CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2016

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
Alcatel-Lucent
E. Dijk
Philips Research

July 3, 2015

Guidelines for HTTP-CoAP Mapping Implementations
draft-ietf-core-http-mapping-07

Abstract

This document provides reference information for implementing a proxy that performs translation between the HTTP protocol and the CoAP protocol, focusing on the reverse proxy case. It describes how a HTTP request is mapped to a CoAP request and how a CoAP response is mapped back to a HTTP response. Furthermore, it defines a template for URI mapping and provides a set of guidelines for HTTP to CoAP protocol translation and related proxy implementations.

Table of Contents

1. Introduction

CoAP [RFC7252] has been designed with the twofold aim to be an application protocol specialized for constrained environments and to be easily used in REST architectures such as the Web. The latter goal has led to define CoAP to easily interoperate with HTTP [RFC7230] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP and the HTTP-to-CoAP cross-protocol mapping process. However, implementing such a cross-protocol proxy can be complex, and many details regarding its internal procedures and design choices require further elaboration. Therefore, a first goal of this document is to provide more detailed information to proxy designers and implementers, to help build proxies that correctly inter-work with existing CoAP and HTTP implementations.

The second goal of this informational document is to define a consistent set of guidelines that a HTTP-to-CoAP proxy implementation MAY adhere to. The main reason for adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability. (For example, a proxy conforming to these guidelines made by vendor A can be easily replaced by a proxy from vendor B that also conforms to the guidelines.)

This document is organized as follows:

- Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- Section 3 introduces the reverse HTTP-CoAP proxy;
- Section 4 lists use cases in which HTTP clients need to contact CoAP servers;

- Section 5 introduces a default HTTP-to-CoAP URI mapping syntax;
- Section 6 describes how to map HTTP media types to CoAP content formats and vice versa;
- Section 7 describes how to map CoAP responses to HTTP responses;
- Section 8 describes additional mapping guidelines related to caching, congestion, timeouts and CoAP blockwise [I-D.ietf-core-block] transfers;
- Section 10 discusses possible security impact of HTTP-CoAP protocol mapping.

2. Terminology

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

HC Proxy: a proxy performing a cross-protocol mapping, in the context of this document a HTTP-CoAP mapping. A Cross-Protocol Proxy can behave as a Forward Proxy, Reverse Proxy or Interception Proxy. In this document we focus on the Reverse Proxy case.

Forward Proxy: a message forwarding agent that is selected by the client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and to attempt to satisfy those requests via translation to the protocol indicated by the absolute URI. The user decides (is willing to) use the proxy as the forwarding/de-referencing agent for a predefined subset of the URI space. In [RFC7230] this is called a Proxy. [RFC7252] defines Forward-Proxy similarly.

Reverse Proxy: as in [RFC7230], a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. A Reverse HC Proxy behaves as an origin (HTTP) server on its connection towards the (HTTP) client and as a (CoAP) client on its connection towards the (CoAP) origin server. The (HTTP) client uses the "origin-form" (Section 5.3.1 of [RFC7230]) as a request-target URI.

Interception Proxy [RFC3040]: a proxy that receives inbound traffic flows through the process of traffic redirection; transparent to the client.

Placement terms: a Server-Side proxy is placed in the same network domain as the server; conversely a Client-Side proxy is placed in the same network domain as the client. In any other case, the proxy is said to be External.

Note that a Reverse Proxy appears to a client as an origin server while a Forward Proxy does not, so, when communicating with a Reverse Proxy a client may be unaware it is communicating with a proxy at all.

3. HTTP-CoAP Reverse Proxy

A Reverse HTTP-CoAP Proxy (HC proxy) is accessed by clients only supporting HTTP, and handles their HTTP requests by mapping these to CoAP requests, which are forwarded to CoAP servers; mapping back received CoAP responses to HTTP responses. This mechanism is transparent to the client, which may assume that it is communicating with the intended target HTTP server. In other words, the client accesses the proxy as an origin server using the "origin-form" (Section 5.3.1 of [RFC7230]) as a request target.

See Figure 1 for an example deployment scenario. Here an HC Proxy is placed server-side, at the boundary of the Constrained Network domain, to avoid any HTTP traffic on the Constrained Network and to avoid any (unsecured) CoAP multicast traffic outside the Constrained Network. The DNS server is used by the HTTP Client to resolve the IP address of the HC Proxy and optionally also by the HC Proxy to resolve IP addresses of CoAP servers.

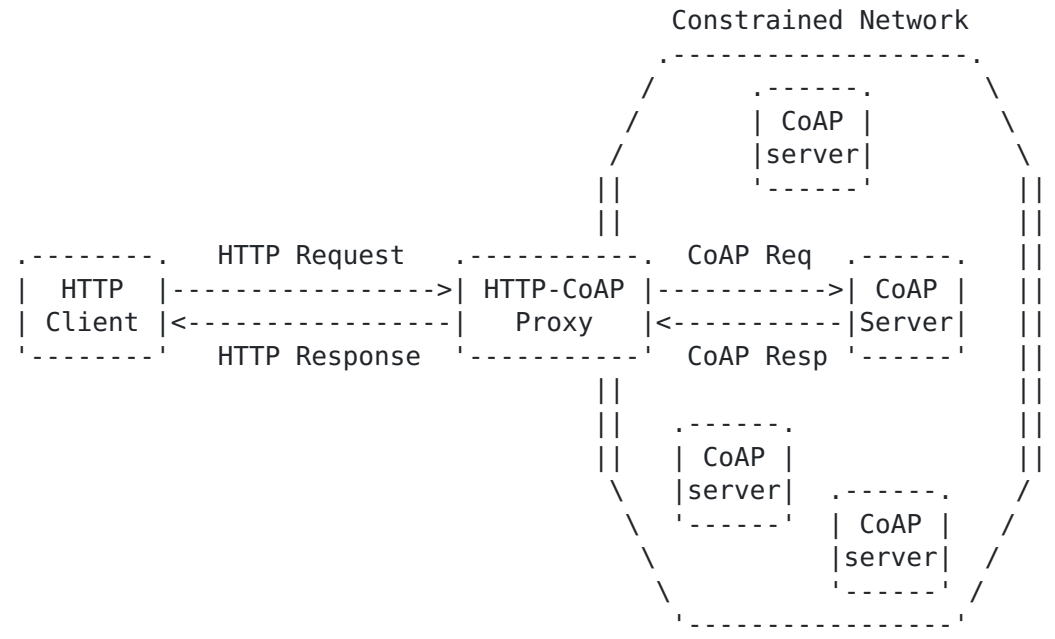

Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

Other placement options for the HC Proxy (not shown) are client-side, which is in the same domain as the HTTP Client; or external, which is both outside the HTTP Client's domain and the CoAP servers' domain.

Normative requirements on the translation of HTTP requests to CoAP requests and of the CoAP responses back to HTTP responses are defined in Section 10.2 of [RFC7252]. However, that section only considers the case of a Forward HC Proxy in which a client explicitly indicates it targets a request to a CoAP server, and does not cover all aspects of proxy implementation in detail. This document provides guidelines and more details for the implementation of a Reverse HC Proxy, which MAY be followed in addition to the normative requirements. Note that most of the guidelines also apply to an Intercepting HC Proxy.

4. Use Cases

To illustrate in which situations HTTP to CoAP protocol translation may be used, three use cases are described below.

1. Smartphone and home sensor: A smartphone can access directly a CoAP home sensor using an authenticated 'https' request, if its home router contains an HC proxy. An HTML5 application on the smartphone can provide a friendly UI to the user using standard (HTTP) networking functions of HTML5.

2. Legacy building control application without CoAP: A building control application that uses HTTP but not CoAP, can check the status of CoAP sensors and/or actuators via an HC proxy.

3. Making sensor data available to 3rd parties: For demonstration or public interest purposes, a HC proxy may be configured to expose the contents of a CoAP sensor to the world via the web (HTTP and/or HTTPS). Some sensors might only handle secure 'coaps' requests, therefore the proxy is configured to translate any request to a 'coaps' secured request. The HC proxy is furthermore configured to only pass through GET requests in order to protect the constrained network. In this way even unattended HTTP clients, such as web crawlers, may index sensor data as regular web pages.

5. URI Mapping

Though, in principle, a CoAP URI could be directly used by a HTTP user agent to de-reference a CoAP resource through an HC proxy, the reality is that all major web browsers, networking libraries and command line tools do not allow making HTTP requests using URIs with a scheme "coap" or "coaps".

Thus, there is a need for web applications to "pack" a CoAP URI into a HTTP URI so that it can be (non-destructively) transported from the user agent to the HC proxy. The HC proxy can then "unpack" the CoAP URI and finally de-reference it via a CoAP request to the target Server.

URI Mapping is the process through which the URI of a CoAP resource is transformed into an HTTP URI so that:

- the requesting HTTP user agent can handle it;
- the receiving HC proxy can extract the intended CoAP URI unambiguously.

To this end, the remainder of this section will identify:

- the default mechanism to map a CoAP URI into a HTTP URI;
- the URI template format to express a class of CoAP-HTTP URI mapping functions;
- the discovery mechanism based on CoRE Link Format [RFC6690] through which clients of an HC proxy can dynamically discover information about the supported URI Mapping Template(s), as well as the base URI where the HC proxy function is anchored.

5.1. URI Terminology

In the remainder of this section, the following terms will be used with a distinctive meaning:

Target CoAP URI:
: URI which refers to the (final) CoAP resource that has to be de-referenced. It conforms to syntax defined in Section 6 of [RFC7252]. Specifically, its scheme is either "coap" or "coaps".

Hosting HTTP URI:
: URI that conforms to syntax in Section 2.7 of [RFC7230]. Its authority component refers to an HC proxy, whereas path (and query) component(s) embed the information used by an HC proxy to extract the Target CoAP URI.

5.2. Default Mapping

The default mapping is for the Target CoAP URI to be appended as-is to a base URI provided by the HC proxy, to form the Hosting HTTP URI.

For example: given a base URI http://p.example.com/hc and a Target CoAP URI coap://s.example.com/light, the resulting Hosting HTTP URI would be http://p.example.com/hc/coap://s.example.com/light.

Provided a correct Target CoAP URI, the Hosting HTTP URI resulting from the default mapping is always syntactically correct. Furthermore, the Target CoAP URI can always be extracted unambiguously from the Hosting HTTP URI. Also, it is worth noting that, using the default mapping, a query component in the target CoAP resource URI is naturally encoded into the query component of the Hosting URI, e.g.: coap://s.example.com/light?dim=5 becomes http://p.example.com/hc/coap://s.example.com/light?dim=5.

There is no default for the base URI. Therefore, it is either known in advance, e.g. as a configuration preset, or dynamically discovered using the mechanism described in Section 5.4.

The default URI mapping function is RECOMMENDED to be implemented and activated by default in an HC proxy, unless there are valid reasons, e.g. application specific, to use a different mapping function.

5.2.1. Optional Scheme Omission

When found in a Hosting HTTP URI, the scheme (i.e., "coap" or "coaps"), the scheme component delimiter (":"), and the double slash

("//") preceding the authority MAY be omitted. In such case, a local default - not defined by this document - applies.

So, http://p.example.com/hc/s.coap.example.com/foo could either represent the target coap://s.coap.example.com/foo or coaps://s.coap.example.com/foo depending on application specific presets.

5.2.2. Encoding Caveats

When the authority of the Target CoAP URI is given as an IPv6address, then the surrounding square brackets MUST be percent-encoded in the Hosting HTTP URI, in order to comply with the syntax defined in Section 3.3. of [RFC3986] for a URI path segment. E.g.: coap://[2001:db8::1]/light?on becomes http://p.example.com/hc/coap://%5B2001:db8::1%5D/light?on.

Everything else can be safely copied verbatim from the Target CoAP URI to the Hosting HTTP URI.

5.3. URI Mapping Template

This section defines a format for the URI template [RFC6570] used by an HC proxy to inform its clients about the expected syntax for the Hosting HTTP URI.

When instantiated, an URI Mapping Template is always concatenated to a base URI provided by the HC proxy via discovery (see Section 5.4), or by other means.

A simple form (Section 5.3.1) and an enhanced form (Section 5.3.2) are provided to fit different users' requirements.

Both forms are expressed as level 2 URI templates [RFC6570] to take care of the expansion of values that are allowed to include reserved URI characters. The syntax of all URI formats is specified in this section in Augmented Backus-Naur Form (ABNF) [RFC5234].

5.3.1. Simple Form

The simple form MUST be used for mappings where the Target CoAP URI is going to be copied (using rules of Section 5.2.2) at some fixed position into the Hosting HTTP URI.

The following template variables MUST be used in mutual exclusion in a template definition:

```
cu = coap-URI   ; from [RFC7252], Section 6.1
su = coaps-URI  ; from [RFC7252], Section 6.2
tu = cu / su
```

The same considerations as in Section 5.2.1 apply, in that the CoAP scheme may be omitted from the Hosting HTTP URI.

5.3.1.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use http://p.example.com/hc as the base URI. Note that these examples all define mapping templates that deviate from the default template of Section 5.2 to be able to illustrate the use of the above template variables.

1. "coap" URI is a query argument of the Hosting HTTP URI:

?coap_target_uri={+cu}

coap://s.example.com/light

http://p.example.com/hc?coap_target_uri=coap://s.example.com/light

2. "coaps" URI is a query argument of the Hosting HTTP URI:

?coaps_target_uri={+su}

coaps://s.example.com/light

http://p.example.com/hc?coaps_target_uri=coaps://s.example.com/light

3. Target CoAP URI as a query argument of the Hosting HTTP URI:

```
?target_uri={+tu}

coap://s.example.com/light

http://p.example.com/hc?target_uri=coap://s.example.com/light

or

coaps://s.example.com/light

http://p.example.com/hc?target_uri=coaps://s.example.com/light
```

4. Target CoAP URI in the path component of the Hosting HTTP URI (i.e., the default URI Mapping template):

```
/{+tu}

coap://s.example.com/light

http://p.example.com/hc/coap://s.example.com/light

or

coaps://s.example.com/light

http://p.example.com/hc/coaps://s.example.com/light
```

5. "coap" URI is a query argument of the Hosting HTTP URI; client decides to omit scheme because a default scheme is agreed beforehand between client and proxy:

```
?coap_uri={+cu}

coap://s.example.com/light

http://p.example.com/hc?coap_uri=s.example.com/light
```

5.3.2. Enhanced Form

The enhanced form can be used to express more sophisticated mappings, i.e., those that do not fit into the simple form.

There MUST be at most one instance of each of the following template variables in a template definition:

```
s  = "coap" / "coaps" ; from [RFC7252], Sections 6.1 and 6.2
hp = host [":" port]  ; from [RFC3986] Sections 3.2.2 and 3.2.3
p  = path-abempty     ; from [RFC3986] Section 3.3
q  = query            ; from [RFC3986] Section 3.4
qq = [ "?" query ]    ; qq is empty iff 'query' is empty
```

5.3.2.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use http://p.example.com/hc as the base URI.

1. Target CoAP URI components in path segments, and optional query in query component:

   {+s}{+hp}{+p}{+qq}

   coap://s.example.com/light

   http://p.example.com/hc/coap/s.example.com/light

   or

   coap://s.example.com/light?on

   http://p.example.com/hc/coap/s.example.com/light?on

2. Target CoAP URI components split in individual query arguments:

?s={+s}&hp={+hp}&p={+p}&q={+q}

coap://s.example.com/light

http://p.example.com/hc?s=coap&hp=s.example.com&p=/light&q=

or

coaps://s.example.com/light?on

http://p.example.com/hc?s=coaps&hp=s.example.com&p=/light&q=on

5.4. Discovery

In order to accommodate site specific needs while allowing third parties to discover the proxy function, the HC proxy SHOULD publish information related to the location and syntax of the HC proxy function using the CoRE Link Format [RFC6690] interface.

To this aim a new Resource Type, "core.hc", is defined in this document. It is associated with a base URI, and can be used as the value for the "rt" attribute in a query to the /.well-known/core in order to locate the base URI where the HC proxy function is anchored.

Along with it, the new target attribute "hct" is defined in this document. This attribute MAY be returned in a "core.hc" link to provide the URI Mapping Template associated to the mapping resource. The default template given in Section 5.2, i.e., {+tu}, MUST be assumed if no "hct" attribute is found in the returned link. If a "hct" attribute is present in the returned link, then a compliant client MUST use it to create the Hosting HTTP URI.

Discovery as specified in [RFC6690] SHOULD be available on both the HTTP and the CoAP side of the HC proxy, with one important difference: on the CoAP side the link associated to the "core.hc" resource needs an explicit anchor referring to the HTTP origin, while on the HTTP interface the link context is already the HTTP origin carried in the request's Host header, and doesn't have to be made explicit.

5.4.1. Discovering CoAP Resources

For a HTTP client, it may be unknown which CoAP resources are available through a HC Proxy. By default an HC Proxy does not support a method to discover all CoAP resources. However, if an HC Proxy is integrated with a Resource Directory ([I-D.ietf-core-resource-directory]) function, an HTTP client can

discover all CoAP resources of its interest by doing an RD Lookup to the HC Proxy, via HTTP. This is possible because a single RD can support both CoAP and HTTP interfaces simultaneously. Of course the HTTP client will this way only discover resources that have been previously registered onto this RD by CoAP devices.

5.4.2. Examples

- The first example exercises the CoAP interface, and assumes that the default template, {+tu}, is used:

```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc

Res:  2.05 Content
      </hc>;anchor="http://p.example.com";rt="core.hc"
```

- The second example - also on the CoAP side of the HC proxy - uses a custom template, i.e., one where the CoAP URI is carried inside the query component, thus the returned link carries the URI template to be used in an explicit "hct" attribute:

```
Req:  GET coap://[ff02::1]/.well-known/core?rt=core.hc

Res:  2.05 Content
      </hc>;anchor="http://p.example.com";
      rt="core.hc";hct="?uri={+tu}"
```

On the HTTP side, link information can be serialized in more than one way:

- using the 'application/link-format' content type:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format
      Content-Length: 18

      </hc>;rt="core.hc"
```

- using the 'application/link-format+json' content type as defined in [I-D.bormann-core-links-json]:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 31

      [{"href":"/hc","rt":"core.hc"}]
```

- using the Link header:

```
Req:  GET /.well-known/core?rt=core.hc HTTP/1.1
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Link: </hc>;rt="core.hc"
```

- An HC proxy may expose two different base URIs to differentiate between Target CoAP resources in the "coap" and "coaps" scheme:

```
Req:  GET /.well-known/core?rt=core.hc
      Host: p.example.com

Res:  HTTP/1.1 200 OK
      Content-Type: application/link-format+json
      Content-Length: 111

      [
        {"href":"/hc/plaintext","rt":"core.hc","hct":"{+cu}"},
        {"href":"/hc/secure","rt":"core.hc","hct":"{+su}"}
      ]
```

6. Media Type Mapping

6.1. Overview

An HC proxy needs to translate HTTP media types (Section 3.1.1.1 of [RFC7231]) and content encodings (Section 3.1.2.2 of [RFC7231]) into CoAP content formats (Section 12.3 of [RFC7252]) and vice versa.

Media type translation can happen in GET, PUT or POST requests going from HTTP to CoAP, and in 2.xx (i.e., successful) responses going from CoAP to HTTP. Specifically, PUT and POST need to map both the Content-Type and Content-Encoding HTTP headers into a single CoAP Content-Format option, whereas GET needs to map Accept and Accept-Encoding HTTP headers into a single CoAP Accept option. To generate the HTTP response, the CoAP Content-Format option is mapped back to a suitable HTTP Content-Type and Content-Encoding combination.

An HTTP request carrying a Content-Type and Content-Encoding combination which the HC proxy is unable to map to an equivalent CoAP Content-Format, SHALL elicit a 415 (Unsupported Media Type) response by the HC proxy.

On the content negotiation side, failure to map Accept and Accept-* headers SHOULD be silently ignored: the HC proxy SHOULD therefore forward as a CoAP request with no Accept option. The HC proxy thus disregards the Accept/Accept-* header fields by treating the response as if it is not subject to content negotiation, as mentioned in Sections 5.3.* of [RFC7231]. However, an HC proxy implementation is free to attempt mapping a single Accept header in a GET request to multiple CoAP GET requests, each with a single Accept option, which are then tried in sequence until one succeeds. Note that an HTTP Accept */* MUST be mapped to a CoAP request without Accept option.

While the CoAP to HTTP direction has always a well defined mapping (with the exception examined in Section 6.2), the HTTP to CoAP direction is more problematic because the source set, i.e., potentially 1000+ IANA registered media types, is much bigger than the destination set, i.e., the mere 6 values initially defined in Section 12.3 of [RFC7252].

Depending on the tight/loose coupling with the application(s) for which it proxies, the HC proxy could implement different media type mappings.

When tightly coupled, the HC proxy knows exactly which content formats are supported by the applications, and can be strict when enforcing its forwarding policies in general, and the media type mapping in particular.

On the other side, when the HC proxy is a general purpose application layer gateway, being too strict could significantly reduce the amount of traffic that it'd be able to successfully forward. In this case, the "loose" media type mapping detailed in Section 6.3 MAY be implemented.

The latter grants more evolution of the surrounding ecosystem, at the cost of allowing more attack surface. In fact, as a result of such strategy, payloads would be forwarded more liberally across the unconstrained/constrained network boundary of the communication path. Therefore, when applied, other forms of access control must be set in place to avoid unauthorized users to deplete or abuse systems and network resources.

6.2. 'application/coap-payload' Media Type

If the HC proxy receives a CoAP response with a Content-Format that it does not recognize (e.g. because the value has been registered after the proxy has been deployed, or the CoAP server uses an experimental value which is not registered), then the HC proxy SHALL return a generic "application/coap-payload" media type with numeric parameter "cf" as defined in Section 9.2.

For example, the CoAP content format '60' ("application/cbor") would be represented by "application/coap-payload;cf=60", would '60' be an unknown content format to the HC Proxy.

A HTTP client MAY use the media type "application/coap-payload" as a means to send a specific content format to a CoAP server via an HC Proxy if the client has determined that the HC Proxy does not directly support the type mapping it needs. This case may happen when dealing for example with newly registered, yet to be registered, or experimental CoAP content formats.

6.3. Loose Media Type Mapping

By structuring the type information in a super-class (e.g. "text") followed by a finer grained sub-class (e.g. "html"), and optional parameters (e.g. "charset=utf-8"), Internet media types provide a rich and scalable framework for encoding the type of any given entity.

This approach is not applicable to CoAP, where Content Formats conflate an Internet media type (potentially with specific parameters) and a content encoding into one small integer value.

To remedy this loss of flexibility, we introduce the concept of a "loose" media type mapping, where media types that are specializations of a more generic media type can be aliased to their super-class and then mapped (if possible) to one of the CoAP content formats. For example, "application/soap+xml" can be aliased to "application/xml", which has a known conversion to CoAP. In the context of this "loose" media type mapping, "application/octet-

stream" can be used as a fallback when no better alias is found for a specific media type.

Table 1 defines the default lookup table for the "loose" media type mapping. Given an input media type, the table returns its best generalized media type using the most specific match i.e. the table entries are compared to the input in top to bottom order until an entry matches.

Internet media type	Generalized media type
application/*+xml	application/xml
application/*+json	application/json
text/xml	application/xml
text/*	text/plain
/	application/octet-stream

Table 1: Media type generalization lookup table

The "loose" media type mapping is an OPTIONAL feature. Implementations supporting this kind of mapping SHOULD provide a flexible way to define the set of media type generalizations allowed.

6.4. Media Type to Content Format Mapping Algorithm

This section defines the algorithm used to map an HTTP Internet media type to its correspondent CoAP content format.

The algorithm uses the mapping table defined in Section 12.3 of [RFC7252] plus, possibly, any locally defined extension of it. Optionally, the table and lookup mechanism described in Section 6.3 can be used if the implementation chooses so.

Note that the algorithm may have side effects on the associated representation (see also Section 6.5).

In the following:

- C-T, C-E, and C-F stand for the values of the Content-Type (or Accept) HTTP header, Content-Encoding (or Accept-Encoding) HTTP header, and Content-Format CoAP option respectively.
- If C-E is not given it is assumed to be "identity".
- MAP is the mandatory lookup table, GMAP is the optional generalized table.

```
INPUT:  C-T and C-E
OUTPUT: C-F or Fail

1.  if no C-T: return Fail
2.  C-F = MAP[C-T, C-E]
3.  if C-F is not None: return C-F
4.  if C-E is not "identity":
5.    if C-E is supported (e.g. gzip):
6.      decode the representation accordingly
7.      set C-E to "identity"
8.    else:
9.      return Fail
10. repeat steps 2. and 3.
11. if C-T allows a non-lossy transformation into \
12.    one of the supported C-F:
13.      transcode the representation accordingly
14.      return C-F
15. if GMAP is defined:
16.   C-F = GMAP[C-T]
17.   if C-F is not None: return C-F
18. return Fail
```

Figure 2

6.5. Content Transcoding

6.5.1. General

Payload content transcoding (e.g. see steps 11-14 of Figure 2) is an OPTIONAL feature. Implementations supporting this feature should provide a flexible way to define the set of transcodings allowed.

As noted in Section 6.4, the process of mapping the media type can have side effects on the forwarded entity body. This may be caused by the removal or addition of a specific content encoding, or because the HC proxy decides to transcode the representation to a different (compatible) format. The latter proves useful when an optimized version of a specific format exists. For example an XML-encoded resource could be transcoded to Efficient XML Interchange (EXI) format, or a JSON-encoded resource into CBOR [RFC7049], effectively achieving compression without losing any information.

However, it should be noted that in certain cases, transcoding can lose information in a non-obvious manner. For example, encoding an XML document using schema-informed EXI encoding leads to a loss of information when the destination does not know the exact schema version used by the encoder, which means that whenever the HC proxy transcodes an application/XML to application/EXI in-band metadata

could be lost. Therefore, the implementer should always carefully verify such lossy payload transformations before triggering the transcoding.

6.5.2. CoRE Link Format

The CoRE Link Format [RFC6690] is a set of links (i.e., URIs and their formal relationships) which is carried as content payload in a CoAP response. These links usually include CoAP URIs that might be translated by the HC proxy to the correspondent HTTP URIs using the implemented URI mapping function (see Section 5). Such a process would inspect the forwarded traffic and attempt to re-write the body of resources with an application/link-format media type, mapping the embedded CoAP URIs to their HTTP counterparts. Some potential issues with this approach are:

1. The client may be interested to retrieve original (unaltered) CoAP payloads through the HC proxy, not modified versions.

2. Tampering with payloads is incompatible with resources that are integrity protected (although this is a problem with transcoding in general).

3. The HC proxy needs to fully understand [RFC6690] syntax and semantics, otherwise there is an inherent risk to corrupt the payloads.

Therefore, CoRE Link Format payload should only be transcoded at the risk and discretion of the proxy implementer.

6.5.3. Diagnostic Messages

CoAP responses may, in certain error cases, contain a diagnostic message in the payload explaining the error situation, as described in Section 5.5.2 of [RFC7252]. In this scenario, the CoAP response diagnostic payload MUST NOT be returned as the regular HTTP payload (message body). Instead, the CoAP diagnostic payload must be used as the HTTP reason-phrase of the HTTP status line, as defined in Section 3.1.2 of [RFC7230], without any alterations, except those needed to comply to the reason-phrase ABNF definition.

7. Response Code Mapping

Table 2 defines the HTTP response status codes to which each CoAP response code SHOULD be mapped. This table complies with the requirements in Section 10.2 of [RFC7252] and is intended to cover all possible cases. Multiple appearances of a HTTP status code in the second column indicates multiple equivalent HTTP responses are

possible based on the same CoAP response code, depending on the conditions cited in the Notes (third column and text below table).

CoAP Response Code	HTTP Status Code	Notes
2.01 Created	201 Created	1
2.02 Deleted	200 OK	2
	204 No Content	2
2.03 Valid	304 Not Modified	3
	200 OK	4
2.04 Changed	200 OK	2
	204 No Content	2
2.05 Content	200 OK	
4.00 Bad Request	400 Bad Request	
4.01 Unauthorized	401 Unauthorized	5
4.02 Bad Option	400 Bad Request	6
4.03 Forbidden	403 Forbidden	
4.04 Not Found	404 Not Found	
4.05 Method Not Allowed	400 Bad Request	7
4.06 Not Acceptable	406 Not Acceptable	
4.12 Precondition Failed	412 Precondition Failed	
4.13 Request Ent. Too Large	413 Request Repr. Too Large	
4.15 Unsupported Media Type	415 Unsupported Media Type	
5.00 Internal Server Error	500 Internal Server Error	
5.01 Not Implemented	501 Not Implemented	
5.02 Bad Gateway	502 Bad Gateway	
5.03 Service Unavailable	503 Service Unavailable	8
5.04 Gateway Timeout	504 Gateway Timeout	
5.05 Proxying Not Supported	502 Bad Gateway	9

Table 2: CoAP-HTTP Response Code Mappings

Notes:

1. A CoAP server may return an arbitrary format payload along with this response. This payload SHOULD be returned as entity in the HTTP 201 response. Section 7.3.2 of [RFC7231] does not put any requirement on the format of the entity. (In the past, [RFC2616] did.)

2. The HTTP code is 200 or 204 respectively for the case that a CoAP server returns a payload or not. [RFC7231] Section 5.3 requires code 200 in case a representation of the action result is returned for DELETE/POST/PUT, and code 204 if not. Hence, a proxy SHOULD transfer any CoAP payload contained in a CoAP 2.02 response to the HTTP client using a 200 OK response.

3. HTTP code 304 (Not Modified) is sent if the HTTP client performed a conditional HTTP request and the CoAP server responded with 2.03 (Valid) to the corresponding CoAP validation request. Note that Section 4.1 of [RFC7232] puts some requirements on header fields that must be present in the HTTP 304 response.

4. A 200 response to a CoAP 2.03 occurs only when the HC proxy, for efficiency reasons, is caching resources and translated a HTTP request (without conditional request) to a CoAP request that includes ETag validation. The proxy receiving 2.03 updates the freshness of its cached representation and returns the entire representation to the HTTP client.

5. A HTTP 401 Unauthorized (Section 3.1 of [RFC7235]) response MUST include a WWW-Authenticate header. Since there is no CoAP equivalent of WWW-Authenticate, the HC proxy must generate this header itself including at least one challenge (Section 4.1 of [RFC7235]). If the HC proxy does not implement a proper authentication method that can be used to gain access to the target CoAP resource, it can include a 'dummy' challenge for example "WWW-Authenticate: None".

6. A proxy receiving 4.02 may first retry the request with less CoAP Options in the hope that the CoAP server will understand the newly formulated request. For example, if the proxy tried using a Block Option [I-D.ietf-core-block] which was not recognized by the CoAP server it may retry without that Block Option. Note that HTTP 402 MUST NOT be returned because it is reserved for future use [RFC7231].

7. A CoAP 4.05 (Method Not Allowed) response SHOULD normally be mapped to a HTTP 400 (Method Not Allowed) code, because the HTTP 405 response would require specifying the supported methods - which are generally unknown. In this case the HC Proxy SHOULD also return a HTTP reason-phrase in the HTTP status line that starts with the string "405" in order to facilitate troubleshooting. However, if the HC proxy has more granular information about the supported methods for the requested resource (e.g. via a Resource Directory ([I-D.ietf-core-resource-directory])) then it MAY send back a HTTP 405 (Method Not Allowed) with a properly filled in "Allow" response-header field (Section 7.4.1 of [RFC7231]).

8. The value of the HTTP "Retry-After" response-header field is taken from the value of the CoAP Max-Age Option, if present.

9. This CoAP response can only happen if the proxy itself is configured to use a CoAP forward-proxy (Section 5.7 of [RFC7252]) to execute some, or all, of its CoAP requests.

8. Additional Mapping Guidelines

8.1. Caching and Congestion Control

An HC proxy SHOULD limit the number of requests to CoAP servers by responding, where applicable, with a cached representation of the resource.

Duplicate idempotent pending requests by an HC proxy to the same CoAP resource SHOULD in general be avoided, by using the same response for multiple requesting HTTP clients without duplicating the CoAP request.

If the HTTP client times out and drops the HTTP session to the HC proxy (closing the TCP connection) after the HTTP request was made, an HC proxy SHOULD wait for the associated CoAP response and cache it if possible. Subsequent requests to the HC proxy for the same resource can use the result present in cache, or, if a response has still to come, the HTTP requests will wait on the open CoAP request.

According to [RFC7252], a proxy MUST limit the number of outstanding interactions to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the HC proxy SHOULD also pose a limit to the number of concurrent CoAP requests pending on the same constrained network; further incoming requests MAY either be queued or dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior SHOULD be configurable. In order to effectively apply above congestion control, the HC proxy should be server-side placed.

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the HC proxy to keep their cached representation fresh while minimizing the number of CoAP traffic in the constrained network. See Section 8.2.

8.2. Cache Refresh via Observe

There are cases where using the CoAP observe protocol [I-D.ietf-core-observe] to handle proxy cache refresh is preferable to the validation mechanism based on ETag as defined in [RFC7252]. Such scenarios include, but are not limited to, sleepy CoAP nodes -- with possibly high variance in requests' distribution -- which would greatly benefit from a server driven cache update mechanism. Ideal candidates for CoAP observe are also crowded or very low throughput

networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether refreshing a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R.

Let T_R be the mean time between two client requests to resource R, let T_C be the mean time between two representation changes of R, and let M_R be the mean number of CoAP messages per second exchanged to and from resource R. If we assume that the initial cost for establishing the observation is negligible, an observation on R reduces M_R iff T_R < 2*T_C with respect to using ETag validation, that is iff the mean arrival rate of requests for resource R is greater than half the change rate of R.

When observing the resource R, M_R is always upper bounded by 2/T_C.

8.3. Use of CoAP Blockwise Transfer

An HC proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding excessive link-layer fragmentation in constrained networks, and to cope with small datagram buffers in CoAP end-points as described in [RFC7252] Section 4.6.

An HC proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. An HC proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than BLOCKWISE_THRESHOLD bytes. The value of BLOCKWISE_THRESHOLD is implementation-specific, for example it can be:

- calculated based on a known or typical UDP datagram buffer size for CoAP end-points, or
- set to N times the known size of a link-layer frame in a constrained network where e.g. N=5, or
- preset to a known IP MTU value, or
- set to a known Path MTU value.

The value BLOCKWISE_THRESHOLD, or the parameters from which it is calculated, should be configurable in a proxy implementation. The maximum block size the proxy will attempt to use in CoAP requests should also be configurable.

The HC proxy SHOULD detect CoAP end-points not supporting blockwise transfers by checking for a 4.02 (Bad Option) response returned by an end-point in response to a CoAP request with a Block* Option, and subsequent absence of the 4.02 in response to the same request without Block* Options. This allows the HC proxy to be more efficient, not attempting repeated blockwise transfers to CoAP servers that do not support it. However, if a request payload is too large to be sent as a single CoAP request and blockwise transfer would be unavoidable, the proxy still SHOULD attempt blockwise transfer on such an end-point before returning the response 413 (Request Entity Too Large) to the HTTP client.

For improved latency an HC proxy MAY initiate a blockwise CoAP request triggered by an incoming HTTP request even when the HTTP request message has not yet been fully received, but enough data has been received to send one or more data blocks to a CoAP server already. This is particularly useful on slow client-to-proxy connections.

8.4. Security Translation

For the guidelines on security context translations for an HC proxy, see Section 10.2. A translation may involve e.g. applying a rule that any "https" request is translated to a "coaps" request, or e.g. applying a rule that a "https" request is translated to an unsecured "coap" request.

8.5. CoAP Multicast

An HC proxy MAY support CoAP multicast. If it does, the HC proxy sends out a multicast CoAP request if the Target CoAP URI's authority is a multicast IP literal or resolves to a multicast IP address; assuming the proper security measures are in place to mitigate security risks of CoAP multicast (Section 10). If the security policies do not allow the specific CoAP multicast request to be made, the HC proxy SHOULD respond 403 (Forbidden).

If an HC proxy does not support CoAP multicast, it SHOULD respond 403 (Forbidden) to any valid HTTP request that maps to a CoAP multicast request.

Details related to supporting CoAP multicast are currently out of scope of this document since in a reverse proxy scenario a HTTP client typically expects to receive a single response, not multiple. However, an HC proxy that implements CoAP multicast MAY include application-specific functions to aggregate multiple CoAP responses into a single HTTP response. We suggest using the "application/http" internet media type (Section 8.3.2 of [RFC7230]) to enclose a set of

one or more HTTP response messages, each representing the mapping of one CoAP response.

8.6. Timeouts

When facing long delays of a CoAP server in responding, the HTTP client or any other proxy in between MAY timeout. Further discussion of timeouts in HTTP is available in Section 6.2.4 of [RFC7230].

An HC proxy MUST define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request. Assuming the Proxy may use confirmable CoAP requests, such timeout value T SHOULD be at least

T = MAX_RTT + MAX_SERVER_RESPONSE_DELAY

where MAX_RTT is defined in [RFC7252] and MAX_SERVER_RESPONSE_DELAY is defined in [RFC7390]. An exception to this rule occurs when the HC proxy is configured with a HTTP response timeout value that is lower than above value T; then the lower value should be also used as the CoAP request timeout.

8.7. Miscellaneous

In certain use cases, constrained CoAP nodes do not make use of the DNS protocol. However even when the DNS protocol is not used in a constrained network, defining valid FQDN (i.e., DNS entries) for constrained CoAP servers, where possible, may help HTTP clients to access the resources offered by these servers via an HC proxy.

HTTP connection pipelining (section 6.3.2 of [RFC7230]) may be supported by an HC proxy. This is transparent to the CoAP servers: the HC proxy will serve the pipelined requests by issuing different CoAP requests. The HC proxy in this case needs to respect the NSTART limit of Section 4.7 of [RFC7252].

9. IANA Considerations

9.1. New 'core.hc' Resource Type

This document registers a new Resource Type (rt=) Link Target Attribute, 'core.hc', in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

Attribute Value: core.hc

Description: HTTP to CoAP mapping base resource.

Reference: See Section 5.4.

9.2. New 'coap-payload' Internet Media Type

This document defines the "application/coap-payload" media type with a single parameter "cf". This media type represents any payload that a CoAP message can carry, having a content format that can be identified by a CoAP Content-Format parameter (an integer in range 0-65535). The parameter "f" is the integer defining the CoAP content format.

Type name: application

Subtype name: coap-payload

Required parameters:

cf - CoAP Content-Format integer in range 0-65535 denoting the content format of the CoAP payload carried.

Optional parameters: None

Encoding considerations:

The specific CoAP content format encoding considerations for the selected Content-Format (cf parameter) apply.

Security considerations:

The specific CoAP content format security considerations for the selected Content-Format (cf parameter) apply.

Interoperability considerations:

Published specification: (this I-D - TBD)

Applications that use this media type:

HTTP-to-CoAP Proxies.

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information:

Esko Dijk ("esko@ieee.org")

Intended usage: COMMON

Restrictions on usage:

An application (or user) can only use this media type if it has to represent a CoAP payload of which the specified CoAP Content-Format is an unrecognized number; such that a proper translation directly to the equivalent HTTP media type is not possible.

Author: CoRE WG

Change controller: IETF

Provisional registration? (standards tree only): N/A

10. Security Considerations

The security concerns raised in Section 9.2 of [RFC7230] also apply to the HC proxy scenario. In fact, the HC proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the HC proxy cannot be dropped, because the protocol translation function is the core duty of the HC proxy: it is a necessarily trusted, impossible to bypass, component in the communication path.

A reverse proxy deployed at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it has fewer limitations than the constrained devices it is serving.

The following sub paragraphs categorize and discuss a set of specific security issues related to the translation, caching and forwarding functionality exposed by an HC proxy.

10.1. Traffic Overflow

Due to the typically constrained nature of CoAP nodes, particular attention SHOULD be given to the implementation of traffic reduction mechanisms (see Section 8.1), because inefficient proxy implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

An amplification attack to the constrained network may be triggered by a multicast request generated by a single HTTP request which is mapped to a CoAP multicast resource, as considered in Section 11.3 of [RFC7252].

The risk likelihood of this amplification technique is higher than an amplification attack carried out by a malicious constrained device (e.g. ICMPv6 flooding, like Packet Too Big, or Parameter Problem on a multicast destination [RFC4732]), since it does not require direct access to the constrained network.

The feasibility of this attack, disruptive in terms of CoAP server availability, can be limited by access controlling the exposed HTTP multicast resources, so that only known/authorized users access such URIs.

10.2. Handling Secured Exchanges

An HTTP request can be sent to the HC proxy over a secured connection. However, there may not always exist a secure connection mapping to CoAP. For example, a secure distribution method for multicast traffic is complex and MAY not be implemented (see [RFC7390]).

An HC proxy SHOULD implement explicit rules for security context translations. A translation may involve e.g. applying a rule that any "https" unicast request is translated to a "coaps" request, or e.g. applying a rule that a "https" request is translated to an unsecured "coap" request. Another rule could specify the security policy and parameters used for DTLS connections. Such rules will largely depend on the application and network context in which a proxy operates. These rules SHOULD be configurable in an HC proxy.

If a policy for access to 'coaps' URIs is configurable in an HC proxy, it is RECOMMENDED that the policy is by default configured to disallow access to any 'coaps' URI by a HTTP client using an unsecured (non-TLS) connection. Naturally, a user MAY reconfigure the policy to allow such access in specific cases.

By default, an HC proxy SHOULD reject any secured client request if there is no configured security policy mapping. This recommendation MAY be relaxed in case the destination network is believed to be secured by other, complementary, means. E.g.: assumed that CoAP nodes are isolated behind a firewall (e.g. as in the SS HC proxy deployment shown in Figure 1), the HC proxy may be configured to translate the incoming HTTPS request using plain CoAP (NoSec mode).

The HTTP-CoAP URI mapping (defined in Section 5) MUST NOT map to HTTP a CoAP resource intended to be only accessed securely.

A secured connection that is terminated at the HC proxy, i.e., the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The HC proxy SHOULD NOT cache any secured content to avoid any leak of secured information. However, in some specific scenario, a security/efficiency trade-off could motivate caching secured information; in that case the caching behavior MAY be tuned to some extent on a per-resource basis.

10.3. Proxy and CoAP Server Resource Exhaustion

If the HC proxy implements the low-latency optimization of Section 8.3 intended for slow client-to-proxy connections, the Proxy may become vulnerable to a resource exhaustion attack. In this case an attacking client could initiate multiple requests using a relatively large message body which is (after an initial fast transfer) transferred very slowly to the Proxy. This would trigger the HC proxy to create state for a blockwise CoAP request per HTTP request, waiting for the arrival of more data over the HTTP/TCP connection. Such attacks can be mitigated in the usual ways for HTTP servers using for example a connection time limit along with a limit on the number of open TCP connections per IP address.

10.4. URI Mapping

The following risks related to the URI mapping described in Section 5 and its use by HC proxies have been identified:

DoS attack on the constrained/CoAP network.
   To mitigate, by default deny any Target CoAP URI whose authority is (or maps to) a multicast address. Then explicitly white-list multicast resources/authorities that are allowed to be de-referenced. See also Section 8.5.

Leaking information on the constrained/CoAP network resources and topology.
   To mitigate, by default deny any Target CoAP URI (especially /.well-known/core is a resource to be protected), and then

explicit white-list resources that are allowed to be seen from outside.

Reduced privacy due to the mechanics of the URI mapping.
The internal CoAP Target resource is totally transparent from outside. An HC proxy can mitigate by implementing a HTTPS-only interface, making the Target CoAP URI totally opaque to a passive attacker.

11. Acknowledgements

An initial version of Table 2 in Section 7 has been provided in revision -05 of the CoRE CoAP I-D. Special thanks to Peter van der Stok for countless comments and discussions on this document, that contributed to its current structure and text.

Thanks to Carsten Bormann, Zach Shelby, Michele Rossi, Nicola Bui, Michele Zorzi, Klaus Hartke, Cullen Jennings, Kepeng Li, Brian Frank, Peter Saint-Andre, Kerry Lynn, Linyi Tian, Dorothy Gellert, Francesco Corazza for helpful comments and discussions that have shaped the document.

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n.251557.

12. References

12.1. Normative References

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-17 (work in progress), March 2015.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-16 (work in progress), December 2014.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

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

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

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

[RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

12.2. Informative References

[I-D.bormann-core-links-json]
Bormann, C., "Representing CoRE Link Collections in JSON", draft-bormann-core-links-json-02 (work in progress), February 2013.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-03 (work in progress), June 2015.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.

[RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

[RFC7390] Rahman, A. and E. Dijk, "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, October 2014.

Appendix A. Change Log

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-06 to ietf-07:

- Addressed Ticket #384 - Section 5.4.1 describes briefly (informative) how to discover CoAP resources from an HTTP client.
- Addressed Ticket #378 - For HTTP media type to CoAP content format mapping and vice versa: a new draft (TBD) may be proposed in CoRE which describes an approach for automatic updating of the media type mapping. This was noted in Section 6.1 but is otherwise outside the scope of this draft.
- Addressed Ticket #377 - Added IANA section that defines a new HTTP media type "application/coap-payload" and created new Section 6.2 on how to use it.
- Addressed Ticket #376 - Updated Table 2 (and corresponding note 7) to indicate that a CoAP 4.05 (Method Not Allowed) Response Code should be mapped to a HTTP 400 (Bad Request).
- Added note to comply to ABNF when translating CoAP diagnostic payload to reason-phrase in Section 6.5.3.

Changes from ietf-05 to ietf-06:

- Fully restructured the draft, bringing introductory text more to the front and allocating main sections to each of the key topics; addressing Ticket #379;
- Addressed Ticket #382, fix of enhanced form URI template definition of q in Section 5.3.2;
- Addressed Ticket #381, found a mapping 4.01 to 401 Unauthorized in Section 7;
- Addressed Ticket #380 (Add IANA registration for "core.hc" Resource Type) in Section 9;
- Addressed Ticket #376 (CoAP 4.05 response can't be translated to HTTP 405 by HC proxy) in Section 7 by use of empty 'Allow' header;

- Removed details on the pros and cons of HC proxy placement options;
- Addressed review comments of Carsten Bormann;
- Clarified failure in mapping of HTTP Accept headers (Section 6.3);
- Clarified detection of CoAP servers not supporting blockwise (Section 8.3);
- Changed CoAP request timeout min value to MAX_RTT + MAX_SERVER_RESPONSE_DELAY (Section 8.6);
- Added security section item (Section 10.3) related to use of CoAP blockwise transfers;
- Many editorial improvements.

Changes from ietf-04 to ietf-05:

- Addressed Ticket #366 (Mapping of CoRE Link Format payloads to be valid in HTTP Domain?) in Section 6.3.3.2 (Content Transcoding - CORE Link Format);
- Addressed Ticket #375 (Add requirement on mapping of CoAP diagnostic payload) in Section 6.3.3.3 (Content Transcoding - Diagnostic Messages);
- Addressed comment from Yusuke (http://www.ietf.org/mail-archive/web/core/current/msg05491.html) in Section 6.3.3.1 (Content Transcoding - General);
- Various editorial improvements.

Changes from ietf-03 to ietf-04:

- Expanded use case descriptions in Section 4;
- Fixed/enhanced discovery examples in Section 5.4.1;
- Addressed Ticket #365 (Add text on media type conversion by HTTP-CoAP proxy) in new Section 6.3.1 (Generalized media type mapping) and new Section 6.3.2 (Content translation);
- Updated HTTPBis WG draft references to recently published RFC numbers.
- Various editorial improvements.

Changes from ietf-02 to ietf-03:

- Closed Ticket #351 "Add security implications of proposed default HTTP-CoAP URI mapping";
- Closed Ticket #363 "Remove CoAP scheme in default HTTP-CoAP URI mapping";
- Closed Ticket #364 "Add discovery of HTTP-CoAP mapping resource(s)".

Changes from ietf-01 to ietf-02:

- Selection of single default URI mapping proposal as proposed to WG mailing list 2013-10-09.

Changes from ietf-00 to ietf-01:

- Added URI mapping proposals to Section 4 as per the Email proposals to WG mailing list from Esko.

Authors' Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal H3A 3G4
Canada

Phone: +1 514 585 0761
Email: Akbar.Rahman@InterDigital.com

Thomas Fossati
Alcatel-Lucent
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: thomas.fossati@alcatel-lucent.com

Esko Dijk
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Email: esko.dijk@philips.com