Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2016

F. Fieau, Ed.
I. Oprescu
Orange

July 6, 2015

HTTPS and delegation of encrypted traffic between interconnected CDNs
draft-fieau-https-delivery-delegation-00

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

Copyright Notice

Table of Contents

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 ontent 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:

- o 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.

- o 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 problem 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 excepts to forge a security association 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

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

9.2. Informative References

[I-D.barnes-hard-problem]
Barnes, R. and P. Saint-Andre, "High Assurance Re-Direction (HARD) Problem Statement", draft-barnes-hard-problem-00 (work in progress), July 2010.

[I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection Interface for CDN Interconnection", draft-ietf-cdni-redirection-10 (work in progress), July 2015.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

Authors' Addresses

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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

F. Fieau
Orange

March 21, 2016

HTTPS and delegation of encrypted traffic between interconnected CDNs
draft-fieau-https-delivery-delegation-02

Abstract

This document discusses 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

Copyright Notice

Table of Contents

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 for some reasons, 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 that are presented in the requirements section.

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.

2. Requirements

A trusted and secured delegation of the delivery of content hosted by an uCDN to a downstream CDN (dCDN) must be guaranteed in the CDN interconnection. Actually, this requirement must entail the following:

- o Legitimacy of redirection: the uCDN must wilfully designate the dCDN to deliver the requested content to ensure the delegation is legitimate. End-users must be warned about any issues in the delegation process, e.g. bad certificate. This is typically ensured by the browser (or player).

- o Revocation use case: A revocation might occur when a dCDN is not trusted anymore by the uCDN in case of policy or security concerns, i.e. the dCDN surrogates has unsolved security issues, or the delegation of HTTPS content delivery for a particular CP has been terminated. For instance, in case of revocation, a dCDN must not be able to present the CP or uCDN certificate to the user agent when UA has directly requested a content to the dCDN.

- o CDN configuration/topology visibility: Reduce CDN configuration, topology and meta-data disclosure/visibility towards the end-users. This means to 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; this may imply for instance to remove data in the redirection messages.

3. Redirection methods

In a secured CDN interconnection where uCDN 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 (please refer to [I-D.ietf-cdni-redirection]):

- o using an HTTP-based mechanism, the UA is redirected by the uCDN to the dCDN. 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 the dCDN domain name.

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

Next the UA 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.

3.1. HTTP-based redirection methods

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

3.1.1. 3xx directives

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

- 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.

- 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, the TLS handshake 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. The 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. Indeed, when tying two sessions together, the authenticated origin uCDN communicates the target URI in the redirection message over an encrypted tunnel, which constitutes a trust delegation endorsement.

However the issue here is when a uCDN invalidates the delegation to dCDN for some reasons. The dCDN will then still be able to stream the content with a valid certificate if an end-user directly requests content to it.

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.

3.1.2. 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 typically be 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.

In the case of HTTP Adaptive Streaming (HAS) techniques where content is 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).

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

3.1.3. API Mode, or scripted redirection

In the API mode (e.g. via AJAX requests, JSONP, etc.), a web page requested by the browser contains a script that "transparently" - from the user’s perspective - requests content on another web page.

As far as CDNI is concerned, the initial web page and scripts would be located on domain A, whereas content 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 security issues likewise in other HTTP-based redirection cases.

3.2. DNS redirection

In the case of DNS-based redirection, prior to any HTTP delivery requests, the UA has to first 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 redirection 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 domain 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 may compromise the delegation security.

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, upon querying the hostname associated with the uCDN URL, the DNS resolver will receive a DNS response (such as CNAME) that will direct the client to the dCDN. However, in an HTTPS environment, the client will end up with a domain different 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 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.

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 explicitly designated the dCDN as its delegate. However, DNSSEC adoption remains limited, 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.

4. Use of Lurk interfaces for CDNI

Delegating the delivery of HTTPS traffic to a third party without any certificate issues 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 and breaks the delegation revocation use cases.

The proposed solution here relies on the introduction of a Private Key Server in the TLS handshake as described in the Lurk draft [I-D.mglt-lurk-tls-use-cases] that suggests several use cases of private key server and protocols implementation.

Some solutions implementing private Key Servers are being standardized. For instance, the Session Key interface - SKI - draft [I-D.cairns-tls-session-key-interface] shows an example of Lurk interface implementation. Others are commercially deployed and are made publicly available.

A Lurk interface implementation for CDNI would allow the private keys to remain under the authority of the content provider (or the uCDN) while the actual content would be served from a dCDN surrogate that is closer to the end user. Indeed, the architecture would introduce a split in the setup of the secure tunnel between the client's browser and the surrogate delivering the content. Since the dCDN would not possess the private keys for the requested certificate, during the setup of the TLS tunnel between the client and the dCDN surrogate. The dCDN would in turn forward a request to get the necessary credentials to establish the secure connection to the end-user and serve the content.

Note that a Lurk interface would allow provisioning certificates to the dCDN and avoids any private keys exchanges between uCDN and the dCDN involved in the delegation. The implementation of Lurk need to be used jointly with DNS redirection.

4.1. Use cases

Two main use cases shall be considered when implementing Lurk in the CDNI context:

a. "Nominal" case for HTTPS delegation: a uCDN delegates the HTTPS content delivery to a dCDN without providing with its private keys for legal/trusting issues. In this case, only the uCDN would need a valid certificate, whereas the dCDN would rely on session key received from the uCDN Key Server (KS) for the TLS session establishment. Therefore the dCDN will deliver HTTPS content using the uCDN certificate.

b. Cascaded HTTPS delivery delegation: This use case has a wider scope that the previous use case. In the context, the content provider delegates the HTTPS content delivery of his content to an uCDN, that in turn delegates the HTTPS delivery to a dCDN. For security reasons, the CP do not want to provide his private keys to all CDNs involved in the interconnection hierarchy. In that case, the uCDN and all dCDN will rely on the CP private keys received from a Key Server (KS) on CP side to deliver HTTPS content.

4.2. Implementation

This section describes basic implementations of Lurk for CDNI secured traffic delegation (case a.). In the following example, the CP has provided his certificate to the uCDN.

As seen on figure 2, once the DNS resolution is done (i.e., UA is able to resolve dCDN surrogate IP@ steps a to d), the user agent connects to the dCDN surrogate. Also please note that DNS cache is not shown on the figure.

1. The client sends a ClientHello, as defined in the TLS protocol [RFC5246].

2. The surrogate sends a ServerHello.

3. The surrogate sends the public certificate to the user agent. The user agent checks the certificate signature with the public key.

In order to prepare the certificate delegation, it may be necessary to provision CP public certificates hosted on the Key Server, on uCDN or CP side, to the dCDN that will be requested the content. Please refer to fig. 3. A request is sent by the dCDN to uCDN KS to get the CP Public certificate.

4. The surrogate requests the key server to sign parameters with the private key.

5. The key server returns the signature to the dCDN surrogate over a secure tunnel.

6. and 7. the client and the surrogate exchange public DH parameters and compute session keys.

8. The end-user terminal and the surrogate 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:

```
 +----+               +----+            +--------+                +----+
 | UA |               |dCDN|            |uCDN KS|                 |uCDN|
 +----+               +----+            +--------+                +----+
   |a. DNS A? FQDN CP   |                   |                        |
   |---------------------------------------------------------------->|
   |                    |                   |                        |
   |b. DNS CNAME dCDN   |                   |                        |
   |<----------------------------------------------------------------|
   |                    |                   |                        |
   |c. DNS A? dCDN      |                   |                        |
   |------------------->|                   |                        |
   |                    |                   |                        |
   |d. DNS A IP Surrogate                   |                        |
   |<-------------------|                   |                        |
   |                    |                   |                        |
   |1. ClientHello (Client Random)          |                        |
   |------------------->|                   |                        |
   |                    |                   |                        |
   |2. ServerHello (Server Random)          |                        |
   |<-------------------|                   |                        |
   |                    |                   |                        |
   |3. Certificate (Server certificate)     |                        |
   |<-------------------|                   |                        |
   |                    |                   |                        |
   |                    |4. API request for Signature (ECDHHParams)  |
   |                    |------------------>|                        |
   |                    |                   |                        |
   |                    |5. API response: signature (ECDHHParams)    |
   |                    |<------------------|                        |
   |                    |                   |                        |
   |6. ServerKeyExchange (ECDHParams, Signature)                     |
   |<-------------------|                   |                        |
   |                    |                   |                        |
   |7. ClientKeyExchange (clientDHpublic)   |                        |
   |------------------->|                   |                        |
   |Finished            |                   |                        |
   |<------------------>|                   |                        |
   |                    |                   |                        |
   |8. HTTPS request    |                   |                        |
   |------------------->|                   |                        |
```

Figure 2: Overview of Lurk interface with DH handshake in case of regional delivery delegation

As shown on figure 4, a similar implementation can be considered for RSA keys handling.

Note that next after step 1, the dCDN may have to retrieve [at least once] the CP public certificate related to the targeted FQDN. This could be done using specific API methods dedicated to certificate retrieval. The certificates may be cached on the dCDN for a given duration. See next figure.

This Lurk API may have to support both RSA (cf. figure 4) and/or Diffie-Helman (cf. figure 2) connection setup.

```
 +----+                +----+            +--------+              +----+
 | UA |                |dCDN|            |uCDN KS|              |uCDN|
 +----+                +----+            +--------+              +----+
   |                     |                   |                       |
   |1. ClientHello (Client Random)          |                       |
   |-------------------->|                   |                       |
   |                     |                   |                       |
   |                     |a. API: Request FQDN CP public certificate  |
   |                     |------------------>|                       |
   |                     |                   |                       |
   |                     |b. API: CP Public certificate (cacheable)   |
   |                     |<------------------|                       |
   | ...                 |                   |                       |
```

Figure 3: Optional steps for a dCDN to get the uCDN public certificate (public certificates may be cached on the dCDN for a given duration)

```
+----+               +----+            +--------+                +----+
| UA |               |dCDN|            |uCDN KS|                 |uCDN|
+----+               +----+            +--------+                +----+
  |a. DNS A? FQDN CP   |                   |                        |
  |---------------------------------------------------------------->|
  |                    |                   |                        |
  |b. DNS CNAME dCDN   |                   |                        |
  |<----------------------------------------------------------------|
  |                    |                   |                        |
  |c. DNS A? dCDN      |                   |                        |
  |------------------->|                   |                        |
  |                    |                   |                        |
  |d. DNS A IP Surrogate                   |                        |
  |<-------------------|                   |                        |
  |                    |                   |                        |
  |1. ClientHello (Client Random)          |                        |
  |------------------->|                   |                        |
  |                    |                   |                        |
  |2. ServerHello (Server Random)          |                        |
  |<-------------------|                   |                        |
  |                    |                   |                        |
  |3. Certificate (Server certificate)     |                        |
  |<-------------------|                   |                        |
  |                    |                   |                        |
  |4. ClientKeyExchange (Encrypted Premaster Secret)                |
  |------------------->|                   |                        |
  |                    |5. API request for Signature + decrypt premaster secret
  |                    |------------------>|                        |
  |                    |                   |                        |
  |                    |6. API response: signature                  |
  |                    |<------------------|                        |
  |                    |                   |                        |
  |Finished            |                   |                        |
  |<------------------>|                   |                        |
```

Figure 4: Overview of Lurk implementation with RSA in case of regional delivery delegation

4.3. Discussions

- o HTTPS: currently, the contents are delivered with the credentials of uCDN or dCDN. The introduction of a key server in the CDNI architecture allows the HTTPS content to be delivered with the origin server certificate. It conforms to the end-to-end HTTPS framework [RFC2818].

- TLS: it does not significantly impact the TLS session establishment duration, while benefiting from TLS session resuming.

- Certificates: a certificate per CP would be stored and managed by the uCDN (or the CP) in a private KS is needed for the CDNI. Therefore it avoids the complexity of having multiple certificates and domain names, e.g. one domain name per CDN of the interconnection. As a side effect, it could also prevent certificates mismatches and help warning the user at user agent side, while ensuring the legitimacy of redirection.

- Security: the dCDN has never access to the uCDN (or CP) private keys, as it rely on the uCDN (or CP) certificate to setup the connection.

- Revocation: in the case of an HTTPS delegation revocation, a dCDN has no longer the delegation right to deliver a content for a given CP. The uCDN would then deny access to CP certificates, and therefore the dCDN would not be able to present the CP certificate to the UA.

- RSA: the use of RSA scheme is not recommended as the UA needs to share premaster secret key which can be a security flaw.

- Use cases: other use cases, e.g. the dCDN would manage its own keys/cert in a Key Server that would be requested by the uCDN, will be described in a further version of this draft.

5. CDNI URI Signing

Another means of enforcing trust delegation would be to use CDNI URI Signing mechanism. The CDNI URI Signing [I-D.leung-cdni-uri-signing] draft specifies a detailed mechanism to ensure the 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 mechanism 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. IANA Considerations

This document has no IANA considerations.

8. Security Considerations

The entire document is about security.

9. Acknowledgments

The authors would like to thank Iuniana Oprescu and Sergey Slovetskiy for the initial authoring of this draft.

Many thanks also to Jon Peterson, Jan Seedorf, and Ben Nivens-Jenkins for their help in putting this draft together.

10. References

10.1. Normative References

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

[RFC3568] Barbir, A., Cain, B., Nair, R., and O. Spatscheck, "Known Content Network (CN) Request-Routing Mechanisms", RFC 3568, DOI 10.17487/RFC3568, July 2003, <http://www.rfc-editor.org/info/rfc3568>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <http://www.rfc-editor.org/info/rfc6698>.

10.2. 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]
Barnes, R. and P. Saint-Andre, "High Assurance Re-Direction (HARD) Problem Statement", draft-barnes-hard-problem-00 (work in progress), July 2010.

[I-D.cairns-tls-session-key-interface]
Cairns, K., Mattsson, J., Skog, R., and D. Migault, "Session Key Interface (SKI) for TLS and DTLS", draft-cairns-tls-session-key-interface-01 (work in progress), October 2015.

[I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-17 (work in progress), February 2016.

[I-D.leung-cdni-uri-signing]
Leung, K., Faucheur, F., Downey, B., Brandenburg, R., and S. Leibrand, "URI Signing for CDN Interconnection (CDNI)", draft-leung-cdni-uri-signing-05 (work in progress), March 2014.

[I-D.mglt-lurk-tls-use-cases]
Migault, D. and K. Ma, "TLS/DTLS Content Provider Edge Server Split Use Case", draft-mglt-lurk-tls-use-cases-00 (work in progress), January 2016.

[Proposed_Lurk_Charter]
Eric Burger, "Proposed Lurk Charter", <https://mailarchive.ietf.org/arch/msg/lurk/tJBcaRJlYBGaOe1X6Hsc-smt4OU>.

[SSL-Challenges]
J. Clark and P. C. van Oorschot, "SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements", in 2013 IEEE Symposium on Security and Privacy (SP), 2013, pp. 511-525.

Author's Address

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

Email: frederic.fieau@orange.com

Network Working Group R. Murray
Internet-Draft B. Niven-Jenkins
Intended status: Standards Track Nokia
Expires: November 20, 2016 May 19, 2016

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

Abstract

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

Requirements Language

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

- o Section 2 outlines the model for the CI/T Interface at a high level.

- o Section 3 describes collections of Trigger Status Resources.

- o Section 4 defines the web service provided by the dCDN.

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

1.1. Terminology

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

2. Model for CDNI Triggers

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

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

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

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

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

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

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

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

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

```
   uCDN                                                       dCDN
     |    (1) POST https://dcdn.example.com/triggers/uCDN       |
    [ ] --------------------------------------------------> [ ]--+
     |                                                      [ ]  | (2)
     |    (3) HTTP 201 Response                             [ ]<-+
    [ ] <-------------------------------------------------- [ ]
     |     Loc: https://dcdn.example.com/triggers/uCDN/123    |
     |                                                        |
     .                            .                           .
     .                            .                           .
     .                            .                           .
     |                                                        |
     |   (4) GET https://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, "https://dcdn.example.com/triggers/uCDN". The URL of this was given to the uCDN when the CI/T interface was established.

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

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

4. The uCDN can poll, possibly repeatedly, the Trigger Status Resource in the dCDN.

5. The dCDN responds with the Trigger Status Resource, describing progress or results of the CI/T Trigger Command.

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

2.1. Timing of Triggered Activity

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

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

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

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

2.2. Scope of Triggered Activity

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

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

2.2.1. Multiple Interconnected CDNs

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

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

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

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

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

Security considerations are discussed further in Section 8.

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

- When the uCDN issues an invalidate CI/T command, a dCDN will receive that command from multiple directly connected uCDNs. The dCDN may schedule multiple those commands separately, and the last may affect content already revalidated following execution of the invalidate command scheduled first.

- If one of a dCDN's directly-connected uCDNs loses its rights to distribute content, it may issue a CI/T purge command. That purge may affect content the dCDN could retain because it's distributed by another directly-connected uCDN. But, that content can be re-acquired by the dCDN from the remaining uCDN.

- When the uCDN originating an item of content issues a CI/T purge followed by a preposition - two directly connected uCDNs will pass those commands to a dCDN. That dCDN implementation need not merge those operations, or notice the repetition. In which case the purge issued by one uCDN will complete before the other. The first uCDN to finish its purge may then forward the preposition trigger, and content pre-positioned as a result might be affected by the still-running purge issued by the other uCDN. However, the dCDN will re-acquire that content as needed, or when it’s asked to pre-position the content by the second uCDN. A dCDN implementation could avoid this interaction by knowing which uCDN it acquired the content from, or it could minimize the consequences by recording the time at which the invalidate/purge command was received and not applying it to content acquired after that time.

2.3. Trigger Results

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

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

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

3. Collections of Trigger Status Resources

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

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

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

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

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

- 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 resources. Therefore uCDNs MUST NOT make any assumptions regarding the structure of CI/T URIs or the mapping between CI/T objects and their associated URIs. URIs present in the examples in this document are purely illustrative and are not intended to impose a definitive structure on CI/T interface implementations.

4.1. Creating Triggers

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

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

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

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

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

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

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

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

4.2. Checking Status

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

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

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

4.2.1. Polling Trigger Status Resource collections

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

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

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

4.2.2. Polling Trigger Status Resources

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

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

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

4.3. Cancelling Triggers

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

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

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

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

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

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

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

4.4. Deleting Triggers

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

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

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

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

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

4.5. Expiry of Trigger Status Resources

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

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

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

4.6. Loop Detection and Prevention

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

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

The CDN Provider Id consists of the two characters "AS" followed by the CDN Provider’s Autonomous System number [RFC1930], then a colon (’:’) and an additional qualifier that is used to guarantee uniqueness in case a particular AS has multiple independent CDNs deployed. For example "AS64496:0".

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

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

4.7. Error Handling

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

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

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

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

- o If the CI/T Command is abandoned by the dCDN, the dCDN SHOULD report an error.

- o A CI/T "invalidate" command may be reported as "complete" when surrogates that may have the data are offline. In this case, surrogates MUST NOT use the affected data without first revalidating it when they are back online.

- o CI/T "preposition" and "purge" commands can be reported as "processed" if affected caches are offline and the activity will complete when they return to service.

- o Otherwise, the dCDN SHOULD keep the Trigger Status Resource in state "pending" or "active" until the CI/T Command is acted upon, or the uCDN chooses to cancel it.

4.8. Content URLs

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

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

5. CI/T Object Properties and Encoding

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

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

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

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

5.1. CI/T Objects

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

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

5.1.1. CI/T Commands

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

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

Name: trigger

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

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

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

Name: cancel

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

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

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

Name: cdn-path

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

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

Mandatory: Yes.

5.1.2. Trigger Status Resource

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

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

Name: trigger

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

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

Mandatory: Yes

Name: ctime

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

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: Yes

Name: mtime

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

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: Yes

Name: etime

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

Value: Absolute Time, as defined in Section 5.2.5.

Mandatory: No

Name: status

Description: Current status of the triggered activity.

Value: Trigger Status, as defined in Section 5.2.3.

Mandatory: Yes

Name: errors

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

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

Mandatory: No.

5.1.3. Trigger Collection

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

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

Name: triggers

Description: Links to Trigger Status Resources in the collection.

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

Mandatory: Yes

Name: staleresourcetime

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

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

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

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

Description: Link to a Trigger Collection.

Value: A URL represented as a JSON string.

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

Name: cdn-id

Description: The CDN Provider Identifier of the dCDN.

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

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

5.2. Properties of CI/T Objects

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

5.2.1. Trigger Specification

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

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

Name: type

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

Value: Trigger Type, as defined in Section 5.2.2.

Mandatory: Yes

Name: metadata.urls

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

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

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

Name: content.urls

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

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

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

Name: content.ccid

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

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

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

Name: metadata.patterns

Description: The metadata the trigger applies to.

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

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

Name: content.patterns

Description: The content data the trigger applies to.

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

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

5.2.2. Trigger Type

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

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

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

The following trigger types are defined by this document:

JSON String	Description
preposition	A request for the dCDN to acquire metadata or content.
invalidate	A request for the dCDN to invalidate metadata or content. After servicing this request the dCDN will not use the specified data without first re-validating it using, for example, an "If-None-Match" HTTP request. The dCDN need not erase the associated data.
purge	A request for the dCDN to erase metadata or content. After servicing the request, the specified data MUST NOT be held on the dCDN (the dCDN should re-acquire the metadata or content from uCDN if it needs it).

5.2.3. Trigger Status

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

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

5.2.4. PatternMatch

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

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

Name: pattern

Description: A pattern for URI matching.

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

Mandatory: Yes.

Name: case-sensitive

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

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

Mandatory: No, default is case-insensitive match.

Name: match-query-string

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

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

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

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

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

5.2.5. Absolute Time

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

5.2.6. Error Description

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

Name: error

Value: Error Code, as defined in Section 5.2.7.

Mandatory: Yes.

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

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

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

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

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

Name: description

Description: A human-readable description of the error.

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

Mandatory: No.

5.2.7. Error Code

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

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

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

6. Examples

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

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

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

6.1. Creating Triggers

Examples of the uCDN triggering activity in the dCDN:

6.1.1. Preposition

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

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

REQUEST:

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


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

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


RESPONSE:

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


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

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


6.1.2. Invalidate

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

REQUEST:

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


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

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

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


RESPONSE:

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

```
     Server: example-server/0.1

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

6.2. Examining Trigger Status

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

6.2.1. Collection of All Triggers

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

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 341
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "-936094426920308378"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```


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


6.2.2. Filtered Collections of Trigger Status Resources

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

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 152
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "4331492443626270781"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```


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


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

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 54
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "7958041393922269003"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

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

6.2.3. Individual Trigger Status Resources

The Trigger Status Resources can also be examined for detail about individual CI/T Trigger Commands. For example, for the CI/T "preposition" and "invalidate" commands from previous examples:

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 467
Expires: Wed, 04 May 2016 08:49:10 GMT
Server: example-server/0.1
ETag: "6990548174277557683"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:10 GMT
Content-Type: application/cdni; ptype=ci-trigger-status
```


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

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 545
Expires: Wed, 04 May 2016 08:49:11 GMT
Server: example-server/0.1
ETag: "-554385204989405469"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:11 GMT
Content-Type: application/cdni; ptype=ci-trigger-status
```


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


6.2.4. Polling for Change

The uCDN SHOULD use the Entity Tags of collections or Trigger Status Resources when polling for change in status, as shown in the following examples:

REQUEST:

```
GET /triggers/pending HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
If-None-Match: "4331492443626270781"
```

RESPONSE:

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

REQUEST:

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

RESPONSE:

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

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

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 54
Expires: Wed, 04 May 2016 08:49:15 GMT
Server: example-server/0.1
ETag: "1337503181677633762"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:15 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```


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

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 152
Expires: Wed, 04 May 2016 08:49:22 GMT
Server: example-server/0.1
ETag: "4481489539378529796"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:22 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection

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

6.2.5. Deleting Trigger Status Resources

The uCDN can delete completed and failed Trigger Status Resources to reduce the size of the collections, as described in Section 4.4. For example, to delete the "preposition" request from earlier examples:

REQUEST:

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

RESPONSE:

```
HTTP/1.1 204 No Content
Date: Wed, 04 May 2016 08:48:22 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Server: example-server/0.1
```

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

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 105
Expires: Wed, 04 May 2016 08:49:22 GMT
Server: example-server/0.1
ETag: "-6938620031669085677"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:22 GMT
Content-Type: application/cdni; ptype=ci-trigger-collection
```


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


6.2.6. Error Reporting

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

REQUEST:

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

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 486
Expires: Wed, 04 May 2016 08:49:26 GMT
Server: example-server/0.1
ETag: "5182824839919043757"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:26 GMT
Content-Type: application/cdni; ptype=ci-trigger-status
```


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


7. IANA Considerations

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

7.1. CDNI Payload Type Parameter Registrations

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

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

7.2. CDNI CI/T Trigger Types Registry

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

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

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

7.3. CDNI CI/T Error Codes Registry

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

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

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

8. Security Considerations

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

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

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

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

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

8.1. Authentication, Authorization, Confidentiality, Integrity Protection

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

TLS MUST be used by the server-side (dCDN) and the client-side (uCDN) of the CI/T interface, including authentication of the remote end, unless alternate methods are used for ensuring the security of the information in the CI/T interface requests and responses (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

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

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

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

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

- o CDNI commands and responses to be transmitted with confidentiality.

- o Protection of the integrity of CDNI commands and responses.

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

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

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

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

8.2. Denial of Service

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

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

8.3. Privacy

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

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

9. Acknowledgements

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

10. References

10.1. Normative References

[I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-16 (work in progress), April 2016.

[RFC1930] Hawkinson, J. and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)", BCP 6, RFC 1930, DOI 10.17487/RFC1930, March 1996, <http://www.rfc-editor.org/info/rfc1930>.

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7232] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, June 2014.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015.

10.2. Informative References

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-08 (work in progress), March 2016.

[I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-18 (work in progress), April 2016.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, August 2014.

[RFC7337] Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, August 2014.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <http://www.rfc-editor.org/info/rfc7736>.

Appendix A. Formalization of the JSON Data

This appendix is non-normative.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

 URI = tstr
```

Authors' Addresses

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

Email: rob.murray@nokia.com

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

Email: ben.niven-jenkins@nokia.com

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

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

May 20, 2016

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

Abstract

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

Requirements Language

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

Status of This Memo

This Internet-Draft will expire on November 21, 2016.

Copyright Notice

Table of Contents

1. Introduction and Scope

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

The goal of this document is to achieve a clear understanding about the semantics associated with the CDNI Request Routing Footprint & Capabilities Advertisement Interface (from now on referred to as FCI), in particular the type of information a downstream CDN (dCDN) 'advertises' regarding its footprint and capabilities. To narrow down undecided aspects of these semantics, this document tries to establish a common understanding of what the FCI needs to offer and accomplish in the context of CDNI.

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

General assumptions in this document:

- The CDNs participating in the interconnected CDN have already performed a boot strap process, i.e., they have connected to each other, either directly or indirectly, and can exchange information amongst each other.

- The upstream CDN (uCDN) receives footprint and/or capability advertisements from a set of dCDNs. Footprint advertisement and capability advertisement need not use the same underlying protocol.

- The uCDN receives the initial request-routing request from the endpoint requesting the resource.

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

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

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

Additionally, the following terms are used throughout this document and are defined as follows:

- Footprint: a description of a CDN's coverage area, i.e., the area from which client requests may originate for, and to which the CDN is willing to deliver, content. Note: There are many ways to describe a footprint, for example, by address range (e.g., IPv4/IPv6 CIDR), by network ID (e.g., ASN), by nation boundaries (e.g., country code), by GPS coordinates, etc. This document does not define or endorse the quality or suitability of any particular footprint description method; this document only defines a method for transporting known footprint descriptions in Footprint and Capabilities Advertisement messages.

- Capability: a feature of a dCDN, upon which a uCDN relies on the dCDN supporting, when making delegation decisions. Support for a given feature can change over time and can be restricted to a limited portion of a dCDN's footprint. Note: There are many possible dCDN features that could be of interest to a uCDN. This document does not presume to define them all; this document describes a scheme for defining new capabilities and how to transport them in Footprint and Capabilities Advertisement messages.

2. Design Decisions for Footprint and Capabilities

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

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

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

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

2.1. Advertising Limited Coverage

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

When such a purpose-built CDN interconnects with other CDNs and advertises its footprint to a uCDN, however, the original intended coverage of the CDN might not represent its actual value to the interconnection of CDNs. Consider an ISP-A and ISP-B that both field their own CDNs, which they interconnect via CDNI. A given user E, who is a customer of ISP-B, might happen to be topologically closer to a surrogate fielded by ISP-A, if E happens to live in a region where ISP-B has few customers and ISP-A has many. In this case, is it ISP-A's CDN that "covers" E? If ISP-B's CDN has a failure condition, is it up to the uCDN to understand that ISP-A's surrogates are potentially available as back-ups - and if so, how does ISP-A advertise itself as a "standby" for E? What about the case where CDNs advertising to the same uCDN express overlapping coverage (for example, mixing global and limited CDNs)?

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

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

2.2. Capabilities and Dynamic Data

In cases where the apparent footprints of dCDNs overlap, uCDNs might also want to rely on other factors to evaluate the respective merits of dCDNs. These include facts related to the surrogates themselves, to the network where the surrogate is deployed, to the nature of the resource sought, and to the administrative policies of the respective networks.

In the absence of network-layer impediments to reaching surrogates, the choice to limit coverage is necessarily an administrative policy. Much policy needs to be agreed upon before CDNs can interconnect, including questions of membership, compensation, volumes, and so on. A uCDN certainly will factor these sorts of considerations into its decision to select a dCDN, but there is probably little need for dCDNs to actually advertise them through an interface - they will be settled out-of-band as a precondition for interconnection.

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

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

2.3. Advertisement versus Queries

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

Were the volume of FCI updates from dCDNs to exceed the volume of requests to the uCDN, it might make more sense for the uCDN to query dCDNs upon receiving requests (as is the case in the recursive redirection mode described in [RFC7336]), instead of receiving advertisements and tracking the state of dCDNs. The advantage of querying dCDNs would be that much of the dynamic data that dCDNs cannot share with the uCDN would now be factored into the uCDN's decision. dCDNs need not replicate any state to the uCDN - uCDNs could effectively operate in a stateless mode.

The semantics of both footprint and capability advertisement depend on the service model here: are there cases where a synchronous query/response model would work better for the uCDN decision than a state replication model?

2.4. Avoiding or Handling 'cheating' dCDNs

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

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

3. Focusing on Capabilities with Footprint Restrictions

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

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

4. Footprint and Capabilities Extension

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

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

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

- Acquisition Protocol (for acquiring content from the uCDN or origin server)

- 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)

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

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

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

- s-ccid

- s-sid

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

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

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

5. Capability Advertisement Object

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

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

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

5.1. Base Advertisement Object

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

Property: capability-type

Description: CDNI Capability object type.

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

Mandatory-to-Specify: Yes.

Property: capability-value

Description: CDNI Capability object.

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

Mandatory-to-Specify: Yes.

Property: footprints

Description: CDNI Capability Footprint.

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

Mandatory-to-Specify: No.

5.2. Encoding

CDNI FCI objects MUST be encoded using JSON [RFC7159] and MUST also follow the recommendations of I-JSON [RFC7493]. FCI objects are composed of a dictionary of (key,value) pairs where the keys are the property names and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e., dependent on the CDNI Payload Type of the capability or the CDNI Metadata Footprint Type of the footprint). Likewise, the values associated with each property (dictionary key) are dependent on the specific object being encoded (i.e., dependent on the CDNI Payload Type of the capability or the CDNI Metadata Footprint Type of the footprint).

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

5.3. Delivery Protocol Capability Object

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

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

Property: delivery-protocols

Description: List of supported CDNI Delivery Protocols.

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

Mandatory-to-Specify: Yes.

5.3.1. Delivery Protocol Capability Object Serialization

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


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


5.4. Acquisition Protocol Capability Object

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

Property: acquisition-protocols

Description: List of supported CDNI Acquisition Protocols.

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

Mandatory-to-Specify: Yes.

5.4.1. Acquisition Protocol Capability Object Serialization

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


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


5.5. Redirection Mode Capability Object

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

Property: redirection-modes

Description: List of supported CDNI Redirection Modes.

Type: List of Redirection Modes (from Section 6.2)

Mandatory-to-Specify: Yes.

5.5.1. Redirection Mode Capability Object Serialization

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

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


5.6. CDNI Logging Capability Object

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

Property: record-type

Description: Supported CDNI Logging record-type.

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

Mandatory-to-Specify: Yes.

Property: fields

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

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

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

5.6.1. CDNI Logging Capability Object Serialization

The following shows an example of CDNI Logging Capability Object Serialization, for a CDN that supports the optional Content Collection ID logging field (but not the optional Session ID logging field) for the "cdni_http_request_v1" record type.


```
{
  "capabilities": [
    {
      "capability-type": "FCI.Logging",
      "capability-value": {
        "record-type": "cdni_http_request_v1",
        "fields": [ "s-ccid" ]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```


The next example shows the CDNI Logging Capability Object Serialization, for a CDN that supports all optional fields for the "cdni_http_request_v1" record type.


```
{
  "capabilities": [
    {
      "capability-type": "FCI.Logging",
      "capability-value": {
        "record-type": "cdni_http_request_v1"
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```


5.7. CDNI Metadata Capability Object

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

Property: metadata

Description: List of supported CDNI GenericMetadata types.

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

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

5.7.1. CDNI Metadata Capability Object Serialization

The following shows an example of CDNI Metadata Capability Object Serialization, for a CDN that supports only the SourceMetadata GenericMetadata type (i.e., it can acquire and deliver content, but cannot enforce and security policies, e.g., time, location, or protocol ACLs).


```
{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": ["MI.SourceMetadata"]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```


The next example shows the CDNI Metadata Capability Object Serialization, for a CDN that supports only structural metadata (i.e., it can parse metadata as a transit CDN, but cannot enforce security policies or deliver content).

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": []
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

6. IANA Considerations

6.1. CDNI Payload Types

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

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

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

6.1.1. CDNI FCI DeliveryProtocol Payload Type

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

Interface: FCI

Encoding: see Section 5.3

6.1.2. CDNI FCI AcquisitionProtocol Payload Type

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

Interface: FCI

Encoding: see Section 5.4

6.1.3. CDNI FCI RedirectionMode Payload Type

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

Interface: FCI

Encoding: see Section 5.5

6.1.4. CDNI FCI Logging Payload Type

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

Interface: FCI

Encoding: see Section 5.6

6.1.5. CDNI FCI Metadata Payload Type

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

Interface: FCI

Encoding: see Section 5.7

6.2. Redirection Mode Registry

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

The following table defines the initial Redirection Modes:

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

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

7. Security Considerations

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

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

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

As specified in this document, the security requirements of the FCI could be met by transport-layer security mechanisms coupled with domain certificates as credentials (e.g., TLS transport for HTTP as per [RFC2818] and [RFC7230], with usage guidance from [RFC7525]) between CDNs. There is no apparent need for further object-level security in this framework, as the trust relationships it defines are bilateral relationships between uCDNs and dCDNs rather than transitive relationships.

8. References

8.1. Normative References

[I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-25 (work in progress), April 2016.

[I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-16 (work in progress), April 2016.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <http://www.rfc-editor.org/info/rfc7336>.

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

8.2. Informative References

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

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <http://www.rfc-editor.org/info/rfc6707>.

[RFC6770] Bertrand, G., Ed., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, DOI 10.17487/RFC6770, November 2012, <http://www.rfc-editor.org/info/rfc6770>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <http://www.rfc-editor.org/info/rfc7337>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <http://www.rfc-editor.org/info/rfc7736>.

Appendix A. Main Use Case to Consider

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

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

Appendix B. Semantics for Footprint Advertisement

Roughly speaking, "footprint" can be defined as "ability and willingness to serve" by a dCDN. However, in addition to simple "ability and willingness to serve", the uCDN could want additional information to make a dCDN selection decision, e.g., "how well" a given dCDN can actually serve a given end user request. The "ability and willingness" to serve SHOULD be distinguished from the subjective qualitative measurement of "how well" it was served. One can imagine that such additional information is implicitly associated with a given footprint, due to contractual agreements, SLAs, business relationships, or past perceptions of dCDN quality. As an alternative, such additional information could also be explicitly tagged along with the footprint.

It is reasonable to assume that a significant part of the actual footprint advertisement will happen in contractual agreements between participating CDNs, prior to the advertisement phase using the CDNI FCI. The reason for this assumption is that any contractual agreement is likely to contain specifics about the dCDN coverage (footprint) to which the contractual agreement applies. In particular, additional information to judge the delivery quality associated with a given dCDN footprint might be defined in contractual agreements, outside of the CDNI FCI. Further, one can assume that dCDN contractual agreements about the delivery quality associated with a given footprint will probably be based on high-level aggregated statistics and not too detailed.

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

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

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

- o Footprint could be defined based on "resources", where resources refers to surrogates a dCDN claims to have (e.g., the location of surrogates/resources). The dCDN claims that 'from this footprint' it can serve incoming end user requests.

For each of these footprint types, there are capabilities associated with a given footprint:

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

- o capabilities of resources, such as delivery protocol, redirection mode, and metadata, which apply to a "resource" defined footprint.

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

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

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

Appendix C. Semantics for Capabilities Advertisement

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

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

At a first glance, several broad categories of capabilities seem useful to convey via an advertisement interface, however, advertising capabilities that change highly dynamically (e.g., real-time delivery performance metrics, CDN resource load, or other highly dynamically changing QoS information) is beyond the scope for CDNI FCI. First, out of the multitude of possible metrics and capabilities, it is hard to agree on a subset and the precise metrics to be used. Second, it seems infeasible to specify such highly dynamically changing capabilities and the corresponding metrics within a reasonable time-frame.

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

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

Given these semantics, it needs to be decided what exact capabilities are useful and how these can be expressed. Since the details of CDNI contracts are not known at the time of this writing (and the CDNI interface are better off being agnostic to these contracts anyway), it remains to be seen what capabilities will be used to define agreements between CDNs in practice. One implication for standardization could be to initially only specify a very limited set of mandatory capabilities for advertisement and have on top of that a flexible data model that allows exchanging additional capabilities when needed. Still, agreement needs to be found on which capabilities (if any) will be mandatory among CDNs.

It is not feasible to enumerate all the possible options for the mandatory capabilities listed above (e.g., all the potential delivery protocols or metadata options) or anticipate all the future needs for additional capabilities. It would be unreasonable to burden the CDNI FCI specification with defining each supported capability. Instead, the CDNI FCI specification should define a generic protocol for conveying any capability information (e.g. with common encoding, error handling, and security mechanism; further requirements for the CDNI FCI Advertisement Interface are listed in [RFC7337]).

Appendix D. Acknowledgment

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

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

Authors' Addresses

Jan Seedorf
NEC
Kurfuerstenanlage 36
Heidelberg 69115
Germany

Phone: +49 6221 4342 221
Fax: +49 6221 4342 155
Email: seedorf@neclab.eu

Jon Peterson
NeuStar
1800 Sutter St Suite 570
Concord CA 94520
USA

Email: jon.peterson@neustar.biz

Stefano Previdi
Cisco Systems
Via Del Serafico 200
Rome 0144
Italy

Email: sprevidi@cisco.com

Ray van Brandenburg
TNO
Brassersplein 2
Delft 2612CT
The Netherlands

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

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

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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: December 10, 2016

F. Le Faucheur, Ed.
G. Bertrand, Ed.
I. Oprescu, Ed.
R. Peterkofsky
Google Inc.
June 8, 2016

CDNI Logging Interface
draft-ietf-cdni-logging-27

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This memo specifies the CDNI Logging interface between a downstream CDN (dCDN) and an upstream CDN (uCDN). First, it describes a reference model for CDNI logging. Then, it specifies the CDNI Logging File format and the actual protocol for exchange of CDNI Logging Files.

The reader should be familiar with the following documents:

- o CDNI problem statement [RFC6707] and framework [RFC7336] identify a Logging interface,
- o Section 8 of [RFC7337] specifies a set of requirements for Logging,
- o [RFC6770] outlines real world use-cases for interconnecting CDNs. These use cases require the exchange of Logging information between the dCDN and the uCDN.

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

The present document describes:

- o The CDNI Logging reference model (Section 2),
- o The CDNI Logging File format (Section 3),
- o The CDNI Logging File Exchange protocol (Section 4).

1.1. Terminology

In this document, the first letter of each CDNI-specific term is capitalized. We adopt the terminology described in [RFC6707] and [RFC7336], and extend it with the additional terms defined below.

Intra-CDN Logging information: logging information generated and collected within a CDN. The format of the Intra-CDN Logging information may be different to the format of the CDNI Logging information.

CDNI Logging information: logging information exchanged across CDNs using the CDNI Logging Interface.

Logging information: logging information generated and collected within a CDN or obtained from another CDN using the CDNI Logging Interface.

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

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

CDNI Logging File: a file containing CDNI Logging Records, as well as additional information facilitating the processing of the CDNI Logging Records.

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

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

1.2. Requirements Language

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

2. CDNI Logging Reference Model

2.1. CDNI Logging interactions

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

- 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 CDNI 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.

```
                +-----+
                | CSP |
                +-----+
                   ^ Reporting and monitoring data
                   * Billing
                ,--*--.
    Logging  ,-'       '-.
    Data  =>(     uCDN    )<=   Logging
       //   '-.       _,-'   \\  Data
       ||      '-'-'-'      ||
    ,-----.                 ,-----.
 ,-'       '-.           ,-'       '-.
(   dCDN-1    )         (   dCDN-2    )<==  Logging
 '-.       ,-'          '-.      _,-'    \\ Data
   '--'--'                  '--'-'        ||
                                       ,-----.
                                     ,'       '-.
                                    (  dCDN-3    )
                                     '.       ,-'
                                       '--'--'

===> CDNI Logging Interface
***> outside the scope of CDNI
```

Figure 1: Interactions in CDNI Logging Reference Model

A downstream CDN relative to uCDN (e.g., dCDN-2) integrates the relevant Logging information obtained from its own downstream CDNs (i.e., dCDN-3) in the Logging information that it provides to the uCDN, so that the uCDN ultimately obtains all Logging information relevant to a CSP for which it acts as the authoritative CDN. Such aggregation is further discussed in Section 3.7.

Note that the format of Logging information that a CDN provides over the CDNI interface might be different from the one that the CDN uses internally. In this case, the CDN needs to reformat the Logging information before it provides this information to the other CDN over the CDNI Logging interface. Similarly, a CDN might reformat the Logging information that it receives over the CDNI Logging interface before injecting it into its log-consuming applications or before providing some of this Logging information to the CSP. Such reformatting operations introduce latency in the logging distribution chain and introduce a processing burden. Therefore, there are benefits in specifying CDNI Logging formats that are suitable for use inside CDNs and also are close to the intra-CDN Logging formats commonly used in CDNs today.

2.2. Overall Logging Chain

This section discusses the overall logging chain within and across CDNs to clarify how CDN Logging information is expected to fit in this overall chain. Figure 2 illustrates the overall logging chain within the dCDN, across CDNs using the CDNI Logging interface and within the uCDN. Note that the logging chain illustrated in the Figure is obviously only an example and varies depending on the specific environments. For example, there may be more or fewer instantiations of each entity (e.g., there may be 4 Log consuming applications in a given CDN). As another example, there may be one instance of Rectification process per Log Consuming Application instead of a shared one.

```
             Log Consuming    Log Consuming
                 App              App
                 ^                ^
                 |                |
           Rectification----------
           ^
           |
           Filtering
            ^
            |
        Collection
        ^        ^
        |        |
        |     Generation
        |
        |                                                     uCDN
   CDNI Logging ---------------------------------------------------
   exchange                                                   dCDN
        ^
        |          Log Consuming    Log Consuming
        |                 App              App
        |                  ^                ^
        |                  |                |
   Rectification     Rectification---------
           ^        ^
           |        |
           Filtering
            ^
            |
         Collection
         ^        ^
         |        |
   Generation    Generation
```

Figure 2: CDNI Logging in the overall Logging Chain

The following subsections describe each of the processes potentially involved in the logging chain of Figure 2.

2.2.1. Logging Generation and During-Generation Aggregation

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

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

[RFC6983] discusses logging for HTTP Adaptive Streaming (HAS). In accordance with the recommendations articulated there, it is expected that a surrogate will generate separate Logging information for delivery of each chunk of HAS content. This ensures that separate Logging information can then be provided to interconnected CDNs over the CDNI Logging interface. Still in line with the recommendations of [RFC6983], the Logging information for per-chunk delivery may include some information (a Content Collection IDentifier and a Session IDentifier) intended to facilitate subsequent post-generation aggregation of per-chunk logs into per-session logs. Note that a CDN may also elect to generate aggregate per-session logs when performing HAS delivery, but this needs to be in addition to, and not instead of, the per-chunk delivery logs. We note that aggregate per-session logs for HAS delivery are for further study and outside the scope of this document.

2.2.2. Logging Collection

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

In a CDNI environment, in addition to collecting Logging information from log-generating entities within the local CDN, the Collection process also collects Logging information provided by another CDN, or other CDNs, through the CDNI Logging interface. This is illustrated in Figure 2 where we see that the Collection process of the uCDN collects Logging information from log-generating entities within the uCDN as well as Logging information coming from the dCDNs through the CDNI Logging interface.

2.2.3. Logging Filtering

A CDN may be required to only present different subsets of the whole Logging information collected to various log-consuming applications. This is achieved by the Filtering process.

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

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

In some use cases described in [RFC6770], the interconnected CDNs do not want to disclose details on their internal topology. The filtering process can then also filter confidential data on the dCDNs' topology (number of servers, location, etc.). In particular, information about the requests served by each Surrogate may be confidential. Therefore, the Logging information needs to be protected so that data such as Surrogates' hostnames are not disclosed to the uCDN. In the "Inter-Affiliates Interconnection" use case, this information may be disclosed to the uCDN because both the dCDN and the uCDN are operated by entities of the same group.

2.2.4. Logging Rectification and Post-Generation Aggregation

If Logging information is generated periodically, it is important that the sessions that start in one Logging period and end in another are correctly reported. If they are reported in the starting period, then the Logging information of this period will be available only after the end of the session, which delays the Logging information generation. A simple approach is to provide the complete Logging Record for a session in the Logging Period of the session end.

A Logging rectification/update mechanism could be useful to reach a good trade-off between the Logging information generation delay and the Logging information accuracy.

In the presence of HAS, some log-consuming applications can benefit from aggregate per-session logs. For example, for analytics, per-session logs allow display of session-related trends which are much more meaningful for some types of analysis than chunk-related trends. In the case where aggregate logs have been generated directly by the log-generating entities, those can be used by the applications. In the case where aggregate logs have not been generated, the Rectification process can be extended with a Post-Generation Aggregation process that generates per-session logs from the per-chunk logs, possibly leveraging the information included in the per-chunk logs for that purpose (Content Collection IDentifier and a Session IDentifier). However, in accordance with [RFC6983], this document does not define exchange of such aggregate logs on the CDNI Logging interface. We note that this is for further study and outside the scope of this document.

2.2.5. Log-Consuming Applications

2.2.5.1. Maintenance/Debugging

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

To detect faults, Logging information needs to report success and failure of CDN delivery operations. The uCDN can summarize such information into KPIs. For instance, Logging information needs to allow the computation of the number of times, during a given time period, that content delivery related to a specific service succeeds/fails.

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

Some of these maintenance and debugging applications only require aggregate logging information highly compatible with use of anonymization of IP addresses (as supported by the present document and specified in the definition of the c-groupid field under Section 3.4.1). However, in some situations, it may be useful, where compatible with privacy protection, to access some CDNI Logging Records containing full non-anonymized IP addresses. This is allowed in the definition of the c-groupid (under Section 3.4.1), with very significant privacy protection limitations that are discussed in the definition of the c-groupid field. For example, this may be useful for detailed fault tracking of a particular end user content delivery issue. Where there is a hard requirement by uCDN or CSP to associate a given enduser to individual CDNI Logging Records (e.g., to allow a-posteriori analysis of individual delivery for example in situations of performance-based penalties), instead of using aggregates containing a single client as discussed in the c-groupid field definition, an alternate approach is to ensure that a client identifier is embedded in the request fields that can be logged in a CDNI Logging Record (for example by including the client identifier in the URI query string or in a HTTP Header). That latter approach offers two strong benefits: first, the aggregate inside the c-groupid can contain more than one client, thereby ensuring stronger privacy protection; second, it allows a reliable identification of the client while IP address does not in many situations (e.g., behind NAT, where dynamic IP addresses are used and reused,...). However, care SHOULD be taken that the client identifiers exposed in other fields of the CDNI Records cannot themselves be linked back to actual users.

2.2.5.2. Accounting

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

2.2.5.3. Analytics and Reporting

The goals of analytics include gathering any relevant information in order to be able to develop statistics on content download, analyze user behavior, and monitor the performance and quality of content delivery. For instance, Logging information enables the CDN providers to report on content consumption (e.g., delivered sessions per content) in a specific geographic area.

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

2.2.5.4. Content Protection

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

2.2.5.5. Notions common to multiple Log Consuming Applications

2.2.5.5.1. Logging Information Views

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

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

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

2.2.5.5.2. Key Performance Indicators (KPIs)

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

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

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

Here is a list of important examples of KPIs:

- o Number of delivery requests received from End Users in a given region for each piece of content, during a given period of time (e.g., hour/day/week/month)
- o Percentage of delivery successes/failures among the aforementioned requests
- o Number of failures listed by failure type (e.g., HTTP error code) for requests received from End Users in a given region and for each piece of content, during a given period of time (e.g., hour/day/week/month)

- 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)

- 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)

- 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)

- 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)

- 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)

- Top 10 most popularly requested contents (during a given day/week/month)

- Terminal type (mobile, PC, STB, if this information can be acquired from the browser type inferred from the User Agent string, for example).

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

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

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

3. CDNI Logging File

3.1. Rules

This specification uses the Augmented Backus-Naur Form (ABNF) notation and core rules of [RFC5234]. In particular, the present document uses the following rules from [RFC5234]:

```
CR = %x0D ; carriage return

ALPHA = %x41-5A / %x61-7A ; A-Z / a-z

DIGIT = %x30-39 ; 0-9

DQUOTE = %x22 ; " (Double Quote)

CRLF = CR LF ; Internet standard newline

HEXDIG = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

HTAB = %x09 ; horizontal tab

LF = %x0A ; linefeed

VCHAR = %x21-7E ; visible (printing) characters

OCTET = %x00-FF ; 8 bits of data
```

The present document also uses the following rules from [RFC3986]:

```
host = as specified in section 3.2.2 of [RFC3986].

IPv4address = as specified in section 3.2.2 of [RFC3986].

IPv6address = as specified in section 3.2.2 of [RFC3986].

partial-time = as specified in [RFC3339].
```

The present document also defines the following additional rules:

```
ADDRESS = IPv4address / IPv6address

ALPHANUM = ALPHA / DIGIT

DATE = 4DIGIT "-" 2DIGIT "-" 2DIGIT

   ; Dates are encoded as "full-date" specified in [RFC3339].
```

```
DEC = 1*DIGIT ["." 1*DIGIT]

NAMEFORMAT = ALPHANUM *(ALPHANUM / "_" / "-")

QSTRING = DQUOTE *(NDQUOTE / PCT-ENCODED) DQUOTE

NDQUOTE = %x20-21 / %x23-24 / %x26-7E / UTF8-2 / UTF8-3 / UTF8-4

   ; whereby a DQUOTE is conveyed inside a QSTRING unambiguously
   by escaping it with PCT-ENCODED.

PCT-ENCODED = "%" HEXDIG HEXDIG

   ; percent encoding is used for escaping octets that might be
   possible in HTTP headers such as bare CR, bare LF, CR LF, HTAB,
   SP or null.  These octets are rendered with percent encoding in
   ABNF as specified by [RFC3986] in order to avoid considering
   them as separators for the logging records.

NHTABSTRING = 1*(SP / VCHAR)

TIME = partial-time

USER-COMMENT = * (SP / VCHAR / UTF8-2 / UTF8-3 / UTF8-4)
```

3.2. CDNI Logging File Structure

As defined in Section 1.1: a CDNI Logging Field is as an atomic logging information element, a CDNI Logging Record is a collection of CDNI Logging fields containing all logging information corresponding to a single logging event, and a CDNI Logging File contains a collection of CDNI Logging Records. This structure is illustrated in Figure 3. The use of a file structure for transfer of CDNI Logging information is selected since this is the most common practise today for exchange of logging information within and across CDNs.

```
+------------------------------------------------------------+
|CDNI Logging File                                           |
|                                                            |
| #Directive 1                                               |
| #Directive 2                                               |
| ...                                                        |
| #Directive P                                               |
|                                                            |
| +--------------------------------------------------------+ |
| |CDNI Logging Record 1                                   | |
| |  +-------------+ +-------------+     +-------------+   | |
| |  |CDNI Logging | |CDNI Logging | ... |CDNI Logging |   | |
| |  |   Field 1   | |   Field 2   |     |   Field N   |   | |
| |  +-------------+ +-------------+     +-------------+   | |
| +--------------------------------------------------------+ |
|                                                            |
| +--------------------------------------------------------+ |
| |CDNI Logging Record 2                                   | |
| |  +-------------+ +-------------+     +-------------+   | |
| |  |CDNI Logging | |CDNI Logging | ... |CDNI Logging |   | |
| |  |   Field 1   | |   Field 2   |     |   Field N   |   | |
| |  +-------------+ +-------------+     +-------------+   | |
| +--------------------------------------------------------+ |
|                                                            |
|  ...                                                       |
|                                                            |
| #Directive P+1                                             |
|                                                            |
|  ...                                                       |
|                                                            |
| +--------------------------------------------------------+ |
| |CDNI Logging Record M                                   | |
| |  +-------------+ +-------------+     +-------------+   | |
| |  |CDNI Logging | |CDNI Logging | ... |CDNI Logging |   | |
| |  |   Field 1   | |   Field 2   |     |   Field N   |   | |
| |  +-------------+ +-------------+     +-------------+   | |
| +--------------------------------------------------------+ |
|                                                            |
|                                                            |
| #Directive P+Q                                             |
+------------------------------------------------------------+
```

Figure 3: Structure of Logging Files

The CDNI Logging File format is inspired from the W3C Extended Log File Format [ELF]. However, it is fully specified by the present document. Where the present document differs from the W3C Extended

Log File Format, an implementation of the CDNI Logging interface MUST comply with the present document. The W3C Extended Log File Format was used as a starting point, reused where possible and expanded when necessary.

Using a format that resembles the W3C Extended Log File Format is intended to keep CDNI logging format close to the intra-CDN Logging information format commonly used in CDNs today, thereby minimizing systematic translation at CDN/CDNI boundary.

A CDNI Logging File MUST contain a sequence of lines containing US-ASCII characters [CHAR_SET] terminated by CRLF. Each line of a CDNI Logging File MUST contain either a directive or a CDNI Logging Record.

Directives record information about the CDNI Logging process itself. Lines containing directives MUST begin with the "#" character. Directives are specified in Section 3.3.

Logging Records provide actual details of the logged event. Logging Records are specified in Section 3.4.

The CDNI Logging File has a specific structure. It always starts with a directive line and the first directive it contains MUST be the version.

The directive lines form together a group that contains at least one directive line. Each directives group is followed by a group of logging records. The records group contains zero or more actual logging record lines about the event being logged. A record line consists of the values corresponding to all or a subset of the possible Logging fields defined within the scope of the record-type directive. These values MUST appear in the order defined by the fields directive.

Note that future extensions MUST be compliant with the previous description. The following examples depict the structure of a CDNILOGFILE as defined currently by the record-type "cdni_http_request_v1."

```
DIRLINE = "#" directive CRLF

DIRGROUP = 1*DIRLINE

RECLINE = <any subset of record values that match what is expected
according to the fields directive within the immediately preceding
DIRGROUP>
```

```
RECGROUP = *RECLINE

CDNILOGFILE = 1*(DIRGROUP RECGROUP)
```

3.3. CDNI Logging Directives

A CDNI Logging directive line contains the directive name followed by ":" HTAB and the directive value.

Directive names MUST be of the format NAMEFORMAT. All directive names MUST be registered in the CDNI Logging Directives Names registry. Directive names are case-insensitive as per the basic ABNF([RFC5234]). Unknown directives MUST be ignored. Directive values can have various formats. All possible directive values for the record-type "cdni_http_request_v1" are further detailed in this section.

The following example shows the structure of a directive and enumerates strictly the directive values presently defined in the version "cdni/1.0" of the CDNI Logging File.

```
directive = DIRNAME ":" HTAB DIRVAL

DIRNAME = NAMEFORMAT

DIRVAL = NHTABSTRING / QSTRING / host / USER-COMMENT / FIENAME *
(HTAB FIENAME) / 64HEXDIG
```

An implementation of the CDNI Logging interface MUST support all of the following directives, listed below by their directive name:

- version:

  * format: NHTABSTRING

  * directive value: indicates the version of the CDNI Logging File format. The entity transmitting a CDNI Logging File as per the present document MUST set the value to "cdni/1.0". In the future, other versions of CDNI Logging File might be specified; those would use a value different to "cdni/1.0" allowing the entity receiving the CDNI Logging File to identify the corresponding version. CDNI Logging File versions are case-insensitive as per the basic ABNF([RFC5234]).

  * occurrence: there MUST be one and only one instance of this directive per CDNI Logging File. It MUST be the first line of the CDNI Logging File.

   * example: "version: HTAB cdni/1.0".

o UUID:

   * format: NHTABSTRING

   * directive value: this a Uniform Resource Name (URN) from the Universally Unique IDentifier (UUID) URN namespace specified in [RFC4122]). The UUID contained in the URN uniquely identifies the CDNI Logging File.

   * occurrence: there MUST be one and only one instance of this directive per CDNI Logging File.

   * example: "UUID: HTAB NHTABSTRING".

o claimed-origin:

   * format: host

   * directive value: this contains the claimed identification of the entity transmitting the CDNI Logging File (e.g., the host in a dCDN supporting the CDNI Logging interface) or the entity responsible for transmitting the CDNI Logging File (e.g., the dCDN).

   * occurrence: there MUST be zero or exactly one instance of this directive per CDNI Logging File. This directive MAY be included by the dCDN. It MUST NOT be included or modified by the uCDN.

   * example: "claimed-origin: HTAB host".

o established-origin:

   * format: host

   * directive value: this contains the identification, as established by the entity receiving the CDNI Logging File, of the entity transmitting the CDNI Logging File (e.g., the host in a dCDN supporting the CDNI Logging interface) or the entity responsible for transmitting the CDNI Logging File (e.g., the dCDN).

   * occurrence: there MUST be zero or exactly one instance of this directive per CDNI Logging File. This directive MAY be added by the uCDN (e.g., before storing the CDNI Logging File). It MUST NOT be included by the dCDN. The mechanisms used by the

uCDN to establish and validate the entity responsible for the CDNI Logging File is outside the scope of the present document. We observe that, in particular, this may be achieved through authentication mechanisms that are part of the transport layer of the CDNI Logging File pull mechanism (Section 4.2).

  * ABNF example: "established-origin: HTAB host".

- 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".

- record-type:

  * format: NAMEFORMAT

  * directive value: indicates the type of the CDNI Logging Records that follow this directive, until another record-type directive (or the end of the CDNI Logging File). This can be any CDNI Logging Record type registered in the CDNI Logging Record-types registry (Section 6.3). For example this may be "cdni_http_request_v1" as specified in Section 3.4.1. CDNI Logging record-types are case-insensitive as per the basic ABNF([RFC5234]).

  * occurrence: there MUST be at least one instance of this directive per CDNI Logging File. The first instance of this directive MUST precede a fields directive and MUST precede all CDNI Logging Records.

  * example: "record-type: HTAB cdni_http_request_v1".

- fields:

  * format: FIENAME *(HTAB FIENAME) ; where FIENAME can take any CDNI Logging field name registered in the CDNI Logging Field Names registry (Section 6.4) that is valid for the record type specified in the record-type directive.

* directive value: this lists the names of all the fields for which a value is to appear in the CDNI Logging Records that follow the instance of this directive (until another instance of this directive). The names of the fields, as well as their occurrences, MUST comply with the corresponding rules specified in the document referenced in the CDNI Logging Record-types registry (Section 6.3) for the corresponding CDNI Logging record-type.

* occurrence: there MUST be at least one instance of this directive per record-type directive. The first instance of this directive for a given record-type MUST appear before any CDNI Logging Record for this record-type. One situation where more than one instance of the fields directive can appear within a given CDNI Logging File, is when there is a change, in the middle of a fairly large logging period, in the agreement between the uCDN and the dCDN about the set of fields that are to be exchanged. The multiple occurrences allow records with the old set of fields and records with the new set of fields to be carried inside the same Logging File.

* example: "fields: HTAB FIENAME * (HTAB FIENAME)".

o SHA256-hash:

* format: 64HEXDIG

* directive value: This directive permits the detection of a corrupted CDNI Logging File. This can be useful, for instance, if a problem occurs on the filesystem of the dCDN Logging system and leads to a truncation of a logging file. The valid SHA256-hash value is included in this directive by the entity that transmits the CDNI Logging File. It MUST be computed by applying the SHA-256 ([RFC6234]) cryptographic hash function on the CDNI Logging File, including all the directives and logging records, up to the SHA256-hash directive itself, excluding the SHA256-hash directive itself. The SHA256-hash value MUST be represented as a US-ASCII encoded hexadecimal number, 64 digits long (representing a 256 bit hash value). The entity receiving the CDNI Logging File also computes in a similar way the SHA-256 hash on the received CDNI Logging File and compares this hash to the value of the SHA256-hash directive. If the two values are equal, then the received CDNI Logging File is to be considered non-corrupted. If the two values are different, the received CDNI Logging File is to be considered corrupted. The behavior of the entity that received a corrupted CDNI Logging File is outside the scope of this specification; we note that the entity MAY attempt to pull again the same CDNI

Logging File from the transmitting entity. If the entity receiving a non-corrupted CDNI Logging File adds an established-origin directive, it MUST then recompute and update the SHA256-hash directive so it also protects the added established-origin directive.

* occurrence: there MUST be zero or exactly one instance of this directive. There SHOULD be exactly one instance of this directive. One situation where that directive could be omitted is where integrity protection is already provided via another mechanism (for example if an integrity hash is associated to the CDNI Logging File out-of-band through the CDNI Logging Feed (Section 4.1) leveraging ATOM extensions such as those proposed in [I-D.snell-atompub-link-extensions]. When present, the SHA256-hash field MUST be the last line of the CDNI Logging File.

* example: "SHA256-hash: HTAB 64HEXDIG".

An uCDN-side implementation of the CDNI Logging interface MUST ignore a CDNI Logging File that does not comply with the occurrences specified above for each and every directive. For example, an uCDN-side implementation of the CDNI Logging interface receiving a CDNI Logging file with zero occurrence of the version directive, or with two occurrences of the SHA256-hash, MUST ignore this CDNI Logging File.

An entity receiving a CDNI Logging File with a value set to "cdni/1.0" MUST process the CDNI Logging File as per the present document. An entity receiving a CDNI Logging File with a value set to a different value MUST process the CDNI Logging File as per the specification referenced in the CDNI Logging File version registry (see Section 6.1) if the implementation supports this specification and MUST ignore the CDNI Logging File otherwise.

3.4. CDNI Logging Records

A CDNI Logging Record consists of a sequence of CDNI Logging fields relating to that single CDNI Logging Record.

CDNI Logging fields MUST be separated by the "horizontal tabulation (HTAB)" character.

To facilitate readability, a prefix scheme is used for CDNI Logging field names in a similar way to the one used in W3C Extended Log File Format [ELF]. The semantics of the prefix in the present document is:

o "c-" refers to the User Agent that issues the request (corresponds to the "client" of W3C Extended Log Format)

o "d-" refers to the dCDN (relative to a given CDN acting as an uCDN)

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

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

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

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

An implementation of the CDNI Logging interface as per the present specification MUST support the CDNI HTTP Request Logging Record as specified in Section 3.4.1.

A CDNI Logging Record contains the corresponding values for the fields that are enumerated in the last fields directive before the current log line. Note that the order in which the field values appear is dictated by the order of the fields names in the fields directive. There SHOULD be no dependency between the various fields values.

3.4.1. HTTP Request Logging Record

This section defines the CDNI Logging Record of record-type "cdni_http_request_v1". It is applicable to content delivery performed by the dCDN using HTTP/1.0([RFC1945]), HTTP/1.1([RFC7230],[RFC7231], [RFC7232], [RFC7233], [RFC7234], [RFC7235]) or HTTPS ([RFC2818], [RFC7230]). We observe that, in the case of HTTPS delivery, there may be value in logging additional information specific to the operation of HTTP over TLS and we note that this is outside the scope of the present document and may be addressed in a future document defining another CDNI Logging Record or another version of the HTTP Request Logging Record.

The "cdni_http_request_v1" record-type is also expected to be applicable to HTTP/2 [RFC7540] since a fundamental design tenet of HTTP/2 is to preserve the HTTP/1.1 semantics. We observe that, in the case of HTTP/2 delivery, there may be value in logging additional information specific to the additional functionality of HTTP/2 (e.g., information related to connection identification, to stream identification, to stream priority and to flow control). We note that such additional information is outside the scope of the present document and may be addressed in a future document defining another CDNI Logging Record or another version of the HTTP Request Logging Record.

The "cdni_http_request_v1" record-type contains the following CDNI Logging fields, listed by their field name:

- o date:
  - * format: DATE
  - * field value: the date at which the processing of request completed on the Surrogate.
  - * occurrence: there MUST be one and only one instance of this field.
- o time:
  - * format: TIME
  - * field value: the time, which MUST be expressed in Coordinated Universal Time (UTC), at which the processing of request completed on the Surrogate.
  - * occurrence: there MUST be one and only one instance of this field.
- o time-taken:
  - * format: DEC
  - * field value: decimal value of the duration, in seconds, between the start of the processing of the request and the completion of the request processing (e.g., completion of delivery) by the Surrogate.
  - * occurrence: there MUST be one and only one instance of this field.
- o c-groupid:
  - * format: NHTABSTRING
  - * field value: an opaque identifier for an aggregate set of clients, derived from the client IPv4 or IPv6 address in the request received by the Surrogate and/or other network-level

identifying information. The c-groupid serves to group clients into aggregates. Example aggregates include civil geolocation information (the country, second-level administrative division, or postal code from which the client is presumed to make the request based on a geolocation database lookup) or network topological information (e.g., the BGP AS number announcing the prefix containing the address). The c-groupid MAY be structured e.g., US/TN/MEM/38138. Agreement between the dCDN and the uCDN on a mapping between IPv4 and IPv6 addresses and aggregates is presumed to occur out-of-band. The aggregation mapping SHOULD be chosen such that each aggregate contains more than one client.

+ When the aggregate is chosen so that it contains a single client (e.g., to allow more detailed analytics, or to allow a-posteriori analysis of individual delivery for example in situations of performance-based penalties) the c-groupid MAY be structured where some elements identify aggregates and one element identifies the client, e.g., US/TN/MEM/38138/43a5bdd6-95c4-4d62-be65-7410df0021e2. In the case where the aggregate is chosen so that it contains a single client:

  - the element identifying the client SHOULD be algorithmically generated (from the client IPv4 or IPv6 address in the request received by the Surrogate and/or other network-level identifying information) in a way that SHOULD NOT be linkable back to the global addressing context and that SHOULD vary over time (to offer protection against long term attacks).

  - It is RECOMMENDED that the mapping varies at least once every 24 hours.

  - The algorithmic mapping and variation over time can, in some cases, allow the uCDN (with the knowledge of the algorithm and time variation and associated attributes and keys) to reconstruct the actual client IPv4 or IPv6 address and/or other network-level identifying information when required (e.g., to allow a-posteriori analysis of individual delivery for example in situations of performance-based penalties). However, these enduser addresses SHOULD only be reconstructed on-demand and the CDNI Logging File SHOULD only be stored with the anonymised c-groupid value.

  - Allowing reconstruction of client address information carries with it grave risks to end-user privacy. Since the c-groupid is in this case equivalent in identification power to a client IP address, its use may be restricted by regulation or law as personally identifiable information. For this reason, such use is NOT RECOMMENDED.

- One method for mapping that MAY be be supported by implementations relies on a symmetric key that is known only to the uCDN and dCDN and HMAC-based Extract-and-Expand Key Derivation Function (HKDF) key derivation ([RFC5869]), as will be used in TLS 1.3 ([I-D.ietf-tls-rfc5246-bis]). When that method is used:

  o The uCDN and dCDN need to agree on the "salt" and "input keying material", as described in Section 2.2 of [RFC5869] and the initial "info" parameter (which could be something like the business names of the two organizations in UTF-8, concatenated), as described in Section 2.3 of [RFC5869]. The hash SHOULD be either SHA-2 or SHA-3 [SHA-3] and the encryption algorithm SHOULD be 128-bit AES [AES] in Galois Counter Mode (GCM) [GCM] (AES-GCM) or better. The PRK SHOULD be chosen by both parties contributing alternate random bytes until sufficient length exists. After the initial setup, client-information can be encrypted using the key generated by the "expand" step of Section 2.3 of [RFC5869]. The encrypted value SHOULD be hex encoded or base64 encoded (as specified in section 4 of [RFC4648]). At the agreed-upon expiration time, a new key SHOULD be generated and used. New keys SHOULD be indicated by prefixing the key with a special character such as exclamation point. In this way, shorter lifetimes can be used as needed.

* occurrence: there MUST be one and only one instance of this field.

o s-ip:

  * format: ADDRESS

  * field value: the IPv4 or IPv6 address of the Surrogate that served the request (i.e., the "server" address).

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

o s-hostname:

  * format: host

  * field value: the hostname of the Surrogate that served the request (i.e., the "server" hostname).

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

o s-port:

  * format: 1*DIGIT

  * field value: the destination TCP port (i.e., the "server" port) in the request received by the Surrogate.

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

o cs-method:

  * format: NHTABSTRING

  * field value: this is the method of the request received by the Surrogate. In the case of HTTP delivery, this is the HTTP method in the request.

  * occurrence: There MUST be one and only one instance of this field.

o cs-uri:

  * format: NHTABSTRING

  * field value: this is the "effective request URI" of the request received by the Surrogate as specified in [RFC7230]. It complies with the "http" URI scheme or the "https" URI scheme as specified in [RFC7230]). Note that cs-uri can be privacy sensitive. In that case, and where appropriate, u-uri could be used instead of cs-uri.

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

o u-uri:

  * format: NHTABSTRING

* field value: this is a complete URI, derived from the "effective request URI" ([RFC7230]) of the request received by the Surrogate (i.e., the cs-uri) but transformed by the entity generating or transmitting the CDNI Logging Record, in a way that is agreed upon between the two ends of the CDNI Logging interface, so the transformed URI is meaningful to the uCDN. For example, the two ends of the CDNI Logging interface could agree that the u-uri is constructed from the cs-uri by removing the part of the hostname that exposes which individual Surrogate actually performed the delivery. The details of modification performed to generate the u-uri, as well as the mechanism to agree on these modifications between the two sides of the CDNI Logging interface are outside the scope of the present document.

* occurrence: there MUST be one and only one instance of this field.

o protocol:

* format: NHTABSTRING

* field value: this is value of the HTTP-Version field as specified in [RFC7230] of the Request-Line of the request received by the Surrogate (e.g., "HTTP/1.1").

* occurrence: there MUST be one and only one instance of this field.

o sc-status:

* format: 3DIGIT

* field value: this is the Status-Code in the response from the Surrogate. In the case of HTTP delivery, this is the HTTP Status-Code in the HTTP response.

* occurrence: There MUST be one and only one instance of this field.

o sc-total-bytes:

* format: 1*DIGIT

* field value: this is the total number of bytes of the response sent by the Surrogate in response to the request. In the case of HTTP delivery, this includes the bytes of the Status-Line,

the bytes of the HTTP headers and the bytes of the message-body.

* occurrence: There MUST be one and only one instance of this field.

o sc-entity-bytes:

* format: 1*DIGIT

* field value: this is the number of bytes of the message-body in the HTTP response sent by the Surrogate in response to the request. This does not include the bytes of the Status-Line or the bytes of the HTTP headers.

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

o cs(insert_HTTP_header_name_here):

* format: QSTRING

* field value: the value of the HTTP header (identified by the insert_HTTP_header_name_here in the CDNI Logging field name) as it appears in the request processed by the Surrogate, but prepended by a DQUOTE and appended by a DQUOTE. For example, when the CDNI Logging field name (FIENAME) listed in the preceding fields directive is cs(User-Agent), this CDNI Logging field value contains the value of the User-Agent HTTP header as received by the Surrogate in the request it processed, but prepended by a DQUOTE and appended by a DQUOTE. If the HTTP header as it appeared in the request processed by the Surrogate contains one or more DQUOTE, each DQUOTE MUST be escaped with percent encoding. For example, if the HTTP header contains My_Header"value", then the field value of the cs(insert_HTTP_header_name_here) is "My_Header%x22value%x22". The entity transmitting the CDNI Logging File MUST ensure that the respective insert_HTTP_header_name_here of the cs(insert_HTTP_header_name_here) listed in the fields directive comply with HTTP specifications. In particular, this field name does not include any HTAB, since this would prevent proper parsing of the fields directive by the entity receiving the CDNI Logging File.

* occurrence: there MAY be zero, one or any number of instance of this field.

o sc(insert_HTTP_header_name_here):

   * format: QSTRING

   * field value: the value of the HTTP header (identified by the insert_HTTP_header_name_here in the CDNI Logging field name) as it appears in the response issued by the Surrogate to serve the request, but prepended by a DQUOTE and appended by a DQUOTE. If the HTTP header as it appeared in the request processed by the Surrogate contains one or more DQUOTE, each DQUOTE MUST be escaped with percent encoding. For example, if the HTTP header contains My_Header"value", then the field value of the sc(insert_HTTP_header_name_here) is "My_Header%x22value%x22". The entity transmitting the CDNI Logging File MUST ensure that the respective insert_HTTP_header_name_here of the cs(insert_HTTP_header_name_here) listed in the fields directive comply with HTTP specifications. In particular, this field name does not include any HTAB, since this would prevent proper parsing of the fields directive by the entity receiving the CDNI Logging File.

   * occurrence: there MAY be zero, one or any number of instances of this field. For a given insert_HTTP_header_name_here, there MUST be zero or exactly one instance of this field.

o s-ccid:

   * format: QSTRING

   * field value: this contains the value of the Content Collection IDentifier (CCID) associated by the uCDN to the content served by the Surrogate via the CDNI Metadata interface ([I-D.ietf-cdni-metadata]), prepended by a DQUOTE and appended by a DQUOTE. If the CCID conveyed in the CDNI Metadata interface contains one or more DQUOTE, each DQUOTE MUST be escaped with percent encoding. For example, if the CCID conveyed in the CDNI Metadata interface is My_CCIDD"value", then the field value of the s-ccid is "My_CCID%x22value%X22".

   * occurrence: there MUST be zero or exactly one instance of this field. For a given insert_HTTP_header_name_here, there MUST be zero or exactly one instance of this field.

o s-sid:

   * format: QSTRING

   * field value: this contains the value of a Session IDentifier (SID) generated by the dCDN for a specific HTTP session, prepended by a DQUOTE and appended by a DQUOTE. In particular,

for HTTP Adaptive Streaming (HAS) session, the Session IDentifier value is included in the Logging record for every content chunk delivery of that session in view of facilitating the later correlation of all the per content chunk log records of a given HAS session. See section 3.4.2.2. of [RFC6983] for more discussion on the concept of Session IDentifier in the context of HAS. If the SID conveyed contains one or more DQUOTE, each DQUOTE MUST be escaped with percent encoding. For example, if the SID is My_SID"value", then the field value of the s-sid is "My_SID%x22value%x22".

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

o s-cached:

* format: 1DIGIT

* field value: this characterises whether the Surrogate served the request using content already stored on its local cache or not. The allowed values are "0" (for miss) and "1" (for hit). "1" MUST be used when the Surrogate did serve the request using exclusively content already stored on its local cache. "0" MUST be used otherwise (including cases where the Surrogate served the request using some, but not all, content already stored on its local cache). Note that a "0" only means a cache miss in the Surrogate and does not provide any information on whether the content was already stored, or not, in another device of the dCDN, i.e., whether this was a "dCDN hit" or "dCDN miss".

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

CDNI Logging field names are case-insensitive as per the basic ABNF([RFC5234]). The "fields" directive corresponding to a HTTP Request Logging Record MUST contain all the fields names whose occurrence is specified above as "There MUST be one and only one instance of this field". The corresponding fields value MUST be present in every HTTP Request Logging Record.

The "fields" directive corresponding to a HTTP Request Logging Record MAY list all the fields value whose occurrence is specified above as "there MUST be zero or exactly one instance of this field" or "there MAY be zero, one or any number of instances of this field". The set of such field names actually listed in the "fields" directive is selected by the CDN generating the CDNI Logging File based on agreements between the interconnected CDNs established through mechanisms outside the scope of this specification (e.g., contractual agreements). When such a field name is not listed in the "fields" directive, the corresponding field value MUST NOT be included in the Logging Record. When such a field name is listed in the "fields" directive, the corresponding field value MUST be included in the Logging Record; if the value for the field is not available, this MUST be conveyed via a dash character ("-").

The fields names listed in the "fields" directive MAY be listed in the order in which they are listed in Section 3.4.1 or MAY be listed in any other order.

Logging some specific fields from HTTP requests and responses can introduce serious security and privacy risks. For example, cookies will often contain (months) long lived token values that can be used to log into a service as the relevant user. Similar values may be included in other header fields or within URLs or elsewhere in HTTP requests and responses. Centralising such values in a CDNI Logging File can therefore represent a significant increase in risk both for the user and the web service provider, but also for the CDNs involved. Implementations ought therefore to attempt to lower the probability of such bad outcomes e.g. by only allowing a configured set of headers to be added to CDNI Logging Records, or by not supporting wildcard selection of HTTP request/response fields to add. Such mechanisms can reduce the probability that security (or privacy) sensitive values are centralised in CDNI Logging Files. Also, when agreeing on which HTTP request/response fields are to be provided in CDNI Logging Files, the uCDN and dCDN administrators ought to consider these risks. Furthermore, CDNs making use of c-groupid to identify an aggregate of clients rather than individual clients ought to realize that by logging certain header fields they may create the possibility to re-identify individual clients. In these cases heeding the above advice, or not logging header fields at all, is particularly important if the goal is to provide logs that do not identify individual clients."

A dCDN-side implementation of the CDNI Logging interface MUST implement all the following Logging fields in a CDNI Logging Record of record-type "cdni_http_request_v1", and MUST support the ability to include valid values for each of them:

- date

- time

- time-taken

- c-groupid

o s-ip

o s-hostname

o s-port

o cs-method

o cs-uri

o u-uri

o protocol

o sc-status

o sc-total-bytes

o sc-entity-bytes

o cs(insert_HTTP_header_name_here)

o sc(insert_HTTP_header_name_here)

o s-cached

A dCDN-side implementation of the CDNI Logging interface MAY support the following Logging fields in a CDNI Logging Record of record-type "cdni_http_request_v1":

o s-ccid

o s-sid

If a dCDN-side implementation of the CDNI Logging interface supports these fields, it MUST support the ability to include valid values for them.

An uCDN-side implementation of the CDNI Logging interface MUST be able to accept CDNI Logging Files with CDNI Logging Records of record-type "cdni_http_request_v1" containing any CDNI Logging Field defined in Section 3.4.1 as long as the CDNI Logging Record and the CDNI Logging File are compliant with the present document.

In case an uCDN-side implementation of the CDNI Logging interface receives a CDNI Logging File with HTTP Request Logging Records that do not contain field values for exactly the set of field names actually listed in the preceding "fields" directive, the implementation MUST ignore those HTTP Request Logging Records, and MUST accept the other HTTP Request Logging Records.

To ensure that the logging file is correct, the text MUST be sanitized before being logged. Null, bare CR, bare LF and HTAB have to be removed by escaping them through percent encoding to avoid confusion with the logging record separators.

3.5. CDNI Logging File Extension

The CDNI Logging File contains blocks of directives and blocks of corresponding records. The supported set of directives is defined relative to the CDNI Logging File Format version. The complete set of directives for version "cdni/1.0" are defined in Section 3.3. The directive list is not expected to require much extension, but when it does, the new directive MUST be defined and registered in the "CDNI Logging Directive Names" registry, as described in Figure 9, and a new version MUST be defined and registered in the "CDNI Logging File version" registry, as described in Section 6.2. For example, adding a new CDNI Logging Directive, e.g., "foo", to the set of directives defined for "cdni/1.0" in Section 3.3, would require registering both the new CDNI Logging Directive "foo" and a new CDNI Logging File version, e.g., "CDNI/2.0", which includes all of the existing CDNI Logging Directives of "cdni/1.0" plus "foo".

It is expected that as new logging requirements arise, the list of fields to log will change and expand. When adding new fields, the new fields MUST be defined and registered in the "CDNI Logging Field Names" registry, as described in Section 6.4, and a new record-type MUST be defined and registered in the "CDNI Logging record-types" registry, as described in Section 6.3. For example, adding a new CDNI Logging Field, e.g., "c-bar", to the set of fields defined for "cdni_http_request_v1" in Section 3.4.1, would require registering both the new CDNI Logging Field "c-bar" and a new CDNI record-type, e.g., "cdni_http_request_v2", which includes all of the existing CDNI Logging Fields of "cdni_http_request_v1" plus "c-bar".

3.6. CDNI Logging File Examples

Let us consider the upstream CDN and the downstream CDN labelled uCDN and dCDN-1 in Figure 1. When dCDN-1 acts as a downstream CDN for uCDN and performs content delivery on behalf of uCDN, dCDN-1 will include the CDNI Logging Records corresponding to the content deliveries performed on behalf of uCDN in the CDNI Logging Files for uCDN. An example CDNI Logging File communicated by dCDN-1 to uCDN is shown below in Figure 4.

```
#version:<HTAB>cdni/1.0<CRLF>

#UUID:<HTAB>urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6<CRLF>

#claimed-origin:<HTAB>cdni-logging-entity.dcdn-1.example.com<CRLF>

#record-type:<HTAB>cdni_http_request_v1<CRLF>

#fields:<HTAB>date<HTAB>time<HTAB>time-taken<HTAB>c-groupid<HTAB>
cs-method<HTAB>u-uri<HTAB>protocol<HTAB>
sc-status<HTAB>sc-total-bytes<HTAB>cs(User-Agent)<HTAB>
cs(Referer)<HTAB>s-cached<CRLF>

2013-05-17<HTAB>00:38:06.825<HTAB>9.058<HTAB>US/TN/MEM/38138<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/movie100.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>6729891<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>00:39:09.145<HTAB>15.32<HTAB>FR/PACA/NCE/06100<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/movie118.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>15799210<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>00:42:53.437<HTAB>52.879<HTAB>US/TN/MEM/38138<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/picture11.mp4<HTAB>
HTTP/1.0<HTAB>200<HTAB>97234724<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host5.example.com"<HTAB>0<CRLF>

#SHA256-hash:<HTAB> 64-hexadecimal-digit hash value <CRLF>
```

Figure 4: CDNI Logging File Example

If uCDN establishes by some means (e.g., via TLS authentication when pulling the CDNI Logging File) the identity of the entity from which it pulled the CDNI Logging File, uCDN can add to the CDNI Logging an established-origin directive as illustrated below:

```
#established-origin:<HTAB>cdni-logging-entity.dcdn-
1.example.com<CRLF>
```

As illustrated in Figure 2, uCDN will then ingest the corresponding CDNI Logging Records into its Collection process, alongside the Logging Records generated locally by the uCDN itself. This allows uCDN to aggregate Logging Records for deliveries performed by itself (through Records generated locally) as well as for deliveries performed by its downstream CDN(s). This aggregate information can then be used (after Filtering and Rectification, as illustrated in Figure 2) by Log Consuming Applications that take into account deliveries performed by uCDN as well as by all of its downstream CDNs.

We observe that the time between

1. when a delivery is completed in dCDN and

2. when the corresponding Logging Record is ingested by the Collection process in uCDN

depends on a number of parameters such as the Logging Period agreed to by uCDN and dCDN, how much time uCDN waits before pulling the CDNI Logging File once it is advertised in the CDNI Logging Feed, and the time to complete the pull of the CDNI Logging File. Therefore, if we consider the set of Logging Records aggregated by the Collection process in uCDN in a given time interval, there could be a permanent significant timing difference between the CDNI Logging Records received from the dCDN and the Logging Records generated locally. For example, in a given time interval, the Collection process in uCDN may be aggregating Logging Records generated locally by uCDN for deliveries performed in the last hour and CDNI Logging Records generated in the dCDN for deliveries in the hour before last.

Say, that for some reason (for example a Surrogate bug), dCDN-1 could not collect the total number of bytes of the responses sent by the Surrogate (in other words, the value for sc-total-bytes is not available). Then the corresponding CDNI Logging records would contain a dash character ("-") in lieu of the value for the sc-total-bytes field (as specified in Section 3.4.1). In that case, the CDNI Logging File that would be communicated by dCDN-1 to uCDN is shown below in Figure 5.

```
#version:<HTAB>cdni/1.0<CRLF>

#UUID:<HTAB>urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6<CRLF>

#claimed-origin:<HTAB>cdni-logging-entity.dcdn-1.example.com<CRLF>

#record-type:<HTAB>cdni_http_request_v1<CRLF>

#fields:<HTAB>date<HTAB>time<HTAB>time-taken<HTAB>c-groupid<HTAB>
cs-method<HTAB>u-uri<HTAB>protocol<HTAB>
sc-status<HTAB>sc-total-bytes<HTAB>cs(User-Agent)<HTAB>
cs(Referer)<HTAB>s-cached<CRLF>

2013-05-17<HTAB>00:38:06.825<HTAB>9.058<HTAB>US/TN/MEM/38138<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/movie100.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>-<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>00:39:09.145<HTAB>15.32<HTAB>FR/PACA/NCE/06100<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/movie118.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>-<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>00:42:53.437<HTAB>52.879<HTAB>US/TN/MEM/38138<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-1.example.com/video/picture11.mp4<HTAB>
HTTP/1.0<HTAB>200<HTAB>-<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari/533.4"<HTAB>
"host5.example.com"<HTAB>0<CRLF>

#SHA256-hash:<HTAB> 64-hexadecimal-digit hash value <CRLF>
```

Figure 5: CDNI Logging File Example With A Missing Field Value

3.7. Cascaded CDNI Logging Files Example

Let us consider the cascaded CDN scenario of uCDN, dCDN-2 and dCDN-3 as depicted in Figure 1. After completion of a delivery by dCDN-3 on behalf of dCDN-2, dCDN-3 will include a corresponding Logging Record in a CDNI Logging File that will be pulled by dCDN-2 and that is illustrated below in Figure 6. In practice, a CDNI Logging File is likely to contain a very high number of CDNI Logging Records. However, for readability, the example in Figure 6 contains a single CDNI Logging Record.

```
#version:<HTAB>cdni/1.0<CRLF>

#UUID:<HTAB>urn:uuid:65718ef-0123-9876-adce4321bcde<CRLF>

#claimed-origin:<HTAB>cdni-logging-entity.dcdn-3.example.com<CRLF>

#record-type:<HTAB>cdni_http_request_v1<CRLF>

#fields:<HTAB>date<HTAB>time<HTAB>time-taken<HTAB>c-groupid<HTAB>
cs-method<HTAB>u-uri<HTAB>protocol<HTAB>
sc-status<HTAB>sc-total-bytes<HTAB>cs(User-Agent)<HTAB>
cs(Referer)<HTAB>s-cached<CRLF>

2013-05-17<HTAB>00:39:09.119<HTAB>14.07<HTAB>US/CA/SFO/94114<HTAB>
GET<HTAB>
http://cdni-dcdn-2.dcdn-3.example.com/video/movie118.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>15799210<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari /533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

#SHA256-hash:<HTAB> 64-hexadecimal-digit hash value <CRLF>
```

Figure 6: Cascaded CDNI Logging File Example (dCDN-3 to dCDN-2)

If dCDN-2 establishes by some means (e.g., via TLS authentication when pulling the CDNI Logging File) the identity of the entity from which it pulled the CDNI Logging File, dCDN-2 can add to the CDNI Logging an established-origin directive as illustrated below:

```
#established-origin:<HTAB>cdni-logging-entity.dcdn-
3.example.com<CRLF>
```

dCDN-2 (behaving as an upstream CDN from the viewpoint of dCDN-3) will then ingest the CDNI Logging Record for the considered dCDN-3 delivery into its Collection process (as illustrated in Figure 2). This Logging Record may be aggregated with Logging Records generated locally by dCDN-2 for deliveries performed by dCDN-2 itself. Say, for illustration, that the content delivery performed by dCDN-3 on behalf of dCDN-2 had actually been redirected to dCDN-2 by uCDN, and say that another content delivery has just been redirected by uCDN to dCDN-2 and that dCDN-2 elected to perform the corresponding delivery itself. Then after Filtering and Rectification (as illustrated in Figure 2), dCDN-2 will include the two Logging Records corresponding respectively to the delivery performed by dCDN-3 and the delivery performed by dCDN-2, in the next CDNI Logging File that will be communicated to uCDN. An example of such CDNI Logging File is illustrated below in Figure 7.

```
#version:<HTAB>cdni/1.0<CRLF>

#UUID:<HTAB>urn:uuid:1234567-8fedc-abab-0987654321ff<CRLF>

#claimed-origin:<HTAB>cdni-logging-entity.dcdn-2.example.com<CRLF>

#record-type:<HTAB>cdni_http_request_v1<CRLF>

#fields:<HTAB>date<HTAB>time<HTAB>time-taken<HTAB>c-groupid<HTAB>
cs-method<HTAB>u-uri<HTAB>protocol<HTAB>
sc-status<HTAB>sc-total-bytes<HTAB>cs(User-Agent)<HTAB>
cs(Referer)<HTAB>s-cached<CRLF>

2013-05-17<HTAB>00:39:09.119<HTAB>14.07<HTAB>US/CA/SFO/94114<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-2.example.com/video/movie118.mp4<HTAB>
HTTP/1.1<HTAB>200<HTAB>15799210<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari /533.4"<HTAB>
"host1.example.com"<HTAB>1<CRLF>

2013-05-17<HTAB>01:42:53.437<HTAB>52.879<HTAB>FR/IDF/PAR/75001<HTAB>
GET<HTAB>
http://cdni-ucdn.dcdn-2.example.com/video/picture11.mp4<HTAB>
HTTP/1.0<HTAB>200<HTAB>97234724<HTAB>"Mozilla/5.0
(Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4 (KHTML, like
Gecko) Chrome/5.0.375.127 Safari /533.4"<HTAB>
"host5.example.com"<HTAB>0<CRLF>

#SHA256-hash:<HTAB> 64-hexadecimal-digit hash value <CRLF>
```

Figure 7: Cascaded CDNI Logging File Example (dCDN-2 to uCDN)

If uCDN establishes by some means (e.g., via TLS authentication when pulling the CDNI Logging File) the identity of the entity from which it pulled the CDNI Logging File, uCDN can add to the CDNI Logging an established-origin directive as illustrated below:

```
#established-origin:<HTAB>cdni-logging-entity.dcdn-
2.example.com<CRLF>
```

In the example of Figure 7, we observe that:

- o the first Logging Record corresponds to the Logging Record communicated earlier to dCDN-2 by dCDN-3, which corresponds to a delivery redirected by uCDN to dCDN-2 and then redirected by dCDN-2 to dCDN-3. The fields values in this Logging Record are copied from the corresponding CDNI Logging REcord communicated to dCDN2 by dCDN-3, with the exception of the u-uri that now reflects the URI convention between uCDN and dCDN-2 and that presents the delivery to uCDN as if it was performed by dCDN-2 itself. This reflects the fact that dCDN-2 had taken the full responsibility of the corresponding delivery (even if in this case, dCDN-2 elected to redirect the delivery to dCDN-3 so it is actually performed by dCDN-3 on behalf of dCDN-2).

- o the second Logging Record corresponds to a delivery redirected by uCDN to dCDN-2 and performed by dCDN-2 itself. The time of the delivery in this Logging Record may be significantly more recent than the first Logging Record since it was generated locally while the first Logging Record was generated by dCDN-3 and had to be advertised , and then pulled and then ingested into the dCDN-2 Collection process, before being aggregated with the second Logging Record.

4. Protocol for Exchange of CDNI Logging File After Full Collection

This section specifies a protocol for the exchange of CDNI Logging Files as specified in Section 3 after the CDNI Logging File is fully collected by the dCDN.

This protocol comprises:

- o a CDNI Logging feed, allowing the dCDN to notify the uCDN about the CDNI Logging Files that can be retrieved by that uCDN from the dCDN, as well as all the information necessary for retrieving each of these CDNI Logging Files. The CDNI Logging feed is specified in Section 4.1.

- o a CDNI Logging File pull mechanism, allowing the uCDN to obtain from the dCDN a given CDNI Logging File at the uCDN’s convenience. The CDNI Logging File pull mechanisms is specified in Section 4.2.

An implementation of the CDNI Logging interface on the dCDN side (the entity generating the CDNI Logging file) MUST support the server side of the CDNI Logging feed (as specified in Section 4.1) and the server side of the CDNI Logging pull mechanism (as specified in Section 4.2).

An implementation of the CDNI Logging interface on the uCDN side (the entity consuming the CDNI Logging file) MUST support the client side of the CDNI Logging feed (as specified in Section 4.1) and the client side of the CDNI Logging pull mechanism (as specified in Section 4.2).

4.1. CDNI Logging Feed

The server-side implementation of the CDNI Logging feed MUST produce an Atom feed [RFC4287]. This feed is used to advertise log files that are available for the client-side to retrieve using the CDNI Logging pull mechanism.

4.1.1. Atom Formatting

A CDNI Logging feed MUST be structured as an Archived feed, as defined in [RFC5005], and MUST be formatted in Atom [RFC4287]. This means it consists of a subscription document that is regularly updated as new CDNI Logging Files become available, and information about older CDNI Logging files is moved into archive documents. Once created, archive documents are never modified.

Each CDNI Logging File listed in an Atom feed MUST be described in an atom:entry container element.

The atom:entry MUST contain an atom:content element whose "src" attribute is a link to the CDNI Logging File and whose "type" attribute is the MIME Media Type indicating that the entry is a CDNI logging file. This MIME Media Type is defined as "application/cdni" (See [RFC7736]) with the Payload Type (ptype) parameter set to "logging-file".

For compatibility with some Atom feed readers the atom:entry MAY also contain an atom:link entry whose "href" attribute is a link to the CDNI Logging File and whose "type" attribute is the MIME Media Type indicating that the entry is a CDNI Logging File using the "application/cdni" MIME Media Type with the Payload Type (ptype) parameter set to "logging-file"(See [RFC7736]).

The URI used in the atom:id of the atom:entry MUST contain the UUID of the CDNI Logging File.

The atom:updated in the atom:entry MUST indicate the time at which the CDNI Logging File was last updated.

4.1.2. Updates to Log Files and the Feed

CDNI Logging Files MUST NOT be modified by the dCDN once published in the CDNI Logging feed.

The frequency with which the subscription feed is updated, the period of time covered by each CDNI Logging File or each archive document, and timeliness of publishing of CDNI Logging Files are outside the scope of the present document and are expected to be agreed upon by uCDN and dCDN via other means (e.g., human agreement).

The server-side implementation MUST be able to set, and SHOULD set, HTTP cache control headers on the subscription feed to indicate the frequency at which the client-side is to poll for updates.

The client-side MAY use HTTP cache control headers (set by the server-side) on the subscription feed to determine the frequency at which to poll for updates. The client-side MAY instead, or in addition, use other information to determine when to poll for updates (e.g., a polling frequency that may have been negotiated between the uCDN and dCDN by mechanisms outside the scope of the present document and that is to override the indications provided in the HTTP cache control headers).

The potential retention limits (e.g., sliding time window) within which the dCDN is to retain and be ready to serve an archive document is outside the scope of the present document and is expected to be agreed upon by uCDN and dCDN via other means (e.g., human agreement). The server-side implementation MUST retain, and be ready to serve, any archive document within the agreed retention limits. Outside these agreed limits, the server-side implementation MAY indicate its inability to serve (e.g., with HTTP status code 404) an archive document or MAY refuse to serve it (e.g., with HTTP status code 403 or 410).

4.1.3. Redundant Feeds

The server-side implementation MAY present more than one CDNI Logging feed for redundancy. Each CDNI Logging File MAY be published in more than one feed.

A client-side implementation MAY support such redundant CDNI Logging feeds. If it supports redundant CDNI Logging feed, the client-side can use the UUID of the CDNI Logging File, presented in the atom:id element of the Atom feed, to avoid unnecessarily pulling and storing a given CDNI Logging File more than once.

4.1.4. Example CDNI Logging Feed

Figure 8 illustrates an example of the subscription document of a CDNI Logging feed.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title type="text">CDNI Logging Feed</title>
  <updated>2013-03-23T14:46:11Z</updated>
  <id>urn:uuid:663ae677-40fb-e99a-049d-c5642916b8ce</id>
  <link href="https://dcdn.example/logfeeds/ucdn1"
     rel="self" type="application/atom+xml" />
  <link href="https://dcdn.example/logfeeds/ucdn1"
     rel="current" type="application/atom+xml" />
  <link href="https://dcdn.example/logfeeds/ucdn1/201303231400"
     rel="prev-archive" type="application/atom+xml" />
  <generator version="example version 1">CDNI Log Feed
     Generator</generator>
  <author><name>dcdn.example</name></author>
  <entry>
    <title type="text">CDNI Logging File for uCDN at
      2013-03-23 14:15:00</title>
      <id>urn:uuid:12345678-1234-abcd-00aa-01234567abcd</id>
      <updated>2013-03-23T14:15:00Z</updated>
      <content src="https://dcdn.example/logs/ucdn/
         http-requests-20130323141500000000"
         type="application/cdni"
         ptype="logging-file"/>
      <summary>CDNI Logging File for uCDN at
      2013-03-23 14:15:00</summary>
  </entry>
  <entry>
    <title type="text">CDNI Logging File for uCDN at
      2013-03-23 14:30:00</title>
      <id>urn:uuid:87654321-4321-dcba-aa00-dcba7654321</id>
      <updated>2013-03-23T14:30:00Z</updated>
      <content src="https://dcdn.example/logs/ucdn/
         http-requests-20130323143000000000"
         type="application/cdni"
         ptype="logging-file"/>
      <summary>CDNI Logging File for uCDN at
      2013-03-23 14:30:00</summary>
  </entry>
  ...
  <entry>
    ...
  </entry>
</feed>
```


Figure 8: Example subscription document of a CDNI Logging Feed

4.2. CDNI Logging File Pull

A client-side implementation of the CDNI Logging interface MAY pull, at its convenience, a CDNI Logging File that is published by the server-side in the CDNI Logging Feed (in the subscription document or an archive document). To do so, the client-side:

- o MUST implement HTTP/1.1 ([RFC7230],[RFC7231], [RFC7232], [RFC7233], [RFC7234], [RFC7235]), MAY also support other HTTP versions (e.g., HTTP/2 [RFC7540]) and MAY negotiate which HTTP version is actually used. This allows operators and implementers to choose to use later versions of HTTP to take advantage of new features, while still ensuring interoperability with systems that only support HTTP/1.1.

- o MUST use the URI that was associated to the CDNI Logging File (within the "src" attribute of the corresponding atom:content element) in the CDNI Logging Feed;

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

- o MUST support exchange of CDNI Logging Files with "gzip" content encoding (as defined in [RFC7230]) applied to the representation.

Note that a client-side implementation of the CDNI Logging interface MAY pull a CDNI Logging File that it has already pulled.

The server-side implementation MUST respond to valid pull request by a client-side implementation for a CDNI Logging File published by the server-side in the CDNI Logging Feed (in the subscription document or an archive document). The server-side implementation:

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

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

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

- o MUST support exchange of CDNI Logging Files with "gzip" content encoding (as defined in [RFC7231]) applied to the representation.

Content negotiation approaches defined in [RFC7231] (e.g., using Accept-Encoding request-header field or Content-Encoding entity-header field) MAY be used by the client-side and server-side implementations to establish the content-coding to be used for a particular exchange of a CDNI Logging File.

Applying compression content encoding (such as "gzip") is expected to mitigate the impact of exchanging the large volumes of logging information expected across CDNs. This is expected to be particularly useful in the presence of HTTP Adaptive Streaming (HAS) which, as per the present version of the document, will result in a separate CDNI Log Record for each HAS segment delivery in the CDNI Logging File.

The potential retention limits (e.g., sliding time window, maximum aggregate file storage quotas) within which the dCDN is to retain and be ready to serve a CDNI Logging File previously advertised in the CDNI Logging Feed is outside the scope of the present document and is expected to be agreed upon by uCDN and dCDN via other means (e.g., human agreement). The server-side implementation MUST retain, and be ready to serve, any CDNI Logging File within the agreed retention limits. Outside these agreed limits, the server-side implementation MAY indicate its inability to serve (e.g., with HTTP status code 404) a CDNI Logging File or MAY refuse to serve it (e.g., with HTTP status code 403 or 410).

5. Protocol for Exchange of CDNI Logging File During Collection

We note that, in addition to the CDNI Logging File exchange protocol specified in Section 4, implementations of the CDNI Logging interface may also support other mechanisms to exchange CDNI Logging Files. In particular, such mechanisms might allow the exchange of the CDNI Logging File to start before the file is fully collected. This can allow CDNI Logging Records to be communicated by the dCDN to the uCDN as they are gathered by the dCDN without having to wait until all the CDNI Logging Records of the same logging period are collected in the corresponding CDNI Logging File. This approach is commonly referred to as "tailing" of the file.

Such an approach could be used, for example, to exchange logging information with a significantly reduced time-lag (e.g., sub-minute or sub-second) between when the event occurred in the dCDN and when the corresponding CDNI Logging Record is made available to the uCDN. This can satisfy log-consuming applications requiring extremely fresh logging information such as near-real-time content delivery monitoring. Such mechanisms are for further study and outside the scope of this document.

6. IANA Considerations

6.1. CDNI Logging Directive Names Registry

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

The initial contents of the CDNI Logging Directives registry comprise the names of the directives specified in Section 3.3 of the present document, and are as follows:

Directive Name	Reference
version	RFC xxxx
UUID	RFC xxxx
claimed-origin	RFC xxxx
established-origin	RFC xxxx
remark	RFC xxxx
record-type	RFC xxxx
fields	RFC xxxx
SHA256-hash	RFC xxxx

Figure 9

[Instructions to IANA: Replace "RFC xxxx" above by the RFC number of the present document]

Within the registry, names are to be allocated by IANA according to the "Specification Required" policy specified in [RFC5226]. Directive names are to be allocated by IANA with a format of NAMEFORMAT (see Section 3.1). All directive names defined in the logging file are case-insensitive as per the basic ABNF([RFC5234]).

Each specification that defines a new CDNI Logging directive needs to contain a description for the new directive with the same set of information as provided in Section 3.3 (i.e., format, directive value and occurrence).

6.2. CDNI Logging File version Registry

The IANA is requested to create a new "CDNI Logging File version" subregistry under the "Content Delivery Networks Interconnection (CDNI) Parameters" registry.

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

version	Reference	Description
cdni/1.0	RFC xxxx	CDNI Logging File version 1.0 as specified in RFC xxxx

Figure 10

[Instructions to IANA: Replace "RFC xxxx" above by the RFC number of the present document]

Within the registry, version values are to be allocated by IANA according to the "Specification Required" policy specified in [RFC5226]. Version values are to be allocated by IANA with a format of NAMEFORMAT (see Section 3.1). All version values defined in the logging file are case-insensitive as per the basic ABNF([RFC5234]).

6.3. CDNI Logging record-types Registry

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

The initial contents of the CDNI Logging record-types registry comprise the names of the CDNI Logging Record types specified in Section 3.4 of the present document, and are as follows:

record-types	Reference	Description
cdni_http_request_v1	RFC xxxx	CDNI Logging Record version 1 for content delivery using HTTP

Figure 11

[Instructions to IANA: Replace "RFC xxxx" above by the RFC number of the present document]

Within the registry, record-types are to be allocated by IANA according to the "Specification Required" policy specified in [RFC5226]. Record-types are to be allocated by IANA with a format of

NAMEFORMAT (see Section 3.1). All record-types defined in the logging file are case-insensitive as per the basic ABNF([RFC5234]).

Each specification that defines a new record-type needs to contain a description for the new record-type with the same set of information as provided in Section 3.4.1. This includes:

- o a list of all the CDNI Logging fields that can appear in a CDNI Logging Record of the new record-type
- o for all these fields: a specification of the occurrence for each Field in the new record-type
- o for every newly defined Field, i.e., for every Field that results in a registration in the CDNI Logging Field Names Registry (Section 6.4): a specification of the field name, format and field value.

6.4. CDNI Logging Field Names Registry

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

This registry is intended to be shared across the currently defined record-type (i.e., cdni_http_request_v1) as well as potential other CDNI Logging record-types that may be defined in separate specifications. When a Field from this registry is used by another CDNI Logging record-type, it is to be used with the exact semantics and format specified in the document that registered this field and that is identified in the Reference column of the registry. If another CDNI Logging record-type requires a Field with semantics that are not strictly identical, or a format that is not strictly identical then this new Field is to be registered in the registry with a different Field name. When a Field from this registry is used by another CDNI Logging record-type, it can be used with different occurrence rules.

The initial contents of the CDNI Logging fields Names registry comprise the names of the CDNI Logging fields specified in Section 3.4 of the present document, and are as follows:

Field Name	Reference
date	RFC xxxx
time	RFC xxxx
time-taken	RFC xxxx
c-groupid	RFC xxxx
s-ip	RFC xxxx
s-hostname	RFC xxxx
s-port	RFC xxxx
cs-method	RFC xxxx
cs-uri	RFC xxxx
u-uri	RFC xxxx
protocol	RFC xxxx
sc-status	RFC xxxx
sc-total-bytes	RFC xxxx
sc-entity-bytes	RFC xxxx
cs(insert_HTTP_header_name_here)	RFC xxxx
sc(insert_HTTP_header_name_here)	RFC xxxx
s-ccid	RFC xxxx
s-sid	RFC xxxx
s-cached	RFC xxxx

Figure 12

[Instructions to IANA: Replace "RFC xxxx" above by the RFC number of the present document]

Within the registry, names are to be allocated by IANA according to the "Specification Required" policy specified in [RFC5226]. Field names are to be allocated by IANA with a format of NHTABSTRING (see Section 3.1). All field names defined in the logging file are case-insensitive as per the basic ABNF([RFC5234]).

6.5. CDNI Logging MIME Media Type

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

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

Payload Type	Specification
logging-file	[RFCthis]

Figure 13: MIME Media Type payload

The purpose of the logging-file payload type is to distinguish between CDNI Logging Files and other CDNI messages.

Interface: LI

Encoding: see Section 3.2, Section 3.3 and Section 3.4

7. Security Considerations

7.1. Authentication, Authorization, Confidentiality, Integrity Protection

An implementation of the CDNI Logging interface MUST support TLS transport of the CDNI Logging feed (Section 4.1) and of the CDNI Logging File pull (Section 4.2) as per [RFC2818] and [RFC7230].

TLS MUST be used by the server-side and the client-side of the CDNI Logging feed, as well as the server-side and the client-side of the CDNI Logging File pull mechanism, including authentication of the remote end, unless alternate methods are used for ensuring the security of the information exchanged over the LI interface (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

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

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

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

- 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

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

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

The SHA256-hash directive inside the CDNI Logging File provides additional integrity protection, this time targeting potential corruption of the CDNI logging information during the CDNI Logging File generation, storage or exchange. This mechanism does not itself allow restoration of the corrupted CDNI Logging information, but it allows detection of such corruption and therefore triggering of appropriate corrective actions (e.g., discard of corrupted information, attempt to re-obtain the CDNI Logging information). Note that the SHA256-hash does not protect against tampering by a third party, since such a third party could have recomputed and updated the SHA256-hash after tampering. Protection against third party tampering, when the CDNI Logging File is communicated over the CDN Logging Interface, can be achieved as discussed above through the use of TLS.

7.2. Denial of Service

This document does not define specific mechanism to protect against Denial of Service (DoS) attacks on the Logging Interface. However, the CDNI Logging feed and CDNI Logging pull endpoints are typically to be accessed only by a very small number of valid remote endpoints and therefore can be easily protected against DoS attacks through the usual conventional DOS protection mechanisms such as firewalling or use of Virtual Private Networks (VPNs).

Protection of dCDN Surrogates against spoofed delivery requests is outside the scope of the CDNI Logging interface.

7.3. Privacy

CDNs have the opportunity to collect detailed information about the downloads performed by End Users. A dCDN is expected to collect such information into CDNI Logging Files, which are then communicated to an uCDN.

Having detailed CDNI logging information known by the dCDN in itself does not represent a particular privacy concern since the dCDN is obviously fully aware of all information logged since it generated the information in the first place.

Transporting detailed CDNI logging information over the HTTP based CDNI Logging Interface does not represent a particular privacy concern because it is protected by usual IETF privacy-protection mechanism (e.g.,TLS).

When HTTP redirection is used between the uCDN and the dCDN, making detailed CDNI logging information known to the uCDN does not represent a particular privacy concern because the uCDN is already exposed at request redirection time to most of the information that shows up as CDNI logging information (e.g., enduser IP@, URL, HTTP headers). When DNS redirection is used between the uCDN and the dCDN, there are cases where there is no privacy concern in making detailed CDNI logging information known to the uCDN; this may be the case, for example, where (1) it is considered that because the uCDN has the authority (with respect to the CSP) and control on how the requests are delivered (including whether it is served by the uCDN itself or by a dCDN), the uCDN is entitled to access all detailed information related to the corresponding deliveries, and (2) there is no legal reasons to restrict access by the uCDN to all these detailed information. Conversely, still when DNS redirection is used between the uCDN and the dCDN, there are cases where there may be some privacy concern in making detailed CDNI logging information known to the uCDN; this may be the case, for example, because the uCDN is in a different jurisdiction to the dCDN resulting is some legal reasons to restrict access by the uCDN to all the detailed information related to the deliveries. In this latter case, the privacy concern can be taken into account when the uCDN and dCDN agree about which fields are to be conveyed inside the CDNI Logging Files and which privacy protection mechanism is to be used as discussed in the definition of the c-groupid field specified in Section 3.4.1.

Another privacy concern arises from the fact that large volumes of detailed information about content delivery to users, potentially traceable back to indvidual 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 personnal information or information that can be exploited to derive personal information. Where this is a concern, the CDNI Logging interface specification allows the dCDN to not include the cs-uri and to include a u-uri that removes (or hides) the sensitive part of the query string and allows the dCDN to not include the cs(<HTTP-header-name>) fields corresponding to HTTP headers associated with cookies.

8. Acknowledgments

This document borrows from the W3C Extended Log Format [ELF].

Rob Murray significantly contributed into the text of Section 4.1.

The authors thank Ben Niven-Jenkins, Kevin Ma, David Mandelberg and Ray van Brandenburg for their ongoing input.

Brian Trammel and Rich Salz made significant contributions into making this interface privacy-friendly.

Finally, we also thank Sebastien Cubaud, Pawel Grochocki, Christian Jacquenet, Yannick Le Louedec, Anne Marrec, Emile Stephan, Fabio Costa, Sara Oueslati, Yvan Massot, Renaud Edel, Joel Favier and the contributors of the EU FP7 OCEAN project for their input in the early versions of this document.

9. References

9.1. Normative References

[AES] NIST, "Advanced Encryption Standard (AES)", August 2015, <FIPS 197>.

[GCM] NIST, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", November 2007, <SP 800-38D>.

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

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <http://www.rfc-editor.org/info/rfc4122>.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <http://www.rfc-editor.org/info/rfc4287>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5005] Nottingham, M., "Feed Paging and Archiving", RFC 5005, DOI 10.17487/RFC5005, September 2007, <http://www.rfc-editor.org/info/rfc5005>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <http://www.rfc-editor.org/info/rfc7232>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <http://www.rfc-editor.org/info/rfc7233>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <http://www.rfc-editor.org/info/rfc7234>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[SHA-3] NIST, "SHA-3 STANDARD: PERMUTATION-BASED HASH AND EXTENDABLE OUTPUT FUNCTIONS", November 2001, <http://dx.doi.org/10.6028/NIST.FIPS.202>.

9.2. Informative References

[CHAR_SET] "IANA Character Sets registry", <http://www.iana.org/assignments/character-sets/character-sets.xml>.

[ELF] Phillip M. Hallam-Baker, and Brian Behlendorf, "Extended Log File Format, W3C (work in progress), WD-logfile-960323", <http://www.w3.org/TR/WD-logfile.html>.

[I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-17 (work in progress), May 2016.

[I-D.ietf-tls-rfc5246-bis]
Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-rfc5246-bis-00 (work in progress), April 2014.

[I-D.snell-atompub-link-extensions]
Snell, J., "Atom Link Extensions", draft-snell-atompub-link-extensions-09 (work in progress), June 2012.

[RFC1945] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, DOI 10.17487/RFC1945, May 1996, <http://www.rfc-editor.org/info/rfc1945>.

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

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://www.rfc-editor.org/info/rfc6265>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <http://www.rfc-editor.org/info/rfc6707>.

[RFC6770] Bertrand, G., Ed., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, DOI 10.17487/RFC6770, November 2012, <http://www.rfc-editor.org/info/rfc6770>.

[RFC6983] van Brandenburg, R., van Deventer, O., Le Faucheur, F., and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware Content Distribution Network Interconnection (CDNI)", RFC 6983, DOI 10.17487/RFC6983, July 2013, <http://www.rfc-editor.org/info/rfc6983>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <http://www.rfc-editor.org/info/rfc7336>.

[RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <http://www.rfc-editor.org/info/rfc7337>.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <http://www.rfc-editor.org/info/rfc7736>.

Authors' Addresses

Francois Le Faucheur (editor)
FR

Phone: +33 6 19 98 50 90
Email: flefauch@gmail.com

Gilles Bertrand (editor)

Phone: +41 76 675 91 44
Email: gilbertrand@gmail.com

Iuniana Oprescu (editor)
FR

Email: iuniana.oprescu@gmail.com

Roy Peterkofsky
Google Inc.
345 Spear St, 4th Floor
San Francisco CA 94105
USA

Email: peterkofsky@google.com

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

Copyright Notice

Table of Contents

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: RFCthis [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.

```
ptype = 1*256(ptype-char)
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:

- o 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).

- o 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.

- o 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.

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, <http://www.rfc-editor.org/info/rfc5226>.

3.2. Informative References

[RFC6770] Bertrand, G., Ed., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, DOI 10.17487/RFC6770, November 2012, <http://www.rfc-editor.org/info/rfc6770>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <http://www.rfc-editor.org/info/rfc7336>.

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 1, 2017

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

August 28, 2016

CDN Interconnection Metadata
draft-ietf-cdni-metadata-21

Abstract

The Content Delivery Networks Interconnection (CDNI) metadata interface enables interconnected Content Delivery Networks (CDNs) to exchange content distribution metadata in order to enable content acquisition and delivery. The CDNI metadata associated with a piece of content provides a downstream CDN with sufficient information for the downstream CDN to service content requests on behalf of an upstream CDN. This document describes both a base set of CDNI metadata and the protocol for exchanging that metadata.

Requirements Language

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

The CDNI metadata interface is discussed in [RFC7336] along with four other interfaces that can be used to compose a CDNI solution (CDNI Control interface, CDNI Request Routing Redirection interface, CDNI Footprint & Capabilities Advertisement interface and CDNI Logging interface). [RFC7336] describes each interface and the relationships between them. The requirements for the CDNI metadata interface are specified in [RFC7337].

The CDNI metadata associated with a piece of content (or with a set of content) provides a dCDN with sufficient information for servicing content requests on behalf of an uCDN, in accordance with the policies defined by the uCDN.

This document defines the CDNI metadata interface which enables a dCDN to obtain CDNI metadata from an uCDN so that the dCDN can properly process and respond to:

- 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).

- A HTTP web service for the transfer of CDNI metadata (Section 6).

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

Additionally, the following terms are used throughout this document and are defined as follows:

- o Object - a collection of properties.
- o Property - a key and value pair where the key is a property name and the value is the property value or another object.

This document uses the phrase "[Object] A contains [Object] B" for simplicity when a strictly accurate phrase would be "[Object] A contains or references (via a Link object) [Object] B".

1.2. Supported Metadata Capabilities

Only the metadata for a small set of initial capabilities is specified in this document. This set provides the minimum amount of metadata for basic CDN interoperability while still meeting the requirements set forth by [RFC7337].

The following high-level functionality can be configured via the CDNI metadata objects specified in Section 4:

- o Acquisition Source: Metadata for allowing a dCDN to fetch content from a uCDN.
- o Delivery Access Control: Metadata for restricting (or permitting) access to content based on any of the following factors:
  - * Location
  - * Time Window
  - * Delivery Protocol
- o Delivery Authorization: Metadata for authorizing dCDN user agent requests.
- o Cache Control: Metadata for controlling cache behavior of the dCDN.

The metadata encoding described by this document is extensible in order to allow for future additions to this list.

The set of metadata specified in this document covers the initial capabilities above. It is only intended to support CDN interconnection for the delivery of content by a dCDN using HTTP/1.1 [RFC7230] and for a dCDN to be able to acquire content from a uCDN using either HTTP/1.1 or HTTP/1.1 over TLS [RFC2818].

Supporting CDN interconnection for the delivery of content using unencrypted HTTP/2 [RFC7540] (as well as for a dCDN to acquire content using unencrypted HTTP/2 or HTTP/2 over TLS) requires the registration of these protocol names in the CDNI Metadata Protocol Types registry Section 7.3.

Delivery of content using HTTP/1.1 over TLS or HTTP/2 over TLS SHOULD follow the guidelines set forth in [RFC7525]. Offline configuration of TLS parameters between CDNs is beyond the scope of this document.

2. Design Principles

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

1. Cacheability of CDNI metadata objects;

2. Deterministic mapping from redirection requests and content requests to CDNI metadata properties;

3. Support for DNS redirection as well as application-specific redirection (for example HTTP redirection);

4. Minimal duplication of CDNI metadata; and

5. Leveraging of existing protocols.

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

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

Support for both HTTP and DNS redirection ensures that the CDNI metadata meets the same design principles for both HTTP and DNS based redirection schemes.

Minimal duplication of CDNI metadata improves storage efficiency in the CDNs.

Leveraging existing protocols avoids reinventing common mechanisms such as data structure encoding (by leveraging I-JSON [RFC7493]) and data transport (by leveraging HTTP [RFC7230]).

3. CDNI Metadata object model

The CDNI metadata object model describes a data structure for mapping redirection requests and content requests to metadata properties. Metadata properties describe how to acquire content from an uCDN, authorize access to content, and deliver content from a dCDN. The object model relies on the assumption that these metadata properties can be grouped based on the hostname of the content and subsequently on the resource path (URI) of the content. The object model associates a set of CDNI metadata properties with a Hostname to form a default set of metadata properties for content delivered on behalf of that Hostname. That default set of metadata properties can be overridden by properties that apply to specific paths within a URI.

Different Hostnames and URI paths will be associated with different sets of CDNI metadata properties in order to describe the required behaviour when a dCDN surrogate or request router is processing User Agent requests for content at that Hostname and URI path. As a result of this structure, significant commonality could exist between the CDNI metadata properties specified for different Hostnames, different URI paths within a Hostname and different URI paths on different Hostnames. For example the definition of which User Agent IP addresses should be grouped together into a single network or geographic location is likely to be common for a number of different Hostnames; although a uCDN is likely to have several different policies configured to express geo-blocking rules, it is likely that a single geo-blocking policy could be applied to multiple Hostnames delivered through the CDN.

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

Important Note: Any CDNI metadata object A that contains another CDNI metadata object B can include a Link object specifying a URI that can be used to retrieve object B, instead of embedding object B within object A. The remainder of this document uses the phrase "[Object] A contains [Object] B" for simplicity when a strictly accurate phrase would be "[Object] A contains or references (via a Link object) [Object] B". It is generally a deployment choice for the uCDN implementation to decide when to embed CDNI metadata objects and when to reference separate resources via Link objects.

Section 3.1 introduces a high level description of the HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects, and describes the relationships between them.

Section 3.2 introduces a high level description of the CDNI GenericMetadata object which represents the level at which CDNI metadata override occurs between HostMetadata and PathMetadata objects.

Section 4 describes in detail the specific CDNI metadata objects and properties specified by this document which can be contained within a CDNI GenericMetadata object.

3.1. HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects

The relationships between the HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects are described in Figure 1.

```
   +---------+      +---------+      +------------+
   |HostIndex+-(*)->|HostMatch+-(1)->|HostMetadata+-------(*)------+
   +---------+      +---------+      +------+-----+                |
                                            |                      |
                                           (*)                     |
                                            |                      V
   --> Contains or References               V         ******************
   (1) One and only one                +---------+    *Generic Metadata*
   (*) Zero or more               +--->|PathMatch|    *     Objects    *
                                  |    +----+---++    ******************
                                  |         |   |                  ^
                                 (*)       (1) (1) +------------+  |
                                  |         |   +->|PatternMatch|  |
                                  |         V      +------------+  |
                                  |  +------------+                |
                                  +--+PathMetadata+-------(*)------+
                                     +------------+
```

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

A HostIndex object (see Section 4.1.1) contains an array of HostMatch objects (see Section 4.1.2) that contain Hostnames (and/or IP addresses) for which content requests might be delegated to the dCDN. The HostIndex is the starting point for accessing the uCDN CDNI metadata data store. It enables the dCDN to deterministically discover which CDNI metadata objects it requires in order to deliver a given piece of content.

The HostIndex links Hostnames (and/or IP addresses) to HostMetadata objects (see Section 4.1.3) via HostMatch objects. A HostMatch object defines a Hostname (or IP address) to match against a requested host and contains a HostMetadata object.

HostMetadata objects contain the default GenericMetadata objects (see Section 4.1.7) required to serve content for that host. When looking up CDNI metadata, the dCDN looks up the requested Hostname (or IP address) against the HostMatch entries in the HostIndex, from there it can find HostMetadata which describes the default metadata properties for each host as well as PathMetadata objects (see Section 4.1.6), via PathMatch objects (see Section 4.1.4). PathMatch objects define patterns, contained inside PatternMatch objects (see Section 4.1.5), to match against the requested URI path. PatternMatch objects contain the pattern strings and flags that describe the URI path that a PathMatch applies to. PathMetadata objects contain the GenericMetadata objects that apply to content requests matching the defined URI path pattern. PathMetadata properties override properties previously defined in HostMetadata or less specific PathMatch paths. PathMetadata objects can contain additional PathMatch objects to recursively define more specific URI paths to which GenericMetadata properties might be applied.

A GenericMetadata object contains individual CDNI metadata objects which define the specific policies and attributes needed to properly deliver the associated content. For example, a GenericMetadata object could describe the source from which a CDN can acquire a piece of content. The GenericMetadata object is an atomic unit that can be referenced by HostMetadata or PathMetadata objects.

For example, if "example.com" is a content provider, a HostMatch object could include an entry for "example.com" with the URI of the associated HostMetadata object. The HostMetadata object for "example.com" describes the metadata properties which apply to "example.com" and could contain PathMatches for "example.com/movies/*" and "example.com/music/*", which in turn reference corresponding PathMetadata objects that contain the properties for those more specific URI paths. The PathMetadata object for "example.com/movies/*" describes the properties which apply to that URI path. It could also contain a PathMatch object for "example.com/movies/hd/*" which would reference the corresponding PathMetadata object for the "example.com/movies/hd/" path prefix.

The relationships in Figure 1 are also represented in tabular format in Table 1 below.

Data Object	Objects it contains or references
HostIndex	0 or more HostMatch objects.
HostMatch	1 HostMetadata object.
HostMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.
PathMatch	1 PatternMatch object. 1 PathMetadata object.
PatternMatch	Does not contain or reference any other objects.
PathMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.

Table 1: Relationships between CDNI Metadata Objects (Table Representation)

3.2. Generic CDNI Metadata Objects

The HostMetadata and PathMetadata objects contain other CDNI metadata objects that contain properties which describe how User Agent requests for content should be processed, for example where to acquire the content from, authorization rules that should be applied, geo-blocking restrictions, and so on. Each such CDNI metadata object is a specialization of a CDNI GenericMetadata object. The GenericMetadata object abstracts the basic information required for metadata override and metadata distribution, from the specifics of any given property (i.e., property semantics, enforcement options, etc.).

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

Although a CDN MUST NOT serve content to a User Agent if a "mandatory-to-enforce" property cannot be enforced, it could still be "safe-to-redistribute" that metadata to another CDN without modification. For example, in the cascaded CDN case, a transit CDN (tCDN) could pass through "mandatory-to-enforce" metadata to a dCDN.

For metadata which does not require customization or translation (i.e., metadata that is "safe-to-redistribute"), the data representation received off the wire MAY be stored and redistributed without being understood or supported by the transit CDN. However, for metadata which requires translation, transparent redistribution of the uCDN metadata values might not be appropriate. Certain metadata can be safely, though perhaps not optimally, redistributed unmodified. For example, source acquisition address might not be optimal if transparently redistributed, but it might still work.

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

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

Table 2 describes the action to be taken by a transit CDN (tCDN) for the different combinations of "mandatory-to-enforce" (MtE) and "safe-to-redistribute" (StR) properties, when the tCDN either does or does not understand the metadata in question:

MtE	StR	Metadata Understood by tCDN	Action
False	True	True	Can serve and redistribute.
False	True	False	Can serve and redistribute.
False	False	False	Can serve. MUST set "incomprehensible" to True when redistributing.
False	False	True	Can serve. Can redistribute after transforming the metadata (if the CDN knows how to do so safely), otherwise MUST set "incomprehensible" to True when redistributing.
True	True	True	Can serve and redistribute.
True	True	False	MUST NOT serve but can redistribute.
True	False	True	Can serve. Can redistribute after transforming the metadata (if the CDN knows how to do so safely), otherwise MUST set "incomprehensible" to True when redistributing.
True	False	False	MUST NOT serve. MUST set "incomprehensible" to True when redistributing.

Table 2: Action to be taken by a tCDN for the different combinations of MtE and StR properties

Table 3 describes the action to be taken by a dCDN for the different combinations of "mandatory-to-enforce" (MtE) and "incomprehensible" (Incomp) properties, when the dCDN either does or does not understand the metadata in question:

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

Table 3: Action to be taken by a dCDN for the different combinations of MtE and Incomp properties

3.3. Metadata Inheritance and Override

In the metadata object model, a HostMetadata object can contain multiple PathMetadata objects (via PathMatch objects). Each PathMetadata object can in turn contain other PathMetadata objects. HostMetadata and PathMetadata objects form an inheritance tree where each node in the tree inherits or overrides the property values set by its parent.

GenericMetadata objects of a given type override all GenericMetadata objects of the same type previously defined by any parent object in the tree. GenericMetadata objects of a given type previously defined by a parent object in the tree are inherited when no object of the same type is defined by the child object. For example, if HostMetadata for the host "example.com" contains GenericMetadata objects of type LocationACL and TimeWindowACL, while a PathMetadata object which applies to "example.com/movies/*" defines an alternate GenericMetadata object of type TimeWindowACL, then:

- o the TimeWindowACL defined in the PathMetadata would override the TimeWindowACL defined in the HostMetadata for all User Agent requests for content under "example.com/movies/", and

- o the LocationACL defined in the HostMetadata would be inherited for all User Agent requests for content under "example.com/movies/".

A single HostMetadata or PathMetadata object MUST NOT contain multiple GenericMetadata objects of the same type. If an array of GenericMetadata contains objects of duplicate types, the receiver MUST ignore all but the first object of each type.

4. CDNI Metadata objects

Section 4.1 provides the definitions of each metadata object type introduced in Section 3. These metadata objects are described as structural metadata objects as they provide the structure for host and URI path-based inheritance and identify which GenericMetadata objects apply to a given User Agent content request.

Section 4.2 provides the definitions for a base set of core metadata objects which can be contained within a GenericMetadata object. These metadata objects govern how User Agent requests for content are handled. GenericMetadata objects can contain other GenericMetadata as properties; these can be referred to as sub-objects). As with all CDNI metadata objects, the value of the GenericMetadata sub-objects can be either a complete serialized representation of the sub-object, or a Link object that contains a URI that can be dereferenced to retrieve the complete serialized representation of the property sub-object.

Section 6.5 discusses the ability to extend the base set of GenericMetadata objects specified in this document with additional standards-based or vendor specific GenericMetadata objects that might be defined in the future in separate documents.

dCDNs and tCDNs MUST support parsing of all CDNI metadata objects specified in this document. A dCDN does not have to implement the underlying functionality represented by non-structural GenericMetadata objects (though that might restrict the content that a given dCDN will be able to serve). uCDNs as generators of CDNI metadata only need to support generating the CDNI metadata that they need in order to express the policies required by the content they are describing. See Section 6.4 for more details on the specific encoding rules for CDNI metadata objects.

Note: In the following sections, the term "mandatory-to-specify" is used to convey which properties MUST be included for a given structural or GenericMetadata object. When mandatory-to-specify is specified as "Yes" for an individual property, it means that if the object containing that property is included in a metadata response, then the mandatory-to-specify property MUST also be included (directly or by reference) in the response, e.g., a HostMatch property object without a host to match against does not make sense, therefore, the host property is mandatory-to-specify inside a HostMatch object.

4.1. Definitions of the CDNI structural metadata objects

Each of the sub-sections below describe the structural objects introduced in Section 3.1.

4.1.1. HostIndex

The HostIndex object is the entry point into the CDNI metadata hierarchy. It contains an array of HostMatch objects. An incoming content request is checked against the Hostname (or IP address) specified by each of the listed HostMatch objects to find the HostMatch object which applies to the request.

Property: hosts

Description: Array of HostMatch objects. Hosts (HostMatch objects) MUST be evaluated in the order they appear and the first HostMatch object that matches the content request being processed MUST be used.

Type: Array of HostMatch objects

Mandatory-to-Specify: Yes.

Example HostIndex object containing two HostMatch objects, where the first HostMatch object is embedded and the second HostMatch object is referenced:

```
{
  "hosts": [
    {
      <Properties of embedded HostMatch object>
    },
    {
      "type": "MI.HostMatch",
      "href": "https://metadata.ucdn.example/hostmatch1234"
    }
  ]
}
```

4.1.2. HostMatch

The HostMatch object contains a Hostname or IP address to match against content requests. The HostMatch object also contains a HostMetadata object to apply if a match is found.

Property: host

Description: Hostname or IP address and optional port to match against the requested host, i.e., the [RFC3986] host and port. In order for a Hostname or IP address in a content request to match the Hostname or IP address in the host property the value from the content request when converted to lowercase MUST be identical to the value of the host property when converted to lowercase. All implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986]. IPv6 addresses MUST be encoded in one of the IPv6 address formats specified in [RFC5952] although receivers MUST support all IPv6 address formats specified in [RFC4291]. Hostnames MUST conform to the Domain Name System (DNS) syntax defined in [RFC1034] and [RFC1123]. Internationalized Domain Names (IDN) must first be transformed to the the A-label form [RFC5890] as per [RFC5891].

Type: Endpoint

Mandatory-to-Specify: Yes.

Property: host-metadata

Description: CDNI metadata to apply when delivering content that matches this host.

Type: HostMetadata

Mandatory-to-Specify: Yes.

Example HostMatch object with an embedded HostMetadata object:

```
{
  "host": "video.example.com",
  "host-metadata" : {
    <Properties of embedded HostMetadata object>
  }
}
```

Example HostMatch object referencing (via a Link object, see Section 4.3.1) a HostMetadata object:

```
{
  "host": "video.example.com",
  "host-metadata" : {
    "type": "MI.HostMetadata",
    "href": "https://metadata.ucdn.example/host1234"
  }
}
```


4.1.3. HostMetadata

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

Property: metadata

Description: Array of host related metadata.

Type: Array of GenericMetadata objects

Mandatory-to-Specify: Yes.

Property: paths

Description: Path specific rules. Path patterns (PathMatch objects) MUST be evaluated in the order they appear and the first (and only the first) PathMatch object that matches the content request being processed MUST be used.

Type: Array of PathMatch objects

Mandatory-to-Specify: No.

Example HostMetadata object containing a number of embedded GenericMetadata objects that will describe the default metadata for the host and an embedded PathMatch object that contains a path for which metadata exists that overrides the default metadata for the host:

```
{
  "metadata": [
    {
      <Properties of 1st embedded GenericMetadata object>
    },
    {
      <Properties of 2nd embedded GenericMetadata object>
    },

 ...

    {
      <Properties of Nth embedded GenericMetadata object>
    }
  ],
  "paths": [
    {
      <Properties of embedded PathMatch object>
    }
  ]
}
```

4.1.4. PathMatch

A PathMatch object contains PatternMatch object with a path to match against a resource's URI path, as well as how to handle URI query parameters. The PathMatch also contains a PathMetadata object with GenericMetadata to apply if the resource's URI matches the pattern within the PatternMatch object.

Property: path-pattern

Description: Pattern to match against the requested resource's URI.

Type: PatternMatch

Mandatory-to-Specify: Yes.

Property: path-metadata

Description: CDNI metadata to apply when delivering content that matches the associated PatternMatch.

Type: PathMetadata

Mandatory-to-Specify: Yes.

Example PathMatch object referencing the PathMetadata object to use for URIs that match the case-sensitive URI path pattern "/movies/*" (contained within an embedded PatternMatch object):

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

4.1.5. PatternMatch

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

Property: pattern

Description: A pattern for matching against the URI path, i.e., against the [RFC3986] path-absolute. The pattern can contain the wildcards * and ?, where * matches any sequence of [RFC3986] pchar or "/" characters (including the empty string) and ? matches exactly one [RFC3986] pchar character. The three literals $, * and ? MUST be escaped as $$, $* and $? (where $ is the designated escape character). All other characters are treated as literals.

Type: String

Mandatory-to-Specify: Yes.

Property: case-sensitive

Description: Flag indicating whether or not case-sensitive matching should be used. Note: Case-insensitivity applies to ALPHA characters in the URI path prior to percent-decoding [RFC3986].

Type: Boolean

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

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


```
{
  "pattern": "/movies/*",
  "case-sensitive": true
}
```


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


```
{
  "pattern": "/movies/*",
  "case-sensitive": true
}
```


4.1.6. PathMetadata

A PathMetadata object contains the CDNI metadata properties for content requests that match against the associated URI path (defined in a PathMatch object).

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

Property: metadata

Description: Array of path related metadata.

Type: Array of GenericMetadata objects

Mandatory-to-Specify: Yes.

Property: paths

Description: Path specific rules. Path patterns (PathMatch objects) MUST be evaluated in the order they appear and the first (and only the first) PathMatch object that matches the content request being processed MUST be used.

Type: Array of PathMatch objects

Mandatory-to-Specify: No.

Example PathMetadata object containing a number of embedded GenericMetadata objects that describe the metadata to apply for the URI path defined in the parent PathMatch object, as well as a more specific PathMatch object.

```
{
  "metadata": [
    {
      <Properties of 1st embedded GenericMetadata object>
    },
    {
      <Properties of 2nd embedded GenericMetadata object>
    },

 ...

    {
      <Properties of Nth embedded GenericMetadata object>
    }
  ],
  "paths": [
    {
      <Properties of embedded PathMatch object>
    }
  ]
}
```

4.1.7. GenericMetadata

A GenericMetadata object is a wrapper for managing individual CDNI metadata properties in an opaque manner.

Property: generic-metadata-type

Description: Case-insensitive CDNI metadata object type.

Type: String containing the CDNI Payload Type [RFC7736] of the object contained in the generic-metadata-value property (see Table 4).

Mandatory-to-Specify: Yes.

Property: generic-metadata-value

Description: CDNI metadata object.

Type: Format/Type is defined by the value of generic-metadata-type property above. Note: generic-metadata-values MUST NOT name any properties "href" (see Section 4.3.1).

Mandatory-to-Specify: Yes.

Property: mandatory-to-enforce

Description: Flag identifying whether or not the enforcement of the property metadata is required.

Type: Boolean

Mandatory-to-Specify: No. Default is to treat metadata as mandatory to enforce (i.e., a value of True).

Property: safe-to-redistribute

Description: Flag identifying whether or not the property metadata can be safely redistributed without modification.

Type: Boolean

Mandatory-to-Specify: No. Default is allow transparent redistribution (i.e., a value of True).

Property: incomprehensible

Description: Flag identifying whether or not any CDN in the chain of delegation has failed to understand and/or failed to properly transform this metadata object. Note: This flag only applies to metadata objects whose safe-to-redistribute property has a value of False.

Type: Boolean

Mandatory-to-Specify: No. Default is comprehensible (i.e., a value of False).

Example GenericMetadata object containing a metadata object that applies to the applicable URI path and/or host (within a parent PathMetadata and/or HostMetadata object, respectively):

```
 {
   "mandatory-to-enforce": true,
   "safe-to-redistribute": true,
   "incomprehensible": false,
   "generic-metadata-type": <CDNI Payload Type of this metadata object>,
   "generic-metadata-value":
     {
       <Properties of this metadata object>
     }
 }
```

4.2. Definitions of the initial set of CDNI Generic Metadata objects

The objects defined below are intended to be used in the GenericMetadata object generic-metadata-value field as defined in Section 4.1.7 and their generic-metadata-type property MUST be set to the appropriate CDNI Payload Type as defined in Table 4.

4.2.1. SourceMetadata

Source metadata provides the dCDN with information about content acquisition, i.e., how to contact an uCDN Surrogate or an Origin Server to obtain the content to be served. The sources are not necessarily the actual Origin Servers operated by the CSP but might be a set of Surrogates in the uCDN.

Property: sources

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

Type: Array of Source objects (see Section 4.2.1.1)

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

Example SourceMetadata object (which contains two Source objects) that describes which servers the dCDN should use for acquiring content for the applicable URI path and/or host:

```
{
  "generic-metadata-type": "MI.SourceMetadata",
  "generic-metadata-value":
    {
      "sources": [
        {
          "endpoints": [
            "a.service123.ucdn.example",
            "b.service123.ucdn.example"
            ],
          "protocol": "http/1.1"
        },
        {
          "endpoints": ["origin.service123.example"],
          "protocol": "http/1.1"
        }
      ]
    }
}
```


4.2.1.1. Source

A Source object describes the source to be used by the dCDN for content acquisition (e.g., a Surrogate within the uCDN or an alternate Origin Server), the protocol to be used, and any authentication method to be used when contacting that source.

Endpoints within a Source object MUST be treated as equivalent/equal. A uCDN can specify an array of sources in preference order within a SourceMetadata object, and then for each preference ranked Source object, a uCDN can specify an array of endpoints that are equivalent (e.g., a pool of servers that are not behind a load balancer).

Property: acquisition-auth

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

Type: Auth (see Section 4.2.7)

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

Property: endpoints

Description: Origins from which the dCDN can acquire content. If multiple endpoints are specified they are all equal, i.e., the list is not in preference order (e.g., a pool of servers behind a load balancer).

Type: Array of Endpoint objects (See Section 4.3.3)

Mandatory-to-Specify: Yes.

Property: protocol

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

Type: Protocol (see Section 4.3.2)

Mandatory-to-Specify: Yes.

Example Source object that describes a pair of endpoints (servers) the dCDN can use for acquiring content for the applicable host and/or URI path:


```
{
  "endpoints": [
    "a.service123.ucdn.example",
    "b.service123.ucdn.example"
  ],
  "protocol": "http/1.1"
}
```


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

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

Property: locations

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

Type: Array of LocationRule objects (see Section 4.2.2.1)

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

Example LocationACL object that allows the dCDN to deliver content to any location/IP address:


```
{
  "generic-metadata-type": "MI.LocationACL",
  "generic-metadata-value":
    {
    }
}
```


Example LocationACL object (which contains a LocationRule object which itself contains a Footprint object) that only allows the dCDN to deliver content to User Agents in the USA:


```
{
  "generic-metadata-type": "MI.LocationACL",
  "generic-metadata-value":
    {
      "locations": [
        {
          "action": "allow",
          "footprints": [
            {
              "footprint-type": "countrycode",
              "footprint-value": ["us"]
            }
          ]
        }
      ]
    }
}
```

4.2.2.1. LocationRule

A LocationRule contains or references an array of Footprint objects and the corresponding action.

Property: footprints

Description: Array of footprints to which the rule applies.

Type: Array of Footprint objects (see Section 4.2.2.2)

Mandatory-to-Specify: Yes.

Property: action

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

Type: Enumeration [allow|deny] encoded as a lowercase string

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

Example LocationRule object (which contains a Footprint object) that allows the dCDN to deliver content to clients in the USA:


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


4.2.2.2. Footprint

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

Property: footprint-type

Description: Registered footprint type (see Section 7.2). The footprint types specified by this document are: "ipv4cidr" (IPv4CIDR, see Section 4.3.5), "ipv6cidr" (IPv6CIDR, see Section 4.3.6), "asn" (Autonomous System Number, see

Section 4.3.7) and "countrycode" (Country Code, see Section 4.3.8).

Type: Lowercase String

Mandatory-to-Specify: Yes.

Property: footprint-value

Description: Array of footprint values conforming to the specification associated with the registered footprint type. Footprint values can be simple strings (e.g., IPv4CIDR, IPv6CIDR, ASN, and CountryCode), however, other Footprint objects can be defined in the future, along with a more complex encoding (e.g., GPS coordinate tuples).

Type: Array of footprints

Mandatory-to-Specify: Yes.

Example Footprint object describing a footprint covering the USA:

```
{
  "footprint-type": "countrycode",
  "footprint-value": ["us"]
}
```

Example Footprint object describing a footprint covering the IP address ranges 192.0.2.0/24 and 198.51.100.0/24:

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

Example Footprint object describing a footprint covering the IP address ranges 2001:db8::/32:

```
{
  "footprint-type": "ipv6cidr",
  "footprint-value": ["2001:db8::/32"]
}
```

Example Footprint object describing a footprint covering the autonomous system 64496:

```
{
  "footprint-type": "asn",
  "footprint-value": ["as64496"]
}
```


4.2.3. TimeWindowACL

TimeWindowACL metadata defines time-based restrictions.

A TimeWindowACL which does not include a times property results in an action of allow all, meaning that delivery can be performed regardless of the time of the User Agent's request, otherwise a CDN MUST take the action from the first window to match against the current time. If two or more windows overlap, the first window that matches against the current time determines the action a CDN MUST take. If the times property is included but is empty, or if none of the listed windows matches the current time, then the result is an action of deny.

Although the LocationACL (see Section 4.2.2), TimeWindowACL, and ProtocolACL (see Section 4.2.4) are independent GenericMetadata objects, they can provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the LocationACL and denied based on the TimeWindowACL. The dCDN MUST use the logical AND of all ACLs (where 'allow' is true and 'deny' is false) to determine whether or not a request should be allowed.

Property: times

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

Type: Array of TimeWindowRule objects (see Section 4.2.3.1)

Mandatory-to-Specify: No. Default is allow all time windows.

Example TimeWIndowACL object (which contains a TimeWindowRule object which itself contains a TimeWIndow object) that only allows the dCDN to deliver content to clients between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC:

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


4.2.3.1. TimeWindowRule

A TimeWindowRule contains or references an array of TimeWindow objects and the corresponding action.

Property: windows

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

Type: Array of TimeWindow objects (see Section 4.2.3.2)

Mandatory-to-Specify: Yes.

Property: action

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

Type: Enumeration [allow|deny] encoded as a lowercase string

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

Example TimeWIndowRule object (which contains a TimeWIndow object) that only allows the dCDN to deliver content to clients between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC:

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


4.2.3.2. TimeWindow

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

Property: start

Description: The start time of the window.

Type: Time (see Section 4.3.4)

Mandatory-to-Specify: Yes.

Property: end

Description: The end time of the window.

Type: Time (see Section 4.3.4)

Mandatory-to-Specify: Yes.

Example TimeWIndow object that describes a time window from 09:00 01/01/2000 UTC to 17:00 01/01/2000 UTC:


```
{
  "start": 946717200,
  "end": 946746000
}
```


4.2.4. ProtocolACL Metadata

ProtocolACL metadata defines delivery protocol restrictions.

A ProtocolACL which does not include a protocol-acl property results in an action of allow all, meaning that delivery can be performed regardless of the protocol in the User Agent’s request, otherwise a CDN MUST take the action from the first protocol to match against the request protocol. If two or more request protocols overlap, the first protocol that matches the request protocol determines the action a CDN MUST take. If the protocol-acl property is included but is empty, or if none of the listed protocol matches the request protocol, then the result is an action of deny.

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

Property: protocol-acl

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

Type: Array of ProtocolRule objects (see Section 4.2.4.1)

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

Example ProtocolACL object (which contains a ProtocolRule object) that only allows the dCDN to deliver content using HTTP/1.1:


```
{
  "generic-metadata-type": "MI.ProtocolACL",
  "generic-metadata-value":
    {
      "protocol-acl": [
        {
          "action": "allow",
          "protocols": ["http/1.1"]
        }
      ]
    }
}
```


4.2.4.1. ProtocolRule

A ProtocolRule contains or references an array of Protocol objects and the corresponding action.

Property: protocols

Description: Array of protocols to which the rule applies.

Type: Array of Protocols (see Section 4.3.2)

Mandatory-to-Specify: Yes.

Property: action

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

Type: Enumeration [allow|deny] encoded as a lowercase string

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

Example ProtocolRule object (which contains a ProtocolRule object) that allows the dCDN to deliver content using HTTP/1.1:


```
{
  "action": "allow",
  "protocols": ["http/1.1"]
}
```


4.2.5. DeliveryAuthorization Metadata

Delivery Authorization defines authorization methods for the delivery of content to User Agents.

Property: delivery-auth-methods

Description: Options for authorizing content requests. Delivery for a content request is authorized if any of the authorization methods in the list is satisfied for that request.

Type: Array of Auth objects (see Section 4.2.7)

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

Example DeliveryAuthorization object (which contains an Auth object):

```
{
  "generic-metadata-type": "MI.DeliveryAuthorization",
  "generic-metadata-value":
    {
      "delivery-auth-methods": [
        {
          "auth-type": <CDNI Payload Type of this Auth object>,
          "auth-value":
          {
            <Properties of this Auth object>
          }
        }
      ]
    }
}
```

4.2.6. Cache

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

Cache keys are generated from the URI of the content request [RFC7234]. In some cases, a CDN or content provider might want certain path segments or query parameters to be excluded from the cache key generation. The Cache object provides guidance on what parts of the path and query string to include.

Property: exclude-path-pattern

Description: A pattern for matching against the URI path, i.e., against the [RFC3986] path-absolute. The pattern can contain the wildcards * and ?, where * matches any sequence of [RFC3986] pchar or "/" characters (including the empty string) and ? matches exactly one [RFC3986] pchar character. The three literals $, * and ? MUST be escaped as $$, $* and $? (where $ is the designated escape character). All other characters are treated as literals. Cache key generation MUST only include the portion of the path-absolute that matches the wildcard portions of the pattern. Note: Inconsistency between the PatternMatch pattern Section 4.1.5 and the exclude-path-pattern can result in inefficient caching.

Type: String

Mandatory-to-Specify: No. Default is to use the full URI path-absolute to generate the cache key.

Property: include-query-strings

Description: Allows a Surrogate to specify the URI query string parameters [RFC3986] to include when comparing the requested URI against the URIs in its cache for equivalence. Matching query parameters MUST be case-insensitive. If all query parameters should be ignored, then the list MUST be specified and MUST be empty. If a query parameter appears multiple times in the query string, each instance value MUST be aggregated prior to comparison. For consistent cache key generation, query parameters SHOULD be evaluated in the order specified in this array.

Type: Array of String

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

Example Cache object that instructs the dCDN to use the full URI path and ignore all query parameters:

```
{
  "generic-metadata-type": "MI.Cache",
  "generic-metadata-value":
  {
    "include-query-strings": []
  }
}
```

Example Cache object that instructs the dCDN to exclude the "CDNX" path prefix and only include the (case-insensitive) query parameters named "mediaid" and "providerid":

```
{
  "generic-metadata-type": "MI.Cache",
  "generic-metadata-value":
  {
    "exclude-path-pattern": "/CDNX/*",
    "include-query-strings": ["mediaid", "providerid"]
  }
}
```

Example Cache object that instructs the dCDN to exclude the "CDNX" path prefix, but includes all query parameters:

```
{
  "generic-metadata-type": "MI.Cache",
  "generic-metadata-value":
  {
    "exclude-path-pattern": "/CDNX/*"
  }
}
```


4.2.7. Auth

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

Note: This document does not define any Auth methods. Individual Auth methods are being defined separately (e.g., URI Signing [I-D.ietf-cdni-uri-signing]). The GenericMetadata which contain Auth objects is defined herein for convenience and so as not to be specific to any particular Auth method.

Property: auth-type

Description: Auth type (The CDNI Payload Type [RFC7736] of the GenericMetadata object contained in the auth-value property).

Type: String

Mandatory-to-Specify: Yes.

Property: auth-value

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

Type: GenericMetadata Object

Mandatory-to-Specify: Yes.

Example Auth object:

```
{
  "generic-metadata-type": "MI.Auth",
  "generic-metadata-value":
  {
    "auth-type": <CDNI Payload Type of this Auth object>,
    "auth-value":
      {
        <Properties of this Auth object>
      }
  }
}
```

4.2.8. Grouping

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

Property: ccid

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

Type: String

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

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

```
{
  "generic-metadata-type": "MI.Grouping",
  "generic-metadata-value":
  {
    "ccid": "ABCD"
  }
}
```

4.3. CDNI Metadata Simple Data Type Descriptions

This section describes the simple data types that are used for properties of CDNI metadata objects.

4.3.1. Link

A Link object can be used in place of any of the objects or properties described above. Link objects can be used to avoid duplication if the same metadata information is repeated within the metadata tree. When a Link object replaces another object, its href property is set to the URI of the resource and its type property is set to the CDNI Payload Type of the object it is replacing.

dCDNs can detect the presence of a Link object by detecting the presence of a property named "href" within the object. This means that GenericMetadata types MUST NOT contain a property named "href" because doing so would conflict with the ability for dCDNs to detect Link objects being used to reference a GenericMetadata object.

Property: href

Description: The URI of the addressable object being referenced.

Type: String

Mandatory-to-Specify: Yes.

Property: type

Description: The CDNI Payload type of the object being referenced.

Type: String

Mandatory-to-Specify: No. If the container specifies the type (e.g., the HostIndex object contains an array of HostMatch objects, so a Link object in the list of HostMatch objects must reference a HostMatch), then it is not necessary to explicitly specify a type.

Example Link object referencing a HostMatch object:

```
{
  "type": "MI.HostMatch",
  "href": "https://metadata.ucdn.example/hostmatch1234"
}
```

Example Link object referencing a HostMatch object, without an explicit type, inside a HostIndex object:

```
{
  "hosts": [
    {
      <Properties of embedded HostMatch object>
    },
    {
      "href": "https://metadata.ucdn.example/hostmatch1234"
    }
  ]
}
```


4.3.1.1. Link Loop Prevention

When following a Link, CDNI metadata clients SHOULD verify that the CDNI Payload Type of the object retrieved matches the expected CDNI Payload Type, as indicated by the link object. For GenericMetadata objects, type checks will prevent self references; however, incorrect linking can result in circular references for structural metadtata objects, specifically, PathMatch and PathMetadata objects Figure 1. To prevent the circular references, CDNI metadata clients SHOULD verify that no duplicate Links occur for PathMatch or PathMetadata objects.

4.3.2. Protocol

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

Type: String

Example:

"http/1.1"

4.3.3. Endpoint

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

All implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986]. IPv6 addresses MUST be encoded in one of the IPv6 address formats specified in [RFC5952] although receivers MUST support all IPv6 address formats specified in [RFC4291]. Hostnames MUST conform to the Domain Name System (DNS) syntax defined in [RFC1034] and [RFC1123]. Internationalized Domain Names (IDN) must first be transformed to the A-label form [RFC5890] as per [RFC5891].

Type: String

Example Hostname:

"metadata.ucdn.example"

Example IPv4 address:

"192.0.2.1"

Example IPv6 address (with port number):

"[2001:db8::1]:81"

4.3.4. Time

A time value expressed in seconds since the Unix epoch (i.e., zero hours, zero minutes, zero seconds, on January 1, 1970) Coordinated Universal Time (UTC) [POSIX].

Type: Integer

Example Time representing 09:00:00 01/01/2000 UTC:

946717200

4.3.5. IPv4CIDR

An IPv4address CIDR block encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986] followed by a / followed by an unsigned integer representing the leading bits of the routing prefix (i.e., IPv4 CIDR notation). Single IP addresses can be expressed as /32.

Type: String

Example IPv4 CIDR:

"192.0.2.0/24"

4.3.6. IPv6CIDR

An IPv6address CIDR block encoded in one of the IPv6 address formats specified in [RFC5952] followed by a / followed by an unsigned integer representing the leading bits of the routing prefix (i.e., IPv6 CIDR notation). Single IP addresses can be expressed as /128.

Type: String

Example IPv6 CIDR:

"2001:db8::/32"

4.3.7. ASN

An Autonomous System Number encoded as a string consisting of the characters "as" (in lowercase) followed by the Autonomous System number [RFC6793].

Type: String

Example ASN:

"as64496"

4.3.8. CountryCode

An ISO 3166-1 alpha-2 code [ISO3166-1] in lowercase.

Type: String

Example Country Code representing the USA:

"us"

5. CDNI Metadata Capabilities

CDNI metadata is used to convey information pertaining to content delivery from uCDN to dCDN. For optional metadata, it can be useful for the uCDN to know if the dCDN supports the underlying functionality described by the metadata, prior to delegating any content requests to the dCDN. If some metadata is "mandatory-to-enforce", and the dCDN does not support it, any delegated requests for content that requires that metadata will fail. The uCDN will likely want to avoid delegating those requests to that dCDN. Likewise, for any metadata which might be assigned optional values, it could be useful for the uCDN to know which values a dCDN supports, prior to delegating any content requests to that dCDN. If the optional value assigned to a given piece of content’s metadata is not supported by the dCDN, any delegated requests for that content can fail, so again the uCDN is likely to want to avoid delegating those requests to that dCDN.

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

6. CDNI Metadata interface

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

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

- o Dynamically as required by the dCDN to process received requests. For example in response to a query from an uCDN over the CDNI Request Routing Redirection interface (RI) [I-D.ietf-cdni-redirection] or in response to receiving a request for content from a User Agent. Or;

- o In advance of being required. For example in the case of pre-positioned CDNI metadata acquisition, initiated through the "CDNI Control interface / Triggers" (CI/T) interface [I-D.ietf-cdni-control-triggers].

The CDNI metadata interface is built on the principles of HTTP web services. In particular, this means that requests and responses over the interface are built around the transfer of representations of hyperlinked resources. A resource in the context of the CDNI metadata interface is any object in the object model (as described in Section 3 and Section 4).

To retrieve CDNI metadata, a CDNI metadata client (i.e., a client in the dCDN) first makes a HTTP GET request for the URI of the HostIndex which provides the CDNI metadata client with an array of Hostnames for which the uCDN can delegate content delivery to the dCDN. The CDNI metadata client can then obtain any other CDNI metadata objects by making a HTTP GET requests for any linked metadata objects it requires.

CDNI metadata servers (i.e., servers in the uCDN) are free to assign whatever structure they desire to the URIs for CDNI metadata objects and CDNI metadata clients MUST NOT make any assumptions regarding the structure of CDNI metadata URIs or the mapping between CDNI metadata objects and their associated URIs. Therefore any URIs present in the examples in this document are purely illustrative and are not intended to impose a definitive structure on CDNI metadata interface implementations.

6.1. Transport

The CDNI metadata interface uses HTTP as the underlying protocol transport [RFC7230].

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

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

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

6.2. Retrieval of CDNI Metadata resources

In the general case, a CDNI metadata server makes CDNI metadata objects available via a unique URIs and thus, in order to retrieve CDNI metadata, a CDNI metadata client first makes a HTTP GET request for the URI of the HostIndex which provides an array of Hostnames for which the uCDN can delegate content delivery to the dCDN.

In order to retrieve the CDNI metadata for a particular request the CDNI metadata client processes the received HostIndex object and finds the corresponding HostMetadata entry (by matching the hostname in the request against the hostnames listed in the HostMatch objects). If the HostMetadata is linked (rather than embedded), the CDNI metadata client then makes a GET request for the URI specified in the href property of the Link object which points to the HostMetadata object itself.

In order to retrieve the most specific metadata for a particular request, the CDNI metadata client inspects the HostMetadata for references to more specific PathMetadata objects (by matching the URI path in the request against the path-patterns in any PathMatch objects listed in the HostMetadata object). If any PathMetadata are found to match (and are linked rather than embedded), the CDNI metadata client makes another GET request for the PathMetadata. Each PathMetadata object can also include references to yet more specific metadata. If this is the case, the CDNI metadata client continues requesting PathMatch and PathMetadata objects recursively. The CDNI metadata client repeats this approach of processing metadata objects and retrieving (via HTTP GETs) any linked objects until it has all the metadata objects it requires in order to process the redirection request from an uCDN or the content request from a User Agent.

In cases where a dCDN is not able to retrieve the entire set of CDNI metadata associated with a User Agent request, or it has retrieved that metadata but it is stale according to standard HTTP caching rules and cannot be revalidated, for example because the uCDN is unreachable or returns a HTTP 4xx or 5xx status in response to some or all of the dCDN's CDNI metadata requests, the dCDN MUST NOT serve the requested content.

Where a dCDN is interconnected with multiple uCDNs, the dCDN needs to determine which uCDN's CDNI metadata should be used to handle a particular User Agent request.

When application level redirection (e.g., HTTP 302 redirects) is being used between CDNs, it is expected that the dCDN will be able to determine the uCDN that redirected a particular request from information contained in the received request (e.g., via the URI). With knowledge of which uCDN routed the request, the dCDN can choose the correct uCDN from which to obtain the HostIndex. Note that the HostIndexes served by each uCDN can be unique.

In the case of DNS redirection there is not always sufficient information carried in the DNS request from User Agents to determine the uCDN that redirected a particular request (e.g., when content from a given host is redirected to a given dCDN by more than one uCDN) and therefore dCDNs will have to apply local policy when deciding which uCDN's metadata to apply.

6.3. Bootstrapping

The URI for the HostIndex object of a given uCDN needs to be configured in the dCDN. All other objects/resources are then discoverable from the HostIndex object by following any links in the HostIndex object and through the referenced HostMetadata and PathMetadata objects and their GenericMetadata sub-objects.

Manual configuration of the URI for the HostIndex object is outside the scope of this document.

6.4. Encoding

CDNI metadata objects MUST be encoded as I-JSON objects [RFC7493] containing a dictionary of (key,value) pairs where the keys are the property names and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e., dependent on the CDNI Payload Type of the returned resource). Likewise, the values associated with each property (dictionary key) are dependent on the specific object being encoded (i.e., dependent on the CDNI Payload Type of the returned resource).

Dictionary keys (properties) in I-JSON are case sensitive. By convention, any dictionary key (property) defined by this document (for example, the names of CDNI metadata object properties) MUST be lowercase.

6.5. Extensibility

The set of GenericMetadata objects can be extended with additional (standards based or vendor specific) metadata objects through the specification of new GenericMetadata objects. The GenericMetadata object defined in Section 4.1.7 specifies a type field and a type-specific value field that allows any metadata to be included in either the HostMetadata or PathMetadata arrays.

As with the initial GenericMetadata types defined in Section 4.2, future GenericMetadata types MUST specify the information necessary for constructing and decoding the GenericMetadata object.

Any document which defines a new GenericMetadata type MUST:

1. Specify and register the CDNI Payload Type [RFC7736] used to identify the new GenericMetadata type being specified.

2. Define the set of properties associated with the new GenericMetadata object. GenericMetadata MUST NOT contain a property named "href" because doing so would conflict with the ability to detect Link objects (see Section 4.3.1).

3. Define a name, description, type, and whether or not the property is mandatory-to-specify.

4. Describe the semantics of the new type including its purpose and example of a use case to which it applies including an example encoded in I-JSON.

5. Describe the security and privacy consequences, for both the user-agent and the CDN, of the new GenericMetadata object.

6. Describe any relation to, conflict with, or obsolescence of other existing CDNI metadata objects.

Note: In the case of vendor specific extensions, vendor-identifying CDNI Payload Type names will decrease the possibility of GenericMetadata type collisions.

6.6. Metadata Enforcement

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

In cases where a uCDN sends metadata containing a GenericMetadata type that a dCDN does not support, the dCDN MUST enforce the semantics of the "mandatory-to-enforce" property. If a dCDN does not understand or is unable to perform the functions associated with any "mandatory-to-enforce" metadata, the dCDN MUST NOT service any requests for the corresponding content.

Note: Ideally, uCDNs would not delegate content requests to a dCDN that does not support the "mandatory-to-enforce" metadata associated with the content being requested. However, even if the uCDN has a priori knowledge of the metadata supported by the dCDN (e.g., via the FCI or through out-of-band negotiation between CDN operators), metadata support can fluctuate or be inconsistent (e.g., due to mis-communication, mis-configuration, or temporary outage). Thus, the dCDN MUST always evaluate all metadata associated with redirection and content requests and reject any requests where "mandatory-to-enforce" metadata associated with the content cannot be enforced.

6.7. Metadata Conflicts

It is possible that new metadata definitions will obsolete or conflict with existing GenericMetadata (e.g., a future revision of the CDNI metadata interface could redefine the Auth GenericMetadata object or a custom vendor extension could implement an alternate Auth metadata option). If multiple metadata (e.g., MI.Auth.v2, vendor1.Auth, and vendor2.Auth) all conflict with an existing GenericMetadata object (i.e., MI.Auth) and all are marked as "mandatory-to-enforce", it could be ambiguous which metadata should be applied, especially if the functionality of the metadata overlap.

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

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

6.8. Versioning

The version of CDNI metadata objects is conveyed inside the CDNI Payload Type that is included in either the HTTP Content-Type header, for example: "Content-Type: application/cdni; ptype=MI.HostIndex", when retrieved via a link, or in the link type (Section 4.3.1), generic-metadata-type (Section 4.1.7), or auth-type (Section 4.2.7) properties in the JSON payload. The CDNI Payload Type uniquely identifies the specification defining that object, including any relation to, conflicts with, or obsolescence of other metadata. There is no explicit version mapping requirement, however, for ease of understanding, metadata creators SHOULD make new versions of metadata easily visible via the CDNI Payload Type, e.g., by appending a version string. Note: A version string is optional on the first version, e.g., MI.HostIndex, but could be added for subsequent versions, e.g., MI.HostIndex.v2, MI.HostIndex.v3, etc.

Except when referenced by a Link object, nested metadata objects (i.e., structural metadata below the HostIndex; Source objects; Location, TimeWindow, and Protocol Rule objects; and Footprint and TimeWindow objects) can be serialized into a JSON payload without explicit CDNI Payload Type information. The type is inferred from the outer structural metadata, generic metadata, or auth object CDNI Payload Type. To avoid ambiguity when revising nestable metadata objects, any outer metadata object(s) MUST be reversioned and allocated new CDNI Payload Type(s) at the same time. For example, the MI.HostIndex object defined in this document contains an array of MI.HostMatch objects, which each in turn contains a MI.HostMetadata object. If a new MI.HostMetadata.v2 object were required, the outer MI.HostIndex and MI.HostMatch objects would need to be revised, e.g., to MI.HostIndex.v2 and MI.HostMatch.v2, respectively. Similarly, if a new MI.TimeWindowRule.v2 object was required, the outer MI.TimeWindowACL object would need to be revised, e.g., to MI.TimeWindowACL.v2; the MI.TimeWindowRule.v2 object, though, could still contain MI.TimeWindow objects, if so specified.

HTTP requests sent to a metadata server SHOULD include an Accept header with the CDNI Payload Type of the expected object. Metadata clients can specify multiple CDNI Payload Types in the Accept header, for example, if a metadata client is capable of processing two different versions of the same type of object (defined by different CDNI Payload Types) it might decide to include both in the Accept header.

6.9. Media Types

All CDNI metadata objects use the Media Type "application/cdni". The CDNI Payload Type for each object then contains the object name of that object as defined by this document, prefixed with "MI.". Table 4 lists the CDNI Payload Type for the metadata objects (resources) specified in this document.

Data Object	CDNI Payload Type
HostIndex	MI.HostIndex
HostMatch	MI.HostMatch
HostMetadata	MI.HostMetadata
PathMatch	MI.PathMatch
PatternMatch	MI.PatternMatch
PathMetadata	MI.PathMetadata
SourceMetadata	MI.SourceMetadata
Source	MI.Source
LocationACL	MI.LocationACL
LocationRule	MI.LocationRule
Footprint	MI.Footprint
TimeWindowACL	MI.TimeWindowACL
TimeWindowRule	MI.TimeWindowRule
TimeWindow	MI.TimeWindow
ProtocolACL	MI.ProtocolACL
ProtocolRule	MI.ProtocolRule
DeliveryAuthorization	MI.DeliveryAuthorization
Cache	MI.Cache
Auth	MI.Auth
Grouping	MI.Grouping

Table 4: CDNI Payload Types for CDNI Metadata objects

6.10. Complete CDNI Metadata Example

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

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


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


```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.SourceMetadata",
      "generic-metadata-value": {
        "sources": [
          {
            "endpoint": ["acq1.ucdn.example"],
            "protocol": "http/1.1"
          },
          {
            "endpoint": ["acq2.ucdn.example"],
            "protocol": "http/1.1"
          }
        ]
      }
    },
    {
      "generic-metadata-type": "MI.LocationACL",
      "generic-metadata-value": {
        "locations": [
          {
            "footprints": [
              {
```

```
                   "footprint-type": "ipv4cidr",
                   "footprint-value": ["192.0.2.0/24"]
                 },
                 {
                   "footprint-type": "ipv6cidr",
                   "footprint-value": ["2001:db8::/32"]
                 },
                 {
                   "footprint-type": "countrycode",
                   "footprint-value": ["us"]
                 },
                 {
                   "footprint-type": "asn",
                   "footprint-value": ["as64496"]
                 }
               ],
               "action": "deny"
             }
           ]
         }
       },
       {
         "generic-metadata-type": "MI.ProtocolACL",
         "generic-metadata-value": {
           "protocol-acl": [
             {
               "protocols": [
                 "http/1.1"
               ],
               "action": "allow"
             }
           ]
         }
       }
     ],
     "paths": [
       {
         "path-pattern": {
           "pattern": "/video/trailers/*"
         },
         "path-metadata": {
           "type": "MI.PathMetadata",
           "href": "https://metadata.ucdn.example/host1234/pathABC"
         }
       },
       {
         "path-pattern": {
           "pattern": "/video/movies/*"
```

```
         },
         "path-metadata": {
           "type": "MI.PathMetadata",
           "href": "https://metadata.ucdn.example/host1234/pathDEF"
         }
       }
     ]
   }
```


Suppose the path of the requested resource matches the "/video/ movies/*" pattern, the next metadata requested would be for "https://metadata.ucdn.example/host1234/pathDCE" with an expected CDNI payload type of "MI.PathMetadata":


```
   {
     "metadata": [],
     "paths": [
       {
         "path-pattern": {
           "pattern": "/videos/movies/hd/*"
         },
         "path-metadata": {
           "type": "MI.PathMetadata",
           "href":
             "https://metadata.ucdn.example/host1234/pathDEF/path123"
         }
       }
     ]
   }
```


Finally, if the path of the requested resource also matches the "/videos/movies/hd/*" pattern, the dCDN would also fetch the following object from "https://metadata.ucdn.example/host1234/pathDEF/path123" with CDNI payload type "MI.PathMetadata":

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.TimeWindowACL",
      "generic-metadata-value": {
        "times": [
          "windows": [
            {
              "start": "1213948800",
              "end": "1327393200"
            }
          ],
          "action": "allow"
        ]
      }
    }
  ]
}
```


The final set of metadata which applies to the requested resource includes a SourceMetadata, a LocationACL, a ProtocolACL, and a TimeWindowACL.

7. IANA Considerations

7.1. CDNI Payload Types

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

Payload Type	Specification
MI.HostIndex	RFCthis
MI.HostMatch	RFCthis
MI.HostMetadata	RFCthis
MI.PathMatch	RFCthis
MI.PatternMatch	RFCthis
MI.PathMetadata	RFCthis
MI.SourceMetadata	RFCthis
MI.Source	RFCthis
MI.LocationACL	RFCthis
MI.LocationRule	RFCthis
MI.Footprint	RFCthis
MI.TimeWindowACL	RFCthis
MI.TimeWindowRule	RFCthis
MI.TimeWindow	RFCthis
MI.ProtocolACL	RFCthis
MI.ProtocolRule	RFCthis
MI.DeliveryAuthorization	RFCthis
MI.Cache	RFCthis
MI.Auth	RFCthis
MI.Grouping	RFCthis

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

7.1.1. CDNI MI HostIndex Payload Type

Purpose: The purpose of this payload type is to distinguish HostIndex MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.1

7.1.2. CDNI MI HostMatch Payload Type

Purpose: The purpose of this payload type is to distinguish HostMatch MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.2

7.1.3. CDNI MI HostMetadata Payload Type

Purpose: The purpose of this payload type is to distinguish HostMetadata MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.3

7.1.4. CDNI MI PathMatch Payload Type

Purpose: The purpose of this payload type is to distinguish PathMatch MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.4

7.1.5. CDNI MI PatternMatch Payload Type

Purpose: The purpose of this payload type is to distinguish PatternMatch MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.5

7.1.6. CDNI MI PathMetadata Payload Type

Purpose: The purpose of this payload type is to distinguish PathMetadata MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.1.6

7.1.7. CDNI MI SourceMetadata Payload Type

Purpose: The purpose of this payload type is to distinguish SourceMetadata MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.1

7.1.8. CDNI MI Source Payload Type

Purpose: The purpose of this payload type is to distinguish Source MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.1.1

7.1.9. CDNI MI LocationACL Payload Type

Purpose: The purpose of this payload type is to distinguish LocationACL MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.2

7.1.10. CDNI MI LocationRule Payload Type

Purpose: The purpose of this payload type is to distinguish LocationRule MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.2.1

7.1.11. CDNI MI Footprint Payload Type

Purpose: The purpose of this payload type is to distinguish Footprint MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.2.2

7.1.12. CDNI MI TimeWindowACL Payload Type

Purpose: The purpose of this payload type is to distinguish TimeWindowACL MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.3

7.1.13. CDNI MI TimeWindowRule Payload Type

Purpose: The purpose of this payload type is to distinguish TimeWindowRule MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.3.1

7.1.14. CDNI MI TimeWindow Payload Type

Purpose: The purpose of this payload type is to distinguish TimeWindow MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.3.2

7.1.15. CDNI MI ProtocolACL Payload Type

Purpose: The purpose of this payload type is to distinguish ProtocolACL MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.4

7.1.16. CDNI MI ProtocolRule Payload Type

Purpose: The purpose of this payload type is to distinguish ProtocolRule MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.4.1

7.1.17. CDNI MI DeliveryAuthorization Payload Type

Purpose: The purpose of this payload type is to distinguish DeliveryAuthorization MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.5

7.1.18. CDNI MI Cache Payload Type

Purpose: The purpose of this payload type is to distinguish Cache MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.6

7.1.19. CDNI MI Auth Payload Type

Purpose: The purpose of this payload type is to distinguish Auth MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.7

7.1.20. CDNI MI Grouping Payload Type

Purpose: The purpose of this payload type is to distinguish Grouping MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.8

7.2. CDNI Metadata Footprint Types Registry

The IANA is requested to create a new "CDNI Metadata Footprint Types" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI Metadata Footprint Types" namespace defines the valid Footprint object type values used by the Footprint object in Section 4.2.2.2. Additions to the Footprint type namespace conform to the "Specification Required" policy as defined in [RFC5226]. The designated expert will verify that new type definitions do not duplicate existing type definitions and prevent gratuitous additions to the namespace. New registrations are required to provide a clear description of how to interpret new footprint types.

The following table defines the initial Footprint Registry values:

Footprint Type	Description	Specification
ipv4cidr	IPv4 CIDR address block	RFCthis
ipv6cidr	IPv6 CIDR address block	RFCthis
asn	Autonomous System (AS) Number	RFCthis
countrycode	ISO 3166-1 alpha-2 code	RFCthis

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

7.3. CDNI Metadata Protocol Types Registry

The IANA is requested to create a new "CDNI Metadata Protocol Types" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI Metadata Protocol Types" namespace defines the valid Protocol object values in Section 4.3.2, used by the SourceMetadata and ProtocolACL objects. Additions to the Protocol namespace conform to the "Specification Required" policy as defined in [RFC5226], where the specification defines the Protocol Type and the protocol to which it is associated. The designated expert will verify that new protocol definitions do not duplicate existing protocol definitions and prevent gratuitous additions to the namespace.

The following table defines the initial Protocol values corresponding to the HTTP and HTTPS protocols:

Protocol Type	Description	Type Specification	Protocol Specifications
http/1.1	Hypertext Transfer Protocol -- HTTP/1.1	RFCthis	RFC7230
https/1.1	HTTP/1.1 Over TLS	RFCthis	RFC7230, RFC2818

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

8. Security Considerations

8.1. Authentication and Integrity

A malicious metadata server, proxy server, or attacker, impersonating an authentic uCDN metadata interface without being detected, could provide false metadata to a dCDN that either:

- o Denies service for one or more pieces of content to one or more User Agents;
- o Directs dCDNs to contact malicious origin servers instead of the actual origin servers, and substitute legitimate content with malware or slanderous alternate content; or
- o Removes delivery restrictions (e.g., LocationACL, TimeWindowACL, ProtocolACL, or Auth metadata), allowing access to content that would otherwise be denied, and thus possibly violating license restrictions and incurring unwarranted delivery costs.

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

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

An implementation of the CDNI metadata interface MUST use mutual authentication and message authentication codes to prevent unauthorized access to and undetected modification of metadata (see Section 8.3).

8.2. Confidentiality and Privacy

Unauthorized viewing of metadata could result in leakage of private information. Content provider origin and policy information is conveyed through the CDNI metadata interface. A third party could intercept metadata transactions in order to gain access to origin servers, as well as information pertaining to content restrictions and usage patterns.

Note: The distribution of metadata by a uCDN to dCDNs could introduce privacy concerns for some content providers, e.g., dCDNs accepting content requests for a content provider’s content might be able to obtain additional information and usage patterns relating to the users of a content provider’s services. Content providers with concerns about divulging information to dCDNs can instruct their uCDN partners not to use CDNI when delivering their content.

An implementation of the CDNI metadata interface MUST use strong encryption to prevent unauthorized interception or monitoring of metadata (see Section 8.3).

8.3. Securing the CDNI Metadata interface

An implementation of the CDNI metadata interface MUST support TLS transport as per [RFC2818] and [RFC7230].

TLS MUST be used by the server-side (dCDN) and the client-side (uCDN) of the CDNI metadata interface, including authentication of the remote end, unless alternate methods are used for ensuring the security of the information in the CDNI metadata interface requests and responses (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

The use of TLS for transport of the CDNI metadata interface messages allows:

- o The dCDN and uCDN to authenticate each other.

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

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

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

- o The integrity of the CDNI metadata interface requests and responses to be protected during the exchange.

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

9. Acknowledgements

The authors would like to thank David Ferguson, Francois Le Faucheur, Jan Seedorf and Matt Miller for their valuable comments and input to this document.

10. Contributing Authors

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

Grant Watson
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: gwatson@velocix.com

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose, 95134
USA

Email: kleung@cisco.com

11. References

11.1. Normative References

[ISO3166-1]
The International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes", ISO 3166-1:2013, 2013.

[POSIX] Institute of Electrical and Electronics Engineers, "Information Technology Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language]", IEEE P1003.1, 1990.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <http://www.rfc-editor.org/info/rfc1034>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <http://www.rfc-editor.org/info/rfc1123>.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <http://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <http://www.rfc-editor.org/info/rfc5891>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <http://www.rfc-editor.org/info/rfc6707>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

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

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

11.2. Informative References

[I-D.ietf-cdni-control-triggers]
Murray, R. and B. Niven-Jenkins, "CDNI Control Interface / Triggers", draft-ietf-cdni-control-triggers-15 (work in progress), May 2016.

[I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-20 (work in progress), August 2016.

[I-D.ietf-cdni-uri-signing]
Leung, K., Faucheur, F., Brandenburg, R., Downey, B., and M. Fisher, "URI Signing for CDN Interconnection (CDNI)", draft-ietf-cdni-uri-signing-09 (work in progress), June 2016.

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

[RFC6793] Vohra, Q. and E. Chen, "BGP Support for Four-Octet Autonomous System (AS) Number Space", RFC 6793, DOI 10.17487/RFC6793, December 2012, <http://www.rfc-editor.org/info/rfc6793>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <http://www.rfc-editor.org/info/rfc7234>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <http://www.rfc-editor.org/info/rfc7336>.

[RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <http://www.rfc-editor.org/info/rfc7337>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <http://www.rfc-editor.org/info/rfc7736>.

Authors' Addresses

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

Email: ben@velocix.com

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

Email: rmurray@velocix.com

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

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

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

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 16, 2017

B. Niven-Jenkins, Ed.
Nokia
R. van Brandenburg, Ed.
TNO

August 15, 2016

Request Routing Redirection interface for CDN Interconnection
draft-ietf-cdni-redirection-20

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

The CDNI Request Routing interface outlined in [RFC7336] comprises of:

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

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

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

2. Terminology

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

This document reuses the terminology defined in [RFC6707].

The following additional terms are introduced by this document:

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

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

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

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

3. Interface function and operation overview

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

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

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

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

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

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

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

The Redirection interface is only relevant in the case of Recursive Request Redirection, as Iterative Request Redirection does not invoke any interaction over the Redirection interface between interconnected CDNs. Therefore, the scope of this document is limited to Recursive Request Redirection.

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

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

For both DNS & HTTP redirection, either HTTP or HTTPS could be used to connect to the Redirection Target. When HTTPS is used to connect to the uCDN, if the uCDN uses DNS redirection to identify the RT to the User Agent, then the new target domain name may not match the domain in the URL dereferenced to reach the uCDN; without operational precautions, and in the absence of DNSSEC, this can make a legitimate redirection look like a DNS-based attack to a User Agent and trigger security warnings. When DNS-based redirection with HTTPS is used, this specification assumes that any RT can complete the necessary TLS handshake with the User Agent. Any operational mechanisms this requires, e.g., private key distribution to surrogates and request routers in dCDNs, are outside the scope of this document.

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

4. HTTP based interface for the Redirection Interface

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

The RI between the uCDN and dCDN uses the same HTTP interface to encapsulate the attributes of both DNS and HTTP requests received from User Agents, although the contents of the RI requests/responses contain data specific to either DNS or HTTP redirection.

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

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

Finally, the RI is easily extendable to support other User Agent request redirection methods (e.g., RTMP 302 redirection) by defining additional protocol specific keys for RI requests and responses along with a specification how to process them.

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

```
User Agent                CDN B RR                  CDN A RR
    |UA Request (DNS or HTTP) |                         |
    |-------------------------------------------------->| (1)
    |                         |                         |
    |                         |HTTP POST to CDN B's RI  |
    |                         |URI encapsulating UA     |
    |                         |request attributes       |
    |                         |<------------------------| (2)
    |                         |                         |
    |                         |HTTP Response with body  |
    |                         |containing RT attributes |
    |                         |of the protocol specific |
    |                         |response to return to UA |
    |                         |------------------------>| (3)
    |                         |                         |
    |            Protocol specific response (redirection)|
    |<--------------------------------------------------| (4)
    |                         |                         |
```

Figure 1: Generic Recursive Request Redirection message flow

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

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

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

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

4.1. Information passed in RI requests & responses

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

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

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

Generally, dCDNs can provide better routing decisions given additional information about the content request, e.g., the URI of the requested content or the User Agent's IP address or subnet. The set of information required to base a routing decision on can be highly dependent on the type of content delivered. A uCDN SHOULD only include information that is absolutely necessary for delivering that type of content. Cookies in particular are particularly sensitive from a security/privacy point of view and in general SHOULD NOT be conveyed in the RI Requests to the dCDN. The set of information necessary to be conveyed for a particular type of request is expected to be conveyed out of band between the uCDN and dCDN. See Section 5.2 for more detail on the privacy aspects of using RI Requests to convey information about UA requests.

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

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

1. The attributes of the RT to return to the User Agent in the DNS response or HTTP response.

2. Parameters/policies that indicate the properties of the response, such as, whether it is cacheable, the scope of the response, etc.

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

These additional policies split into two basic categories:

- o Cacheability information signaled via the HTTP response headers of the RI response (to reduce the number of subsequent RI requests the uCDN needs to make).

- o The scope of a cacheable response signaled in the HTTP response body of the RI response, for example, whether the response applies to a wider range of IP addresses than what was included in the RI request.

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

4.2. JSON encoding of RI requests & responses

The body of RI requests and responses is a JSON object [RFC7159] that MUST conform to [RFC7493] containing a dictionary of key:value pairs. Senders MUST encode all (top level object and sub-object) keys specified in this document in lowercase. Receivers MUST ignore any keys that are unknown or invalid.

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

Key	Request/Response	Description
dns	Both	The attributes of the UA's DNS request or the attributes of the RT(s) to return in a DNS response.
http	Both	The attributes of the UA's HTTP request or the attributes of the RT to return in a HTTP response.
scope	Response	The scope of the response (if it is cacheable). For example, whether the response applies to a wider range of IP addresses than what was included in the RI request.
error	Response	Additional details if the response is an error response.
cdn-path	Both	A List of Strings. Contains a list of the CDN Provider IDs of previous CDNs that have participated in the request routing for the associated User Agent request. On RI requests it contains the list of previous CDNs that this RI request has passed through. On RI responses it contains the list of CDNs that were involved in obtaining the final redirection included in the RI response. See Section 4.8
max-hops	Request	Integer specifying the maximum number of hops (CDN Provider IDs) this request is allowed to be propagated along. This allows the uCDN to coarsely constrain the latency of the request routing chain.

Top-Level keys in RI requests/responses

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

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

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

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

All implementations that support IPv4 addresses MUST support the encoding specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986]. Likewise, implementations that support IPv6 addresses MUST support all IPv6 address formats specified in [RFC4291]. Server implementations SHOULD use IPv6 address formats specified in [RFC5952].

4.3. MIME Media Types used by the RI interface

RI requests MUST use a MIME Media Type of application/cdni as specified in [RFC7736], with the Payload Type (ptype) parameter set to 'redirection-request'.

RI responses MUST use a MIME Media Type of application/cdni as specified in [RFC7736], with the Payload Type (ptype) parameter set to 'redirection-response'.

4.4. DNS redirection

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

4.4.1. DNS Redirection requests

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

o The IP address of the DNS resolver that made the DNS request to the uCDN.

o The type of DNS query made (usually either A or AAAA).

o The class of DNS query made (usually IN).

o The fully qualified domain name for which DNS redirection is being requested.

o The IP address or prefix of the User Agent (if known to the uCDN).

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

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

An RI request for DNS-based redirection MUST include a dns dictionary. This dns dictionary MUST contain the following keys: resolver-ip, qtype, qclass, qname and the value of each MUST be the value of the appropriate part of the User Agent's DNS query/request. For internationalized domain names containing non-ASCII characters, the value of the qname field MUST be the ASCII-compatible encoded (ACE) representation (A-label) of the domain name [RFC5890].

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

```
POST /dcdn/ri HTTP/1.1
Host: 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:

Key	Value	Mandatory	Description
rcode	Integer	Yes	DNS response code (see [RFC6895]).
name	String	Yes	The fully qualified domain name the response relates to.
a	List of String	No	Set of IPv4 Addresses of RT(s).
aaaa	List of String	No	Set of IPv6 Addresses of RT(s).
cname	List of String	No	Set of fully qualified domain names of RT(s).
ttl	Integer	No	TTL in seconds of DNS response. Default is 0.

A successful RI response for DNS-based redirection MUST include a dns dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for DNS-based redirection MUST include an error dictionary. If a dns dictionary is included in the RI response, it MUST include the rcode and name keys and it MUST include at least one of the following keys: a, aaaa, cname. The dns dictionary MAY include both 'a' and 'aaaa' keys. If the dns dictionary contains a cname key it MUST NOT contain either an a or aaaa key. For internationalized domain names containing non-ASCII characters, the value of the cname field MUST be the ASCII-compatible encoded (ACE) representation (A-label) of the domain name.

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

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

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

A further example of a successful RI response (dCDN->uCDN) for DNS based redirection is listed below, in this case with a cname key containing the FQDN of the RT.

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


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


4.5. HTTP Redirection

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

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

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

4.5.1. HTTP Redirection requests

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

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

The uCDN may also decide to pass the presence and value of particular HTTP headers included in the User Agent request to the dCDN.

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

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

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

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

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

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

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

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

4.5.2. HTTP Redirection responses

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

- o A URI pointing to an RT that is the selected dCDN surrogate(s) (if the dCDN is performing HTTP based redirection directly to a surrogate); or

- o A URI pointing to an RT that is a Request Router (if the dCDN will perform request redirection itself).

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

Key	Value	Mandatory	Description
sc-status	Integer	Yes	The status-code part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA (usually set to 302).
sc-version	String	Yes	The HTTP-version part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.
sc-reason	String	Yes	The reason-phrase part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.
cs-uri	String	Yes	The URI requested by the UA/client.
sc-(location)	String	Yes	The contents of the Location header to return to the UA (i.e., a URI pointing to the RT(s)).
sc-(<headername>)	String	No	The field-value of the HTTP header field named <HeaderName> to return to the UA. For example, sc-(expires) would contain the value of the HTTP Expires header.

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

A successful RI response for HTTP-based redirection MUST include an http dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for HTTP-based redirection MUST include an error dictionary. If an http dictionary is included in the RI response, it MUST include at least the following keys: sc-status, sc-version, sc-reason, cs-uri and sc-(location).

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

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

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

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

{
  "http": {
    "sc-status": 302,
    "sc-version": "HTTP/1.1",
    "sc-reason": "Found",
    "cs-uri": "http://www.example.com"
    "sc-(location)":
      "http://sur1.dcdn.example/ucdn/example.com",
  }
}
```

4.6. Cacheability and scope of responses

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

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

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

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

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

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

Key	Value	Mandatory	Description
error-code	Integer	Yes	A three-digit numeric code defined by the server to indicate the error(s) that occurred.
reason	String	No	A string providing further information related to the error.

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

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

2xx: Reserved.

3xx: Reserved.

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

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

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

Error codes with a "Reason" of "<reason>" do not have a defined value for their 'reason'-key. Depending on the error-code semantics, the value of this field may be determined dynamically.

Code	Reason	Description
100	<reason> (see Description)	Generic informational error-code meant for carrying a human-readable string
400	<reason> (see Description)	Generic error-code for uCDN errors where the dCDN can not or will not process the request due to something that is perceived to be a uCDN error. The reason field may be used to provide more details about the source of the error.
500	<reason> (see Description)	Generic error-code for dCDN errors where the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason. The reason field may be used to provide more details about the source of the error.
501	Unable to retrieve metadata	The dCDN is unable to retrieve the metadata associated with the content requested by the UA. This may indicate a configuration error or the content requested by the UA not existing.
502	Loop detected	The dCDN detected a redirection loop (see Section 4.8).
503	Maximum hops exceeded	The dCDN detected the maximum number of redirection hops exceeding max-hops (see Section 4.8).
504	Out of capacity	The dCDN does not currently have sufficient capacity to handle the UA request.
505	Delivery protocol not supported	The dCDN does not support the (set of) delivery protocols indicated in the CDNI Metadata of the content requested content by the UA.
506	Redirection protocol not supported	The dCDN does not support the requested redirection protocol. This error-code is also used when the RI request has the dns-only flag set to True and the dCDN is not support or is not prepared to return a RT of a surrogate directly.

Table 1

The following is an example of an unsuccessful RI response (dCDN->uCDN) for a DNS based User Agent request:

```
HTTP/1.1 500 Internal Server Error
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: private, no-cache

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

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

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

{
   "http": {
    "sc-status": 302,
    "sc-version": "HTTP/1.1",
    "sc-reason": "Found",
    "cs-uri": "http://www.example.com"
    "sc-(location)":
      "http://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 NOT propagate to any downstream CDNs if the number of CDN Provider IDs in cdn-path (before adding its own Provider ID) is equal to or greater than max-hops.

The CDN Provider ID uniquely identifies each CDN provider during the course of request routing redirection. It consists of the characters AS followed by the CDN Provider’s AS number, then a colon (’:’) and an additional qualifier that is used to guarantee uniqueness in case a particular AS has multiple independent CDNs deployed. For example, "AS64496:0".

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

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

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

5. Security Considerations

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

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

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

5.1. Authentication, Authorization, Confidentiality, Integrity Protection

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

- o The dCDN and uCDN to authenticate each other

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

- o The dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI Redirection messages to/from an authorized CDN);
- o CDNI Redirection interface messages to be transmitted with confidentiality; and
- o The integrity of the CDNI Redirection interface messages to be protected during the exchange.

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

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

5.2. Privacy

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

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

While it is technically possible to mask some information in the RI Request, such as the last bits of the UA IP address, it is important to note that this will reduce the effectivess of the RI in certain cases. CDN deployments need to strike a balance between end-user privacy and the features impacted by such masking. This balance is likely to vary from one deployment to another. As an example, when the UA and its DNS resolver is behind a Carrier-grade NAT, and the RI is used to find an appropriate delivery node behind the same NAT, the full IP address might be necessary. Another potential issue when using IP anonymization is that it is no longer possible to correlate an RI Request with a subsequent UA request.

6. IANA Considerations

6.1. CDNI Payload Type Parameter registrations

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

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

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

6.1.1. CDNI RI Redirection Request Payload Type

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

Interface: RI

Encoding: see Section 4.4.1 and Section 4.5.1

6.1.2. CDNI RI Redirection Response Payload Type

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

Interface: RI

Encoding: see Section 4.4.2 and Section 4.5.2

6.2. RI Error response registry

IANA is requested to create a new "CDNI RI Error response code" subregistry within the "Content Delivery Network Interconnection (CDNI) Parameters" registry. The "CDNI RI Error response code" namespace defines the valid values for the error-code key in RI error responses. The CDNI RI Error response code MUST be a three digit integer.

Additions to the "RI Error response registry" will be made via "Specification Required" as defined in [RFC5226].

The Designated Expert will verify that new error code registrations do not duplicate existing error code definitions (in name or functionality), ensure that the new error code is in accordance with the error classes defined in section Section 4.7 of this document, prevent gratuitous additions to the namespace, and prevent any additions to the namespace that would impair the interoperability of CDNI implementations.

New registrations are required to provide the following information:

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

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

Description: A brief description of the error code semantics.

Specification: Reference to the specification that defines the error code in more detail.

The entries in Table 1 are registered by this document, with the value of the 'Specification' field set to [RFCThis].

7. Contributors

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

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

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

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

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

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

Yunfei Zhang, Email:hishigh@gmail.com

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

8. Acknowledgements

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

9. References

9.1. Normative References

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895, April 2013, <http://www.rfc-editor.org/info/rfc6895>.

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

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <http://www.rfc-editor.org/info/rfc6707>.

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

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <http://www.rfc-editor.org/info/rfc5890>.

[RTMP] Adobe Systems Incorporated, "Real-Time Messaging Protocol (RTMP) specification", December 2012, <http://www.adobe.com/go/spec_rtmp>.

9.2. Informative References

[RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <http://www.rfc-editor.org/info/rfc7337>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <http://www.rfc-editor.org/info/rfc7336>.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <http://www.rfc-editor.org/info/rfc7736>.

Authors' Addresses

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

Email: ben.niven-jenkins@nokia.com

Ray van Brandenburg (editor)
TNO
Anna van Buerenplein 1
The Hague 2595DA
the Netherlands

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

CDNI R. van Brandenburg
Internet-Draft Tiledmedia
Intended status: Standards Track K. Leung
Expires: April 10, 2020 Cisco Systems, Inc.
P. Sorber
Apple, Inc.
October 8, 2019

URI Signing for CDN Interconnection (CDNI)
draft-ietf-cdni-uri-signing-19

Abstract

This document describes how the concept of URI signing supports the content access control requirements of CDNI and proposes a URI signing method as a JSON Web Token (JWT) profile.

The proposed URI signing method specifies the information needed to be included in the URI to transmit the signed JWT, as well as the claims needed by the signed JWT to authorize a UA. The mechanism described can be used both in CDNI and single CDN scenarios.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of redirection between interconnected CDNs (CDNI) and between a Content Service Provider (CSP) and a CDN. The primary goal of URI Signing is to make sure that only authorized User Agents (UAs) are able to access the content, with a CSP being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as Digital Rights Management (DRM), are more appropriate. In addition to access control, URI Signing also has benefits in reducing the impact of denial-of-service attacks.

The overall problem space for CDN Interconnection (CDNI) is described in CDNI Problem Statement [RFC6707]. This document, along with the CDNI Requirements [RFC7337] document and the CDNI Framework [RFC7336], describes the need for interconnected CDNs to be able to implement an access control mechanism that enforces a CSP's distribution policies.

Specifically, the CDNI Framework [RFC7336] states:

The CSP may also trust the CDN operator to perform actions such as delegating traffic to additional downstream CDNs, and to enforce per-request authorization performed by the CSP using techniques such as URI signing.

In particular, the following requirement is listed in the CDNI Requirements [RFC7337]:

MI-16 {HIGH} The CDNI Metadata interface shall allow signaling of authorization checks and verification that are to be performed by the Surrogate before delivery. For example, this could potentially include the need to verify information (e.g., Expiry time, Client IP address) required for access authorization.

This document defines a method of signing URIs that allows Surrogates in interconnected CDNs to enforce a per-request authorization initiated by the CSP. Splitting the role of initiating per-request authorization by the CSP and the role of verifying this authorization by the CDN allows any arbitrary distribution policy to be enforced across CDNs without the need of CDNs to have any awareness of the specific CSP distribution policies.

The method is implemented using Signed JSON Web Tokens (JWTs) [RFC7519].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the terminology defined in the CDNI Problem Statement [RFC6707].

This document also uses the terminology of the JSON Web Token (JWT) [RFC7519].

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

- Signed URI: A URI for which a signed JWT is provided.

- Target CDN URI: URI created by the CSP to direct a UA towards the Upstream CDN (uCDN). The Target CDN URI can be signed by the CSP and verified by the uCDN and possibly further Downstream CDNs (dCDNs).

- o Redirection URI: URI created by the uCDN to redirect a UA towards the dCDN. The Redirection URI can be signed by the uCDN and verified by the dCDN. In a cascaded CDNI scenario, there can be more than one Redirection URI.

- o Signed Token Renewal: A series of signed JWTs that are used for subsequent access to a set of related resources in a CDN, such as a set of HTTP Adaptive Streaming files. Every time a signed JWT is used to access a particular resource, a new signed JWT is sent along with the resource that can be used to request the next resource in the set. When generating a new signed JWT in Signed Token Renewal, parameters are carried over from one signed JWT to the next.

1.2. Background and overview on URI Signing

A CSP and CDN are assumed to have a trust relationship that enables the CSP to authorize access to a content item by including a set of claims in the form of a signed JWT in the URI before redirecting a UA to the CDN. Using these attributes, it is possible for a CDN to check an incoming content request to see whether it was authorized by the CSP (e.g., based on the UA's IP address or a time window). To prevent the UA from altering the claims a JWT MUST be signed.

Figure 1, shown below, presents an overview of the URI Signing mechanism in the case of a CSP with a single CDN. When the UA browses for content on CSP's website (#1), it receives HTML web pages with embedded content URIs. Upon requesting these URIs, the CSP redirects to a CDN, creating a Target CDN URI (#2) (alternatively, the Target CDN URI itself is embedded in the HTML). The Target CDN URI is the Signed URI which may include the IP address of the UA and/ or a time window. The signed URI always contains a signed JWT generated by the CSP using a shared secret or private key. Once the UA receives the response with the Signed URI, it sends a new HTTP request using the Signed URI to the CDN (#3). Upon receiving the request, the CDN authenticates the Signed URI by verifying the signed JWT. If applicable, the CDN checks whether the source IP address of the HTTP request matches the one in the Signed URI and/or if the time window is still valid. After these claims are verified, the CDN delivers the content (#4).

```
                  --------
                 /        \
                 |   CSP  |< * * * * * * * * * * * *
                 \        /        Trust           *
                  --------      relationship       *
                     ^  |                          *
                     |  |                          *
         1. Browse   |  | 2. Signed                *
              for    |  |    URI                   *
            content  |  |                          *
                     |  v                          v
                  +------+ 3. Signed URI     --------
                  | User |----------------->/        \
                  | Agent|                  |  CDN   |
                  |      |<-----------------\        /
                  +------+ 4. Content        --------
                              Delivery
```

Figure 1: Figure 1: URI Signing in a CDN Environment

1.3. CDNI URI Signing Overview

In a CDNI environment, as shown in Figure 2 below, URI Signing operates the same way in the initial steps #1 and #2 but the later steps involve multiple CDNs delivering the content. The main difference from the single CDN case is a redirection step between the uCDN and the dCDN. In step #3, the UA may send an HTTP request or a DNS request. Depending on whether HTTP-based or DNS-based request routing is used. The uCDN responds by directing the UA towards the dCDN using either a Redirection URI (i.e., a Signed URI generated by the uCDN) or a DNS reply, respectively (#4). Once the UA receives the response, it sends the Redirection URI/Target CDN URI to the dCDN (#5). The received URI is verified by the dCDN before delivering the content (#6). Note: The CDNI call flows are covered in Detailed URI Signing Operation (Section 5).

```
                                      +-------------------------+
                                      |Request Redirection Modes|
                                      +-------------------------+
                                      | a) HTTP                 |
                                      | b) DNS                  |
                                      +-------------------------+
                   --------
                  /        \< * * * * * * * * * * * * * *
                  |   CSP  |< * * * * * * * * * * *     *
                  \        /        Trust           *     *
                   --------      relationship       *     *
                     ^  |                           *     *
                     |  |  2. Signed                *     *
          1. Browse  |  |     URI in                *     *
               for   |  |     HTML                  *     *
             content |  |                           *     *
                     |  v   3.a)Signed URI          v     *
                   +------+   b)DNS request   --------  * Trust
                   | User |----------------->/        \ * relationship
                   | Agent|                  |  uCDN  | * (optional)
                   |      |<-----------------\        / *
                   +------+ 4.a)Redirection URI-------  *
                     ^  |     b)DNS Reply         ^     *
                     |  |                         *     *
                     |  |      Trust relationship *     *
                     |  |                         *     *
         6. Content  |  | 5.a)Redirection URI     *     *
            delivery |  |   b)Signed URI(after    v     v
                     |  |     DNS exchange)      --------
                     |  +----------------------->/        \ [May be
                     |                          |  dCDN  |  cascaded
                     +--------------------------\        /  CDNs]
                                                 --------
```

Figure 2: URI Signing in a CDNI Environment

The trust relationships between CSP, uCDN, and dCDN have direct implications for URI Signing. In the case shown in Figure 2, the CSP has a trust relationship with the uCDN. The delivery of the content may be delegated to a dCDN, which has a relationship with the uCDN but may have no relationship with the CSP.

In CDNI, there are two methods for request routing: DNS-based and HTTP-based. For DNS-based request routing, the Signed URI (i.e., the Target CDN URI) provided by the CSP reaches the dCDN directly. In the case where the dCDN does not have a trust relationship with the CSP, this means that either an asymmetric public/private key method needs to be used for computing the signed JWT (because the CSP and dCDN are not able to exchange symmetric shared secret keys), or the CSP needs to allow the uCDN to redistribute shared keys to a subset of their dCDNs.

For HTTP-based request routing, the Signed URI (i.e., the Target CDN URI) provided by the CSP reaches the uCDN. After this URI has been verified by the uCDN, the uCDN creates and signs a new Redirection URI, redirecting the UA to the dCDN. Since this new URI can have a new signed JWT, the relationship between the dCDN and CSP is not relevant. Because a relationship between uCDN and dCDN always exists, either asymmetric public/private keys or symmetric shared secret keys can be used for URI Signing with HTTP-based request routing. Note that the signed Redirection URI MUST maintain the same (or higher) level of security as the original Signed URI.

Mode	Asymmetric Key	Symmetric Key
HTTP	Public key (uCDN)	Shared key (uCDN)
DNS	Public key (CSP)	Shared key (CSP)

Figure 3: CDNI URI Signing Key

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 known only to the entity signing the URI (either CSP or uCDN) and a public key for the verification of the Signed URI. With symmetric keys, the same key is used by both the signing entity for signing the URI as well as by the verifying entity for verifying the Signed URI. Regardless of the type of keys used, the verifying entity has to obtain the key (either the public or the symmetric key). There are very different requirements (outside the scope of this document) for distributing asymmetric keys and symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent third parties from getting access to the key, since they could then generate valid Signed URIs for unauthorized requests. Key distribution for asymmetric keys does not require confidentiality since public keys can typically be distributed openly (because they cannot be used to sign URIs) and private keys are kept secret by the URI signer.

1.4. URI Signing in a non-CDNI context

While the URI signing method defined in this document was primarily created for the purpose of allowing URI Signing in CDNI scenarios, i.e., between a uCDN and a dCDN, there is nothing in the defined URI Signing method that precludes it from being used in a non-CDNI context. As such, the described mechanism could be used in a single-CDN scenario such as shown in Figure 1 in Section 1.2, for example to allow a CSP that uses different CDNs to only have to implement a single URI Signing mechanism.

2. JWT Format and Processing Requirements

The concept behind URI Signing is based on embedding a signed JSON Web Token (JWT) [RFC7519] in an HTTP or HTTPS URI [RFC7230] (Section 2.7). The signed JWT contains a number of claims that can be verified to ensure the UA has legitimate access to the content.

This document specifies the following attribute for embedding a signed JWT in a Target CDN URI or Redirection URI:

- URI Signing Package (URISigningPackage): The URI attribute that encapsulates all the URI Signing claims in a signed JWT encoded format. This attribute is exposed in the Signed URI as a URI query parameter or as a URL path parameter.

The parameter name of the URI Signing Package Attribute is defined in the CDNI Metadata (Section 4.4). If the CDNI Metadata interface is not used, or does not include a parameter name for the URI Signing Package Attribute, the parameter name can be set by configuration (out of scope of this document).

The URI Signing Package will be found by searching the URI, left-to-right, for the following sequence:

- a reserved character (as defined in [RFC3986] Section 2.2),

- the URI Signing Package Attribute name,

- if the last character of the URI Signing Package Attribute name is not a reserved character, an equal symbol ('='),

- and a sequence of zero or more non-reserved characters that will be interpreted as a signed JWT,

- terminated by either a reserved character or the end of the URI.

The first such match will be taken to provide the signed JWT; the URI will not be searched for multiple signed JWTs.

2.1. JWT Claims

This section identifies the set of claims that can be used to enforce the CSP distribution policy. New claims can be introduced in the future to extend the distribution policy capabilities.

In order to provide distribution policy flexibility, the exact subset of claims used in a given signed JWT is a runtime decision. Claim requirements are defined in the CDNI Metadata (Section 4.4). If the CDNI Metadata interface is not used, or does not include claim requirements, the claim requirements can be set by configuration (out of scope of this document).

The following claims (where the "JSON Web Token Claims" registry claim name is specified in parenthesis below) are used to enforce the distribution policies. All of the listed claims are mandatory to implement in a URI Signing implementation, but are not mandatory to use in a given signed JWT. (The "optional" and "mandatory" identifiers in square brackets refer to whether or not a given claim MUST be present in a URI Signing JWT.) A CDN MUST be able to parse and process all of the claims listed below.

Note: See the Security Considerations (Section 7) section on the limitations of using an expiration time and client IP address for distribution policy enforcement.

2.1.1. Issuer (iss) claim

Issuer (iss) [optional] - The semantics in [RFC7519] Section 4.1.1 MUST be followed. This claim MAY be used to verify authorization of the issuer of a signed JWT and also MAY be used to confirm that the indicated key was provided by said issuer. If the CDN verifying the signed JWT does not support Issuer verification, or if the Issuer in the signed JWT does not match the list of known acceptable Issuers, the CDN MUST reject the request. If the received signed JWT contains an Issuer claim, then any JWT subsequently generated for CDNI redirection MUST also contain an Issuer claim, and the Issuer value MUST be updated to identify the redirecting CDN. If the received signed JWT does not contain an Issuer claim, an Issuer claim MAY be added to a signed JWT generated for CDNI redirection.

2.1.2. Subject (sub) claim

Subject (sub) [optional] - The semantics in [RFC7519] Section 4.1.2 MUST be followed. If this claim is used, it MUST be a JSON Web Encryption (JWE [RFC7516]) Object in compact serialization form, because it contains personally identifiable information. This claim contains information about the subject (for example, a user or an agent) that MAY be used to verify the signed JWT. If the received signed JWT contains a Subject claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Subject claim, and the Subject value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Subject claim if no Subject claim existed in the received signed JWT.

2.1.3. Audience (aud) claim

Audience (aud) [optional] - The semantics in [RFC7519] Section 4.1.3 MUST be followed. This claim is used to ensure that the CDN verifing the JWT is an intended recipient of the request. The claim should contain an identity on behalf of whom the CDN can verify the token (e.g., the CSP or any uCDN in the chain). A dCDN MAY modify the claim as long it can generate a valid signature.

2.1.4. Expiry Time (exp) claim

Expiry Time (exp) [optional] - The semantics in [RFC7519] Section 4.1.4 MUST be followed, though URI Signing implementations MUST NOT allow for any time synchronization "leeway". Note: The time on the entities that generate and verify the signed URI SHOULD be in sync. In the CDNI case, this means that CSP, uCDN, and dCDN servers need to be time-synchronized. It is RECOMMENDED to use NTP [RFC5905] for time synchronization. If the CDN verifying the signed JWT does not support Expiry Time verification, or if the Expiry Time in the signed JWT corresponds to a time equal to or earlier than the time of the content request, the CDN MUST reject the request. If the received signed JWT contains a Expiry Time claim, then any JWT subsequently generated for CDNI redirection MUST also contain an Expiry Time claim, and the Expiry Time value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add an Expiry Time claim if no Expiry Time claim existed in the received signed JWT.

2.1.5. Not Before (nbf) claim

Not Before (nbf) [optional] - The semantics in [RFC7519] Section 4.1.5 MUST be followed, though URI Signing implementations MUST NOT allow for any time synchronization "leeway". Note: The time on the entities that generate and verify the signed URI SHOULD be in sync. In the CDNI case, this means that the CSP, uCDN, and dCDN servers need to be time-synchronized. It is RECOMMENDED to use NTP [RFC5905] for time synchronization. If the CDN verifying the signed JWT does not support Not Before time verification, or if the Not Before time in the signed JWT corresponds to a time later than the time of the content request, the CDN MUST reject the request. If the received signed JWT contains a Not Before time claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Not Before time claim, and the Not Before time value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Not Before time claim if no Not Before time claim existed in the received signed JWT.

2.1.6. Issued At (iat) claim

Issued At (iat) [optional] - The semantics in [RFC7519] Section 4.1.6 MUST be followed. Note: The time on the entities that generate and verify the signed URI SHOULD be in sync. In the CDNI case, this means that CSP, uCDN, and dCDN servers need to be time-synchronized. It is RECOMMENDED to use NTP [RFC5905] for time synchronization. If the received signed JWT contains an Issued At claim, then any JWT subsequently generated for CDNI redirection MUST also contain an Issued At claim, and the Issuer value MUST be updated to identify the time the new JWT was generated. If the received signed JWT does not contain an Issued At claim, an Issued At claim MAY be added to a signed JWT generated for CDNI redirection.

2.1.7. Nonce (jti) claim

Nonce (jti) [optional] - The semantics in [RFC7519] Section 4.1.7 MUST be followed. A Nonce can be used to prevent replay attacks if the CDN stores a list of all previously used Nonce values, and verifies that the Nonce in the current JWT has never been used before. If the signed JWT contains a Nonce claim and the CDN verifying the signed JWT either does not support Nonce storage or has previously seen the nonce used in a request for the same content, then the CDN MUST reject the request. If the received signed JWT contains a Nonce claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Nonce claim, and the Nonce value MUST be the same as in the received signed JWT. If the received signed JWT does not contain a Nonce claim, a Nonce claim MUST NOT be added to a signed JWT generated for CDNI redirection.

2.1.8. CDNI Claim Set Version (cdniv) claim

CDNI Claim Set Version (cdniv) [optional] - The CDNI Claim Set Version (cdniv) claim provides a means within a signed JWT to tie the claim set to a specific version of this specification. The cdniv claim is intended to allow changes in and facilitate upgrades across specifications. The type is JSON integer and the value MUST be set to "1", for this version of the specification. In the absence of this claim, the value is assumed to be "1". For future versions this claim will be mandatory. Implementations MUST reject signed JWTs with unsupported CDNI Claim Set versions.

2.1.9. CDNI Critical Claims Set (cdnicrit) claim

CDNI Critical Claims Set (cdnicrit) [optional] - The CDNI Critical Claims Set (cdnicrit) claim indicates that extensions to this specification are being used that MUST be understood and processed. Its value is a comma separated listing of claims in the Signed JWT that use those extensions. If any of the listed extension claims are not understood and supported by the recipient, then the Signed JWT is invalid. Producers MUST NOT include claim names defined by this specification, duplicate names, or names that do not occur as claim names within the Signed JWT in the cdnicrit list. Producers MUST NOT use the empty list "" as the cdnicrit value. Recipients MAY consider the Signed JWT to be invalid if the cdnicrit list contains any claim names defined by this specification or if any other constraints on its use are violated. This claim MUST be understood and processed by all implementations.

2.1.10. Client IP (cdniip) claim

Client IP (cdniip) [optional] - The Client IP (cdniip) claim hold an IP address or IP prefix for which the Signed URI is valid. This is represented in CIDR notation, with dotted decimal format for IPv4 addresses [RFC0791] or canonical text representation for IPv6 addresses [RFC5952]. The request MUST be rejected if sourced from a client outside of the specified IP range. Since the client IP is considered personally identifiable information this field MUST be a JSON Web Encryption (JWE [RFC7516]) Object in compact serialization form. If the CDN verifying the signed JWT does not support Client IP verification, or if the Client IP in the signed JWT does not match the source IP address in the content request, the CDN MUST reject the request. The type of this claim is a JSON string that contains the JWE. If the received signed JWT contains a Client IP claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Client IP claim, and the Client IP value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Client IP claim if no Client IP claim existed in the received signed JWT.

2.1.11. CDNI URI Container (cdniuc) claim

URI Container (cdniuc) [optional] - The URI Container (cdniuc) holds the URI representation before a URI Signing Package is added. This representation can take one of several forms detailed in Section 2.1.15. If the URI Container used in the signed JWT does not match the URI of the content request, the CDN verifying the signed JWT MUST reject the request. When comparing the URI, the percent encoded form as defined in [RFC3986] Section 2.1 MUST be used. When redirecting a URI, the CDN generating the new signed JWT MAY change the URI Container to comport with the URI being used in the redirection.

2.1.12. CDNI Expiration Time Setting (cdniets) claim

CDNI Expiration Time Setting (cdniets) [optional] - The CDNI Expiration Time Setting (cdniets) claim provides a means for setting the value of the Expiry Time (exp) claim when generating a subsequent signed JWT in Signed Token Renewal. Its type is a JSON numeric value. It denotes the number of seconds to be added to the time at which the JWT is verified that gives the value of the Expiry Time (exp) claim of the next signed JWT. The CDNI Expiration Time Setting (cdniets) SHOULD NOT be used when not using Signed Token Renewal and MUST be present when using Signed Token Renewal.

2.1.13. CDNI Signed Token Transport (cdnistt) claim

CDNI Signed Token Transport (cdnistt) [optional] - The CDNI Signed Token Transport (cdnistt) claim provides a means of signalling the method through which a new signed JWT is transported from the CDN to the UA and vice versa for the purpose of Signed Token Renewal. Its type is a JSON integer. Values for this claim are defined in Section 6.5. If using this claim you MUST also specify a CDNI Expiration Time Setting (cdniets) as noted above.

2.1.14. CDNI Signed Token Depth (cdnistd) claim

CDNI Signed Token Depth (cdnistd) [optional] - The CDNI Signed Token Depth (cdnistd) claim is used to associate a subsequent signed JWT, generated as the result of a CDNI Signed Token Transport claim, with a specific URI subset. Its type is a JSON integer. Signed JWTs MUST NOT use a negative value for the CDNI Signed Token Depth claim.

If the transport used for Signed Token Transport allows the CDN to associate the path component of a URI with tokens (e.g., an HTTP Cookie Path as described in section 4.1.2.4 of [RFC6265]), the CDNI Signed Token Depth value is the number of path segments that should be considered significant for this association. A CDNI Signed Token Depth of zero means that the client SHOULD be directed to return the token with requests for any path. If the CDNI Signed Token Depth is greater than zero, then the client SHOULD be directed to return the token for future requests wherein the first CDNI Signed Token Depth segments of the path match the first CDNI Signed Token Depth segments of the signed URI path. This matching MUST use the URI with the token removed, as specified in Section 2.1.15.

If the URI path to match contains fewer segments than the CDNI Signed Token Depth claim, a signed JWT MUST NOT be generated for the purposes of Signed Token Renewal. If the CDNI Signed Token Depth claim is omitted, it means the same thing as if its value were zero. If the received signed JWT contains a CDNI Signed Token Depth claim, then any JWT subsequently generated for CDNI redirection or Signed Token Transport MUST also contain a CDNI Signed Token Depth claim, and the value MUST be the same as in the received signed JWT.

2.1.15. URI Container Forms

The URI Container (cdniuc) claim takes one of the following forms: 'hash:' or 'regex:'. More forms may be added in the future to extend the capabilities.

Before comparing a URI with contents of this container, the following steps MUST be performed:

- Prior to verification, remove the signed JWT from the URI. This removal is only for the purpose of determining if the URI matches; all other purposes will use the original URI. If the signed JWT is terminated by anything other than a sub-delimiter (as defined in [RFC3986] Section 2.2), everything from the reserved character (as defined in [RFC3986] Section 2.2) that precedes the URI Signing Package Attribute to the last character of the signed JWT will be removed, inclusive. Otherwise, everything from the first character of the URI Signing Package Attribute to the sub-delimiter that terminates the signed JWT will be removed, inclusive.

- Normalize the URI according to section 2.7.3 [RFC7230] and sections 6.2.2 and 6.2.3 [RFC3986]. This applies to both generation and verification of the signed JWT.

2.1.15.1. URI Hash Container (hash:)

Prefixed with 'hash:', this string is a URL Segment form ([RFC6920] Section 5) of the URI.

2.1.15.2. URI Regular Expression Container (regex:)

Prefixed with 'regex:', this string is any POSIX Section 9 [POSIX.1] Extended Regular Expression compatible regular expression used to match against the requested URI. These regular expressions MUST be evaluated in the POSIX locale (POSIX Section 7.2 [POSIX.1]).

Note: Because '\' has special meaning in JSON [RFC8259] as the escape character within JSON strings, the regular expression character '\' MUST be escaped as '\\'.

An example of a 'regex:' is the following:

```
[^:]*\\://[^/]*/folder/content/quality_[^/]*/segment.{3}\\.mp4(\\?.*)?
```

Note: Due to computational complexity of executing arbitrary regular expressions, it is RECOMMENDED to only execute after verifying the JWT to ensure its authenticity.

2.2. JWT Header

The header of the JWT MAY be passed via the CDNI Metadata interface instead of being included in the URISigningPackage. The header value must be transmitted in the serialized encoded form and prepended to the JWT payload and signature passed in the URISigningPackage prior to verification. This reduces the size of the signed JWT token.

3. URI Signing Token Renewal

3.1. Overview

For content that is delivered via HTTP in a segmented fashion, such as MPEG-DASH [MPEG-DASH] or HTTP Live Streaming (HLS) [RFC8216], special provisions need to be made in order to ensure URI Signing can be applied. In general, segmented protocols work by breaking large objects (e.g., videos) into a sequence of small independent segments. Such segments are then referenced by a separate manifest file, which either includes a list of URLs to the segments or specifies an algorithm through which a User Agent can construct the URLs to the segments. Requests for segments therefore originate from the manifest file and, unless the URLs in the manifest file point to the CSP, are not subjected to redirection and URI Signing. This opens up a vulnerability to malicious User Agents sharing the manifest file and deep-linking to the segments.

One method for dealing with this vulnerability would be to include, in the manifest itself, Signed URIs that point to the individual segments. There exist a number of issues with that approach. First, it requires the CDN delivering the manifest to rewrite the manifest file for each User Agent, which would require the CDN to be aware of the exact segmentation protocol used. Secondly, it could also require the expiration time of the Signed URIs to be valid for an extended duration if the content described by the manifest is meant to be consumed in real time. For instance, if the manifest file were to contain a segmented video stream of more than 30 minutes in length, Signed URIs would require to be valid for a at least 30 minutes, thereby reducing their effectiveness and that of the URI Signing mechanism in general. For a more detailed analysis of how segmented protocols such as HTTP Adaptive Streaming protocols affect CDNI, see Models for HTTP-Adaptive-Streaming-Aware CDNI [RFC6983].

The method described in this section allows CDNs to use URI Signing for segmented content without having to include the Signed URIs in the manifest files themselves.

3.2. Signed Token Renewal mechanism

In order to allow for effective access control of segmented content, the URI signing mechanism defined in this section is based on a method through which subsequent segment requests can be linked together. As part of the JWT verification procedure, the CDN can generate a new signed JWT that the UA can use to do a subsequent request. More specifically, whenever a UA successfully retrieves a segment, it receives, in the HTTP 2xx Successful message, a signed JWT that it can use whenever it requests the next segment. As long as each successive signed JWT is correctly verified before a new one is generated, the model is not broken and the User Agent can successfully retrieve additional segments. Given the fact that with segmented protocols, it is usually not possible to determine a priori which segment will be requested next (i.e., to allow for seeking within the content and for switching to a different representation), the Signed Token Renewal uses the URI Regular Expression Container scoping mechanisms in the URI Container (cdniuc) claim to allow a signed JWT to be valid for more than one URL.

In order for this renewal of signed JWTs to work, it is necessary for a UA to extract the signed JWT from the HTTP 2xx Successful message of an earlier request and use it to retrieve the next segment. The exact mechanism by which the client does this is outside the scope of this document. However, in order to also support legacy UAs that do not include any specific provisions for the handling of signed JWTs, the following section defines a mechanism using HTTP Cookies [RFC6265] that allows such UAs to support the concept of renewing signed JWTs without requiring any additional UA support.

3.2.1. Required Claims

The cdnistt claim (Section 2.1.13) and cdniets claim (Section 2.1.12) MUST both be present for Signed Token Renewal. You MAY set cdnistt to a value of '0' to mean no Signed Token Renewal, but you still MUST have a corresponding cdniets that verifies as a JSON number. However, if you do not want to use Signed Token Renewal, it is RECOMMENDED to simply omit both.

3.3. Communicating a signed JWTs in Signed Token Renewal

This section assumes the value of the CDNI Signed Token Transport (cdnistt) claim has been set to 1. Other values of cdnistt are out of scope of this document.

When using the Signed Token Renewal mechanism, the signed JWT is transported to the UA via a 'URISigningPackage' cookie added to the HTTP 2xx Successful message along with the content being returned to the UA, or to the HTTP 3xx Redirection message in case the UA is redirected to a different server.

3.3.1. Support for cross-domain redirection

For security purposes, the use of cross-domain cookies is not supported in some application environments. As a result, the Cookie-based method for transport of the Signed Token described in the previous section might break if used in combination with an HTTP 3xx Redirection response where the target URL is in a different domain. In such scenarios, Signed Token Renewal of a signed JWT SHOULD be communicated via the query string instead, in a similar fashion to how regular signed JWTs (outside of Signed Token Renewal) are communicated. Note that the use of URL embedded signed JWTs SHOULD NOT be used in HTTP 2xx Successful messages, since UAs might not know how to extract the signed JWTs.

Note that the process described herein only works in cases where both the manifest file and segments constituting the segmented content are delivered from the same domain. In other words, any redirection between different domains needs to be carried out while retrieving the manifest file.

4. Relationship with CDNI Interfaces

Some of the CDNI Interfaces need enhancements to support URI Signing. A dCDN that supports URI Signing needs to be able to advertise this capability to the uCDN. The uCDN needs to select a dCDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the uCDN needs to be able to distribute via the CDNI Metadata interface the information necessary to allow the dCDN to verify a Signed URI. Events that pertain to URI Signing (e.g., request denial or delivery after access authorization) need to be included in the logs communicated through the CDNI Logging interface.

4.1. CDNI Control Interface

URI Signing has no impact on this interface.

4.2. CDNI Footprint & Capabilities Advertisement Interface

The CDNI Request Routing: Footprint and Capabilities Semantics document [RFC8008] defines support for advertising CDNI Metadata capabilities, via CDNI Payload Type. The CDNI Payload Type registered in Section 6.1 can be used for capability advertisement.

4.3. CDNI Request Routing Redirection Interface

The CDNI Request Routing Redirection Interface [RFC7975] describes the recursive request redirection method. For URI Signing, the uCDN signs the URI provided by the dCDN. URI Signing therefore has has no impact on this interface.

4.4. CDNI Metadata Interface

The CDNI Metadata Interface [RFC8006] describes the CDNI metadata distribution needed to enable content acquisition and delivery. For URI Signing, a new CDNI metadata object is specified.

The UriSigning Metadata object contains information to enable URI signing and verification by a dCDN. The UriSigning properties are defined below.

Property: enforce

Description: URI Signing enforcement flag. Specifically, this flag indicates if the access to content is subject to URI Signing. URI Signing requires the dCDN to ensure that the URI is signed and verified before delivering content. Otherwise, the dCDN does not perform verification, regardless of whether or not the URI is signed.

Type: Boolean

Mandatory-to-Specify: No. The default is true.

Property: issuers

Description: A list of valid Issuers against which the Issuer claim in the signed JWT may be verified.

Type: Array of Strings

Mandatory-to-Specify: No. The default is an empty list. An empty list means that any Issuer is acceptable.

Property: package-attribute

Description: The name to use for the URI Signing Package.

Type: String

Mandatory-to-Specify: No. The default is "URISigningPackage".

Property: jwt-header

Description: The header part of JWT that is used for generating or verifying a signed JWT when the JWT token in the URI Signing Package does not contain a header part.

Type: String

Mandatory-to-Specify: No. By default, the header is assumed to be included in the JWT token.

The following is an example of a URI Signing metadata payload with all default values:


```
{
  "generic-metadata-type": "MI.UriSigning"
  "generic-metadata-value": {}
}
```


The following is an example of a URI Signing metadata payload with explicit values:

```
{
  "generic-metadata-type": "MI.UriSigning"
  "generic-metadata-value":
    {
      "enforce": true,
      "issuers": ["csp", "ucdn1", "ucdn2"],
      "package-attribute": "usp",
      "jwt-header":
        {
            "alg": "ES256",
            "kid": "P5UpOv0eMq1wcxLf7WxIg09JdSYGYFDOWkldueaImf0"
        }
    }
}
```

4.5. CDNI Logging Interface

For URI Signing, the dCDN reports that enforcement of the access control was applied to the request for content delivery. When the request is denied due to enforcement of URI Signing, the reason is logged.

The following CDNI Logging field for URI Signing SHOULD be supported in the HTTP Request Logging Record as specified in CDNI Logging Interface [RFC7937], using the new "cdni_http_request_v2" record-type registered in Section 6.2.1.

- s-uri-signing (mandatory):
  - format: 3DIGIT
  - field value: this characterises the URI signing verification performed by the Surrogate on the request. The allowed values are registered in Section 6.4.
  - occurrence: there MUST be zero or exactly one instance of this field.
- s-uri-signing-deny-reason (optional):
  - format: QSTRING
  - field value: a string for providing further information in case the signed JWT was rejected, e.g., for debugging purposes.
  - occurrence: there MUST be zero or exactly one instance of this field.

5. URI Signing Message Flow

URI Signing supports both HTTP-based and DNS-based request routing. JSON Web Token (JWT) [RFC7519] defines a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a signed JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

5.1. HTTP Redirection

For HTTP-based request routing, a set of information that is unique to a given end user content request is included in a signed JWT, using key information that is specific to a pair of adjacent CDNI hops (e.g., between the CSP and the uCDN or between the uCDN and a dCDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI signing method (assuming HTTP redirection, iterative request routing, and a CDN path with two CDNs) includes the following steps:

```
        End-User           dCDN                 uCDN                CSP
        |                    |                    |                    |
        |            1.CDNI FCI interface used to |                    |
        |         advertise URI Signing capability|                    |
        |                    |------------------->|                    |
        |                    |                    |                    |
        |              2.Provides information to verify signed JWT     |
        |                    |                    |<-------------------|
        |                    |                    |                    |
        |        3.CDNI Metadata interface used to|                    |
        |           provide URI Signing attributes|                    |
        |                    |<-------------------|                    |
        :                    :                    :                    :
        :   (Later in time)  :                    :                    :
        |4.Authorization request                  |                    |
        |------------------------------------------------------------->|
        |                    |                    |  [Apply distribution
        |                    |                    |   policy]          |
        |                    |                    |                    |
        |                    |             (ALT: Authorization decision)
        |5.Request is denied |                    |         <Negative> |
        |<-------------------------------------------------------------|
        |                    |                    |                    |
        |6.CSP provides signed URI                |         <Positive> |
        |<-------------------------------------------------------------|
```

```
        |                    |                    |                    |
        |7.Content request   |                    |                    |
        |---------------------------------------->| [Verifiy URI       |
        |                    |                    |  signature]        |
        |                    |                    |                    |
        |                    |    (ALT: Verification result)           |
        |8.Request is denied |          <Negative>|                    |
        |<----------------------------------------|                    |
        |                    |                    |                    |
        |9.Re-sign URI and redirect to  <Positive>|                    |
        |  dCDN (newly signed URI)                |                    |
        |<----------------------------------------|                    |
        |                    |                    |                    |
        |10.Content request  |                    |                    |
        |------------------->| [Verify URI        |                    |
        |                    |  signature]        |                    |
        |                    |                    |                    |
        |    (ALT: Verification result)           |                    |
        |11.Request is denied| <Negative>         |                    |
        |<-------------------|                    |                    |
        |                    |                    |                    |
        |12.Content delivery | <Positive>         |                    |
        |<-------------------|                    |                    |
        :                    :                    :                    :
        :   (Later in time)  :                    :                    :
        |13.CDNI Logging interface to include URI Signing information  |
        |                    |------------------->|                    |
```

Figure 4: HTTP-based Request Routing with URI Signing

1.   Using the CDNI Footprint & Capabilities Advertisement interface, the dCDN advertises its capabilities including URI Signing support to the uCDN.

2.   CSP provides to the uCDN the information needed to verify signed JWTs from that CSP. For example, this information may include a key value.

3.   Using the CDNI Metadata interface, the uCDN communicates to a dCDN the information needed to verify signed JWTs from the uCDN for the given CSP. For example, this information may include the URI query string parameter name for the URI Signing Package Attribute.

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 personal distribution policy.

5. If the authorization decision is negative, the CSP rejects the request and sends an error code (e.g., 403 Forbidden) in the HTTP response.

6. If the authorization decision is positive, the CSP computes a Signed 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 uCDN verifies the signed JWT in the URI using the information provided by the CSP.

8. If the verification is negative, the uCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.

9. If the verification is positive, the uCDN computes a Signed URI that is based on unique parameters of that request and provides it to the end user as the URI to use to further request the content from the dCDN.

10. On receipt of the corresponding content request, the dCDN verifies the signed JWT in the Signed URI using the information provided by the uCDN in the CDNI Metadata.

11. If the verification is negative, the dCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.

12. If the verification is positive, the dCDN serves the request and delivers the content.

13. At a later time, the dCDN reports logging events that include URI signing information.

With HTTP-based request routing, URI Signing matches well the general chain of trust model of CDNI both with symmetric and asymmetric keys because the key information only needs to be specific to a pair of adjacent CDNI hops.

5.2. DNS Redirection

For DNS-based request routing, the CSP and uCDN must agree on a trust model appropriate to the security requirements of the CSP's particular content. Use of asymmetric public/private keys allows for unlimited distribution of the public key to dCDNs. However, if a shared secret key is preferred, then the CSP may want to restrict the distribution of the key to a (possibly empty) subset of trusted dCDNs. Authorized Delivery CDNs need to obtain the key information to verify the Signed URI.

The URI signing method (assuming iterative DNS request routing and a CDN path with two CDNs) includes the following steps.

```
     End-User            dCDN                 uCDN                CSP
     |                    |                    |                    |
     |            1.CDNI FCI interface used to |                    |
     |         advertise URI Signing capability|                    |
     |                    |------------------->|                    |
     |                    |                    |                    |
     |              2.Provides information to verify signed JWT     |
     |                    |                    |<-------------------|
     |        3.CDNI Metadata interface used to|                    |
     |           provide URI Signing attributes|                    |
     |                    |<-------------------|                    |
     :                    :                    :                    :
     :   (Later in time)  :                    :                    :
     |4.Authorization request                  |                    |
     |------------------------------------------------------------->|
     |                    |                    |  [Apply distribution
     |                    |                    |   policy]          |
     |                    |                    |                    |
     |                    |             (ALT: Authorization decision)
     |5.Request is denied |                    |      <Negative>    |
     |<-------------------------------------------------------------|
     |                    |                    |                    |
     |6.Provides signed URI                    |      <Positive>    |
     |<-------------------------------------------------------------|
     |                    |                    |                    |
     |7.DNS request       |                    |                    |
     |---------------------------------------->|                    |
     |                    |                    |                    |
     |8.Redirect DNS to dCDN                   |                    |
     |<----------------------------------------|                    |
     |                    |                    |                    |
     |9.DNS request       |                    |                    |
     |------------------->|                    |                    |
     |                    |                    |                    |
     |10.IP address of Surrogate               |                    |
     |<-------------------|                    |                    |
     |                    |                    |                    |
     |11.Content request  |                    |                    |
     |------------------->| [Verify URI        |                    |
     |                    |  signature]        |                    |
     |                    |                    |                    |
     |    (ALT: Verification result)           |                    |
```

```
 |12.Request is denied| <Negative>         |                    |
 |<-------------------|                    |                    |
 |                    |                    |                    |
 |13.Content delivery | <Positive>         |                    |
 |<-------------------|                    |                    |
 :                    :                    :                    :
 :   (Later in time)  :                    :                    :
 |14.CDNI Logging interface to report URI Signing information   |
 |                    |------------------->|                    |
```

Figure 5: DNS-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the dCDN advertises its capabilities including URI Signing support to the uCDN.

2. CSP provides to the uCDN the information needed to verify cryptographic signatures from that CSP. For example, this information may include a key.

3. Using the CDNI Metadata interface, the uCDN communicates to a dCDN the information needed to verify 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 uCDN checks if the dCDN 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 and sends an error code (e.g., 403 Forbidden) in the HTTP response.

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 uCDN.

8. On receipt of the DNS request, the uCDN redirects the request to the dCDN.

9. End user sends DNS request to the dCDN.

10. On receipt of the DNS request, the dCDN responds with IP address of one of its Surrogates.

11. On receipt of the corresponding content request, the dCDN verifies the cryptographic signature in the URI using the information provided by the uCDN in the CDNI Metadata.

12. If the verification is negative, the dCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.

13. If the verification is positive, the dCDN serves the request and delivers the content.

14. At a later time, dCDN reports logging events that includes URI signing information.

With DNS-based request routing, URI Signing matches well the general chain of trust model of CDNI when used with asymmetric keys because the only key information that needs to be distributed across multiple, possibly untrusted, CDNI hops is the public key, which is generally not confidential.

With DNS-based request routing, URI Signing does not match well the general chain of trust model of CDNI when used with symmetric keys because the symmetric key information needs to be distributed across multiple CDNI hops, to CDNs with which the CSP may not have a trust relationship. This raises a security concern for applicability of URI Signing with symmetric keys in case of DNS-based inter-CDN request routing.

6. IANA Considerations

6.1. CDNI Payload Type

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

Payload Type	Specification
MI.UriSigning	RFCthis

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

6.1.1. CDNI UriSigning Payload Type

Purpose: The purpose of this payload type is to distinguish UriSigning MI objects (and any associated capability advertisement).

Interface: MI/FCI

Encoding: see Section 4.4

6.2. CDNI Logging Record Type

This document requests the registration of the following CDNI Logging record-type under the IANA "CDNI Logging record-types" registry:

record-types	Reference	Description
cdni_http_request_v2	RFCthis	Extension to CDNI Logging Record version 1 for content delivery using HTTP, to include URI Signing logging fields

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

6.2.1. CDNI Logging Record Version 2 for HTTP

The "cdni_http_request_v2" record-type supports all of the fields supported by the "cdni_http_request_v1" record-type [RFC7937] plus the two additional fields "s-uri-signing" and "s-uri-signing-deny-reason", registered by this document in Section 6.3. The name, format, field value, and occurence information for the two new fields can be found in Section 4.5 of this document.

6.3. CDNI Logging Field Names

This document requests the registration of the following CDNI Logging fields under the IANA "CDNI Logging Field Names" registry:

Field Name	Reference
s-uri-signing	RFCthis
s-uri-signing-deny-reason	RFCthis

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

6.4. CDNI URI Signing Verification Code

The IANA is requested to create a new "CDNI URI Signing Verification Code" subregistry, in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Verification Code" namespace defines the valid values associated with the s-uri-signing CDNI Logging Field. The CDNI URI Signing Verification Code is a 3DIGIT value as defined in Section 4.5. Additions to the CDNI URI Signing Verification Code namespace will conform to the "Specification Required" policy as defined in [RFC8126]. Updates to this subregistry are expected to be sparse.

Value	Reference	Description
000	RFCthis	No signed JWT verification performed
200	RFCthis	Signed JWT verification performed and verified
400	RFCthis	Signed JWT verification performed and rejected because of incorrect signature
401	RFCthis	Signed JWT verification performed and rejected because of Issuer enforcement
402	RFCthis	Signed JWT verification performed and rejected because of Subject enforcement
403	RFCthis	Signed JWT verification performed and rejected because of Audience enforcement
404	RFCthis	Signed JWT verification performed and rejected because of Expiration Time enforcement
405	RFCthis	Signed JWT verification performed and rejected because of Not Before enforcement
406	RFCthis	Signed JWT verification performed and rejected because of Issued At enforcement
407	RFCthis	Signed JWT verification performed and rejected because of Nonce enforcement
408	RFCthis	Signed JWT verification performed and rejected because of Version enforcement
409	RFCthis	Signed JWT verification performed and rejected because of Critical Extention enforcement
410	RFCthis	Signed JWT verification performed and rejected because of Client IP enforcement
411	RFCthis	Signed JWT verification performed and rejected because of URI Container enforcement
500	RFCthis	Unable to perform signed JWT verification because of malformed URI

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

6.5. CDNI URI Signing Signed Token Transport

The IANA is requested to create a new "CDNI URI Signing Signed Token Transport" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Signed Token Transport" namespace defines the valid values that may be in the Signed Token Transport (cdnistt) JWT claim. Additions to the Signed Token Transport namespace conform to the "Specification

Required" policy as defined in [RFC8126]. Updates to this subregistry are expected to be sparse.

The following table defines the initial Enforcement Information Elements:

Value	Description	RFC
0	Designates token transport is not enabled	RFCthis
1	Designates token transport via cookie	RFCthis

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

6.6. JSON Web Token Claims Registration

This specification registers the following Claims in the IANA "JSON Web Token Claims" registry [IANA.JWT.Claims] established by [RFC7519].

6.6.1. Registry Contents

- o Claim Name: "cdniv"
- o Claim Description: CDNI Claim Set Version
- o Change Controller: IESG
- o Specification Document(s): Section 2.1.8 of [[this specification]]

- o Claim Name: "cdnicrit"
- o Claim Description: CDNI Critical Claims Set
- o Change Controller: IESG
- o Specification Document(s): Section 2.1.9 of [[this specification]]

- o Claim Name: "cdniip"
- o Claim Description: CDNI IP Address
- o Change Controller: IESG
- o Specification Document(s): Section 2.1.10 of [[this specification]]

- o Claim Name: "cdniuc"
- o Claim Description: CDNI URI Container
- o Change Controller: IESG
- o Specification Document(s): Section 2.1.11 of [[this specification]]

- o Claim Name: "cdniets"
- o Claim Description: CDNI Expiration Time Setting for Signed Token Renewal
- o Change Controller: IESG
- o Specification Document(s): Section 2.1.12 of [[this specification]]

- o Claim Name: "cdnistt"
- o Claim Description: CDNI Signed Token Transport Method for Signed Token Renewal
- o Change Controller: IESG
- o Specification Document(s): Section 2.1.13 of [[this specification]]

- o Claim Name: "cdnistd"
- o Claim Description: CDNI Signed Token Depth
- o Change Controller: IESG
- o Specification Document(s): Section 2.1.14 of [[this specification]]

6.7. Expert Review Guidance

Generally speaking, we should determine the registration has a rational justification and does not duplicate a previous registration. Early assignment should be permissible as long as there is a reasonable expectation that the specification will become formalized. Expert Reviewers should be empowered to pass judgements as they see fit, but should follow the guidelines of [RFC8126] Section 5 when reasonable.

7. Security Considerations

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of CDNI. The primary goal of URI Signing is to make sure that only authorized UAs are able to access the content, with a CSP being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as DRM, are more appropriate.

In general, it holds that the level of protection against illegitimate access can be increased by including more claims in the signed JWT. The current version of this document includes claims for enforcing Issuer, Client IP Address, Not Before time, and Expiration Time, however this list can be extended with other, more complex, attributes that are able to provide some form of protection against some of the vulnerabilities highlighted below.

That said, there are a number of aspects that limit the level of security offered by URI Signing and that anybody implementing URI Signing should be aware of.

o Replay attacks: A (valid) Signed URI may be used to perform replay attacks. The vulnerability to replay attacks can be reduced by picking a relatively short window between the Not Before time and Expiration Time attributes, although this is limited by the fact that any HTTP-based request needs a window of at least a couple of seconds to prevent a sudden network issues from denying legitimate UAs access to the content. One may also reduce exposure to replay attacks by including a unique one-time access ID via the Nonce attribute (jti claim). Whenever the dCDN receives a request with a given unique ID, it adds that ID to the list of 'used' IDs. In the case an illegitimate UA tries to use the same URI through a replay attack, the dCDN can deny the request based on the already-used access ID.

o Illegitimate clients behind a NAT: In cases where there are multiple users behind the same NAT, all users will have the same IP address from the point of view of the dCDN. This results in the dCDN not being able to distinguish between different users based on Client IP Address which can lead to illegitimate users being able to access the content. One way to reduce exposure to this kind of attack is to not only check for Client IP but also for other attributes, e.g., attributes that can be found in HTTP headers.

The shared key between CSP and uCDN may be distributed to dCDNs - including cascaded CDNs. Since this key can be used to legitimately sign a URL for content access authorization, it is important to know the implications of a compromised shared key.

If a shared key usable for signing is compromised, an attacker can use it to perform a denial-of-service attack by forcing the CDN to evaluate prohibitively expensive regular expressions embedded in a cdniuc claim. As a result, compromised keys should be timely revoked in order to prevent exploitation.

8. Privacy

The privacy protection concerns described in CDNI Logging Interface [RFC7937] apply when the client's IP address (cdniip) is embedded in the Signed URI. For this reason, the mechanism described in Section 2 encrypts the Client IP before including it in the URI Signing Package (and thus the URL itself).

9. Acknowledgements

The authors would like to thank the following people for their contributions in reviewing this document and providing feedback: Scott Leibrand, Kevin Ma, Ben Niven-Jenkins, Thierry Magnien, Dan York, Bhaskar Bhupalam, Matt Caulfield, Samuel Rajakumar, Iuniana Oprescu, Leif Hedstrom, Gancho Tenev, Brian Campbell, and Chris Lemmons.

10. Contributors

In addition, the authors would also like to make special mentions for certain people who contributed significant sections to this document.

- o Matt Caulfield provided content for the CDNI Metadata Interface section.

- o Emmanuel Thomas provided content for HTTP Adaptive Streaming.

- o Matt Miller provided consultation on JWT usage as well as code to generate working JWT examples.

11. References

11.1. Normative References

[POSIX.1] "The Open Group Base Specifications Issue 7", IEEE Std 1003.1 2018 Edition, Jan 2018, <http://pubs.opengroup.org/onlinepubs/9699919799/>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7937] Le Faucheur, F., Ed., Bertrand, G., Ed., Oprescu, I., Ed., and R. Peterkofsky, "Content Distribution Network Interconnection (CDNI) Logging Interface", RFC 7937, DOI 10.17487/RFC7937, August 2016, <https://www.rfc-editor.org/info/rfc7937>.

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/info/rfc8006>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

11.2. Informative References

[IANA.JWT.Claims]
IANA, "JSON Web Token Claims", <http://www.iana.org/assignments/jwt>.

[MPEG-DASH]
ISO, "Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment format", ISO/IEC 23009-1:2014, Edition 2, 05 2014, <http://www.iso.org/standard/65274.html>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <https://www.rfc-editor.org/info/rfc6707>.

[RFC6983] van Brandenburg, R., van Deventer, O., Le Faucheur, F., and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware Content Distribution Network Interconnection (CDNI)", RFC 6983, DOI 10.17487/RFC6983, July 2013, <https://www.rfc-editor.org/info/rfc6983>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <https://www.rfc-editor.org/info/rfc7336>.

[RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <https://www.rfc-editor.org/info/rfc7337>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7975] Niven-Jenkins, B., Ed. and R. van Brandenburg, Ed., "Request Routing Redirection Interface for Content Delivery Network (CDN) Interconnection", RFC 7975, DOI 10.17487/RFC7975, October 2016, <https://www.rfc-editor.org/info/rfc7975>.

[RFC8008] Seedorf, J., Peterson, J., Previdi, S., van Brandenburg, R., and K. Ma, "Content Delivery Network Interconnection (CDNI) Request Routing: Footprint and Capabilities Semantics", RFC 8008, DOI 10.17487/RFC8008, December 2016, <https://www.rfc-editor.org/info/rfc8008>.

[RFC8216] Pantos, R., Ed. and W. May, "HTTP Live Streaming", RFC 8216, DOI 10.17487/RFC8216, August 2017, <https://www.rfc-editor.org/info/rfc8216>.

Appendix A. Signed URI Package Example

This section contains three examples of token usage: a simple example with only the required claim present, a complex example which demonstrates the full JWT claims set, including an encrypted Client IP (cdniip), and one that uses a Signed Token Renewal.

Note: All of the examples have whitespace added to improve formatting and readability, but are not present in the generated content.

All examples use the following JWK Set [RFC7517]:

```
{ "keys": [
  {
    "kty": "EC",
    "kid": "P5UpOv0eMq1wcxLf7WxIg09JdSYGYFDOWklduea Imf0",
    "use": "sig",
    "alg": "ES256",
    "crv": "P-256",
    "x": "be807S4O7dzB6I4hTiCUvmxCI6FuxWba1xYBlLSSsZ8",
    "y": "rOGC4vI69g-WF9AGEVI37sNNwbjIzBxSjLvIL7f3RBA"
  },
  {
    "kty": "EC",
    "kid": "P5UpOv0eMq1wcxLf7WxIg09JdSYGYFDOWklduea Imf0",
    "use": "sig",
    "alg": "ES256",
    "crv": "P-256",
    "x": "be807S4O7dzB6I4hTiCUvmxCI6FuxWba1xYBlLSSsZ8",
    "y": "rOGC4vI69g-WF9AGEVI37sNNwbjIzBxSjLvIL7f3RBA",
    "d": "yaowezrCLTU6yIwUL5RQw67cHgvZeMTLVZXjUGb1A1M"
  },
  {
    "kty": "oct",
    "kid": "f-WbjxBC3dPuI3d24kP2hfvos7Qz688UTi6aB0hN998",
    "use": "enc",
    "alg": "A128GCM",
    "k": "4uFxxV7fhNmrtiah2d1fFg"
  }
]}
```


Note: They are the public signing key, the private signing key, and the shared secret enctyption key, respectively. The public and private signing keys have the same fingerprint and only vary by the 'd' parameter that is missing from the public signing key.

A.1. Simple Example

This example is a simple common usage example containing a minimal subset of claims that the authors find most useful.

The JWT Claim Set before signing:

Note: "sha-256;2tderfWPa86Ku7YnzW51YUp7dGUjBS_3SW3ELx4hmWY" is the URL Segment form ([RFC6920] Section 5) of "http://cdni.example/foo/bar".

```
{
  "exp": 1474243500,
  "iss": "uCDN Inc",
  "cdniuc": "hash:sha-256;2tderfWPa86Ku7YnzW51YUp7dGUjBS_3SW3ELx4hmWY"
}
```


The signed JWT:


```
eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJleHAiOjE0NzQyNDM1MDAsImlzcyI6InVDRE4gS
W5jIiwiY2RuaXVjIjoiaGFzaDpzaGEtMjU2OzJ0ZGVyZldQYTg2S3U3WW56VzUxWVV
wN2RHVWpCU18zU1czRUx4NGhtV1kifQ.qzzAB9akC-HoEzQrkOoODWjMC0PEZRrmWz
2rSMcpLtvxyxVodlB2xcpl4J4ABhLLOJzgzL9B39TljTqZApSOpQ
```


A.2. Complex Example

This example uses all fields except for those dealing with Signed Token Renewal, including Client IP (cdniip) and Subject (sub) which are encrpyted. This significantly increases the size of the signed JWT token.

JWE for Client IP (cdniip) of [2001:db8::1/32]:


```
eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiZGlyIiwia2lkIjoiZi1XYmp4QkMzZFB1ST
NkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..SuzoOnfg-GVh-BOc.wQ9iS
R1sTj-A04CiDmvcgg.9Ts_cIEUw6Yc6U5HaH1UPQ
```


JWE for Subject (sub) of "UserToken":


```
eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiZGlyIiwia2lkIjoiZi1XYmp4QkMzZFB1ST
NkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..XsJ7ySeChORSIojp.R1U8E
SGU2NnW.DWR8pTbeCwQZca6SitfX_g
```


The JWT Claim Set before signing:

```
{
  "aud": "dCDN LLC",
  "sub": "eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiZGlyIiwia2lkIjoiZi1XYmp4
QkMzZFB1STNkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..XsJ7ySeChORS
Iojp.R1U8ESGU2NnW.DWR8pTbeCwQZca6SitfX_g",
  "cdniip": "eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiZGlyIiwia2lkIjoiZi1XY
mp4QkMzZFB1STNkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..SuzoOnfg-
GVh-BOc.wQ9iSR1sTj-A04CiDmvcgg.9Ts_cIEUw6Yc6U5HaH1UPQ",
  "cdniv": 1,
  "exp": 1474243500,
  "iat": 1474243200,
  "iss": "uCDN Inc",
  "jti": "5DAafLhZAfhsbe",
  "nbf": 1474243200,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\.png"
}
```


The signed JWT:


```
eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJhdWQiOiJkQ0ROIExMQyIsInN1YiI6ImV5Smxib
U1pT2lKQk1USTRSME5OSWl3aVlXeG5Jam9pWkdseUlpd2lhMmxrSWpvaVppMVhZbXA
0UWtNelpGQjFTVE5rTWpSclVESm9ablp2Y3pkUmVqWTRPRlZVYVRaaFFqQm9Uams1T
0NKOS4uWHNKN3lTZUNoT1JTSW9qcC5SMVU4RVNHVTJOblcuRFdSOHBUYmVDd1FaY2E
2U2l0ZlhfZyIsImNkbmlpcCI6ImV5SmxibU1pT2lKQk1USTRSME5OSWl3aVlXeG5Ja
m9pWkdseUlpd2lhMmxrSWpvaVppMVhZbXA0UWtNelpGQjFTVE5rTWpSclVESm9ablp
2Y3pkUmVqWTRPRlZVYVRaaFFqQm9Uams1T0NKOS4uU3V6b09uZmctR1ZoLUJPYy53U
TlpU1Ixc1RqLUEwNENpRG12Y2dnLjljUc19jSUVVdzZZYzZVNUhhSDFVUFEiLCJjZG5
pdiI6MSwiZXhwIjoxNDc0MjQzNTAwLCJpYXQiOjE0NzQyNDMyMDAsImlzcyI6InVDR
E4gSW5jIiwianRpIjoiNURBYWZMaFpBZmhzYmUiLCJuYmYiOjE0NzQyNDMyMDAsImN
kbml1YyI6InJlZ2V4Omh0dHA6Ly9jZG5pXFwuZXhhbXBsZS9mb28vYmFyL1swLTlde
zN9XFwucG5nIn0.XEi1NeP8Lzh6ECcbp6EoqYlnJGikaGp6F3lIJ7ZJt3bim6tOtuD
pCQxmEQxobzIpWOCNdpB8kvxM_s95brKjNQ
```


A.3. Signed Token Renewal Example

This example uses fields for Signed Token Renewal.

The JWT Claim Set before signing:


```
{
  "cdniets": 30,
  "cdnistt": 1,
  "cdnistd": 2,
  "exp": 1474243500,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\.ts"
}
```

The signed JWT:

eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJjZG5pZXRzIjozMCwiY2RuaXN0dCI6MSwiY2Rua
XN0ZCI6MiwiZXhwIjoxNDc0MjQzNTAwLCJjZG5pdWMiOiJyZWdleDpodHRwOi8vY2R
uaVxcLmV4YW1wbGUvZm9vL2Jhci9bMC05XXszfVxcLnRzIn0.wsSvwxY8mtRax7HK_
dro_l6m-mM-HYdeaUoTSgVS5XTIhXBsCPsYQncsradmgmOWHDDOxsSMVVTjHe5E5YH
ZlQ

Once the server verifies the signed JWT it will return a new signed JWT with an updated expiry time (exp) as shown below. Note the expiry time is increased by the expiration time setting (cdniets) value.

The JWT Claim Set before signing:


```
{
  "cdniets": 30,
  "cdnistt": 1,
  "cdnistd": 2,
  "exp": 1474243530,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\.ts"
}
```


The signed JWT:

eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJjZG5pZXRzIjozMCwiY2RuaXN0dCI6MSwiY2Rua
XN0ZCI6MiwiZXhwIjoxNDc0MjQzNTMwLCJjZG5pdWMiOiJyZWdleDpodHRwOi8vY2R
uaVxcLmV4YW1wbGUvZm9vL2Jhci9bMC05XXszfVxcLnRzIn0.SITeoIVZ8-yeE_GBV
jYEo1P2LN-EId1gEJ6baR3Au7Dzh2o_O7LhH3k6wHY081sYMdXHucB0P5ocp-r7gqe
ruQ

Authors' Addresses

Ray van Brandenburg
Tiledmedia
Anna van Buerenplein 1
Den Haag 2595DA
The Netherlands

Phone: +31 88 866 7000
Email: ray@tiledmedia.com

Kent Leung
Cisco Systems, Inc.
3625 Cisco Way
San Jose, CA 95134
United States

Phone: +1 408 526 5030
Email: kleung@cisco.com

Phil Sorber
Apple, Inc.
1800 Wazee Street
Suite 410
Denver, CO 80202
United States

Email: sorber@apple.com