SVA Configuration Interface

IETF/CDNi Metadata Model Extensions

March 2022 (IETF 113)

streaming video alliance

Background

CDN Configuration Metadata Challenge

- **Problem Statement:** The need for an industry-standard API and configuration metadata model becomes increasingly important as content and service providers automate more of their operations, and as technologies such as open caching require coordination of content delivery configurations.

- **Motivational Drivers:** The SVA Membership (which includes Content Providers, CDNs, ISPs, and Open Caching technology vendors) agreed on the following drivers for defining a standard configuration interface:

    - Make integrations more efficient
    - Enable self-service configuration (DevOps)
    - Standardize CDN feature definitions
    - Improve governance & compliance
    - Create opportunities for differentiation

Wasn’t CDNI Metadata Sufficient As-Is?

Quick Answer: Almost, but not quite.

- The CDNi Metadata and Control Interfaces (RFCs 8006 & 8007) were designed for the limited scope use cases related interchanges between upstream CDNs and downstream CDNs.

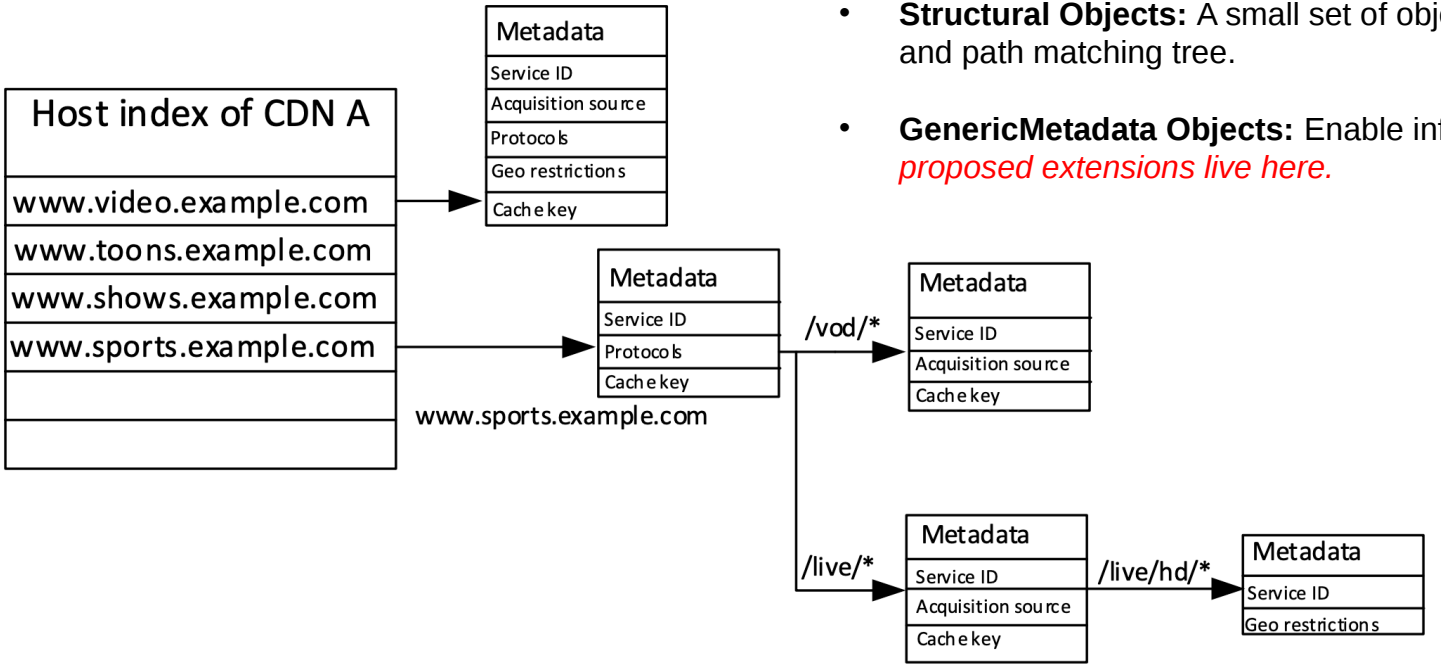
- As we look at the wider set of use cases involving Content Providers managing multiple CDNS, along with use cases in the Open Caching ecosystem, we see gaps.

What CDNI Metadata Provides	**Gaps - What’s Missing**
Simple CDN Metadata Object Model	Metadata Object Model meeting more complex requirements of CDN and Open Caching industries
Interfaces for retrieving metadata, and triggering metadata prepositioning, invalidation and purging	Simple push-style metadata publishing (POST)
Interfaces to check status or cancel trigger requests	Advanced configuration publishing capabilities required by Content Providers (publishing, versioning, deployment)

The SVA Configuration Interface initiative addresses these gaps!

streaming video alliance

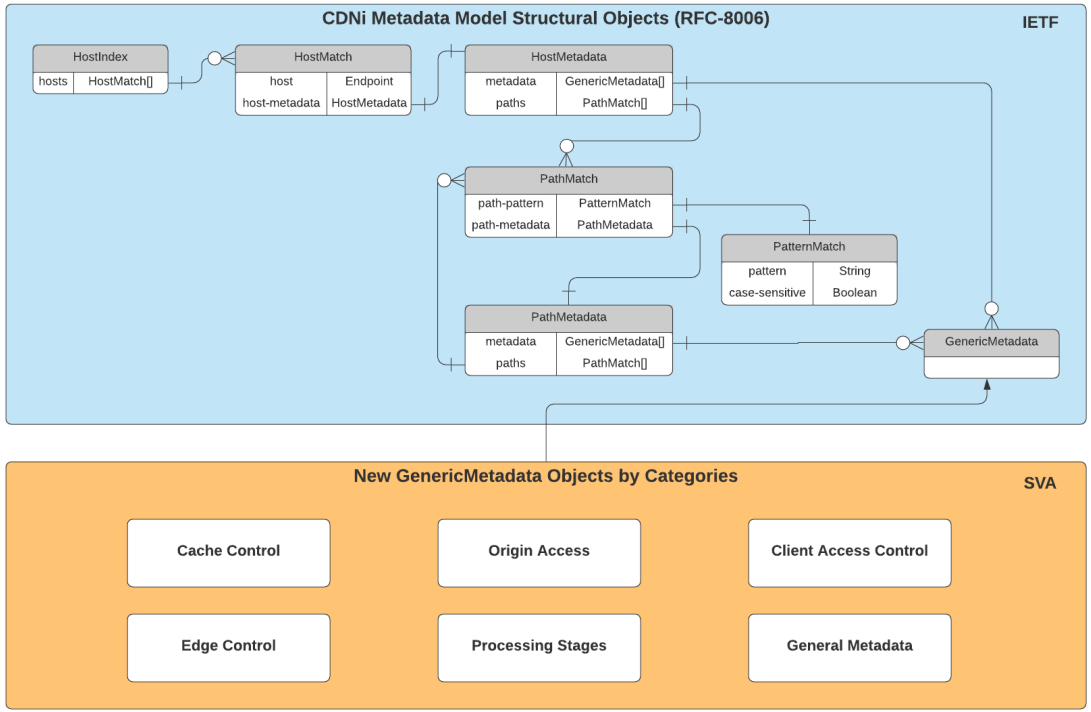
CDNI Metadata Model (RFC 8006)



Key Concepts:

- **Inheritance model**: Content Delivery Metadata (caching and access rules) defined at the host level can be overridden at the path level.
- **Structural Objects:** A small set of objects that define the host and path matching tree.
- **GenericMetadata Objects:** Enable infinite extensibility. *All our proposed extensions live here.*

CDNi Metadata Model Schema



streaming video alliance

Metadata Model Extensions

The following requirements and proposed extensions are documented in detail in the *SVA Configuration Interface Part 2* specification and are being submitted to the IETF as extensions to RFC-8006:

- Enhanced Source/Origin Definitions:
  - Origin Load Balancing, Failover
  - Origin Authentication Methods

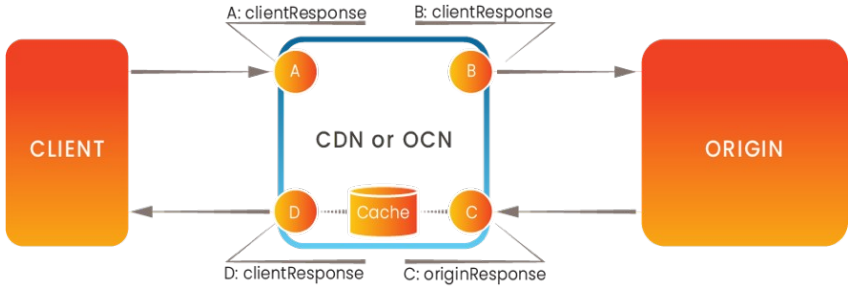
- Cache Control Policies & Computed Keys
- Dynamic CORS Headers
- Traffic Type Metadata
- Service ID Metadata
- SVA Open Caching Configuration Metadata
- Private Features for extensibility
- Processing Stages with an Expression Language(see next slide)

CDNi Metadata Extension: Processing Stages

Allows metadata rules to be applied conditionally at a specific stage in the pipeline, based on matching elements of HTTP requests & responses. A rich expression language is provided to specify matching rules and synthesis dynamic values.

Stage-specific processing enables:

- Application of metadata (such as cache policies)
- Request Transformations (Header modifications, URI re-writes)
- Response Transformations (Header modifications, status code overrides)
- Generating Synthetic Responses



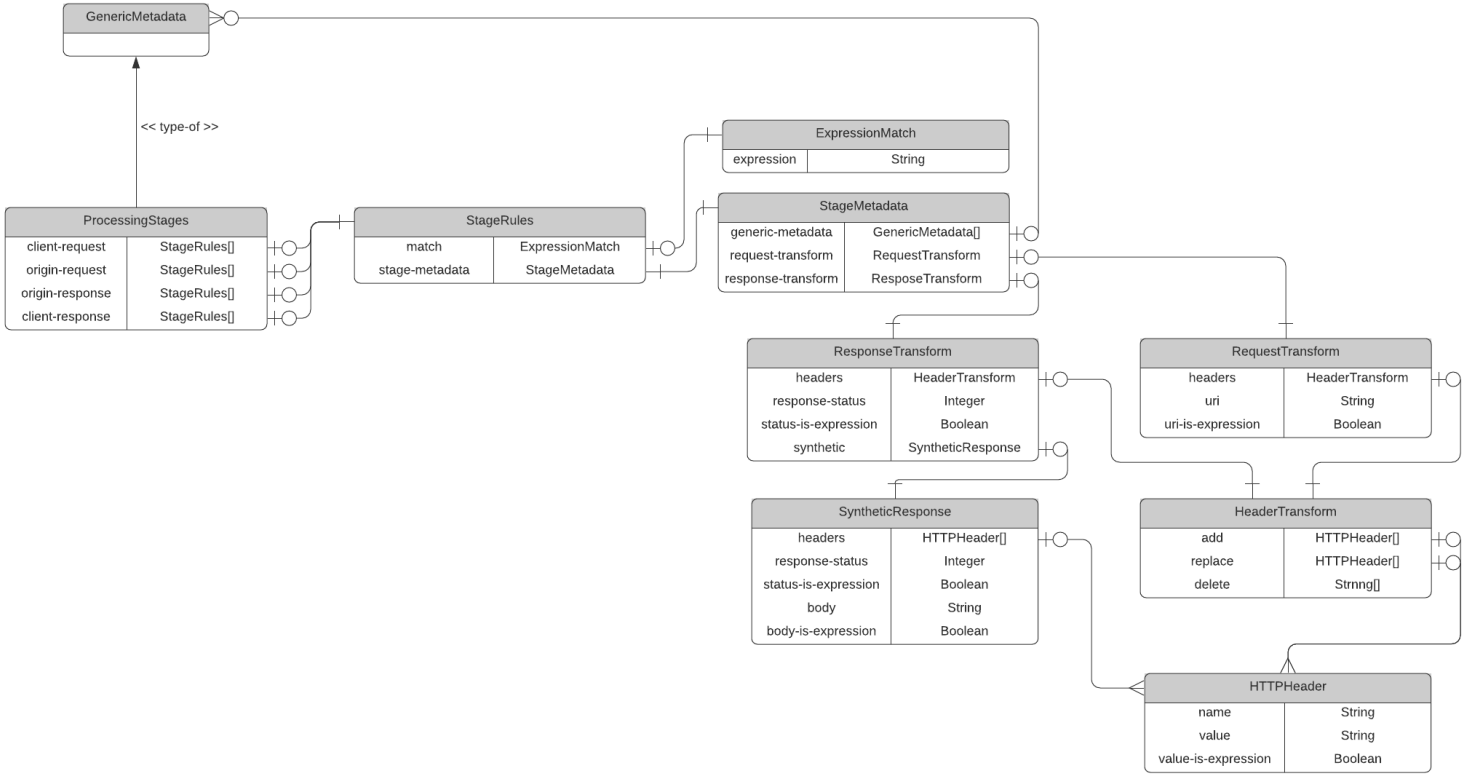
clientRequest - Rules run on the client request prior to further processing.

originRequest - Rules run prior to making a request to the origin.

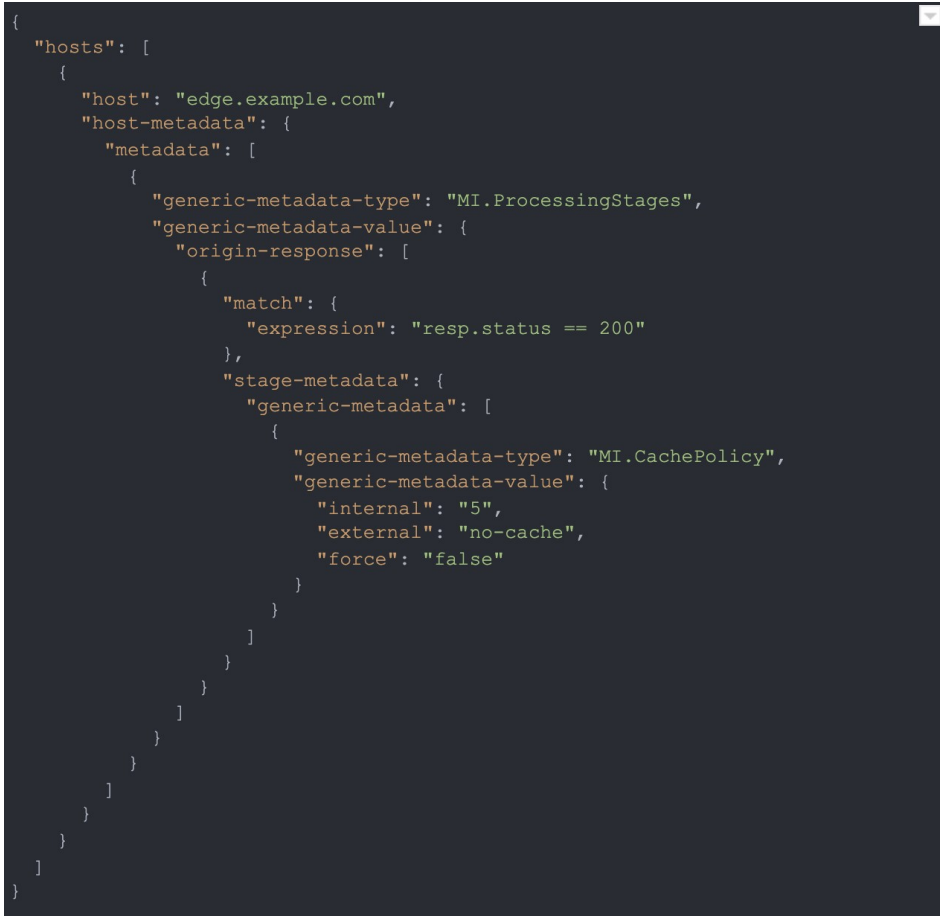
originResponse - Rules run after response is received from the origin and before being placed in cache.

clientResponse - Rules run prior to sending the response to the client. If response is from cache, rules are applied to the response retrieved from cache prior to sending to the client.

Stage Processing Object Model



Processing Stages Example



```
{
  "hosts": [
    {
      "host": "edge.example.com",
      "host-metadata": {
        "metadata": [
          {
            "generic-metadata-type": "MI.ProcessingStages",
            "generic-metadata-value": {
              "origin-response": [
                {
                  "match": {
                    "expression": "resp.status == 200"
                  },
                  "stage-metadata": {
                    "generic-metadata": [
                      {
                        "generic-metadata-type": "MI.CachePolicy",
                        "generic-metadata-value": {
                          "internal": "5",
                          "external": "no-cache",
                          "force": "false"
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
      }
    }
  ]
}
```

A complete example using the SVA `ProcessingStages` and `CachePolicy` extensions to the CDNI metadata model.

In this example, clients are directed to not cache content when there is a 200 response from the origin, with the CDN maintaining internal caches for 5 seconds to protect the origin from being overwhelmed.

Expression Language Examples

The CDNI Metadata Expression Language provides a syntax with a rich set of variables, operators, and built-in functions to facilitate use cases within the extended CDNi metadata model.

ExpressionMatch where the expression is true if the user-agent (glob) matches *Safari* and the referrer equals www.example.com.

```
{
  "expression": "req.h.user-agent *= '*Safari*'
      and req.h.referrer == 'www.example.com'"
}
```

Add a Set-Cookie header with a dynamically computed cookie value (concatenating user agent and host name).

```
{
    "response-transform":{
       "headers":{
          "add":[
             {
                "name":"Set-Cookie",
                "value":"$req.h.user-agent - $req.h.host",
                "value-is-expression":true
             }
          ]
       }
    }
}
```

Capabilities Interface

- The proposed CDNi Metadata Model extensions are optional, with dCDNs able to advertise their support via the Footprint & Capabilities Interface (FCI).
- Any extension that is embodied as a new GenericMetadata object can be advertised as supported via the CDNi standard FCI.Metadata object.
- Some proposed extensions entail many features, and it is quite possible that a dCDN may support some (but not all) of these features.
- To allow for more fine-grained advertisement of feature support, additional FCI objects will be defined containing feature flags that are specific to each extended GenericMetadata object.

Extending the Metadata Interface

In addition to extending the CDNi Metadata model, the SVA is also working on APIs that extend the interface:

- Extend the basic metadata retrieval interface defined in RFC-8006 with metadata publishing capabilities to allow a uCDN to publish and delete metadata on a dCDN.
- Add capabilities to publish and reference sets of named GenericMetadata Objects (extending the current HREF concept).
- Provide CDN configuration life cycle management capabilities such as publishing, versioning, staged deployments, profiles, and templates.
- Provide configuration for additional aspects of CDN operation such as provisioning of Certificates and configuration of log delivery endpoints.

Draft Status:

draft-goldstein-cdni-metadata-model-extensions-02

streaming video alliance

Changes from Revision 01

- As suggested on IETF 112, new GenericMetadata included in this draft have been categorized in different groups by functionality:
  - **Cache Control**: Metadata related to Cache properties management
    - MI.CachePolicy, MI.NegativeCachePolicy, MI.StaleCachePolicy, MI.CacheByPassPolicy, MI.ComputedCacheKey
  - **Origin Access**: metadata related to acquisition of the content
    - MI.SourceMetadataExtended, MI.SourceExtended, MI.LoadBalance, MI.HeaderAuth, MI.AWSv4Auth
  - **Client Access Control**: Currently it does not include any object but will come in the next version
  - **Edge Control**: GenericMetadata to inform processing of responses downstream (at the edge and in the user agent)
    - MI.CrossoriginPolicy, MI.AllowCompress, MI.TrafficType, MI.OcnSelection
  - **Processing Stages**: All GenericMetadata required for Processing Stages
    - MI.ProcessingStages, MI.StageRules, MI.ExpressionMatch, MI.StageMetadata,MI.RequestTransform, MI.ResponseTransform, MI.SyntheticResponse, MI.HeaderTransform, MI.HTTPHeader,
  - **General Metadata**: Uncategorized metadata:
    - Mi.ServiceIDs, MI.PrivateFeatureList, MI.RequestRouting

Changes from Revision 01

- Added examples for every GenericMetadata object
- Minor changes in Metadata Expression Language.
- More detailed description for ProcessingStages
- Removed MI.RequestedCapacityLimits. Being part of the definition for Capacity limits, this topic is related to draft-ryan-cdni-capacity-insights-extensions. Any configuration metadata related to that draft will not be included in this one onwards.
- Corrections based on previous versions questions and comments

Questions from Revision 01

draft v01	draft v02	Question
2.2	2.3.2	add cross reference for HeaderTransform
		Missing in v2. Set for v3
2.2	2.3.2	MI.AllowCompress: Without this metadata today, a dCDN is free to choose to compress? t feels more natural to make the default true or change the metadata to DisallowCompress? Otherwise, without an FCI advertisement of whether the dCDN supports the metadata, it's hard to know what its default behavior will be?
		Without it, there is no enforcement from the uCDN about dCDN behavior for compression. The purpouse of the functionality is to force compression in dCDN independently of how the uCDN content has been acquired. Agreed probably the name does not reflect precisely the functionality, and something like EdgeCompress or ForceCompress would be more suitable. In discussion in the SVA OCWG
2.3	2.1.1	I'm curious about the use case. The uCDN could just set the desired cache policy in the response, rather than setting the metadata. Is the primary use case where the uCDN does not want to set the cache policy in each response and wants a default in the metadata? Or is there something else? Is it to have a push interface (note: the triggers API provides a way to invalidate)?
		Cache Control directives condition how any cache system in the path between the uCDN origin and the user agent should behave. As the uCDN can only use one specific Cache-Control header, it has no flexibility to control the TTL of the objects in a dCDN cache. It would be the same for the dCDN cache and for an intermediate cache system. As the uCDN can have control over the objects the dCDN stores (as with the trigger interface mentioned) it can be of the uCDN interest to have a longer TTL for the dCDN Caches (so there is less traffic to the uCDN origins) but having a smaller TTL for the end users. Thus, the dCDN supports a higher impact on requests, protecting the uCDN origin. For that, CachePOlicy permits to define modifications on the Cache-Control directives sent by the uCDN origin. Apart of that, the force property permits to mandate the dCDN to use those values only in the case the uCDN origin does not set the Cache Control directives. (as a default value)

Questions from Revision 01

draft v01	draft v02	Question
2.5.1	2.3.1.1	I'm not sure what "List of valid URLs only scheme and host name" means. Is that to say the URLs only have a scheme and hostname (i.e., no uri path)? And is the port explicity excluded, or is it an RFC3986 "authority" (i.e., user, host or ip, and port https://datatracker.ietf.org/doc/html/rfc3986#section-3.2)? Perhaps "List of valid URLs (where the URLs contain only a scheme and authority)"? Note: In order to find the metadata, a request must already have matched a host in the HostMatch object, so depending on what is in the HostMatch, this could be redundant? Is this something that could be simplified?
		The description has been modified to clarify the reasons. The Origin header is a HTTP extension. Its value is a version of the Referer header in some specific requests, and used for Cross Origin requests. . Permitted values are schema://hostname[:port] We have not used the RFC linked as there is no such reference in the HTTP Fetch standard that defined the logic about CORS headers. https://fetch.spec.whatwg.org But seems to be the same concept. The value of the allow-lilst is not directly related to the HostMatch, because the value that a player or browser set for the Origin header is not the hostname where it is making the request (the HostMatch in CDNI prospective) but the “authority” of the video portal where the content has been requested in first time. As an example, the player access a video portal to videoportal.example.com, and when trying to play, the player is commanded to get a manifest file to http://hostmatch.example.com . In this case, the Host header in the request is hostmatch.example.com (the HostMatch used in the metadata configuration), but the Origin header is set to http://videoportal.example.com

Questions from Revision 01

draft v01	draft v02	Question
2.6	2.1.2	"it may be desirable to cache error responses at the uCDN for a short period of time to prevent an overwhelmed origin service from being flooded with requests"? Replace 's/uCDN/dCDN/' and 's/origin service/uCDN/' ? Could the uCDN specify the cache policies in the response?
		Removed that sentence. For the second questions: Same case as for MI.CachePolicy above. In fact, this could be considered a particular case of the MI.CachePolicy for errors. The uCDN can of course set the cache control directive for those errors. But it is typical the uCDN prefers to protect its origin server of a flood of requests when a non-desirable situation occur (an HTTP 500 error for example), but wanting the UA to try a faster revalidation. Other example, in the opposite way: A video fragment is not available yet in the uCDN Origin but published in a manifest (Live DASH). The UA requests a video fragment to the dCDN, and the dCDN requests it to the uCDN. When having an HTTP 404 error (because maybe the object is in the process of being available in the origin). If it sets a max-age of 10s in the response, the dCDN will not revalidate the object until 10 seconds has passed, so every UA trying to get the object will receive a 404 during 10 seconds. But maybe the resource became available after just 1 second of the first request. The first UA that requested the fragment will wait 10s to get it again, but the rest of UA trying to access the same fragment will get it right away.
2.7	2.1.4	Is the intent that this metadata would be pulled from the uCDN on every request and that the uCDN would made decisions based on the requestor: "CacheBypassPolicy only applies to the current request"? Though not explicity prohibited by the MI, it goes a bit beyond the original intent of the metadata cacheability.
		Corrected the description. No relationship with the UA current request. This metadata applies after is received in a configuration, for all the requests that arrived since that moment that match the MI.ExpressionMatch and onwards. For the same resource requested, the dCDN can have a cached copy for those requests that don´t match the condition, while requesting a new copy for those that match.
2.8.1	2.3.4.1	Combine the two Mandatory-to-Specify bullets?
		Corrected

Questions from Revision 01

draft v01	draft v02	Question
2.9.1	2.5.2.1	Do you envision a registry for the private features?
		By definition, the possible PrivateFeatures are implementation specific. As there could be a great variability on the possible properties we have not discussed the idea of having a registry. An uCDN configuring a HostMatch on multiple dCDNs should send a MI.PrivateFeatures metadata in the configuration only for those that support it, and the content particularized for each of those. As being a point to point configuration does not seem needed to have a registry.
2.12	2.5.3	I would expect the type to come from the redirection modes registry (https://www.iana.org/assignments/cdni-parameters/cdni-parameters.xhtml#capabilities-redirection-modes)?
		It is not exactly the same. FCI.RequestRouting and MI.RequestRouting are related to how the dCDN will make the Request Routing internally to direct the user to the suitable streamer. In the reference mentioned, the types HTTP-I or HTTP-R include the interative or recursive methods, while the internal request routing in a dCDN is independent of those techniques. The most typical use case is that a uCDN make a Traffic delegation using DNS-I or DNS-R, but wants to be sure that the dCDN is not going to do a internal HTTP redirection. Thus, sending MI.RequestRouting as “DNS” force the dCDN to use DNS RR internally and to avoid HTTP 302.
2.13	2.5.1	I'm not clear what the two tiers are. In the case of a static ID, would the existing ccid metadata (https://datatracker.ietf.org/doc/html/rfc8006#section-4.2.8) suffice?
		Description expanded to clarify that point. This extension permits to identify services AND properties in the service. So in the case of a static ID without any property identification, Grouping could be sufficient.

Questions from Revision 01

draft v01	draft v02	Question
2.14.1	2.2.1.1	For ease of understanding, should probably just duplicate the acquistion-auth, endpoints, and protocol properties
		Done
2.14.2	2.2.1.2	- section 2.14.2: Is the intent to create a registry for load balancing algorithms?
		In discussion
2.16	2.3.3	"vod" and "live" are fairly video specific; could this be more generically categorized? is the intent to convey information about source acquisition (e.g., you may have to wait for live content to become available vs you may be able to prefetch vod), delivery pacing or rate limiting, both, neither, or other?
		The intention is related to the content delivery, not the content acquisition. In HTTP Adaptative Streaming protocols, the behaviour of the end users consumers (aka 'players') is quite different and it affects the content delivery resources depending on this type. While a live content is mainly RAM consuming, a VoD content is storage consuming. CDNs typically can apply different optimizations, including using different hardware elements depending on the type of the content. Thus, it is very important that in video streaming processes the dCDN knows that information in advance so it can apply those strategies, so the QoE will be superior.

Conclusion

Based on the contents of this presentation, Can the CDNI working group accept version 02 of this document as a Working Group Draft?