Workgroup: Network Working Group
Internet-Draft:
draft-siloniz-cdni-edge-control-metadata-00
Updates: 8006 (if approved)
Published: 13 March 2023
Intended Status: Standards Track
Expires: 14 September 2023
Authors: A. Siloniz   G. Goldstein
         Telefonica   Lumen Technologies

# CDNI Edge Control Metadata

## Abstract

This specification defines MI configuration metadata objects to extend RFC 8006 related to controlling edge access to resources via CDNs and Open Caching systems. Configuring Cross-Origin Resource Sharing (CORS) access rules and the dynamic generation of CORS headers is a key feature of typical configurations, as are the ability to define response body compression rules, client connection timeouts, and traffic type hints for optimized caching.

## Status of This Memo

## Copyright Notice

## Table of Contents

## 1. Introduction

CDNs typically require a set of configuration metadata to inform processing of responses downstream (at the edge and in the user agent). This section specifies GenericMetadata objects to meet those requirements, defining edge processing rules such as CORS handling, response compressions, and client connection failures.

## 2. Requirements

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].

## 3. MI.CrossoriginPolicy

Delegation of traffic between a CDN over an open caching node based on HTTP redirection does change the domain name in the client requests. This represents a cross-origin request that must be managed appropriately using Cross-Origin Resource Sharing (CORS) headers in the responses.

The dynamic generation of CORS headers is typical in modern HTTP request processing and avoids CORS validation forwarded to the CDN origin servers, particularly with the preflight OPTIONS requests. The CDNI metadata model requires extensions to specify how a CDN or open caching node should generate and evaluate these headers.

Required capabilities:

Set a default value for CORS response headers independent of the origin request header value.

1. Set a default value for CORS response headers independent of the origin request header value.

2. Match the origin request header with a list of valid values, including PatternMatch, to return or not return the CORS response headers.

3. Set a list of custom headers that can be exposed to the client (expose headers).

4. Support for preflight requests using the OPTIONS method, including custom header validation, expose headers, and methods.

5. Support for credentials validation within CORS.

Simple CORS requests are those where both HTTP method and headers in the request are included in the safe list defined by the World Wide Web Consortium [W3C]. The user agent (UA) request can include an origin header set to the URL domain of the webpage that the player runs. Depending on the metadata configuration, the logic to apply by the open caching node (OCN) is:

Validation of the origin header - Metadata can include a list of valid domains to validate the request origin header. If it does not match, the CORS header must not be included in the response.

1. Validation of the origin header - Metadata can include a list of valid domains to validate the request origin header. If it does not match, the CORS header must not be included in the response.

2. WIldcard usage - Depending on the configuration, the resultant CORS header to include in the response will be the same as the request origin header, or a wildcard.

3. If no validation of request is included in the origin header, set a default value for CORS response headers independent of the origin request header value.

When a UA makes a request that includes a method or headers that are not included in the safe-list, the client will make a CORS preflight request using the OPTIONS method to the resource including the origin header. If CORS is enabled and the requests passes the origin validation, the OCN SHOULD respond with the set of headers that indicate what is permitted for that resource, including one or more of the following:

Allowed methods

1. Allowed methods

2. Allowed credentials

3. Allowed request headers

4. Max age that the OPTIONS request is valid

5. Headers that can be exposed to the client

When an uCDN configures any of those advanced parameters it is requesting the dCDN to generate synthetic responses to OPTIONS requests. Thus, no conditional request is performed to the uCDN origin. uCDN should configure these values taking that into account. If some of the advanced parameters are empty, the dCDN would not send the corresponding header into the UA OPTIONS request.

In case the uCDN only configures the MI.AccessControlAllowOrigin subobject, the dCDN will not generate synthetic responses to OPTIONS requests, thus it will validate we the uCDN every OPTIONS request to obtain the response.

CrossoriginPolicy is a GenericMetadata object that allows for the specification of dynamically generated CORS headers.

Property: allow-origin

- -Description: Validation of simple CORS requests.
- -Type: Object MI.AccessControlAllowOrigin
- -Mandatory-to-Specify: Yes

Property: expose-headers

- -Description: A list of values the OCN will include in the Access-Control-Expose-Headers response header to a preflight request.
- -Type: Array of strings
- -Mandatory-to-Specify: No

Property: allow-methods

- Description: A list of values the OCN will include in the Access-Control-Allow-Methods response header to a preflight request.

- Type: Array of strings

- Mandatory-to-Specify: No

Property: allow-headers

- Description: A list of values the OCN will include in the Access-Control-Allow-Headers response header to a preflight request.

- Type: Array of strings

- Mandatory-to-Specify: No

Property: allow-credentials

- Description: The value the OCN will include in the Access-Control-Allow-Credentials response header to a preflight request.

- Type: Boolean

- Mandatory-to-Specify: No

Property: max-age

- Description: The value the OCN will include in the Access-Control-Max-Age response header to a preflight request.

- Type: Integer

- Mandatory-to-Specify: No

Property: no-origin-response-headers

- Description: In the case of a request that has no Origin field, return this set of headers with the response.

- Type: Array of MI.HTTPHeader

- Mandatory-to-Specify: No

Property: apply-to-all-methods

-Description: By default the CORS configuration refers to OPTIONS requests. Setting this flag to true applies the entire CORS configuration to the other methods as well

-Type: Bollean

-Default: false

-Mandatory-to-Specify: No

## 3.1. MI.AccessControlAllowOrigin

The MI.AccessControlAllowOrigin object has the following properties:

Property: allow-list

-Description: List of valid URLs that will be used to match the request origin header. 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]

-Type: Array of PatternMatch objects

-Mandatory-to-Specify: Yes

Property: wildcard-return

-Description: If "True", the OCN will include a wildcard (*) in the Access-Control-Allow-Origin response header. If "False", the OCN will reflect the request origin header in the Access-Control-Allow-Origin response header.

-Type: Boolean

-Mandatory-to-Specify: Yes

The examples below demonstrate how to configure response headers dynamically for CORS validation.

Example 1: A simple CORS validation configuration:

```
{
  "generic-metadata-type": "MI.CrossoriginPolicy",
  "generic-metadata-value": {
    "allow-origin": {
      "allow-list": [
        {
          "pattern": "*"
        }
      ],
      "wildcard-return": true
    }
  }
}
```

Example 2: Validation of a preflight request when some of the headers included in the subsequent object request are not included in the CORS specification safelist:

```
{
  "generic-metadata-type": "MI.CrossoriginPolicy",
  "generic-metadata-value": {
    "allow-origin": {
      "allow-list": [
        {
          "pattern": "*://sourcepage.example.com"
        },
        "wildcard-return": false
      },
      "allow-methods": [ "GET", "POST" ],
      "allow-credentials": true,
      "allow-headers": [ "X-PINGOTHER", "Content-Type" ],
      "expose-headers": [ "X-User", "Authorization" ],
      "max-age": 3600
    }
  }
}
```

## 4. MI.AllowCompress

Downstream CDNs often have the ability to compress HTTP response bodies in cases where the client has declared that it can accept compressed responses (via an Accept-Encoding header), but the source/origin has returned an uncompressed response.

The specific compression algorithm used by the dCDN is negotiated by the client's Accept-Encoding header according to [RFC9110] (including q= preferences) and the compression capabilities available on the dCDN.

In addition, HeaderTransform allows the uCDN to normalize, or modify, the Accept-Encoding header to allow for fine-grain control over the selection of the compression algorithm (e.g., gzip, compress, deflate, br, etc.).

AllowCompress is a new GenericMetadata object that allows the dCDN to compress content before sending to the client.

Property: allow-compress

-Description: If set to "True", then the dCDN will try to compress the response to the client based on the Accept-Encoding request header.

-Type: Boolean

-Values: True or False

-Mandatory-to-Specify: No. The default is "False".

The following examples illustrate the use of MI.AllowCompress in the context of the Processing Stages model that allowed for metadata to be applied conditionally based on evaluation of HTTP request headers. See the Processing Stages Metadata Specification and Metadata Model Expression Language (MEL) Specification.

Example 1: An MI.AllowCompress that allows manifests (*.m3u8) to be compressed by the dCDN:

```
{
  "match": {
    "expression": "req.h.uri *= '*.m3u8'"
  },
  "stage-metadata": {
    "generic-metadata": [
      {
        "generic-metadata-type": "MI.AllowCompress",
        "generic-metadata-value": {
          "allow-compress": "true"
        }
      }
    ]
  }
}
```

Example 2: An MI.AllowCompress that allows manifests (*.m3u8) to be compressed by the dCDN but normalizing the client’s Accept-Encoding header:

```
{
  "match": {
    "expression": "req.h.accept-encoding *= '*gzip*'"
  },
  "stage-metadata": {
    "generic-metadata": [
      {
        "generic-metadata-type": "MI.AllowCompress",
        "generic-metadata-value": {
          "allow-compress": "true"
        }
      }
    ]
  }
}
```

## 5. MI.ClientConnectionControl

Configuration metadata is required to define how connections against a client are maintained by a dCDN. Since the clients are typically owned/operated by a uCDN, giving this control to a uCDN allows it to accommodate device specific constraints and performance optimizations. A dCDN can also benefit from this configuration metadata to meet its security and resource consumption requirements.

ClientConnectionControl is a new GenericMetadata object that specifies how a dCDN manages its connections to clients/players.

Property: connection-keep-alive-time-ms

-Description: Specifies the time in milliseconds to keep an idle connection open.

-Type: Integer

-Mandatory-to-Specify: No. When not specified, a default value selected by the dCDN will be used.

Following example shows how a connection setup and keep alive timeout can be set for client connections against a dCDN:

```
{
  "generic-metadata-type": "MI.ClientConnectionControl",
  "generic-metadata-value": {
    "connection-keep-alive-time-ms": 3
  }
}
```

## 6. MI.TrafficType

Content delivery networks often apply different infrastructure, network routes, and internal metadata for different types of traffic. Delivery of large static objects (such as software downloads), may, for example, use different edge servers and network routes than video stream delivery. In an HTTP adaptive bitrate video service, every video title corresponds to a set of video files and descriptors according to different video protocols, and this is independent of the type of service (Video-on-Demand, Live, Catch-up, etc.).

The way the video service is consumed by the user agents can vary. For instance, a segment that belongs to a Video-on-Demand (VoD) title can be requested for every moment the content is available for the user agents to consume, while a segment of live content will be only requested as long as the time-shift duration is configured for that service. Knowing those differences, a CDN or OCN provider can implement specific strategies that will maximize performance and thereby provide more available capacity to the upstream provider. It should be noted that the dCDNs handling of the traffic types is implementation-specific and not prescribed here.

MI.TrafficType metadata defines a set of descriptors that characterize either the type or usage of the traffic, enabling CDNs and OCNs to apply any internal configuration rules without exposing an unnecessary number of internal details. Note that the interpretation of these traffic types and application of rules such as rate limiting or delivery pacing are implementation specific.

Property: traffic-type

-Description: A literal that defines the traffic type. uCDN will use the literal that is most representative of the traffic being delegated.

-Type: Enumeration [vod, live, object-download] encoded as lowercase string

-Mandatory-to-Specify: Yes

Property: hints

-Description: Other traffic characteristics that the uCDN can indicate to the dCDN as suggestions for service optimization. Accepts free-form unconstrained values.

-Type: Array of strings

-Mandatory-to-Specify: No

A TrafficType definition example for HostMetadata:

```
{
  "generic-metadata-type": "MI.TrafficType",
  "generic-metadata-value": {
    "traffic-type": "vod",
    "hints": [ "low-latency", "catch-up" ]
  }
}
```

## 7. Conclusion

The specification has extended the basic CDNI configuration metadata objects defined in [RFC8006], and can be extended in the future with additional Edge Control Metadata object definitions.

## 8. Security Considerations

The FCI and MI objects defined in the present document are transferred via the interfaces defined in CDNI [RFC8006]. [RFC8006] describes how to secure these interfaces, protecting the integrity, confidentiality and ensuring the authenticity of the dCDN and uCDN. The security provide by [RFC8006] should therefore address the above security concerns.

## 9. IANA Considerations

### 9.1. CDNI Payload Types

TBD.

## 10. Acknowledgements

The authors would like to express their gratitude to the members of the Streaming Video Technology Alliance [SVTA] Open Caching Working Group for their guidance / contribution / reviews ...)

Particulary the following people contribute in one or other way to the content of this draft:

Guillaume Bichot - Broadpeak

Christoph Neumann - Broadpeak

Pankaj Chaudhari - Disney Streaming Services

Rajeev RK - picoNETS

Yoav Gressel - Qwilt

Arnon Warshavsky - QWilt

Shmuel Asafi . Qwilt

Nir Sopher - Qwilt

Arnon Warshavsky - Qwilt

Ben Rosenblum - Vecima

## 11. References

### 11.1. Normative References

**[RFC9110]** Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

**[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>.

**[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>.

### 11.2. Informative References

**[W3C]** "Cross-Origin Resource Sharing", <https://www.w3.org/TR/2020/SPSD-cors-20200602/>.

**[SVTA]** "Streaming Video Technology Alliance Home Page", <https://www.svta.org>.

## Authors' Addresses

Alfonso Siloniz
Telefonica
Spain

Email: alfonsosiloniz@gmail.com

Glenn Goldstein
Lumen Technologies
United States of America

Email: glenng1215@gmail.com