234/movies/hd" with CDNI payload type "MI.PathMetadata.v1":

```json
```
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.v1 | RFCthis
MI.HostMatch.v1 | RFCthis
MI.HostMetadata.v1 | RFCthis
MI.PathMatch.v1 | RFCthis
MI.PatternMatch.v1 | RFCthis
MI.PathMetadata.v1 | RFCthis
MI.SourceMetadata.v1 | RFCthis
MI.Source.v1 | RFCthis
MI.LocationACL.v1 | RFCthis
MI.LocationRule.v1 | RFCthis
MI.Footprint.v1 | RFCthis
MI.TimeWindowACL.v1 | RFCthis
MI.TimeWindowRule.v1 | RFCthis
MI.TimeWindow.v1 | RFCthis
MI.ProtocolACL.v1 | RFCthis
MI.ProtocolRule.v1 | RFCthis
MI.DeliveryAuthorization.v1 | RFCthis
MI.Cache.v1 | RFCthis
MI.Auth.v1 | RFCthis
MI.Grouping.v1 | 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" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" category. 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.

The following table defines the initial Footprint Registry values:
### Footprint Type

<table>
<thead>
<tr>
<th>Footprint Type</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipv4cidr</td>
<td>IPv4 CIDR address block</td>
<td>RFCthis</td>
</tr>
<tr>
<td>ipv6cidr</td>
<td>IPv6 CIDR address block</td>
<td>RFCthis</td>
</tr>
<tr>
<td>asn</td>
<td>Autonomous System (AS) Number</td>
<td>RFCthis</td>
</tr>
<tr>
<td>countrycode</td>
<td>ISO 3166-1 alpha-2 code</td>
<td>RFCthis</td>
</tr>
</tbody>
</table>

[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" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" category. 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:

<table>
<thead>
<tr>
<th>Protocol Type</th>
<th>Description</th>
<th>Type Specification</th>
<th>Protocol Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>http1.1</td>
<td>Hypertext Transfer Protocol -- HTTP/1.1</td>
<td>RFCthis</td>
<td>RFC7230</td>
</tr>
<tr>
<td>https1.1</td>
<td>HTTP/1.1 Over TLS</td>
<td>RFCthis</td>
<td>RFC2818</td>
</tr>
</tbody>
</table>

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

#### 7.4. CDNI Metadata Auth Types Registry

The IANA is requested to create a new "CDNI Metadata Auth Types" registry in the "Content Delivery Networks Interconnection (CDNI) Parameters" category. The "CDNI Metadata Auth Type" namespace defines the valid Auth object types used by the Auth object in Section 4.2.7. Additions to the Auth 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.

The registry will initially be unpopulated:

+-----------+-------------+---------------+
| Auth Type | Description | Specification |
+-----------+-------------+---------------+

8. Security Considerations

8.1. Authentication

Unauthorized access to metadata could result in denial of service. A malicious metadata server, proxy server or an attacker performing a "man in the middle" attack could provide malicious metadata to a dCDN that either:

- Denies service for one or more pieces of content to one or more User Agents; or
- Directs dCDNs to contact malicious origin servers instead of the actual origin servers.

Unauthorized access to metadata could also enable a malicious metadata client to continuously issue large metadata requests in order to overload a uCDN’s metadata server(s).

Unauthorized access to metadata could result in leakage of private information. A malicious metadata client could request metadata in order to gain access to origin servers, as well as information pertaining to content restrictions.

An implementation of the CDNI metadata interface SHOULD use mutual authentication to prevent unauthorized access to metadata.

8.2. Confidentiality

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

An implementation of the CDNI metadata interface SHOULD use strong encryption to prevent unauthorized interception of metadata.
8.3.  Integrity

Unauthorized modification of metadata could result in denial of service. A malicious metadata server, proxy server or an attacker performing a "man in the middle" attack could modify metadata destined to a dCDN in order to deny service for one or more pieces of content to one or more user agents. A malicious metadata server, proxy server or an attacker performing a "Man in the middle" attack could modify metadata so that dCDNs are directed to contact to malicious origin servers instead of the actual origin servers.

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

8.4.  Privacy

Content provider origin and policy information is conveyed through the CDNI metadata interface. The distribution of this information to another CDN may introduce potential privacy concerns for some content providers, for example because dCDNs accepting content requests for a content provider’s content may be able to obtain additional information & usage patterns relating to the users of a content provider’s services. Content providers with such concerns can instruct their CDN partners not to use CDN interconnects when delivering that content provider’s content.

8.5.  Securing the CDNI Metadata interface

An implementation of the CDNI metadata interface MUST support TLS transport as per [RFC2818] and [RFC7230]. The use of TLS for transport of the CDNI metadata interface messages allows:

- The dCDN and uCDN to authenticate each other.

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

- The dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI metadata requests & responses from an authorized CDN).

- CDNI metadata interface requests and responses to be transmitted with confidentiality.

- The integrity of the CDNI metadata interface requests and responses 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 metadata interface unless alternate methods are used for ensuring the confidentiality 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).

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]

"https://www.iso.org/obp/ui/#search".
11.2. Informative References


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
Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 11, 2016.

Copyright Notice

Copyright (c) 2015 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.
1. Introduction

A Content Delivery Network (CDN) is a system built on an existing IP network which is used for large scale content delivery, via prefetching or dynamically caching content on its distributed surrogates (caching servers). [RFC6707] describes the problem area of interconnecting CDNs.
The CDNI Request Routing interface outlined in [RFC7336] comprises of:

1. The asynchronous advertisement of footprint and capabilities by a downstream CDN (dCDN) that allows an upstream CDN (uCDN) to decide whether to redirect particular user requests to that dCDN.

2. The synchronous operation of a uCDN requesting whether a dCDN is prepared to accept a user request and of a dCDN responding with how to actually redirect the user request.

This document describes an interface for the latter part, i.e. the CDNI Request Routing Redirection interface (RI).

2. Terminology

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

This document reuses the terminology defined in [RFC6707].

The following additional terms are introduced by this document:

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

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

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

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

3. Interface function and operation overview

The main function of the CDNI Redirection interface (RI) is to allow the request routing systems in interconnected CDNs to communicate to facilitate the redirection of User Agent requests between interconnected CDNs.

The detailed requirements for the Redirection Interface and their relative priorities are described in section 5 of [RFC7337].

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

- The final Surrogate, which may be in the dCDN that returned the RI response to the uCDN, or another CDN (if the dCDN delegates the delivery to another CDN).

- A request router (in the dCDN or another CDN), which may use a different redirection protocol (DNS or HTTP) than the one included in the RI request.

The Redirection interface operates between the request routing systems of a pair of interconnected CDNs. To enable communication over the Redirection Interface, the uCDN needs to know the URI (end point) in the dCDN to send CDNI request routing queries.

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

The Redirection Interface is only relevant in the case of Recursive Request Redirection, as Iterative Request Redirection does not invoke any interaction over the Redirection Interface between interconnected CDNs. Therefore the scope of this document is limited to Recursive Request Redirection.
In the case of Recursive Request Redirection, in order to perform redirection of a request received from a User Agent, the uCDN queries the dCDN so that the dCDN can select and provide a Redirection Target. In cases where a uCDN has a choice of dCDNs it is up to the uCDN to decide (for example via configured policies) which dCDN(s) to query and in which order to query them. A number of strategies are possible including selecting a preferred dCDN based on local policy, possibly falling back to querying an alternative dCDN(s) if the first dCDN does not return a Redirection Target or otherwise rejects the uCDN's RI request. A more complex strategy could be to query multiple dCDNs in parallel before selecting one and using the Redirection Target provided by that dCDN.

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

This document also defines an RI loop prevention and detection mechanism as part of the Redirection Interface.

3.1. Redirection of encrypted traffic

The Redirection Interface defined in this document might be used to redirect a request where the User Agent will subsequently attempt to establish a TLS session with the Redirection Target. In such a case, any surrogate or request router to which the User Agent is redirected needs to be able to successfully complete the TLS handshake and to perform encryption of the TLS channel. Mechanisms to distribute the required information and/or configuration, such as private keys, to surrogates and request routers in dCDNs are outside the scope of this document.

4. HTTP based interface for the Redirection Interface

This document defines a simple interface for the Redirection Interface based on HTTP 1.1 [RFC7230], where the attributes of a User Agent’s requests are encapsulated along with any other data that can aid the dCDN in processing the requests. The RI response encapsulates the attributes of the RT(s) that the uCDN should return to the User Agent (if it decides to utilize the dCDN for delivery) along with the policy for how the response can be reused. The examples of RI requests and responses below do not contain a complete set of HTTP headers for brevity; only the pertinent HTTP headers are shown.
The same HTTP interface is used for both DNS and HTTP redirection of User Agent requests, although the contents of the RI requests/responses contain data specific to either DNS or HTTP redirection.

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

In addition, having a single RI where the attributes of the User Agent’s DNS or HTTP request are encapsulated along with the other data required for the dCDN 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:

1. The User Agent sends its (DNS or HTTP) request to CDN A. The Request Routing System of CDN A processes the request and,
through local policy, recognizes that the request is best served by another CDN, specifically CDN B (or that CDN B may be one of a number of candidate dCDNs it could use).

2. The Request Routing System of CDN A sends an HTTP POST to CDN B’s RI URI containing the attributes of the User Agent’s request.

3. The Request Routing System of CDN B processes the RI request and assuming the request is well formed, responds with an HTTP "200" response with a message body containing the RT(s) to return to the User Agent as well as parameters that indicate the properties of the response (cacheability and scope).

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

4.1. Information passed in RI requests & responses

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

1. The attributes of the User Agent’s request to the uCDN.

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

To assist the routing decision of a dCDN, the uCDN SHOULD convey as much information as possible to the dCDN, for example the URI of the requested content and the User Agent’s IP address or subnet, when those are known by the uCDN Request Routing system.

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

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

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

In addition to details of how to redirect the User Agent, the dCDN may wish to return additional policy information to the uCDN to it with future RI requests. For example the dCDN may wish to return a policy that expresses "this response can be reused without requiring an RI request for 60 seconds provided the User Agent’s IP address is in the range 198.51.100.0 - 198.51.100.255".

These additional policies split into two basic categories:

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

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

4.2. JSON encoding of RI requests & responses

The body of RI requests and responses is a JSON object [RFC7159] that MUST conform to [RFC7493] containing a dictionary of key:value pairs.

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

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

The following top level keys are defined along with whether they are applicable to RI requests, RI responses or both:
<table>
<thead>
<tr>
<th>Key</th>
<th>Request/Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dns</td>
<td>Both</td>
<td>The attributes of the UA’s DNS request or the attributes of the RT(s) to return in a DNS response.</td>
</tr>
<tr>
<td>http</td>
<td>Both</td>
<td>The attributes of the UA’s HTTP request or the attributes of the RT to return in a HTTP response.</td>
</tr>
<tr>
<td>scope</td>
<td>Response</td>
<td>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.</td>
</tr>
<tr>
<td>error</td>
<td>Response</td>
<td>Additional details if the response is an error response.</td>
</tr>
<tr>
<td>cdn-path</td>
<td>Both</td>
<td>A List of Strings. Contains a list of the CDN Provider IDs of previous CDNs that have participated in the request routing for the associated User Agent request. On RI requests it contains the list of previous CDNs that this RI request has passed through. On RI responses it contains the list of CDNs that were involved in obtaining the final redirection included in the RI response.</td>
</tr>
<tr>
<td>max-hops</td>
<td>Request</td>
<td>Integer specifying the maximum number of hops (CDN Provider IDs) this request is allowed to be propagated along. This allows the uCDN to coarsely constrain the latency of the request routing chain.</td>
</tr>
</tbody>
</table>

Top-Level keys in RI requests/responses

A single request or response MUST contain only one of the dns or http keys. Requests MUST contain a cdn-path key and responses MAY contain a cdn-path key. If the max-hops key is not present then there is no limit on the number of CDN hops that the RI request can be propagated along. If the first uCDN does not wish the RI request to be propagated beyond the dCDN it is making the request to, then the uCDN MUST set max-hops to 1.
When cascading an RI request, a transit CDN MUST append its own CDN Provider ID to the list in cdn-path so that dCDNs can detect loops in the RI request chain. Transit CDNs MUST check the cdn-path and MUST NOT cascade the RI request to dCDNs that are already listed in cdn-path. Transit CDNs MUST NOT modify the cdn-path when cascading an RI request, except to append its own CDN Provider ID.

The cdn-path MAY be reflected back in RI responses, although doing so could expose information to the uCDN that a dCDN may not wish to expose (for example, the existence of business relationships between a dCDN and other CDNs).

If the cdn-path is reflected back in the RI response it MUST contain the value of cdn-path received in the associated RI request with the final dCDN’s CDN Provider ID appended. Transit CDNs MAY remove the cdn-path from RI responses but MUST NOT modify the cdn-path in other ways.

The presence of an error key within a response that also contains either a dns or http key does not automatically indicate that the RI request was unsuccessful as the error key MAY be used for communicating additional (e.g. debugging) information. When a response contains an error key as well as either a dns or http key, the error-code SHOULD be 1xx (e.g. 100). See Section 4.7 for more details of encoding error information in RI responses.

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

4.3. MIME Media Types used by the RI interface

RI requests MUST use a MIME Media Type of application/cdni as specified in [I-D.ietf-cdni-media-type], with the Payload Type (ptype) parameter set to ‘redirection-request’.

RI requests MUST use a MIME Media Type of application/cdni as specified in [I-D.ietf-cdni-media-type], with the Payload Type (ptype) parameter set to ‘redirection-response’.

4.4. DNS redirection

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

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

- The IP address of the DNS resolver that made the DNS request to the uCDN.
- The type of DNS query made (usually either A or AAAA).
- The class of DNS query made (usually IN).
- The fully qualified domain name for which DNS redirection is being requested.
- The IP address or prefix of the User Agent (if known to the uCDN).

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

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

An RI request for DNS-based redirection MUST include a dns dictionary. This dns dictionary MUST contain the following keys: resolver-ip, qtype, qclass, qname and the value of each MUST be the value of the appropriate part of the User Agent’s DNS query/request.
An example RI request (uCDN->dCDN) for DNS based redirection:

POST /dcdn/ri HTTP/1.1  
Host: rr1.dcdn.example.net  
Content-Type: application/cdni; ptype=redirection-request  
Accept: application/cdni; ptype=redirection-response

{
  "dns": {
    "resolver-ip": "192.0.2.1",
    "c-subnet": "198.51.100.0/24",
    "qtype": "A",
    "qclass": "IN",
    "qname": "www.example.com"
  },
  "cdn-path": ["AS64496:0"],
  "max-hops": 3
}

4.4.2. DNS Redirection responses

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

- The IP address(es) of (or the CNAME of) RTs that are dCDN surrogates (if the dCDN is performing DNS based redirection directly to a surrogate); or
- The IP address(es) of (or the CNAME of) RTs that are Request Routers (if the dCDN will perform request redirection itself). A dCDN MUST NOT return a RT which is a Request Router if the dns-only key is set to True in the RI request.

The information above is encoded as a set of key:value pairs within the dns dictionary as follows:
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcode</td>
<td>Integer</td>
<td>Yes</td>
<td>DNS response code (see [RFC6895]).</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Yes</td>
<td>The fully qualified domain name the response relates to.</td>
</tr>
<tr>
<td>a</td>
<td>List of String</td>
<td>No</td>
<td>Set of IPv4 Addresses of RT(s).</td>
</tr>
<tr>
<td>aaaa</td>
<td>List of String</td>
<td>No</td>
<td>Set of IPv6 Addresses of RT(s).</td>
</tr>
<tr>
<td>cname</td>
<td>List of String</td>
<td>No</td>
<td>Set of fully qualified domain names of RT(s).</td>
</tr>
<tr>
<td>ttl</td>
<td>Integer</td>
<td>No</td>
<td>TTL in seconds of DNS response. Default is 0.</td>
</tr>
</tbody>
</table>

A successful RI response for DNS-based redirection MUST include a dns dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for DNS-based redirection MUST include an error dictionary. If a dns dictionary is included in the RI response, it MUST include at least one of the following keys: a, aaaa, cname. The dns dictionary MAY include both ‘a’ and ‘aaaa’ keys. If the dns dictionary contains a cname key it MUST NOT contain either an a or aaaa key.

An example of a successful RI response (dCDN->uCDN) for DNS based redirection with both a and aaaa keys is listed below:

HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response

{
    "dns" : {
        "rcode" : 0,
        "name" : "www.example.com",
        "a" : ["203.0.113.200", "203.0.113.201", "203.0.113.202"],
        "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
        "ttl" : 60
    }
}

A further example of a successful RI response (dCDN->uCDN) for DNS based redirection is listed below, in this case with a cname key containing the FQDN of the RT.
HTTP/1.1 200 OK  
Date: Mon, 06 Aug 2012 18:41:38 GMT  
Content-Type: application/cdni; ptype=redirection-response

{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "cname" : ["rr1.dcdn.example"],
    "ttl" : 20
   }
}

4.5. HTTP Redirection

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

The dictionary keys used in HTTP Redirection requests and responses use the following conventions for their prefixes:

- **c-** is prefixed to keys for information related to the Client (User Agent).
- **cs-** is prefixed to keys for information passed by the Client (User Agent) to the Server (uCDN).
- **sc-** is prefixed to keys for information to be passed by the Server (uCDN) to the Client (User Agent).

4.5.1. HTTP Redirection requests

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

- The IP address of the User Agent.
- The URI requested by the User Agent.
- The HTTP method requested by the User Agent.
- The HTTP version number requested by the User Agent.

The uCDN may also decide to pass the presence and value of particular HTTP headers included in the User Agent request to the dCDN.
The information above is encoded as a set of key:value pairs within the `http` dictionary as follows:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c-ip</td>
<td>String</td>
<td>Yes</td>
<td>The IP address of the UA.</td>
</tr>
<tr>
<td>cs-uri</td>
<td>String</td>
<td>Yes</td>
<td>The Effective Request URI [RFC7230] requested by the UA.</td>
</tr>
<tr>
<td>cs-method</td>
<td>String</td>
<td>Yes</td>
<td>The method part of the request-line as defined in Section 3.1.1 of [RFC7230].</td>
</tr>
<tr>
<td>cs-version</td>
<td>String</td>
<td>Yes</td>
<td>The HTTP-version part of the request-line as defined in Section 3.1.1 of [RFC7230].</td>
</tr>
<tr>
<td>cs-(&lt;headername&gt;)</td>
<td>String</td>
<td>No</td>
<td>The field-value of the HTTP header field named &lt;HeaderName&gt; as a string, for example cs-(cookie) would contain the value of the HTTP Cookie header from the UA request.</td>
</tr>
</tbody>
</table>

An RI request for HTTP-based redirection MUST include an `http` dictionary. This `http` dictionary MUST contain the following keys: `c-ip`, `cs-method`, `cs-version` and `cs-uri` and the value of each MUST be the value of the appropriate part of the User Agent’s HTTP request.

The `http` dictionary of an RI request MUST contain a maximum of one `cs-(<headername>)` key for each unique header field-name (HTTP header field). `<headername>` MUST be identical to the equivalent HTTP header field-name encoded in all lowercase.

In the case where the User Agent request includes multiple HTTP header fields with the same field-name, it is RECOMMENDED that the uCDN combines these different HTTP headers into a single value according to Section 3.2.2 of [RFC7230]. However, because of the plurality of already defined HTTP header fields, and inconsistency of some of these header fields concerning the combination mechanism defined in RFC 7230, the uCDN MAY have to deviate from using the combination mechanism where appropriate. For example, it MAY only
send the contents of the first occurrence of the HTTP Headers instead.

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

POST /dcdn/rrri HTTP/1.1
Host: rr1.dcdn.example.net
Content-Type: application/cdni; ptype=redirection-request
Accept: application/cdni; ptype=redirection-response

{
    "http": {
        "c-ip": "198.51.100.1",
        "cs-uri": "http://www.example.com",
        "cs-version": "HTTP/1.1",
        "cs-method": "GET"
    },
    "cdn-path": ["AS64496:0"],
    "max-hops": 3
}

4.5.2. HTTP Redirection responses

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

- A URI pointing to an RT that is the selected dCDN surrogate(s) (if the dCDN is performing HTTP based redirection directly to a surrogate); or
- A URI pointing to an RT that is a Request Router (if the dCDN will perform request redirection itself).

The information above is encoded as a set of key:value pairs within the http dictionary as follows:
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc-status</td>
<td>Integer</td>
<td>Yes</td>
<td>The status-code part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA (usually set to 302).</td>
</tr>
<tr>
<td>sc-version</td>
<td>String</td>
<td>Yes</td>
<td>The HTTP-version part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.</td>
</tr>
<tr>
<td>sc-reason</td>
<td>String</td>
<td>Yes</td>
<td>The reason-phrase part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.</td>
</tr>
<tr>
<td>cs-uri</td>
<td>String</td>
<td>Yes</td>
<td>The URI requested by the UA/client.</td>
</tr>
<tr>
<td>sc-(location)</td>
<td>String</td>
<td>Yes</td>
<td>Location header to return to the UA (i.e. a URI pointing to the RT(s)).</td>
</tr>
<tr>
<td>sc-(&lt;headername&gt;)</td>
<td>String</td>
<td>No</td>
<td>The field-value of the HTTP header field named &lt;HeaderText&gt; to return to the UA. For example, sc-(expires) would contain the value of the HTTP Expires header.</td>
</tr>
</tbody>
</table>

Note: The sc-(location) key in the table above is an example of sc-(<headername>) that has been called out separately as its presence is mandatory in RI responses.

A successful RI response for HTTP-based redirection MUST include an http dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for HTTP-based redirection MUST include an error dictionary. If an http dictionary is included in the RI response, it MUST include at least the following keys: sc-status, sc-version, sc-reason, cs-uri, sc-(location).
The http dictionary of an RI response MUST contain a maximum of one sc-⟨<headername⟩⟩ key for each unique header field-name (HTTP header field). ⟨headername⟩ MUST be identical to the equivalent HTTP header field-name encoded in all lowercase.

The uCDN MAY decide to not return, override or alter any or all of the HTTP headers defined by sc-⟨<headername⟩⟩ keys before sending the HTTP response to the UA. It should be noted that in some cases, sending the HTTP Headers indicated by the dCDN transparently on to the UA might result in, for the uCDN, undesired behaviour. As an example, the dCDN might include sc-(cache-control), sc-(last-modified) and sc-(expires) keys in the http dictionary, through which the dCDN may try to influence the cacheability of the response by the UA. If the uCDN would pass these HTTP headers on to the UA, this could mean that further requests from the uCDN would go directly to the dCDN, bypassing the uCDN and any logging it may perform on incoming requests. The uCDN is therefore recommended to carefully consider which HTTP headers to pass on, and which to either override or not pass on at all.

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

HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response

{
    "http": {
        "sc-status": 302,
        "sc-version": "HTTP/1.1",
        "sc-reason": "Found",
        "cs-uri": "http://www.example.com"
        "sc-(location)":
            "http://surf1.dcdn.example/ucdn/example.com",
    }
}

4.6. Cacheability and scope of responses

RI responses may be cacheable. As long as a cached RI response is not stale according to standard HTTP Cache-Control or other applicable mechanisms, it may be reused by the uCDN in response to User Agent requests without sending another RI request to the dCDN.

An RI response MUST NOT be reused unless the request from the User Agent would generate an identical RI request to the dCDN as the one that resulted in the cached RI response (except for the c-ip field.
provided that the User Agent’s c-ip is covered by the scope in the original RI response, as elaborated upon below).

Additionally, although RI requests only encode a single User Agent request to be redirected there may be cases where a dCDN wishes to indicate to the uCDN that the RI response can be reused for other User Agent requests without the uCDN having to make another request via the RI. For example a dCDN may know that it will always select the same Surrogates for a given set of User Agent IP addresses and in order to reduce request volume across the RI or to remove the additional latency associated with an RI request, the dCDN may wish to indicate that set of User Agent IP addresses to the uCDN in the initial RI response. This is achieved by including an optional scope dictionary in the RI response.

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iprange</td>
<td>List</td>
<td>No</td>
<td>A List of IP subnets in CIDR notation that this RI response can be reused for, provided the RI response is still considered fresh.</td>
</tr>
</tbody>
</table>

If a uCDN has multiple cached responses with overlapping scopes and a UA request comes in for which the User Agent’s IP matches with the IP subnets in multiple of these cached responses, the uCDN SHOULD use the most recent cached response when determining the appropriate RI response to use.

The following is an example of a DNS redirection response from Section 4.4.2 that is cacheable by the uCDN for 30 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/24.
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: public, max-age=30

{
    "dns": {
        "rcode": 0,
        "name": "www.example.com",
        "a": ["203.0.113.200", "203.0.113.201"],
        "aaaa": ["2001:DB8::C8", "2001:DB8::C9"],
        "ttl": 60
    }
    "scope": {
        "iprange": ["198.51.100.0/24"]
    }
}

Example of HTTP redirection response from Section 4.5.2 that is cacheable by the uCDN for 60 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/24.

Note: The response to the UA is only valid for 30 seconds, whereas the uCDN can cache the RI response for 60 seconds.

HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: public, max-age=60

{
    "http": {
        "sc-status": 302,
        "cs-uri": "http://www.example.com",
        "sc-(location)": "http://sur1.dcdn.example/ucdn/example.com",
        "sc-(cache-control)": "public, max-age=30"
    }
    "scope": {
        "iprange": ["198.51.100.0/24"]
    }
}

4.7. Error responses

From a uCDN perspective, there are two types of errors that can be the result of the transmission of an RI request to a dCDN:
1. An HTTP protocol error signaled via an HTTP status code, indicating a problem with the reception or parsing of the RI request or the generation of the RI response by the dCDN, and

2. An RI-level error specified in an RI response message

This section deals with the latter type. The former type is outside the scope of this document.

There are numerous reasons for a dCDN to be unable to return an affirmative RI response to a uCDN. Reasons may include both dCDN internal issues such as capacity problems, as well as reasons outside the influence of the dCDN, such as a malformed RI request. To aid with diagnosing the cause of errors, RI responses SHOULD include an error dictionary to provide additional information to the uCDN as to the reason/cause of the error. The intention behind the error dictionary is to aid with either manual or automatic diagnosis of issues. The resolution of such issues is outside the scope of this document; this document does not specify any consequent actions a uCDN should take upon receiving a particular error code.

Error information (if present) is encoded as a set of key:value pairs within a JSON-encoded error dictionary as follows:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error-code</td>
<td>Integer</td>
<td>Yes</td>
<td>A three-digit numeric code defined by the server to indicate the error(s) that occurred.</td>
</tr>
<tr>
<td>reason</td>
<td>String</td>
<td>No</td>
<td>A string providing further information related to the error.</td>
</tr>
</tbody>
</table>

The first digit of the error-code defines the class of error. There are 5 classes of error distinguished by the first digit of the error-code:

1xx: Informational (no error): The response should not be considered an error by the uCDN, which may proceed by redirecting the UA according to the values in the RI response. The error code and accompanying description may be used for informational purposes, e.g. for logging.

2xx: Reserved.
3xx: Reserved.

4xx: uCDN error: The dCDN can not or will not process the request due to something that is perceived to be a uCDN error, for example the RI request could not be parsed successfully by the dCDN. The last two-digits may be used to more specifically indicate the source of the problem.

5xx: dCDN error: Indicates that the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason, for example the dCDN was able to parse the RI request but encountered an error for some reason. Examples include the dCDN not being able to retrieve the associated metadata or the dCDN being out of capacity.

The following error codes are defined and maintained by IANA (see Section 6):

Error codes with a "Reason" of "<reason>" do not have a defined value for their 'reason'-key. Depending on the error-code semantics, the value of this field may be determined dynamically.
The following is an example of an unsuccessful RI response (dCDN->uCDN) for a DNS based User Agent request:

<table>
<thead>
<tr>
<th>Code</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>&lt;reason&gt; (see Description)</td>
<td>Generic informational error-code meant for carrying a human-readable string</td>
</tr>
<tr>
<td>400</td>
<td>&lt;reason&gt; (see Description)</td>
<td>Generic error-code for uCDN errors where the dCDN can not or will not process the request due to something that is perceived to be a uCDN error. The reason field may be used to provide more details about the source of the error.</td>
</tr>
<tr>
<td>500</td>
<td>&lt;reason&gt; (see Description)</td>
<td>Generic error-code for dCDN errors where the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason. The reason field may be used to provide more details about the source of the error.</td>
</tr>
<tr>
<td>501</td>
<td>Unable to retrieve metadata</td>
<td>The dCDN is unable to retrieve the metadata associated with the content requested by the UA. This may indicate a configuration error or the content requested by the UA not existing.</td>
</tr>
<tr>
<td>502</td>
<td>Loop detected</td>
<td>The dCDN detected a redirection loop (see Section 4.8).</td>
</tr>
<tr>
<td>503</td>
<td>Maximum hops exceeded</td>
<td>The dCDN detected the maximum number of redirection hops exceeding max-hops (see Section 4.8).</td>
</tr>
<tr>
<td>504</td>
<td>Out of capacity</td>
<td>The dCDN does not currently have sufficient capacity to handle the UA request.</td>
</tr>
<tr>
<td>505</td>
<td>Delivery protocol not supported</td>
<td>The dCDN does not support the (set of) delivery protocols indicated in the CDNI Metadata of the content requested content by the UA.</td>
</tr>
<tr>
<td>506</td>
<td>Redirection protocol not supported</td>
<td>The dCDN does not support the requested redirection protocol. This error-code is also used when the RI request has the dns-only flag set to True and the dCDN is not support or is not prepared to return a RT of a surrogate directly.</td>
</tr>
</tbody>
</table>
HTTP/1.1 500 Internal Server Error
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: private, no-cache

{
    "error": {
        "error-code": 504,
        "description": "Out of capacity"
    }
}

The following is an example of a successful RI response (dCDN->uCDN)
for a HTTP based User Agent request containing an error dictionary
for informational purposes:

HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: private, no-cache

{
    "http": {
        "sc-status": 302,
        "sc-version": "HTTP/1.1",
        "sc-reason": "Found",
        "cs-uri": "http://www.example.com",
        "sc-(location)": "http://sur1.dcdn.example/ucdn/example.com",
    },
    "error": {
        "error-code": 100,
        "description": "This is a human-readable message meant for debugging purposes"
    }
}

4.8. Loop detection & prevention

In order to prevent and detect RI request loops, each CDN MUST insert
its CDN Provider ID into the cdn-path key of every RI request it
originates or cascades. When receiving RI requests a dCDN MUST check
the cdn-path and reject any RI requests which already contain the
dCDN’s Provider ID in the cdn-path. Transit CDNs MUST check the cdn-
path and not cascade the RI request to dCDNs that are already listed
in cdn-path. Transit CDNs MUST NOT propagate to any downstream CDNs
if the number of CDN Provider IDs in cdn-path (before adding its own
Provider ID) is equal to or greater than max-hops.
The CDN Provider ID uniquely identifies each CDN provider during the course of request routing redirection. It consists of the characters AS followed by the CDN Provider’s AS number, then a colon (’::’) and an additional qualifier that is used to guarantee uniqueness in case a particular AS has multiple independent CDNs deployed. For example "AS64496:0".

If a dCDN receives an RI request whose cdn-path already contains that dCDN’s Provider ID the dCDN SHOULD send an RI response with an error code of 502.

If a dCDN receives an RI request where the number of CDN Provider IDs in cdn-path is greater than max-hops, the dCDN SHOULD send an RI response with an error code of 503.

It should be noted that the loop detection & prevention mechanisms described above only cover preventing and detecting loops within the RI itself. As well as loops within the RI itself, there is also the possibility of loops in the data plane, for example if the IP address(es) or URI(s) returned in RI responses do not resolve directly to a surrogate in the final dCDN there is the possibility that a User Agent may be continuously redirected through a loop of CDNs. The specification of solutions to address data plane request redirection loops between CDNs is outside of the scope of this document.

5. Security Considerations

Information passed over the RI could be considered personal or sensitive, for example RI requests contain parts of a User Agent’s original request and RI responses reveal information about the dCDN’s policy for which surrogates should serve which content/user locations.

The RI interface also provides a mechanism whereby a uCDN could probe a dCDN and infer the dCDN’s edge topology by making repeated RI requests for different content and/or UA IP addresses and correlating the responses from the dCDN. Additionally the ability for a dCDN to indicate that an RI response applies more widely than the original request (via the scope dictionary) may significantly reduce the number of RI requests required to probe and infer the dCDN’s edge topology.

The same information could be obtained in the absence of the RI interface, but it could be more difficult to gather as it would require a distributed set of machines with a range of different IP addresses each making requests directly to the dCDN. However, the RI facilitates easier collection of such information as it enables a
single client to query the dCDN for a redirection/surrogate selection on behalf of any UA IP address.

5.1. Authentication, Authorization, Confidentiality, Integrity Protection

An implementation of the CDNI Redirection interface MUST support TLS transport as per [RFC2818] and [RFC7230]. The use of TLS for transport of the CDNI Redirection interface messages allows:

- The dCDN and uCDN to authenticate each other and, once they have mutually authenticated each other, it allows:
  - the dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI Redirection messages to/from an authorized CDN)
  - CDNI Redirection interface messages to be transmitted with confidentiality.
  - The integrity of the CDNI Redirection interface messages to be protected during the exchange.

In an environment where any such protection is required, mutually authenticated encrypted transport MUST be used to ensure confidentiality of the redirection information. To that end, TLS MUST be used (including authentication of the remote end) by the server-side (dCDN) and the client-side (uCDN) of the CDNI Redirection interface.

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

5.2. Privacy

Information passed over the RI could be considered personal or sensitive. In particular, parts of a User Agent’s original request, most notably the UA’s IP address and requested URI, are transmitted over the RI to the dCDN. The use of mutually authenticated TLS, as described in the previous section, prevents any other party than the authorized dCDN from gaining access to this information.

Regardless of whether the uCDN and dCDN use the RI, a successful redirect from a uCDN to a dCDN will make that dCDN aware of the UA’s IP address. As such, the fact that this information is transmitted across the RI does not allow the dCDN to learn new information. On the other hand, if a uCDN uses the RI to check with multiple
candidate dCDNs, those candidates that do not end up getting redirected to, do obtain information regarding End User IP addresses and requested URIs that they wouldn’t had the RI not been used.

While it is technically possible to mask some information in the RI Request, such as the last bits of the UA IP address, it is important to note that this will reduce the effectiveness of the RI in certain cases. As an example, when the UA is behind a Carrier-grade NAT, and the RI is used to find an appropriate delivery node behind the same NAT, the full IP address might be necessary. Another potential issue when using IP anonymization is that it is no longer possible to correlate a RI Request with a subsequent UA request.

6. IANA Considerations

6.1. CDNI Payload Type Parameter registrations

The IANA is requested to register the following two new Payload Types in the CDNI Payload Type Parameter registry for use with the application/cdni MIME media type.

[RFC Editor Note: Please replace the references to [RFCthis] below with this document’s RFC number before publication.]

+----------------------+---------------+
| Payload Type         | Specification |
+----------------------+---------------+
| redirection-request  | [RFCthis]      |
| redirection-response | [RFCthis]      |
+----------------------+---------------+

6.1.1. CDNI RI Redirection Request Payload Type

Purpose: The purpose of this payload type is to distinguish RI request messages.

Interface: RI

Encoding: see Section 4.4.1 and Section 4.5.1

6.1.2. CDNI RI Redirection Response Payload Type

Purpose: The purpose of this payload type is to distinguish RI response messages.

Interface: RI

Encoding: see Section 4.4.2 and Section 4.5.2
6.2. RI Error response registry

This document establishes a new IANA registry for CDNI RI Error response codes.

An expert reviewer is advised to examine new registrations for possible duplication with existing error codes and to ensure that the new code is in accordance with the error classes defined in section Section 4.7 of this document.

New registrations are required to provide the following information:

Code: A three-digit numeric error-code, in accordance with the error classes defined in section Section 4.7 of this document.

Reason: A string that provides further information related to the error that will be included in the JSON error dictionary with the 'reason’-key. Depending on the error-code semantics, the value of this field may be determined dynamically. In that case, the registration should set this value to '<reason>' and define its semantics in the description field.

Description: A brief description of the error code semantics.

Specification: An optional reference to a specification that defines the error code in more detail.

The entries in Table 1 are registered by this document.

7. Contributors

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

The following persons have participated as co-authors to this document:

Wang Danhua, Huawei, Email: wangdanhua@huawei.com

He Xiaoyan, Huawei, Email: hexiaoyan@huawei.com

Ge Chen, China Telecom, Email: cheng@gsta.com

Ni Wei, China Mobile, Email: niwei@chinamobile.com

Yunfei Zhang, Email: hishigh@gmail.com

Spencer Dawkins, Huawei, Email: spencer@wonderhamster.org
8. Acknowledgements

The authors would like to thank Taesang Choi, Francois le Faucheur, Matt Miller, Scott Wainner and Kevin J. Ma for their valuable comments and input to this document.

9. References

9.1. Normative References


9.2. Informative References


Authors’ Addresses

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

Email: ben.niven-jenkins@alcatel-lucent.com
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 February 13, 2016.
Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ................................................. 3
  1.1. Terminology ............................................. 4
  1.2. Background on URI Signing ............................... 4
  1.3. CDNI URI Signing Overview ............................. 6
  1.4. URI Signing in a non-CDNI context ...................... 8
2. Signed URI Information Elements ............................... 8
  2.1. Enforcement Information Elements ....................... 10
  2.2. Signature Computation Information Elements .......... 11
  2.3. URI Signature Information Elements ................... 12
  2.4. URI Signing Package Attribute ........................ 13
  2.5. User Agent Attributes ................................ 14
3. Creating the Signed URI ..................................... 14
  3.1. Calculating the URI Signature .......................... 15
  3.2. Packaging the URI Signature ............................ 18
4. Validating a URI Signature .................................. 19
  4.1. Information Element Extraction ........................ 19
  4.2. Signature Validation ................................... 20
  4.3. Distribution Policy Enforcement ....................... 22
5. Relationship with CDNI Interfaces ........................... 23
  5.1. CDNI Control Interface ................................. 23
  5.2. CDNI Footprint & Capabilities Advertisement Interface .. 23
  5.3. CDNI Request Routing Redirection Interface ........... 24
  5.4. CDNI Metadata Interface ................................ 24
  5.5. CDNI Logging Interface ................................. 27
6. URI Signing Message Flow .................................... 28
  6.1. HTTP Redirection ....................................... 28
  6.2. DNS Redirection ........................................ 31
7. HTTP Adaptive Streaming ..................................... 34
8. IANA Considerations ......................................... 34
9. Security Considerations ..................................... 35
10. Privacy ..................................................... 36
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 [RFC7337] document and the CDNI Framework [RFC7336] 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 [RFC7336] 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 [RFC7337]:

"MI-16 [HIGH] The CDN 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):

- MAC: message authentication code.
- 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.
- HMAC-SHA256: HMAC instantiation using SHA-256 as the cryptographic hash function.
- SHA-1: Secure Hash Algorithm 1 (SHA-1) [RFC3174] is the cryptographic hash function.

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

- URI Signature: Message digest or digital signature that is computed with an algorithm for protecting the URI.
- Original URI: The URI before URI Signing is applied.
- Signed URI: Any URI that contains a URI Signature.
- 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.
- 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).
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).
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:

- 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.

- 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.

- 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:

- URI Signing Package Attribute: The URI attribute that encapsulates all the URI Signing information elements in an encoded format. Only this attribute is exposed in the Signed URI as a URI query parameter.

Two types of keys can be used for URI Signing: asymmetric keys and symmetric keys. Asymmetric keys are based on a public/private key pair mechanism and always contain a private key only known to the entity signing the URI (either CSP or uCDN) and a public key for the verification of the Signed URI. With symmetric keys, the same key is used by both the signing entity for signing the URI as well as by the validating entity for validating the Signed URI. Regardless of the type of keys used, the validating entity has to obtain the key (either the public or the symmetric key). There are very different requirements for key distribution (out of scope of this document) with asymmetric keys and with symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent another party from getting access to the key, since it could then generate valid Signed URIs for unauthorized requests. Key distribution for asymmetric keys does not require confidentiality since public keys can typically be distributed openly (because they cannot be used for URI signing) and private keys are kept by the URI signing function.
Note that all the URI Signing information elements and the URI query attribute are mandatory to implement, but not mandatory to use.

### 2.1. Enforcement Information Elements

This section identifies the set of information elements that may be needed to enforce the CSP distribution policy. New information elements may be introduced in the future to extend the capabilities of the distribution policy.

In order to provide flexibility in distribution policies to be enforced, the exact subset of information elements used in the URI Signature of a given request is a deployment decision. The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to enforce the distribution policy:

- **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).

- **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.

- **Version (VER) [optional]** - An 8-bit unsigned 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.

- **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. The KID and KID_NUM information elements MUST NOT be present in the same URI Signing Package Attribute.

- **Numerical Key ID (KID_NUM) [optional]** - A 64-bit unsigned integer used as an optional alternative for KID. The KID and KID_NUM information elements MUST NOT be present in the same URI Signing Package Attribute.

- **Hash Function (HF) [optional]** - A string used for identifying the hash function to compute the URI signature with HMAC. If this Information Element is not present in the URI Signing Package Attribute, the default hash function is SHA-256.

- **Digital Signature Algorithm (DSA) [optional]** - Algorithm used to calculate the Digital Signature. If this Information Element is not present in the URI Signing Package Attribute, the default is EC-DSA.

The Version Information Element indicates which version of URI signing scheme is used (including which attributes and algorithms are supported). The present document specifies Version 1. If the Version attribute is not present in the Signed URI, then the version is obtained from the CDNI metadata, else it is considered to have been set to the default value of 1. More versions may be defined in the future.

The Key ID Information Element is used to retrieve the key which is needed as input to the algorithm for validating the Signed URI. The
method used for obtaining the actual key from the reference included in the Key ID Information Element is outside the scope of this document. Instead of using the KID element, which is a string, it is possible to use the KID_NUM element for numerical Key identifiers instead. The KID_NUM element is a 64-bit unsigned integer. In cases where numerical KEY IDs are used, it is RECOMMENDED to use KID_NUM instead of KID.

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.

- Message Digest (MD) [mandatory for symmetric key] - A string used for the message digest generated by the URI signing entity.

- Digital Signature (DS) [mandatory for asymmetric keys] - A string used for the digital signature provided by the URI signing entity.

The Message Digest attribute contains the message digest used to validate the Signed URI when symmetric keys are used.

The Digital Signature attribute contains the digital signature used to verify the Signed URI when asymmetric keys are used.

In the case of symmetric key, HMAC algorithm is used for the following reasons: 1) Ability to use hash functions (i.e. no changes needed) with well understood cryptographic properties that perform well and for which code is freely and widely available, 2) Easy to replace the embedded hash function in case faster or more secure hash
functions are found or required, 3) Original performance of the hash function is maintained without incurring a significant degradation, and 4) Simple way to use and handle keys. The default HMAC algorithm used is SHA-256.

In the case of asymmetric keys, Elliptic Curve Digital Signature Algorithm (EC DSA) - a variant of DSA - is used because of the following reasons: 1) Key size is small while still offering good security, 2) Key is easy to store, and 3) Computation is faster than DSA or RSA.

2.4. URI Signing Package Attribute

The URI Signing Package Attribute is an encapsulation container for the URI Signing Information Elements defined in the previous sections. The URI Signing Information Elements are encoded and stored in this attribute. URI Signing Package Attribute is appended to the Original URI to create the Signed URI.

The primary advantage of the URI Signing Package Attribute is that it avoids having to expose the URI Signing Information Elements directly in the query string of the URI, thereby reducing the potential for a namespace collision space within the URI query string. A side-benefit of the attribute is the obfuscation performed by the URI Signing Package Attribute hides the information (e.g. client IP address) from view of the common user, who is not aware of the encoding scheme. Obviously, this is not a security method since anyone who knows the encoding scheme is able to obtain the clear text. Note that any parameters appended to the query string after the URI Signing Package Attribute are not validated and hence do not affect URI Signing.

The following attribute is used to carry the encoded set of URI Signing attributes in the Signed URI.

- 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 is not used, or does not include a parameter name for the URI Signing Package Attribute, the parameter name is set by configuration (out of scope of this document).

2.5. User Agent Attributes

For some use cases, such as logging, it might be useful to allow the UA, or another entity, add one or more attributes to the Signed URI for purposes other than URI Signing without causing URI Signing to fail. In order to do so, such attributes MUST be appended after the URI Signing Package Attribute. Any attributes appended in such way after the URI Signature has been calculated are not validated for the purpose of content access authorization. Adding any such attributes to the Signed URI before the URI Signing Package Attribute will cause the URI Signing validation to fail.

Note that a malicious UA might potentially use the ability to append attributes to the Signed URI in order to try to influence the content that is delivered. For example, the UA might append ‘&quality=HD’ to try to make the dCDN deliver an HD version of the requested content. Since such an additional attribute is appended after the URI Signing Package Attribute it is not validated and will not affect the outcome of the URI validation. In order to deal with this vulnerability, a dCDN is RECOMMENDED to ignore any query strings appended after the URI Signing Package Attribute for the purpose of content selection.

3. Creating the Signed URI

The following procedure for signing a URI defines the algorithms in this version of URI Signing. Note that some steps may be skipped if the CSP does not enforce a distribution policy and the Enforcement Information Elements are therefore not necessary. A URI (as defined in URI Generic Syntax [RFC3986]) contains the following parts: scheme name, authority, path, query, and fragment. The entire URI except the "scheme name" part is protected by the URI signature. This allows the URI signature to be validated correctly in the case when a client performs a fallback to another scheme (e.g. HTTP) for a content item referenced by a URI with a specific scheme (e.g. RTSP). The benefit is that the content access is protected regardless of the type of transport used for delivery. If the CSP wants to ensure a specific protocol is used for content delivery, that information is passed by CDNI metadata. Note: Support for changing of the URL

scheme requires that the default port is used, or that the protocols must both run on the same non-standard port.

The process of generating a Signed URI can be divided into two sets of steps: first, calculating the URI Signature and then, packaging the URI Signature and appending it to the Original URI. Note it is possible to use some other algorithm and implementation as long as the same result is achieved. An example for the Original URI, "http://example.com/content.mov", is used to clarify the steps.

3.1. Calculating the URI Signature

Calculate the URI Signature by following the procedure below.

1. Copy the Original URI, excluding the "scheme name" part, into a buffer to hold the message for performing the operations below.

2. Check if the URI already contains a query string. If not, append a "?" character. If yes, append an "&" character.

3. If the version is the default value (i.e. "1"), skip this step. Otherwise, specify the version by appending the string "VER=#", where '#' represents the new version number. The following steps in the procedure is based on the initial version of URI Signing specified by this document. For other versions, reference the associated RFC for the URI signing procedure.

4. If time window enforcement is not needed, step 4 can be skipped.

   A. If an information element was added to the message, append an "&" character. Append the string "ET=". Note in the case of re-signing a URI, the information element is carried over from the received Signed URI.

   B. Get the current time in seconds since epoch (as an integer). Add the validity time in seconds as an integer. Note in the case of re-signing a URI, the value MUST remain the same as the received Signed URI.

   C. Convert this integer to a string and append to the message.

5. If client IP enforcement is not needed, step 5 can be skipped.

   A. If an information element was added to the message, append an "&" character. Append the string "CIP=". Note in the case of re-signing a URI, the attribute is carried over from the received Signed URI.
B. Convert the client’s IP address in dotted decimal notation format (i.e. for IPv4 address) or canonical text representation (for IPv6 address [RFC5952]) to a string and append to the message. Note in the case of re-signing an URI, the value MUST remain the same as the received Signed URI.

6. Depending on the type of key used to sign the URI, compute the message digest or digital signature for symmetric key or asymmetric keys, respectively.

A. For symmetric key, HMAC is used.

1. Obtain the shared key to be used for signing the URI.

2. If the key identifier is not needed, skip this step. If an information element was added to the message, append an "&" character. Append the string "KID=" in case a string-based Key ID is used, or "KID_NUM=" in case a numerical Key ID is used. Append the key identifier (e.g. "example:keys:123" or "56128239") needed by the entity to locate the shared key for validating the URI signature.

3. Optional: If the hash function for the HMAC uses the default value ("SHA-256"), skip this step. If an information element was added to the message, append an "&" character. Append the string "HF=". Append the string for the new type of hash function to be used. Note that re-signing a URI MUST use the same hash function as the received Signed URI or one of the allowable hash functions designated by the CDNI metadata.

4. If an information element was added to the message, append an "&" character. Append the string "MD=". The message now contains the complete section of the URI that is protected (e.g. "://example.com/content.mov?ET=1209422976&CIP=192.0.2.1&KID=example:keys:123&MD=").

5. Compute the message digest using the HMAC algorithm and the default SHA-256 hash function, or another hash function if specified by the HF Information Element, with the shared key and message as the two inputs to the hash function.

6. Convert the message digest to its equivalent hexadecimal format.
7. Append the string for the message digest (e.g. "://example.com/content.mov?ET=1209422976&CIP=192.0.2.1&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bdddaf").

B. For asymmetric keys, EC DSA is used.

1. Generate the EC private and public key pair. Store the EC public key in a location that's reachable for any entity that needs to validate the URI signature.

2. If the key identifier is not needed, skip this step. If an information element was added to the message, append an "&" character. Append the string "KID=" in case a string-based Key ID is used, or "KID_NUM=" in case a numerical Key ID is used. 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 that in the case the Key ID URI is a URL to a public key, the Key ID URI SHOULD only contain the "scheme name", "authority", and "path" parts (i.e. query string is not allowed).

3. Optional: If the digital signature algorithm uses the default value ("EC-DSA"), skip this step. If an information element was added to the message, append an "&" character. Append the string "DSA=". Append the string denoting the new digital signature function.

4. If an information element was added to the message, append an "&" character. Append the string "DS=". The message now contains the complete section of the URI that is protected. (e.g. "://example.com/content.mov?ET=1209422976&CIP=192.0.2.1&KID=http://example.com/public/keys/123&DS=").

5. Compute the message digest using SHA-1 (without a key) for the message. Note: The digital signature generated in the next step is calculated over the SHA-1 message digest, instead of over the cleartype message, to reduce the length of the digital signature, and thereby the length of the URI Signing Package Attribute and the resulting Signed URI. Since SHA-1 is not used for cryptographic purposes here, the security concerns around SHA-1 do not apply.

6. Compute the digital signature, using the EC-DSA algorithm by default or another algorithm if specified by the DSA...
Information Element, with the private EC key and message digest (obtained in previous step) as inputs.

7. Convert the digital signature to its equivalent hexadecimal format.

8. Append the string for the digital signature. In the case where EC-DSA algorithm is used, this string contains the values for the 'r' and 's' parameters, delimited by ':' (e.g. "://example.com/content.mov?ET=1209422976&CIP=192.0.2.1&KID=http://example.com/public/keys/123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bda"f)

3.2. Packaging the URI Signature

Apply the URI Signing Package Attribute by following the procedure below to generate the Signed URI.

1. Remove the Original URI portion from the message to obtain all the URI Signing Information Elements, including the URI signature (e.g. "ET=1209422976&CIP=192.0.2.1&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bda")

2. Compute the URI Signing Package Attribute using Base-64 Data Encoding [RFC4648] on the message (e.g. "VkVSPTEmRVQ9MTIwOTQyMjk3NiZDSVA9MTkyLjAuMi4xJktJRD1leGFtcGxlOmtleXM6MTIzJk1EPTFlY2IzNDQ2YTY0MzEyW1IwZmI2ZTBkY2EzMGUzMDM1NjU5M2E5N2FjYjkJ3MjIwMjEyMGRjNDgyYmRkYWY="). 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=VkVSPTEmRVQ9MTIwOTQyMjk3NiZDSVA9MTkyLjAuMi4xJktJRD1leGFtcGxlOmtleXM6MTIzJk1EPTFlY2IzNDQ2YTY0MzEyW1IwZmI2ZTBkY2EzMGUzMDM1NjU5M2E5N2FjYjkJ3MjIwMjEyMGRjNDgyYmRkYWY="
4. Validating a URI Signature

The process of validating a Signed URI can be divided into three sets of steps: first, extraction of the URI Signing information elements, then validation of the URI signature to ensure the integrity of the Signed URI, and finally, validation of the information elements to ensure proper enforcement of the distribution policy. The integrity of the Signed URI is confirmed before distribution policy enforcement because validation procedure would detect the right event when the URI is tampered with. Note it is possible to use some other algorithm and implementation as long as the same result is achieved.

4.1. Information Element Extraction

Extract the information elements embedded in the URI. Note that some steps are to be skipped if the corresponding URI Signing information elements are not embedded in the Signed URI.

1. Extract the value from ‘URISigningPackage’ attribute. This value is the encoded URI Signing Package Attribute. If there are multiple instances of this attribute, the first one is used and the remaining ones are ignored. This ensures that the Signed URI can be validated despite a client appending another instance of the ‘URISigningPackage’ attribute.

2. Decode the string using Base-64 Data Encoding [RFC4648] to obtain all the URI Signing information elements (e.g. "ET=1209422976&CIP=192.0.2.1&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dc30030356593a97ac872202120dc482bdddaf").

3. Extract the value from "VER" if the information element exists in the query string. Determine the version of the URI Signing algorithm used to process the Signed URI. If the CDNI Metadata interface is used, check to see if the used version of the URI Signing algorithm is among the allowed set of URI Signing versions specified by the metadata. If this is not the case, the request is denied. If the information element is not in the URI, then obtain the version number in another manner (e.g. configuration, CDNI metadata or default value).

4. Extract the value from "MD" if the information element exists in the query string. The existence of this information element indicates a symmetric key is used.
5. Extract the value from "DS" if the information element exists in the query string. The existence of this information element indicates an asymmetric key is used.

6. If neither "MD" or "DS" attribute is in the URI, then no URI Signature exists and the request is denied. If both the "MD" and the "DS" information elements are present, the Signed URI is considered to be malformed and the request is denied.

7. Extract the value from "CIP" if the information element exists in the query string. The existence of this information element indicates content delivery is enforced based on client IP address.

8. Extract the value from "ET" if the information element exists in the query string. The existence of this information element indicates content delivery is enforced based on time.

9. Extract the value from the "KID" or "KID_NUM" information element, if they exist. The existence of either of these information elements indicates a key can be referenced. If both the "KID" and the "KID_NUM" information elements are present, the Signed URI is considered to be malformed and the request is denied.

10. Extract the value from the "HF" information element, if it exists. The existence of this information element indicates a different hash function than the default.

11. Extract the value from the "DSA" information element, if it exists. The existence of this information element indicates a different digital signature algorithm than the default.

4.2. Signature Validation

Validate the URI Signature for the Signed URI.

1. Copy the Original URI, excluding the "scheme name" part, into a buffer to hold the message for performing the operations below.

2. Remove the "URISigningPackage" attribute from the message. Remove any subsequent part of the query string after the "URISigningPackage" attribute.

3. Append the decoded value from "URISigningPackage" attribute (which contains all the URI Signing Information Elements).
4. Depending on the type of key used to sign the URI, validate the message digest or digital signature for symmetric key or asymmetric keys, respectively.

A. For symmetric key, HMAC algorithm is used.
   
   a. If either the "KID" or "KID_NUM" information element exists, validate that the key identifier is in the allowable KID set as listed in the CDNI metadata or configuration. The request is denied when the key identifier is not allowed. If neither the "KID" or "KID_NUM" information element is present in the Signed URI, obtain the shared key via CDNI metadata or configuration.

   b. If "HF" information element exists, validate that the hash function is in the allowable "HF" set as listed in the CDNI metadata or configuration. The request is denied when the hash function is not allowed. Otherwise, the "HF" information element is not in the Signed URI. In this case, the default hash function is SHA-256.

   c. Extract the value from the "MD" information element. This is the received message digest.

   d. Convert the message digest to binary format. This will be used to compare with the computed value later.

   e. Remove the value part of the "MD" information element (but not the '=' character) from the message. The message is ready for validation of the message digest (e.g. "://example.com/content.mov?ET=1209422976&CIP=192.0.2.1&KID=example:keys:123&MD=").

   f. Compute the message digest using the HMAC algorithm with the shared key and message as the two inputs to the hash function.

   g. Compare the result with the received message digest to validate the Signed URI.

B. For asymmetric keys, a digital signature function is used.

   a. If either the "KID" or "KID_NUM" information element exists, validate that the key identifier is in the allowable KID set as listed in the CDNI metadata or configuration. The request is denied when the key identifier is not allowed. If neither the "KID" or
"KID_NUM" information element is present in the Signed URI, obtain the public key via CDNI metadata or configuration.

b. If "DSA" information element exists, validate that the digital signature algorithm is in the allowable "DSA" set as listed in the CDNI metadata or configuration. The request is denied when the DSA is not allowed. Otherwise, the "DSA" information element is not in the Signed URI. In this case, the default DSA is EC-DSA.

c. Extract the value from the "DS" information element. This is the digital signature.

d. Convert the digital signature to binary format. This will be used for verification later.

e. Remove the value part of the "DS" information element (but not the ‘=’ character) from the message. The message is ready for validation of the digital signature (e.g. "://example.com/content.mov?ET=1209422976&CIP=192.0.2.1&KID=http://example.com/public/keys/123&DS=").

f. Compute the message digest using SHA-1 (without a key) for the message.

g. Verify the digital signature using the digital signature function (e.g. EC-DSA) with the public key, received digital signature, and message digest (obtained in previous step) as inputs. This validates the Signed URI.

4.3. Distribution Policy Enforcement

Note the steps are to be skipped if the corresponding URI Signing information elements are not in the Signed URI. The absence of a given Enforcement Information Element indicates enforcement of its purpose is not necessary in the CSP’s distribution policy.

1. If the "CIP" information element exists, validate that the request came from the same IP address as indicated in the "CIP" information element. If the IP address is incorrect, then the request is denied.

2. If the "ET" information element exists, validate that the request arrived before expiration time based on the "ET" information element. If the time expired, then the request is denied.
5. Relationship with CDNI Interfaces

Some of the CDNI Interfaces need enhancements to support URI Signing. As an example: A Downstream CDN that supports URI Signing needs to be able to advertise this capability to the Upstream CDN. The Upstream CDN needs to select a Downstream CDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the Upstream CDN needs to be able to distribute via the CDNI Metadata interface the information necessary to allow the Downstream CDN to validate a Signed URI. Events that pertain to URI Signing (e.g. request denial or delivery after access authorization) need to be included in the logs communicated through the CDNI Logging interface (Editor’s Note: Is this within the scope of the CDNI Logging interface?).

5.1. CDNI Control Interface

URI Signing has no impact on this interface.

5.2. CDNI Footprint & Capabilities Advertisement Interface

The Downstream CDN advertises its capability to support URI Signing via the CDNI Footprint & Capabilities Advertisement interface (FCI). The supported version of URI Signing needs to be included to allow for future extensibility.

In general, new information elements introduced to enhance URI Signing requires a draft and a new version. For Information Elements,

   For Enforcement Information Elements, there is no need to advertise the based information elements such as "CIP" and "ET".

   For Signature Computation Information Elements:

   No need to advertise "VER" Information Element unless it’s not "1". In this case, a draft is needed to describe the new version.

   Advertise value of the "HF" Information Element (i.e. SHA-256) to indicate support for the hash function; Need IANA assignment for new hash function.

   Advertise value of the "DSA" Information Element (i.e. EC-DSA) to indicate support for the DSA; Need IANA assignment for new digital signature algorithm.
Advertise "MD" Information Element (i.e. EC-DSA) to indicate support for symmetric key method; A new draft is needed for an alternative method.

Advertise "DS" Information Element (i.e. EC-DSA) to indicate support for asymmetric key method; A new draft is needed for an alternative method.

For URI Signing Package Attribute, there is no need to advertise the base attribute.

5.3. CDNI Request Routing Redirection Interface

The CDNI Request Routing Redirection Interface [I-D.ietf-cdni-redirection] describes the recursive request redirection method. For URI Signing, the Upstream CDN signs the URI provided by the Downstream CDN. This approach has the following benefits:

- Consistency with iterative request routing method
- URI Signing is fully operational even when Downstream CDN does not have the signing function (which may be the case when the Downstream CDN operates only as a delivering CDN)
- Upstream CDN can act as a conversion gateway for the requesting routing interface between Upstream CDN and CSP and request routing interface between Upstream CDN and Downstream CDN since these two interfaces may not be the same

5.4. CDNI Metadata Interface

The CDNI Metadata Interface [I-D.ietf-cdni-metadata] describes the CDNI metadata distribution in order to enable content acquisition and delivery. For URI Signing, additional CDNI metadata objects are specified. In general, an Empty set means "all". These are the CDNI metadata objects used for URI Signing.

The UriSigning Metadata object contains information to enable URI signing and validation by a dCDN. The UriSigning properties are defined below.

Property: enforce

Description: URI Signing enforcement flag. Specifically, this flag indicates if the access to content is subject to URI Signing. URI Signing requires the Downstream CDN to ensure that the URI must be signed and validated before content...
delivery. Otherwise, Downstream CDN does not perform validation regardless if URI is signed or not.

Type: Boolean

Mandatory-to-Specify: No. If a UriSigning object is present in the metadata for a piece of content (even if the object is empty), then URI signing should be enforced. If no UriSigning object is present in the metadata for a piece of content, then the URI signature should not be validated.

Property: key-id

Description: Designated key identifier used for URI Signing computation when the Signed URI does not contain the Key ID information element.

Type: String

Mandatory-to-Specify: No. A Key ID is not essential for all implementations of URI signing.

Property: key-id-set

Description: Allowable Key ID set that the Signed URI’s Key ID information element can reference.

Type: List of Strings

Mandatory-to-Specify: No. Default is to allow any Key ID.

Property: hash-function

Description: Designated hash function used for URI Signing computation when the Signed URI does not contain the Hash Function information element.

Type: String (limited to the hash function strings in the registry defined by the IANA Considerations (Section 8) section)

Mandatory-to-Specify: No. Default is SHA-256.

Property: hash-function-set

Description: Allowable Hash Function set that the Signed URI’s Hash Function information element can reference.
Type: List of Strings
Mandatory-to-Specify: No. Default is to allow any hash function.

Property: digital-signature-algorithm
Description: Designated digital signature function used for URI Signing computation when the Signed URI does not contain the Digital Signature Algorithm information element.

Type: String (limited to the digital signature algorithm strings in the registry defined by the IANA Considerations (Section 8) section).
Mandatory-to-Specify: No. Default is EC-DSA.

Property: digital-signature-algorithm-set
Description: Allowable digital signature function set that the Signed URI’s Digital Signature Algorithm information element can reference.

Type: List of Strings
Mandatory-to-Specify: No. Default is to allow any DSA.

Property: version
Description: Designated version used for URI Signing computation when the Signed URI does not contain the VER attribute.

Type: Integer
Mandatory-to-Specify: No. Default is 1.

Property: version-set
Description: Allowable version set that the Signed URI’s VER attribute can reference.

Type: List of Integers
Mandatory-to-Specify: No. Default is to allow any version.
Description: Overwrite the default name for the URL Signing Package Attribute.

Type: String

Mandatory-to-Specify: No. Default is "URISigningPackage".

Note that the Key ID information element is not needed if only one key is provided by the CSP or the Upstream CDN for the content item or set of content items covered by the CDNI Metadata object. In the case of asymmetric keys, it’s easy for any entity to sign the URI for content with a private key and provide the public key in the Signed URI. This just confirms that the URI Signer authorized the delivery. But it’s necessary for the URI Signer to be the content owner. So, the CDNI Metadata interface or configuration MUST provide the allowable Key ID set to authorize the Key ID information element embedded in the Signed URI.

5.5. CDNI Logging Interface

For URI Signing, the Downstream CDN reports that enforcement of the access control was applied to the request for content delivery. When the request is denied due to enforcement of URI Signing, the reason is logged.

The following CDNI Logging field for URI Signing SHOULD be supported in the HTTP Request Logging Record as specified in CDNI Logging Interface [I-D.ietf-cdni-logging].

- s-uri-signing (mandatory):
  * format: 3DIGIT
  * field value: this characterises the uri signing validation performed by the Surrogate on the request. The allowed values are:
    + "0": no uri signature validation performed
    + "1": uri signature validation performed and validated
    + "2": uri signature validation performed and rejected
  * occurrence: there MUST be zero or exactly one instance of this field.

- s-uri-signing-deny-reason (optional):
* format: QSTRING

* field value: the rejection reason when uri signature performed by the Surrogate on the request. Examples:
  + "invalid client IP address"
  + "expired signed URI"
  + "incorrect URI signature"

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

6. URI Signing Message Flow

URI Signing supports both HTTP-based and DNS-based request routing. HMAC [RFC2104] defines a hash-based message authentication code allowing two parties that share a symmetric key or asymmetric keys to establish the integrity and authenticity of a set of information (e.g. a message) through a cryptographic hash function.

6.1. HTTP Redirection

For HTTP-based request routing, HMAC is applied to a set of information that is unique to a given end user content request using key information that is specific to a pair of adjacent CDNI hops (e.g. between the CSP and the Authoritative CDN, between the Authoritative CDN and a Downstream CDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI signing scheme described below is based on the following steps (assuming HTTP redirection, iterative request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.

```
<table>
<thead>
<tr>
<th>End-User</th>
<th>dCDN</th>
<th>uCDN</th>
<th>CSP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.CDNI FCI interface used to advertise URI Signing capability</td>
<td></td>
<td>&lt;-----------------------</td>
</tr>
<tr>
<td></td>
<td>2.Provides information to validate URI signature</td>
<td></td>
<td>&lt;-----------------------</td>
</tr>
<tr>
<td></td>
<td>3.CDNI Metadata interface used to provide URI Signing attributes</td>
<td></td>
<td>----------</td>
</tr>
</tbody>
</table>
```
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.

```
<table>
<thead>
<tr>
<th>End-User</th>
<th>dCDN</th>
<th>uCDN</th>
<th>CSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CDNI FCI interface used to advertise URI Signing capability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Provides information to validate URI signature</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. CDNI Metadata interface used to provide URI Signing attributes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Authorization request</td>
<td></td>
<td></td>
<td>[Apply distribution policy]</td>
</tr>
<tr>
<td>5. Request is denied</td>
<td></td>
<td></td>
<td>[Negative]</td>
</tr>
<tr>
<td>6. Provides signed URI</td>
<td></td>
<td></td>
<td>[Positive]</td>
</tr>
</tbody>
</table>
```
1. Using the CDNI Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.

2. CSP provides to the Authoritative CDN the information needed to validate cryptographic signatures from that CSP. For example, this information may include a hash function, algorithm, and a key.

3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate cryptographic signatures from the CSP (e.g. the URI query string parameter name for the URI Signing Package Attribute). In the case of symmetric key, the Authoritative CDN checks if the Downstream CDN is allowed by CSP to obtain the shared secret key.
4. When a UA requests a piece of protected content from the CSP, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy.

5. If the authorization decision is negative, the CSP rejects the request.

6. If the authorization decision is positive, the CSP computes a cryptographic signature that is based on unique parameters of that request and includes it in the URI provided to the end user to request the content.

7. End user sends DNS request to the authoritative CDN.

8. On receipt of the DNS request, the authoritative CDN redirects the request to the Downstream CDN.

9. End user sends DNS request to the Downstream CDN.

10. On receipt of the DNS request, the Downstream CDN responds with IP address of one of its Surrogates.

11. On receipt of the corresponding content request, the Downstream CDN validates the cryptographic signature in the URI using the information provided by the Authoritative CDN in the CDNI Metadata.

12. If the validation is negative, the Downstream CDN rejects the request and sends an error code (e.g. 403) in the HTTP response.

13. If the validation is positive, the Downstream CDN serves the request and delivers the content.

14. At a later time, Downstream CDN reports logging events that includes URI signing information.

With DNS-based request routing, URI Signing matches well the general chain of trust model of CDNI when used with asymmetric keys because the only key information that need to be distributed across multiple CDNI hops including non-adjacent hops is the public key, that is generally not confidential.

With DNS-based request routing, URI Signing does not match well the general chain of trust model of CDNI when used with symmetric keys because the symmetric key information needs to be distributed across multiple CDNI hops including non-adjacent hops. This raises a security concern for applicability of URI Signing with symmetric keys in case of DNS-based inter-CDN request routing.
7. HTTP Adaptive Streaming

The authors note that in order to perform URI signing for individual content segments of HTTP Adaptive Bitrate content, specific URI signing mechanisms are needed. Such mechanisms are currently out-of-scope of this document. More details on this topic is covered in Models for HTTP-Adaptive-Streaming-Aware CDNI [RFC6983]. [Editor note: DASH draft discussion]

8. IANA Considerations

[Editor’s note: (Is there a need to) register default value for URI Signing Package Attribute URI query string parameter name (i.e. URISigningPackage) to be used for URI Signing? Need anything from IANA?]

[Editor’s note: To do: Convert to proper IANA Registry format]

This document requests IANA to create three new URI Signing registries for the Information Elements and their defined values to be used for URI Signing.

The following Enforcement Information Element names are allocated:

- ET (Expiry time)
- CIP (Client IP address)

The following Signature Computation Information Element names are allocated:

- VER (Version): 1 (Base)
- KID (Key ID)
- KID_NUM (Numerical Key ID)
- HF (Hash Function): "SHA-256"
- DSA (Digital Signature Algorithm): "EC-DSA"

The following URI Signature Information Element names are allocated:

- MD (Message Digest for Symmetric Key)
- DS (Digital Signature for Asymmetric Keys)
The IANA is requested to allocate a new entry to the CDNI Logging Field Names Registry as specified in CDNI Logging Interface [I-D.ietf-cdni-logging] in accordance to the "Specification Required" policy [RFC5226]

- s-url-signing
- s-url-signing-deny-reason

The IANA is requested to allocate a new entry to the "CDNI GenericMetadata Types" Registry as specified in CDNI Metadata Interface [I-D.ietf-cdni-metadata] in accordance to the "Specification Required" policy [RFC5226]:

```
+------------+---------------+---------+------+------+
<table>
<thead>
<tr>
<th>Type name</th>
<th>Specification</th>
<th>Version</th>
<th>MTE</th>
<th>STR</th>
</tr>
</thead>
<tbody>
<tr>
<td>UriSigning</td>
<td>RFCthis</td>
<td>1</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
```

The IANA is also requested to allocate a new MIME type under the IANA MIME Media Type registry for the UriSigning metadata object:

application/cdni.UriSigning.v1

9. Security Considerations

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of interconnected CDNs (CDNI). The primary goal of URI Signing is to make sure that only authorized UAs are able to access the content, with a Content Service Provider (CSP) being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as DRM, are more appropriate.

In general, it holds that the level of protection against illegitimate access can be increased by including more Enforcement Information Elements in the URI. The current version of this document includes elements for enforcing Client IP Address and Expiration Time, however this list can be extended with other, more complex, attributes that are able to provide some form of protection against some of the vulnerabilities highlighted below.

That said, there are a number of aspects that limit the level of security offered by URI signing and that anybody implementing URI signing should be aware of.
Replay attacks: Any (valid) Signed URI can be used to perform replay attacks. The vulnerability to replay attacks can be reduced by picking a relatively short window for the Expiration Time attribute, although this is limited by the fact that any HTTP-based request needs a window of at least a couple of seconds to prevent any sudden network issues from preventing legitimate UAs access to the content. One way to reduce exposure to replay attacks is to include in the URI a unique one-time access ID. Whenever the Downstream CDN receives a request with a given unique access ID, it adds that access ID to the list of ‘used’ IDs. In the case an illegitimate UA tries to use the same URI through a replay attack, the Downstream CDN can deny the request based on the already-used access ID.

Illegitimate client behind a NAT: In cases where there are multiple users behind the same NAT, all users will have the same IP address from the point of view of the Downstream CDN. This results in the Downstream CDN not being able to distinguish between the different users based on Client IP Address and illegitimate users being able to access the content. One way to reduce exposure to this kind of attack is to not only check for Client IP but also for other attributes that can be found in the HTTP headers.

The shared key between CSP and Authoritative CDN may be distributed to Downstream CDNs – including cascaded CDNs. Since this key can be used to legitimately sign a URL for content access authorization, it’s important to know the implications of a compromised shared key.

In the case where asymmetric keys are used, the KID information element might contain the URL to the public key. To prevent malicious clients from signing their own URIs and inserting the associated public key URL in the KID field, thereby passing URI validation, it is important that CDNs check whether the URI conveyed in the KID field is in the allowable set of KIDs as listed in the CDNI metadata or set via configuration.

10. Privacy

The privacy protection concerns described in CDNI Logging Interface [I-D.ietf-cdni-logging] apply when the client’s IP address (CIP attribute) is embedded in the Signed URI. This means that, when anonymization is enabled, the value of the URI Signing Package Attribute MUST be removed from the logging record.
11. Acknowledgements

The authors would like to thank the following people for their contributions in reviewing this document and providing feedback: Scott Leibrand, Kevin Ma, Ben Niven-Jenkins, Thierry Magnien, Dan York, Bhaskar Bhupalam, Matt Caulfield, Samuel Rajakumar, Iuniana Oprescu and Leif Hedstrom. In addition, Matt Caulfield provided content for the CDNI Metadata Interface section.

12. References

12.1. Normative References

[I-D.ietf-cdni-logging]


12.2. Informative References

[I-D.ietf-cdni-metadata]

[I-D.ietf-cdni-redirection]


Authors’ Addresses

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose  95134
USA

Phone: +1 408 526 5030
Email: kleung@cisco.com