Content Delivery Networks Interconnection
Internet-Draft
Intended status: Standards Track
Expires: 31 July 2026

A. Siloniz
Telefonica
G. Goldstein
Lumen Technologies

27 January 2026

CDNI Edge Control Metadata
draft-ietf-cdni-edge-control-metadata-08

Abstract

This specification defines configuration metadata objects related to controlling edge access to resources via content delivery networks (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 and client connection timeouts.

Status of This Memo

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

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

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

This Internet-Draft will expire on 31 July 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction

CDNs typically require a set of configuration metadata to provide directives for processing responses downstream (at the edge and in the user agent). This document specifies GenericMetadata objects to meet those requirements, defining edge processing rules such as Cross-Origin Resource Sharing (CORS) handling, response compression, 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

3.1. Overview

When an application of a Content Provider (CP) requests content that is being distributed by a CDN, it generally makes an initial request to a CP's URL using a Fully Qualified Domain Name (FQDN) in the CP's realm. Depending on the CDN architecture used, the response can be an HTTP redirection to a different FQDN, typically one that belongs to the CDN realm. This situation is known as a shared cross-origin response, but many User Agents apply same-origin restrictions to these network requests.

Cross-origin resource sharing (CORS) is a protocol that allows a HTTP client application to access restricted resources from a server on a domain different from the domain that served the original request. It is defined by the Web Hypertext Application Technology Working Group (WHATWG) Fetch Living Standard [WHATWG-FETCH].

CORS is based on HTTP request and response headers, which must be managed appropriately.

For clarification, it is important to note that the concept of an Origin in the CDN context is different from that in the CORS context. The former indicates the source from where the CDN acquires content, while the latter is related to the source from where the User Agent downloaded the instructions to request the content, also known as the Referer.

To permit shared cross-origin network requests, CORS protocol defines a set of request and response headers that MUST be present to permit access to these restricted resources. Two situations can occur when one or more CDNs are involved in processing these requests:

* CDNs forward the User-Agent (UA) request headers to the CP servers, and they forward the response headers generated back to the UA

* CDNs can include logic to dynamically generate the required response headers based on the UA request without contacting the CP servers

The dynamic generation of CORS headers is typical in modern HTTP request processing and avoids forwarding CORS headers to the uCDN, thereby reducing the load between dCDN and uCDN when the content to be delivered is already cached in the dCDN.

Also consider that there are two types of requests involved in the CORS protocol as defined in Section 3.2.2 of [WHATWG-FETCH]:

* A CORS request is any HTTP request that includes an Origin header. The Origin header is a version of the Referer header that does not reveal a path, as defined in Section 3.1 of [WHATWG-FETCH]

* A CORS preflight request is a CORS request to check whether the CORS protocol is understood by the delivery server, and to ensure that it includes valid CORS response headers. It uses the OPTIONS method along with some other specific request headers to indicate which method and headers will be used in a future CORS request to the same resource.

The CDN Interconnection (CDNI) metadata model requires extensions to permit a uCDN to declare how a dCDN MUST evaluate and dynamically generate the necessary CORS response headers, avoiding the forwarding of CORS requests and/or preflight requests to the uCDN.

Required capabilities:

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

* Match the value of the Origin request header with a list of valid values. If it is successful, the dCDN will inject the appropriate CORS response headers.

* Set a list of custom response headers that are allowed to be exposed to the client using CORS response headers.

* Dynamically generate CORS response headers to CORS preflight requests including custom header validation, expose headers, and methods.

* Support credentials validation within CORS.

Depending on the metadata configuration, the dCDN MUST apply the following logic:

* Validation of the Origin request header - Metadata can include a list of valid domains to validate the Origin request header. If it does not match, the CORS response headers MUST NOT be included in the dCDN response. UA with same origin restrictions will prevent access to the resource.

* Wildcard usage - If the Origin request header is valid, depending on the configuration, the value of the CORS response header to include in the response will be the same as the Origin request header, or a wildcard.

* Possibility to define default values for CORS response headers independent of the value of the Origin request header.

* If the UA is expected to use CORS preflight requests, uCDN can also configure the dCDN to dynamically generate synthetic responses to OPTIONS requests.

When an uCDN configures one or more of the expose-headers, allow-methods, allow-headers, allow-credentials and max-age properties the dCDN MUST generate synthetic responses to any CORS preflight request without contacting the uCDN servers. In this case the dCDN will add the corresponding CORS response headers to every non-empty parameter.

If none of those parameters are set in the configuration, the dCDN MUST forward every CORS preflight request (OPTIONS method) to the uCDN servers and process their responses before responding to the UAs.

3.2. MI.CrossoriginPolicy

MI.CrossoriginPolicy is a GenericMetadata object that allows configuring dynamically generated CORS headers.

Property: allow-origin

* Description: Validation of simple CORS requests.

* Type: Object MI.AccessControlAllowOrigin (Section 3.2.1)

* Mandatory-to-Specify: Yes

Property: expose-headers

* Description: A list of header names the dCDN will include in the Access-Control-Expose-Headers response header of a CORS preflight request.

* Type: Array of Field Names as defined in Section 5.1 [RFC9110]

* Mandatory-to-Specify: No. If not specified, the default behavior is not to add the header in the response

Property: allow-methods

* Description: A list of request method tokens the dCDN will include in the Access-Control-Allow-Methods response header to a CORS preflight request.

* Type: Array of method tokens as defined in Section 9.1 [RFC9110]

* Mandatory-to-Specify: No. If not specified, the default behavior is not to add the header in the response

Property: allow-headers

* Description: A list of header names the dCDN will include in the Access-Control-Allow-Headers response header to a CORS preflight request.

* Type: Array of Field Names as defined in Section 5.1 [RFC9110]

* Mandatory-to-Specify: No. If not specified, the default behavior is not to add the header in the response

Property: allow-credentials

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

* Type: Boolean

* Mandatory-to-Specify: No. If not specified, the default behavior is not to add the header in the response

Property: max-age

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

* Type: Integer

* Mandatory-to-Specify: No. If not specified, the default behavior is not to add the header in the response

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. Content providers need to deal with devices or applications that, even not sending the Origin header in the request, are expecting to receive the Access-Control-* headers in the response.

* Type: Array of MI.AccessControlHeaders objects.

* Mandatory-to-Specify: No. If not specified, the default behavior is not to add any CORS response headers

Property: preflight-only

* Description: If this flag is set to "true" the dCDN will only generate synthetic responses to OPTIONS requests with proper CORS response headers

* Type: Boolean

* Mandatory-to-Specify: No. The default is "false", so CORS response headers logic will be injected for all HTTP methods

3.2.1. MI.AccessControlAllowOrigin

The MI.AccessControlAllowOrigin object has the following properties:

Property: allow-list

* Description: List of valid expressions that will be used to match the request Origin header. The Origin header is an HTTP extension. Its value is a variation of the Referer header that does not reveal a path in some specific requests and is used for cross-origin requests. Permitted values for the Origin header are of the form schema://host[:port] as defined in Sections 3.1, 3.2.2, and 3.2.3 of [RFC3986]

* Type: Array of Strings, where each String is a pattern as defined in Section 4.1.5 [RFC8006]

* Mandatory-to-Specify: Yes

Property: wildcard-return

* Description: If set to "true", the dCDN will include a wildcard (*) in the Access-Control-Allow-Origin response header. If "false", the dCDN MUST reflect the value of the Origin request header in the Access-Control-Allow-Origin response header.

* Type: Boolean

* Mandatory-to-Specify: Yes

3.2.2. MI.AccessControlHeaders

The MI.AccessControlAllowHeaders object has the following properties:

Property: name

* Description: The name of the HTTP header.

* Type: String

* Mandatory-to-Specify: Yes

Property: value

* Description: The new value of the named HTTP header.

* Type: String. A static string .

* Mandatory-to-Specify: Yes

3.3. Examples

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

The following is an example of a CORS validation configuration that does not include CORS preflight requests:

```
{
  "generic-metadata-type": "MI.CrossoriginPolicy",
  "generic-metadata-value": {
    "allow-origin": {
      "allow-list": [
        {
          "pattern": "https://sourcepage.example.com"
        }
      ],
      "wildcard-return": true
    }
  }
}
```

Figure 1

The following is an example of configuration to the dCDN to generate synthetic responses to CORS requests and CORS preflight requests:

```
{
  "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,
    "no-origin-response-headers": [
        {
          "name": "Access-Control-Allow-Origin",
          "value": "https://sourcepage.example.com"
        }
      ],
    "preflight-only": false
  }
}
```

Figure 2

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 Section 12.5.3 [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.).

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

Property: allow-compress

* Description: If set to "true", the dCDN SHOULD try to compress the response to the client based on the Accept-Encoding request header.

* Type: Boolean.

* Mandatory-to-Specify: No. The default is "false".

4.1. Examples

The following is an example of the usage of MI.AllowCompress:

```
{
    "generic-metadata-type": "MI.AllowCompress",
    "generic-metadata-value": {
        "allow-compress": true
    }
}
```

Figure 3

5. MI.ClientConnectionControl

Configuration metadata is required to define how connections to a client are maintained by a dCDN. In some use cases, like video streaming or other critical object delivery, UA applications' connection to the cache server must be managed to ensure the best possible user experience. This metadata allows a uCDN to accommodate device-specific constraints and performance optimization. A dCDN can also benefit from this configuration metadata to meet its security and resource consumption requirements.

MI.ClientConnectionControl is a new GenericMetadata object that specifies how a dCDN SHOULD manage its connections to UAs.

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.

5.1. Examples

The following example shows how a connection setup and a keep alive timeout of 3 seconds can be set for client connections to a dCDN:

```
{
  "generic-metadata-type": "MI.ClientConnectionControl",
  "generic-metadata-value": {
    "connection-keep-alive-time-ms": 3000
  }
}
```

Figure 4

6. Security Considerations

The FCI and MI objects defined in this document are transferred via the interfaces defined in CDNI [RFC8006] which describes how to secure these interfaces by protecting integrity and confidentiality while ensuring the authenticity of the dCDN and uCDN.

7. IANA Considerations

7.1. CDNI Payload Types

This document requests the registration of the following entries under the "CDNI Payload Types" registry hosted by IANA:

Payload Type	Specification
MI.CrossoriginPolicy	RFCthis
MI.AccessControlAllowOrigin	RFCthis
MI.AllowCompress	RFCthis
MI.ClientConnectionControl	RFCthis
MI.AccessControlHeaders	RFCthis

Table 1: CDNI Payload Types

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

7.1.1. CDNI MI CrossoriginPolicy Payload Type

Purpose: The purpose of this Payload Type is to distinguish CrossoriginPolicy MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: See Section 3

7.1.2. CDNI MI AccessControlAllowOrigin Payload Type

Purpose: The purpose of this Payload Type is to distinguish AccessControlAllowOrigin MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: See Section 3.2.1

7.1.3. CDNI MI AllowCompress Payload Type

Purpose: The purpose of this Payload Type is to distinguish AllowCompress MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: See Section 4

7.1.4. CDNI MI ClientConnectionControl Payload Type

Purpose: The purpose of this Payload Type is to distinguish ClientConnectionControl MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: See Section 5

7.1.5. CDNI MI AccessControlHeaders Payload Type

Purpose: The purpose of this Payload Type is to distinguish AccessControlHeaders MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: See Section 5

8. 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 contributions and guidance.

Particularly the following people contribute in vairous ways to the content of this draft:

* Guillaume Bichot (Broadpeak)
* Christoph Neumann (Broadpeak)
* Chris Lemmons (Comcast)
* Pankaj Chaudhari (Disney Streaming)
* Robert Colantuoni (Disney Streaming)

* Will Power (Lumen)

* Rajeev RK (picoNETS)

* Shmuel Asafi (Qwilt)

* Yoav Gressel (Qwilt)

* Nir Sopher (Qwilt)

* Arnon Warshavsky (Qwilt)

* Eric Klein (Sirius XM)

* Francisco Cano Hila (Telefonica)

* Ben Rosenblum (Vecima)

9. 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, <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>.

[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>.

[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>.

10. Informative References

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

[WHATWG-FETCH]
Web Hypertext Application Technology Working Group (WHATWG), "Fetch Standard", <https://fetch.spec.whatwg.org>.

Authors' Addresses

Alfonso Siloniz
Telefonica
Spain
Email: alfonsosiloniz@gmail.com

Glenn Goldstein
Lumen Technologies
United States of America
Email: glenng1215@gmail.com