Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

O. Finkelman
Qwilt
S. Mishra
Verizon

October 30, 2017

CDNI SVA Extensions
draft-finkelman-cdni-sva-extensions-00

Abstract

The Open Caching working group of the Streaming Video Alliance is focused on the delegation of video delivery request from commercial CDNs to a caching layer at the ISP. In that aspect, Open Caching is a specific use case of CDNI, where the commercial CDN is the upstream CDN (uCDN) and the ISP caching layer is the downstream CDN (dCDN).

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

In this document, we describe the different use cases of Open Caching and the interface and functionality extensions they require, compared to the existing CDNI RFCs. For consistency, this document follows the CDNI notation of uCDN (the commercial CDN) and dCDN (the ISP caching layer). When using the term CP in this document we refer to a video content provider.

The CDNI Logging interface is described in [RFC7937].

The CDNI metadata interface is described in [RFC8006].

The CDNI footprint and capability interface is described in [RFC8008].

The CDNI control interface / triggers is described in [RFC8007].

1.1. Terminology

This document reuses the terminology defined in [RFC6707], [RFC8006], [RFC8007], and [RFC8008].

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

o SVA - Streaming Video Alliance.

o OC - SVA Open Caching.

o RR - Request Router.

o CP - Content Provider.

2. Request routing

This section lists extensions required by request routing features.

2.1. Request router address

Open Caching uses iterative request redirect as defined in [RFC7336]. In order for the uCDN to redirect to the dCDN it requires a request router address. CDNI RFCs do not specify how the request router address is advertised and suggests it may be passed via a bootstrap protocol / interface, which is currently not defined.

We propose to add the request router address as a capability under the Footprint and Capabilities interface.

Use cases

* Footprint: The dCDN may want to have different RR addresses per footprint. Note that a dCDN may spread across multiple geographies. This makes it easier to route client request to a nearby RR. Though this can be achieved using a single canonical name and geo DNS, that approach has limitations, for example a client may be using third party DNS resolver, making it impossible for the redirector to detect where the client is located.

* Scaling: The dCDN may choose to scale its RR service by deploying more RRs in new locations and advertise them via an updatable interface like the FCI.

Proposal

Advertise request router address in an FCI capability object.

Example FCI.RequestRouterAddress object:

```
{
     "capabilities": [
       {
         "capability-type": "FCI.RequestRouterAddress",
         "capability-value": {
           "address": <endpoint object>
         },
         "footprints": [
           <Footprint objects>
         ]
       }
     ]
}
```

2.2. uCDN fallback address

Open Caching requires that the uCDN should provide a fallback address to the dCDN to be used in cases where the dCDN cannot properly handle the request. To avoid redirect loops, the dCDN would redirect the request back to the uCDN but to a different location than the original uCDN address, the uCDN will not redirect requests coming to that other address.

Use cases

* Failover: A dCDN request router receives a request but has no caches to which it can route the request to. This can happen

in the case of failures, or temporary network overload. In these cases, the router may choose to redirect the request back to the uCDN fallback address.

* Error: A cache may receive a request that it cannot properly serve, for example, some of the metadata objects for that service were not properly acquired. In this case the cache may resolve to redirect back to uCDN.

Proposal

Add a generic metadata object for fallback address similar to the source metadata.

Example MI.FallbackAddress object:

```
{
     "generic-metadata-type": "MI.FallbackAddress",
     "generic-metadata-value":
       {
         "sources": [
           {
             "endpoints": [
               "fallback-a.service123.ucdn.example",
               "fallback-b.service123.ucdn.example"
               ],
             "protocol": "http/1.1"
           },
           {
             "endpoints": ["origin.service123.example"],
             "protocol": "http/1.1"
           }
         ]
       }
}
```

3. Content management

Open Caching uses the CDNI CI/T [RFC8007] as an interface for content management operations. The basic operations are the ones defined in the RFC (i.e. purge, invalidate, pre-position).

3.1. Content matching rules

RFC8007 provides means to match on full content URL or patterns with wildcards. The Open Caching working group proposes to add two more match rule types.

3.1.1. Regular expresssion

Using regexp one can create more complex rules to match on objects for the cases of invalidation and purge.

Use cases

* Purge: Purging specific content within a specific directory path. In some cases wildcard MAY be used but it can be a constraining or overreaching variable that exposes the assets to purge further than desired.

Proposal

Add content.regexs to trigger specification.

Name: content.regexs

Description: Regexs of content the CI/T Trigger Command applies to.

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

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

3.1.2. Playlist

Using video playlist files, one can trigger an operation that will work on a collection of distinct media files in a representation that is natural for the content provider. A playlist may have several formats, specifically HLS *.m3u8 manifest [RFC8216], MSS *.ismc client manifest, and DASH XML MPD file [ISO/IEC 23009-1:2014].

Use cases

* Pre-position: Pre-position of content requires passing the full list of media files to the dCDN. Passing the manifest instead is a more natural interface for both sides as they are both supposed to be able to properly read and understand the manifest files.

Proposal

Add playlist.urls to trigger specification.

Name: playlist.urls

Description: URLs of video playlist the CI/T Trigger Command applies to.

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

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

3.2. Geo limits

A content operation may apply for a specific geographical region, or need to be excluded from a specific region. In this case, the trigger should be applied only to parts of the network that are included or not excluded by the geo limit. Note that the limit here is on the cache location rather than client location.

Use cases

* Pre-position: Certain contracts allow for prepositioning or availability of contract in all regions except for certain excluded regions in the world, including caches. For example, some CPs content cannot ever knowingly touch servers in a specific country, including caches. Therefore, these regions MUST be excluded from a pre-positioning operation.

* Purge: In certain cases, content may have been located on servers in regions where the content MUST not reside on. In such cases a purge operation to remove content specifically from that region, is required.

Proposal

Add GEO locations as an option in the trigger specification. We should consider where this locations object is defined. Should this a part of CI/T or there can be a way we can use metadata objects. The generic metadata object MI.LocationAcl has the same syntax, though the meaning is different as the limit here is on caches rather than end user locations.

Example of trigger specification with a geo limit:

```
        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",
            "content.urls": [
                "https://www.example.com/a/b/c/1",
                "https://www.example.com/a/b/c/2"
              ]
          },
          "locations": [
            {
              "action": "allow" / "deny",
              "footprints": [
                {
                  "footprint-type": "countrycode",
                  "footprint-value": ["us"]
                }
              ]
            }
          ],
          "cdn-path": [ "AS64496:1" ]
        }
```

3.3. Scheduled operations

A uCDN may wish to perform content management operation on the dCDN with a defined local time schedule.

Use cases

* Pre-position: A content provider wishes to pre-populate a new episode at off-peak time so that it would be ready on caches (for example home caches) at prime time when the episode is released for viewing. This requires an interface that directs the dCDN when to pre-position the content; the time frame is local time per area as the off-peak time is also localized.

Proposal

Add an execution time window as an option in the trigger specification.

Example of trigger specification with a schedule limit:

```
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",
    "content.urls": [
        "https://www.example.com/a/b/c/1",
        "https://www.example.com/a/b/c/2"
      ]
  },
  "time-windows": [
    {
      "time-type": "local" / "UTC",
      "start": "<seconds since UNIX epoch>",
      "end": "<seconds since UNIX epoch>"

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

3.4. Trigger extensibility

There are cases in which some new data has to pass in the trigger which was not thought of in advance. We propose the add a mechanism to the trigger spec which will be similar to the MI generic metadata, allowing parties to easily add more information, that can later be standardized if required.

Use cases

* Purge content by acquisition time: A uCDN finds that due to configuration mistake it has delivered wrong content, in the past two hours. The uCDN would like to instruct the dCDN to invalidate all content that was acquired in the past two hours. However, there is no such primitive in the trigger specification. If this would be a common use case it may require the addition of a new generic trigger spec object that restrict the match to be on content which was acquired in some time spec.

* Pre-position by cache type: The uCDN would like the dCDN to pre-populate some content, but only on a specific layer of the caching network, for example, only on home caches. There is currently no such option in the interface. By using a generic object parties may define such object and implement it between them, and later standardize it, if required.

Proposal

Add trigger extensibility mechanism to the trigger specification.

Example of trigger extension:

```
     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": "purge",
         "content.patterns": [
             "https://www.example.com/*"
           ]
       },
       "generic-trigger-spec-type": <type-name>,
       "generic-trigger-spec-value":
         {
           <properties of this object>
         }
     }
```

3.5. Capabilties

The capabilities added to the triggers interface are not mandatory to support and are, therefore, best negotiated via the FCI.

Use cases

* Content management operations: Advertise which content operations are supported by the dCDN. CDNI defines three operations (purge, invalidate, pre-position), but it does not necessarily mean that all dCDNs support all of them. The uCDN

may prefer to work only with dCDN that support what the uCDN needs.

* Content mapping types: Advertise which mapping types are supported, for example, if adding content regexp and possibly playlists, not all dCDN would support it. For playlist, advertise which types and versions of protocols are supported, e.g. HLS/DASH/SS, DASH templates.

* Trigger spec objects: Advertise which trigger spec object are supported, for example time-window, geo-limit etc.

Proposal

Define the non-mandatory objects as generic objects, similar to the metadata generic objects, and then the FCI can declare which ones of the trigger spec objects are supported. .

4. Split authentication

Different CDNs and Content Providers apply different access control and authentication of user requests. It is not feasible for a dCDN, or ISP cache layer, to implement every scheme a uCDN may have thought of, and, unfortunately, it is not reasonable to expect that uCDNs and CPs will move from their current implementation to a new standard, any time soon. In some cases, existing implementation also include secrets under NDA; sharing them with a third party dCDN is unlikely to happen. Therefore, we aim to look for a solid, generic solution that keeps the access control, authentication and authorization logic in the origin/uCDN.

Use cases

* URI signing: There are numerous methods in which a CP signs its URIs such that the uCDN can verify the signatures. In most cases, symmetric keys are being used and require some key exchange. Expecting the dCDN caches to implement every method used by commercial CDNs is problematic, and sharing of content provider keys is unlikely.

* Token based authentication: Some CPs and CDNs are using token based client / session authentication. The token is passed either as a URI query parameter or as a cookie. The dCDN / ISP cannot implement the token validation, as it has no knowledge of the identity and validation methods used by the CP / uCDN. Also, if using cookies with HTTP redirect, the cookie will be omitted after the redirect, so a solution for cookie based authentication is necessary.

* CORS delegation: CORS may also be a use case of split authentication, see explanation in the CORS delegation section.

Proposal

Split authentication is a mechanism that leverages the fact that video sessions are very long and chunked into very small requests, comparing the overall session time and volume. The dCDN cache relays the authentication verification to the uCDN by sending the uCDN a HEAD request for every new session. The dCDN cache saves the session state for some time and uses it for subsequent requests of the same session.

As this is a general problem when delegating traffic between CDNs, and in-fact, can become a blocker for CDNI deployments. We propose to consider this concept for the general CDNI use case, and draft it for RFC.

The following diagram gives a high level sequence view of the URI signing use case.

```
     +------+          +------+               +------+           +-----+
     |Client|          |dCDN  |               |uCDN  |           | CP  |
     |      |          |      |               |      |           |     |
     +---+--+          +---+--+               +---+--+           +--+--+
         |                 |                      |                 |
 +----------------+        |                      |                 |
 |Access video on |        |                      |                 |
 |CP web site     |        |                      |                 |
 +-------+--------+        |                      |                 |
         | Get master manifest location           |                 |
         +-----------------+----------------------+----------------->
         |                 |Respond with signed URI to manifest     |
         <----------------------------------------+-----------------+
         | Get manifest    |                      |                 |
         +---------------------------------------->                 |
         |                 |                      |                 |
         |                 |              +-------+----------+      |
         |                 |              |Verify URI signing|      |
         |                 |              +-------+----------+      |
         |                 | Redirect to dCDN     |                 |
         <----------------------------------------+                 |
         | Get manifest    |                      |                 |
         +----------------->                      |                 |
         |                 |Authenticate URI      |                 |
         |                 +---------------------->                 |
         |                 |Authentication success|                 |
         | Master manifest <----------------------+                 |
```

```
         <-----------------+                      |                 |
         | Get sub manifest|                      |                 |
         +----------------->                      |                 |
         |                 |Authenticate URI      |                 |
         |                 +---------------------->                 |
         |                 |Authentication success|                 |
         |                 <----------------------+                 |
         |                 |                      |                 |
         |        +------------------+            |                 |
         |        |Save authenticated|            |                 |
         |        |session token     |            |                 |
         |        +--------+---------+            |                 |
         | Sub manifest    |                      |                 |
         <-----------------+                      |                 |
         | Request chunk 1 |                      |                 |
         +----------------->                      |                 |
         |                 |                      |                 |
         |      +---------------------+           |                 |
         |      | Use session state to|           |                 |
         |      | authenticate client |           |                 |
         |      | chunk requests      |           |                 |
         |      +----------+----------+           |                 |
         |     chunk 1     |                      |                 |
         <-----------------+                      |                 |
         |-Request chunk 2->                      |                 |
         <------chunk 2----|                      |                 |
         |-Request chunk 3->                      |                 |
         <------chunk 3----+                      |                 |
         |                 |                      |                 |
         +                 +                      +                 +
```

Figure 1

5. CORS delegation

CORS (Cross Origin Resource Sharing) is a mechanism designed to allow a resource from domain A to access other resources in domain B, overriding the same-origin policy. When a uCDN delegate traffic to a dCDN (or ISP) the dCDN is required to comply with the same CORS server behavior the uCDN would have had. For example, if a resource from domain A is accessible for request coming from a resource domain B, but not accessible to requests coming from a resource of domain C, the same logic must be done by the dCDN.

Though CORS can possibly be handled by simply echoing the Origin header value, or *, back to the client, in some cases it is not sufficient, and it also breaks the concept of CORS as an access control mechanism. As proper CORS handling is not possible without a

delegation scheme, the Open Caching working group sees it as an essential part of inter-CDN delegation, and therefore propose to adopt it under CDNI and draft it for CDNI RFC.

Use cases

* A simple use case example is a when resource from Origin: www.video.example.com points to the media file on domain: www.cdn.com. The uCDN is supposed to deliver the content if the Origin is video.example.com otherwise it should be rejected. In this case, for a request header "Origin: www.video.example.com" the CDN should reply with "Access-Control-Allow-Origin: www.video.example.com". OTOH, if the origin is www.video.other.com then the CDN should not allow it by omitting the ACAO header. When delegating the session to a dCDN cache, it should maintain the same behavior.

Proposals

There are several alternatives for the dCDN / ISP cache to learn the allowed origins for a content item.

1. Caching: Caching of CORS headers per content. If the cache receives a request using an origin it does not already approve for that content, the cache sends a HEAD request to the CDN with the client's CORS request headers. The cache saves the response information in a content database and uses it for subsequent requests for the same content. .

2. Metadata: the uCDN can provide the dCDN the metadata referring the content of a specific domain. This metadata holds, for example, all the information required to take CORS decisions at the Open Cache.

3. Split authentication: Using split authentication, the dCDN cache can send the CORS headers to the uCDN in the initial session request, the uCDN responds to the CORS request properly, the dCDN forwards the CORS response to the client and caches it for rest of the client session.

The following diagram gives a high level sequence view of CORS delegation from uCDN to dCDN using the CORS caching alternative.

```
     +------+          +------+               +------+           +-----+
   +-|Client|          |dCDN  |               | uCDN |           | CP  |
   | |1     |          |      |               |  B   |           |  A  |
   |2+---+--+          +---+--+               +---+--+           +--+--+
   +--+--|+                |                      |                 |
 +-------+-------------+   |                      |                 |
 |Access resource on CP|   |                      |                 |
 |www.example.com      |   |                      |                 |
 +-------+-------------+   |                      |                 |
      |  | Get resource A from example.com        |                 |
      |  +-----------------+----------------------+----------------->
      |  |     CP resource A points to a resource B on uCDN cdn.com |
      |  <----------------------------------------+-----------------+
      |  | Get B from uCDN ucdn.com               |                 |
      |  | Origin: example.com                    |                 |
      |  +--------------------------------------->                  |
      |  |                 |                      |                 |
      |  |                 |           +----------+-----------+     |
      |  |                 |           |uCDN Delegate to dCDN |     |
      |  |                 |           +----------+-----------+     |
      |  |                 | Redirect to dCDN     |                 |
      |  <----------------------------------------+                 |
      |  |Get B from dCDN  |                      |                 |
      |  |Origin: example.com                     |                 |
      |  +----------------->                      |                 |
      |  |                 | Request CORS for B   |                 |
      |  |                 | Origin: example.com  |                 |
      |  |                 +---------------------->                 |
      |  |                 | Provide CORS for B   |                 |
      |  |                 | Origin: example.com  |                 |
      |  |                 <----------------------+                 |
      |  |        +--------+----------+           |                 |
      |  |        + cache B CORS rules |          |                 |
      |  |        + Origin: example.com|          |                 |
      |  |        +--------+----------+           |                 |
      |  | Provide B with  |                      |                 |
      |  | CORS headers    |                      |                 |
      |  <-----------------+                      |                 |
      | Get B from uCDN ucdn.com                  |                 |
      | Origin: example.com                       |                 |
      |------------------------------------------->                 |
      |  |                 |                      |                 |
      |  |                 |           +----------+-----------+     |
      |  |                 |           |uCDN delegate to dCDN |     |
      |  |                 |           +----------+-----------+     |
      |  |                 | Redirect to dCDN     |                 |
      <-------------------------------------------+                 |
      | Get B from dCDN    |                      |                 |
```

```
      | Origin: example.com|                      |                 |
      |-------------------->                      |                 |
      |  |        +--------+-----------+          |                 |
      |  |        + use B cached CORS  |          |                 |
      |  |        + Origin: example.com|          |                 |
      |  |        +--------+-----------+          |                 |
      |  | Provide B with  |                      |                 |
      |  | CORS headers    |                      |                 |
      <--------------------+                      |                 |
      +  +                 +                      +                 +
```

Figure 2

In the above simplified example, we depict the caching alternative for CORS solution.

Client 1 accesses resource A on CP domain example.com. Resource A, refers client 1 to resource B on uCDN ucdn.com. Without delegation, at this points uCDN has to resolve CORS and decide if a resource from example.com is allowed to access a resource at ucdn.com. However, once delegated to dCDN, it becomes the dCDNs duty to resolve it for the client request arrives at the dCDN cache. The dCDN sends a CORS request to the uCDN, for resource B with origin example.com, it then uses the response to respond to client 1, and caches the response. When client 2's request arrives at the dCDN, the required CORS information is already in cache and the dCDN can serve client 2 without reiterating to uCDN.

For simplicity, in this diagram, we have ignored some of the challenges of CORS delegation like preflight requests and "null" origin after HTTP redirect.

6. Logging

This section outlines creation of service delivery logs at the dCDN (ISP) and transmittal of the logs by the dCDN to the uCDN. The key motivation for logging outlined below as compared to CDNI Logging Interface [RFC7937] is the ability for dCDN and uCDN to negotiate and agree on a log transport mechanism.

The logging mechanism provides the flexibility for CDNs to leverage common transport mechanism in-use already. Second, the open caching working group has selected Squid based file format given its wide usage within the CDN environments for access and cache logs, result codes and error messages. As an example, the result codes in squid return both the status code returned by downstream as well as result code indicator such as HIT, MISS, REFRESH_HIT, etc. Between the two statuses, it is easier to discern the delivery status. As an example, if the request was forbidden by the origin, the status field

will likely be MISS/403 or if it is a cache error response, it will be HIT/503. So, leveraging the Squid log already in use within the CDN environment and, equally important, the ability for CDNs to negotiate and agree on a file transport mechanisms, were the key motivations for open caching. These are therefore proposed as complementary extensions to the CDNI Logging Interface [RFC7937].

The sub-sections below explain extensions to the Footprint and Capabilities [RFC8008] and Metadata Interface [RFC8006]. The specific extension includes FCI announcement of supported log file transport types by dCDN and metadata response by uCDN to provision one or more log file types from the list sent by the dCDN.

Use cases

* Transport: Delivery logs are to be supplied by the dCDN to the uCDN via a transport mechanism of choice, supported by both dCDN and uCDN.

* Record format: Log record format is advertised by the dCDN and interpreted correctly by the uCDN. The dCDN in this case shall announce to uCDN one or more transport format that it supports. The uCDN, in turn, will select one format from the potential candidates and set up a provisioning process.

* Log destination: The uCDN configures a log receiving system tied to a specific delivery service it has delegated to a dCDN. The uCDN will provision log destination at its end where it will route the returned logs by delivery service associated with the log file.

The diagram below illustrates the use cases:

```
    Delivery Service A (VOD)    Delivery Service C    RR Logs
    Delivery Service B (Live)   (Linear)
    +----------------------+   +-----------------+   +-----------------+
    |                      |   |                 |   |                 |
    |Log Destination 1(VOD)|   |Log Destination 2|   |Log Destination 3|
    |       Logstash       |   |       Kafka     |   |       SFTP      |
    |                      |   |                 |   |                 |
    +------------^---------+   +----------^------+   +----------^------+
                 |                        |                     |
                 |                        |                     |
                 +----------------------------------------------+
  uCDN                                    |
+----------------------------------------------------------------------+
  dCDN                                    |
            Delivery Logs        +----------+-------+
            Service A/B/C        |                  |
          +-------+--------+----->                  |   RR Logs
          |       |        |     | dCDN Open Cache  <-----------+
          |       |        |     |  Controller      |           |
     +----+-----+ |        |     |                  |           |
     |          | |        |     +-------^-------+--+           |
     |       +--+-+----+   |             |       ^         +----+-----+
     |       |         |   |         +---+----+  |         |  Request |
     |       |      +--+---+--+      |        |  |         |  Router  |
     |       |      |         |      |     +--+--+-+       |          |
     +-------+      |         |      |     |       |       +----------+
             +------+  Cache  |      |     | Cache |
                    |         |      +-----+       |
                    +---------+            +-------+
```

Figure 3

Proposal

Delivery logs are created and then transferred from log producing entities at the dCDN premises (mainly caches and Request Router) to log destinations at the uCDN premises. The dCDN may offload logs from these entities to logging at the dCDN premises to facilitate log transfers, or, logs may be transferred directly from log producing entities to uCDN.

Various transport mechanisms may suit the use case of transferring log data, for example SFTP, HTTP upload, Kafka, Logstash or other methods as per the agreement between a dCDN and a uCDN.

In compliance with the CDNI Footprint and Capabilities Interface, and therefore, as per the above use cases, the dCDN is responsible to advertise supported Logging "record-types", as well as Logging

"fields" which are marked as optional for the s pecified "record-types" as defined by the CDNI "Logging Capability Object".

The CDNI Logging Capability Object is extended to contain additional properties that hold information on record format, such as fields that should be obfuscated by the dCDN. Note that the uCDN can further control field obfuscation when configuring a logging integration.

During provisioning process the dCDN may reject configuration if a selected record format is not available for a selected Log Integration Type.

6.1. FCI extension for Logging

This is a proposal of a Logging Capability object that extends the CDNI "FCI.Logging" object.

The following shows an example of Logging Capability object serialization, for a dCDN that supports the optional fields "hostname" and "cache-key", for the "oc_http_request_v1" record type. The "client-address" field is hashed.

In this example, the logging integration types that are supported are named "kafka" and "logstash"

```
    {
     "capabilities": [
       {
         "capability-type": "FCI.Logging",
         "capability-value": {
           "transport-types": [
             "kafka",
             "logstash"
           ],
           "record-type": "oc_http_request_v1",
           "fields": [
             "hostname",
             "cache-key"
           ],
           "hash-fields": [
             "client-address"
           ]
         },
         "footprints": [
            <footprint-objects>
         ]
       }
     ]
   }
```

6.2. Metadata Interface extension for Logging

This is a proposal of Logging Metadata and Transport Metadata objects that comply with the CDNI "Service Metadata" interface

6.2.1. Logging Configuration object

The following shows an example of Logging Configuration MI.Logging Metadata object serialization, for a logging integration that includes the optional field "hostname" in the log record.

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.Logging",
      "generic-metadata-value": {
        "include-fields": [
          "hostname"
        ]
      },
      "footprints": [
         <footprint-objects>
      ]
    }
  ]
}
```

6.2.2. Transport Configuration object

An initial set of logging transport types and their respective configuration objects should be defined. More types can be added in the future as needed. The following shows an example of Transport Configuration MI.LoggingTransport Metadata object serialization, for a "kafka" logging integration type.

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.LoggingTransport",
      "generic-metadata-value": {
        "type": [
          "kafka",
        ],
        "config":
          <kafka-integration-config-object>
        ]
      },
      "footprints": [
         <footprint-objects>
      ]
    }
  ]
}
```

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 [RFC7736]:

Payload Type	Specification
FCI.RequestRouterAddress	RFCthis
MI.FallbackAddress	RFCthis
MI.Logging	RFCthis
MI.LoggingTransport	RFCthis

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

7.1.1. CDNI FCI RequestRouterAddress Payload Type

Purpose: The purpose of this payload type is to distinguish RequestRouterAddress FCI objects (and any associated capability advertisement)

Interface: FCI

Encoding: see Section 2.1

7.1.2. CDNI MI FallbackAddress Payload Type

Purpose: The purpose of this payload type is to distinguish FallbackAddress MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 2.2

7.1.3. CDNI MI Logging Payload Type

Purpose: The purpose of this payload type is to distinguish Logging MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 6.2.1

7.1.4. CDNI MI LoggingTransport Payload Type

Purpose: The purpose of this payload type is to distinguish LoggingTransport MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 6.2.2

8. Security Considerations

TBD.

9. Acknowledgements

The authors would like to thank Kevin J. Ma for his guidance and support.

10. Contributors

The authors would like to thank all members of the SVA's Open Caching Working Group for their contribution in support of this document.

11. References

11.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://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, <https://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, <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>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6707>.

[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>.

[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>.

[RFC8007] Murray, R. and B. Niven-Jenkins, "Content Delivery Network Interconnection (CDNI) Control Interface / Triggers", RFC 8007, DOI 10.17487/RFC8007, December 2016, <https://www.rfc-editor.org/info/rfc8007>.

[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>.

11.2. Informative References

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

Authors' Addresses

Ori Finkelman
Qwilt
6, Ha'harash
Hod HaSharon 4524079
Israel

Phone: +972-72-2221647
Email: orif@qwilt.com

Sanjay Mishra
Verizon
13100 Columbia Pike
Silver Spring, MD 20904
USA

Email: sanjay.mishra@verizon.com