Internet Engineering Task Force (IETF)
Request for Comments: 8075
Category: Standards Track
ISSN: 2070-1721

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
Nokia
E. Dijk
Philips Lighting

February 2017

Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)

Abstract

This document provides reference information for implementing a cross-protocol network proxy that performs translation from the HTTP protocol to the Constrained Application Protocol (CoAP). This will enable an HTTP client to access resources on a CoAP server through the proxy. This document describes how an HTTP request is mapped to a CoAP request and how a CoAP response is mapped back to an HTTP response. This includes guidelines for status code, URI, and media type mappings, as well as additional interworking advice.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8075.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] has been designed with a twofold aim: it's an application protocol specialized for constrained environments and it's easily used in architectures based on Representational State Transfer (REST) [Fielding], such as the web. The latter goal has led to defining CoAP to easily interoperate with HTTP [RFC7230] through an intermediary proxy that 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, [RFC7252] focuses on the basic mapping of request methods and simple response code mapping between HTTP and CoAP, while leaving many details of the cross-protocol proxy for future definition. Therefore, a primary goal of this document is to define a consistent set of guidelines that an HTTP-to-CoAP proxy implementation should adhere to. The key benefit to adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability between an HTTP client and a CoAP server independent of the proxy that implements the cross-protocol mapping. (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 without breaking API semantics.)

This document describes HTTP mappings that apply to protocol elements defined in the base CoAP specification [RFC7252] and in the CoAP block-wise transfer specification [RFC7959]. It is up to CoAP protocol extensions (new methods, response codes, options, content-formats) to describe their own HTTP mappings, if applicable.

The rest of this document is organized as follows:

- Section 2 defines proxy terminology;

- Section 3 introduces the HTTP-to-CoAP proxy;

- Section 4 lists use cases in which HTTP clients need to contact CoAP servers;

- Section 5 introduces a null, default, and advanced 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, multicast, timeouts, etc.; and

- Section 10 discusses the possible security impact of HTTP-to-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].

This specification requires readers to be familiar with the vocabulary and concepts discussed in [RFC7228], in particular, the terms "constrained nodes" and "constrained networks". Readers must also be familiar with all of the terminology of the normative references listed in this document, in particular [RFC7252] (CoAP) and [RFC7230] (HTTP). In addition, this specification makes use of the following terms:

HC Proxy
A proxy performing a cross-protocol mapping, in the context of this document an HTTP-to-CoAP (HC) mapping. Specifically, the HC Proxy acts as an HTTP server and a CoAP client. The HC Proxy can take on the role of a forward, reverse, or interception Proxy.

Application Level Gateway (ALG)
An application-specific translation agent that allows an application on a host in one address realm to connect to its counterpart running on a host in a different realm transparently. See Section 2.9 of [RFC2663].

forward-proxy
A message-forwarding agent that is selected by the HTTP 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 agent decides (is willing) to use the proxy as the forwarding/dereferencing 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-proxy behaves as an origin (HTTP) server on its connection from the HTTP client. The

HTTP client uses the "origin-form" (Section 5.3.1 of [RFC7230]) as a request-target URI. (Note that a reverse-proxy appears to an HTTP 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.)

interception proxy
As in [RFC3040], a proxy that receives inbound HTTP traffic flows through the process of traffic redirection, transparent to the HTTP client.

3. HTTP-to-CoAP Proxy

An HC Proxy is accessed by an HTTP client that needs to fetch a resource on a CoAP server. The HC Proxy handles the HTTP request by mapping it to the equivalent CoAP request, which is then forwarded to the appropriate CoAP server. The received CoAP response is then mapped to an appropriate HTTP response and finally sent back to the originating HTTP client.

Section 10.2 of [RFC7252] defines basic normative requirements on HTTP-to-CoAP mapping. This document provides additional details and guidelines for the implementation of an HC Proxy.

```
                                               Constrained Network
                                              .-------------------.
                                             /      .------.       \
                                            /       | CoAP |        \
                                           /        |server|         \
                                          ||        '------'         ||
                                          ||                         ||
     .--------.  HTTP Request   .------------.  CoAP Req  .------.   ||
     |  HTTP  |---------------->|HTTP-to-CoAP|----------->| CoAP |   ||
     | Client |<----------------|   Proxy    |<-----------|server|   ||
     '--------'  HTTP Response  '------------'  CoAP Resp '------'   ||
                                          ||                         ||
                                          ||   .------.              ||
                                          ||   | CoAP |              ||
                                           \   |server|  .------.    /
                                            \  '------'  | CoAP |   /
                                             \           |server|  /
                                              \          '------' /
                                               '-----------------'
```

Figure 1: HTTP-To-CoAP Proxy Deployment Scenario

Figure 1 illustrates an example deployment scenario. There, an HC Proxy is located at the boundary of the constrained network domain and acts as an ALG that allows only a very specific type of traffic (i.e., authorized inbound HTTP requests and their associated outbound CoAP responses) to pass through. All other kinds of traffic are segregated within the respective network segments.

4. Use Cases

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

1. 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 control actuators via an HC Proxy.

2. Making sensor data available to third parties on the web: For demonstration or public interest purposes, an 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 may only accept secure 'coaps' requests; therefore, the proxy is configured to translate requests to those devices accordingly. The HC Proxy is furthermore configured to only pass through GET requests in order to protect the constrained network.

3. Smartphone and home sensor: A smartphone can access directly a CoAP home sensor using a mutually authenticated 'https' request, provided its home router runs an HC Proxy and is configured with the appropriate certificate. An HTML5 [W3C.REC-html5-20141028] application on the smartphone can provide a friendly UI using the standard (HTTP) networking functions of HTML5.

A key point in the above use cases is the expected nature of the URI to be used by the HTTP client initiating the HTTP request to the HC Proxy. Specifically, in use case #1, there will be no information related to 'coap' or 'coaps' embedded in the HTTP URI as it is a legacy HTTP client sending the request. Use case #2 is also expected to be similar. In contrast, in use case #3, it is likely that the HTTP client will specifically embed information related to 'coap' or 'coaps' in the HTTP URI of the HTTP request to the HC Proxy.

5. URI Mapping

Though, in principle, a CoAP URI could be directly used by an HTTP client to dereference 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 embed or "pack" a CoAP URI into an HTTP URI so that it can be (non-destructively) transported from the HTTP client to the HC Proxy. The HC Proxy can then "unpack" the CoAP URI and finally dereference it via a CoAP request to the target server.

URI mapping is the term used in this document to describe the process through which the URI of a CoAP resource is transformed into an HTTP URI so that:

- The requesting HTTP client can handle it; and
- 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 an HTTP URI;
- The URI Template format to express a class of CoAP-HTTP URI mapping functions; and
- The discovery mechanism based on "Constrained RESTful Environments (CoRE) Link Format" [RFC6690] through which clients of an HC Proxy can dynamically learn about the supported URI mapping template(s), as well as the 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:

HC Proxy URI:
URI that refers to the HC Proxy function. It conforms to syntax defined in Section 2.7 of [RFC7230].

Target CoAP URI:
URI that refers to the (final) CoAP resource that has to be dereferenced. 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 a path and/or query component(s) embed the information used by an HC Proxy to extract the Target CoAP URI.

5.2. Null Mapping

The null mapping is the case where there is no Target CoAP URI appended to the HC Proxy URI. In other words, it is a "pure" HTTP URI that is sent to the HC Proxy. This would typically occur in situations like use case #1 described in Section 4, and the proxy would typically be a reverse-proxy. In this scenario, the HC Proxy will determine through its own private algorithms what the Target CoAP URI should be.

5.3. Default Mapping

The default mapping is for the Target CoAP URI to be appended as is (with the only caveat discussed in Section 5.3.2) to the HC Proxy URI, to form the Hosting HTTP URI. This is the effective request URI (see Section 5.5 of [RFC7230]) that will then be sent by the HTTP client in the HTTP request to the HC Proxy.

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

Provided a correct Target CoAP URI, the Hosting HTTP URI resulting from the default mapping will be a syntactically valid HTTP URI. Furthermore, the Target CoAP URI can always be extracted unambiguously from the Hosting HTTP URI.

There is no default for the HC Proxy URI. Therefore, it is either known in advance, e.g., as a configuration preset, or dynamically discovered using the mechanism described in Section 5.5.

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

5.3.1. Optional Scheme Omission

When constructing a Hosting HTTP URI by embedding a Target CoAP URI, the scheme (i.e., ’coap’ or ’coaps’), the scheme component delimiter (":"), and the double slash ("//") preceding the authority MAY be omitted if a local default -- not defined by this document -- applies. If no prior mutual agreement exists between the client and the HC Proxy, then a Target CoAP URI without the scheme component is syntactically incorrect, and therefore:

- It MUST NOT be emitted by clients; and

- It MUST elicit a suitable client error status (i.e., 4xx) by the HC Proxy.

5.3.2. 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. For example: coap://[2001:db8::1]/light?on becomes https://p.example.com/hc/coap://%5B2001:db8::1%5D/light?on. (Note that the percent-encoded square brackets shall be reverted to their non-percent-encoded form when the HC Proxy unpacks the Target CoAP URI.)

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

5.4. 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. This can then be used by the HTTP client to construct the effective request URI to be sent in the HTTP request to the HC Proxy.

When instantiated, a URI mapping template is always concatenated to an HC Proxy URI provided by the HC Proxy via discovery (see Section 5.5), or by other means.

A simple form (Section 5.4.1) and an enhanced form (Section 5.4.2) 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].

5.4.1. Simple Form

The simple form MUST be used for mappings where the Target CoAP URI is going to be copied (using rules of Section 5.3.2) at some fixed position into the Hosting HTTP URI.

The "tu" template variable is defined below using the ABNF rules from [RFC3986], Sections 3.2.2, 3.2.3, 3.3, and 3.4. It is intended to be used in a template definition to represent a Target CoAP URI:

```
  tu = [ ( "coap:" / "coaps:" ) "//" ] host [ ":" port ] path-abempty
       [ "?" query ]
```

Note that the same considerations as in Section 5.3.1 apply, in that the CoAP scheme may be omitted from the Hosting HTTP URI.

5.4.1.1. Examples

All the following examples (given as a specific URI mapping template, a Target CoAP URI, and the produced Hosting HTTP URI) use https://p.example.com/hc/ as the HC Proxy URI. Note that these examples all define mapping templates that deviate from the default template of Section 5.3 in order to illustrate the use of the above template variables.

1. Target CoAP URI is a query argument of the Hosting HTTP URI:

```
?target_uri={+tu}

coap://s.example.com/light

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

whereas

coaps://s.example.com/light

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

2. Target CoAP URI in the path component of the Hosting HTTP URI:

```
forward/{+tu}

coap://s.example.com/light

=> https://p.example.com/hc/forward/coap://s.example.com/light

whereas

coaps://s.example.com/light

=> https://p.example.com/hc/forward/coaps://s.example.com/light
```

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

```
?coap_uri={+tu}

coap://s.example.com/light

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

5.4.2. Enhanced Form

The enhanced form can be used to express more sophisticated mappings of the Target CoAP URI into the Hosting HTTP URI, i.e., mappings that do not fit into the simple form.

There MUST be at most one instance of each of the following template variables in a URI mapping 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 if and only if 'query' is empty
```

The qq form is used when the path and the (optional) query components are to be copied verbatim from the Target CoAP URI into the Hosting HTTP URI, i.e., as "{+p}{+qq}". Instead, the q form is used when the query and path are mapped as separate entities, e.g., as in "coap_path={+p}&coap_query={+q}". So q and qq MUST be used in mutual exclusion in a template definition.

5.4.2.1. Examples

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

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

```
{+s}/{+hp}{+p}{+qq}

coap://s.example.com/light

=> https://p.example.com/hc/coap/s.example.com/light
```

```
        whereas

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

        => https://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

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

      whereas

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

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

5.5. 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 can be used as the value for the "rt" attribute in a query to the "/.well-known/core" resource in order to locate the URI where the HC Proxy function is anchored, i.e., the HC Proxy URI.

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 with the mapping resource. The default template given in Section 5.3, i.e., {+tu}, MUST be assumed if no "hct" attribute is found in a returned link. If a "hct" attribute is present in a returned link, the client MUST use it to create a Hosting HTTP URI.

The URI mapping SHOULD be discoverable (as specified in [RFC6690]) on both the HTTP and the CoAP side of the HC Proxy, with one important difference: on the CoAP side, the link associated with the "core.hc" resource always needs an explicit anchor parameter referring to the HTTP origin [RFC6454], while on the HTTP interface, the context URI of the link may be equal to the HTTP origin of the discovery request: in that case, the anchor parameter is not needed.

5.5.1. Examples

- The first example exercises the CoAP interface and assumes that the default template, {+tu}, is used. For example, a smartphone may discover the public HC Proxy before leaving the home network. Then, when outside the home network, the smartphone will be able to query the appropriate home sensor.

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

Res:  2.05 Content
      </hc/>;anchor="https://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::fd]/.well-known/core?rt=core.hc

Res:  2.05 Content
      </hc/>;anchor="https://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: 19

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

- using the 'application/link-format+json' content type as defined in [CoRE-JSON-CBOR]:

```
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: 32

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

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 codings (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, in 2.xx (i.e., successful) responses going from CoAP to HTTP, and in 4.xx/5.xx error responses with a diagnostic payload. 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 that 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 Section 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 an Accept option.

While the CoAP-to-HTTP direction always has 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 six 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 hand, when the HC Proxy is a general purpose ALG, being too strict could significantly reduce the amount of traffic that it would 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.

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 that 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", if the HC Proxy doesn't recognize the content-format '60'.

An 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. However, unless explicitly configured to allow pass-through of unknown content-formats, the HC Proxy SHOULD NOT forward requests carrying a Content-Type or Accept header with an "application/coap-payload", and return an appropriate client error instead.

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 coding 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. It is expected that an implementation can refine it because either application-specific knowledge is given or new Content-Formats are defined. 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 pattern	Generalized media type
application/*+xml	application/xml
application/*+json	application/json
application/*+cbor	application/cbor
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; it can be used as a building block for translating HTTP Content-Type and Accept headers into CoAP Content-Format and Accept Options.

The algorithm uses an IANA-maintained table, "CoAP Content-Formats", as established by 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 assumes an "identity" Content-Encoding and expects the resource body has been already successfully content decoded or transcoded to the desired format.

In the following (Figure 2):

- o media_type is the media type to translate;
- o coap_cf_registry is a lookup table matching the "CoAP Content-Formats" registry; and
- o loose_mapper is an optional lookup table describing the loose media type mappings (e.g., the one defined in Table 1).

The full source code is provided in Appendix A.

```
def mt2cf(media_type, encoding=None,
          coap_cf_registry=CoAPContentFormatRegistry(),
          loose_mapper=None):
    """Return a CoAP Content-Format given an Internet media type and
       its optional encoding.  The current (as of 2016/10/24) "CoAP
       Content-Formats" registry is supplied by default.  An optional
       ’loose-mapping’ implementation can be supplied by the caller."""
    assert media_type is not None
    assert coap_cf_registry is not None

    # Lookup the "CoAP Content-Formats" registry
    content_format = coap_cf_registry.lookup(media_type, encoding)

    # If an exact match is not found and a loose mapper has been
    # supplied, try to use it to get a media type with which to
    # retry the "CoAP Content-Formats" registry lookup.
    if content_format is None and loose_mapper is not None:
        content_format = coap_cf_registry.lookup(
            loose_mapper.lookup(media_type), encoding)

    return content_format
```

Figure 2

6.5. Content Transcoding

6.5.1. General

Payload content transcoding is an OPTIONAL feature. Implementations supporting this feature should provide a flexible way to define the set of transcodings allowed.

The HC Proxy might decide to transcode the received representation to a different (compatible) format 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 Concise Binary Object Representation (CBOR) [RFC7049], effectively achieving compression without losing any information.

However, there are a few important factors to keep in mind when enabling a transcoding function:

1. Maliciously crafted inputs coming from the HTTP side might inflate in size (see, for example, Section 4.2 of [RFC7049]), therefore creating a security threat for both the HC Proxy and the target resource.

2. Transcoding can lose information in non-obvious ways. 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. That means that whenever the HC Proxy transcodes "application/xml" to "application/exi", in-band metadata could be lost.

3. When the Content-Type is mapped, there is a risk that the content with the destination type would have malware not active in the source type.

It is crucial that these risks are well understood and carefully weighed against the actual benefits before deploying the transcoding function.

6.5.2. CoRE Link Format

The CoRE Link Format [RFC6690] is a set of links (i.e., URIs and their formal relationships) that 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 translation process would inspect the forwarded traffic and attempt to rewrite 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 in retrieving 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 syntax and semantics defined in [RFC6690], 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.

6.6. Diagnostic Payloads

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]. If present, the CoAP diagnostic payload SHOULD be copied into the HTTP response body with the media type of the response set to "text/plain;charset=utf-8". The CoAP diagnostic payload MUST NOT be copied into the HTTP reason-phrase, since it potentially contains CR-LF characters that are incompatible with HTTP reason-phrase syntax.

7. Response Code Mapping

Table 2 defines the HTTP response status codes to which each CoAP response code SHOULD be mapped. Multiple HTTP status codes in the second column for a given CoAP response code indicates that multiple HTTP responses are possible for the same CoAP response code, depending on the conditions cited in the Notes (see the third column and text below the table).

CoAP Response Code	HTTP Status Code	Note
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	
2.31 Continue	N/A	10
4.00 Bad Request	400 Bad Request	
4.01 Unauthorized	403 Forbidden	5
4.02 Bad Option	400 Bad Request	6
	500 Internal Server Error	6
4.03 Forbidden	403 Forbidden	
4.04 Not Found	404 Not Found	
4.05 Method Not Allowed	400 Bad Request	7
	405 Method Not Allowed	7
4.06 Not Acceptable	406 Not Acceptable	
4.08 Request Entity Incomplt.	N/A	10
4.12 Precondition Failed	412 Precondition Failed	
4.13 Request Ent. Too Large	413 Payload Too Large	11
4.15 Unsupported Content-Fmt.	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. If present, this payload MUST be returned as an entity in the HTTP 201 response. Section 6.3.2 of [RFC7231] does not put any requirement on the format of the entity. (In the past, [RFC2616] did. Note that [RFC2616] has been obsoleted by [RFC7230].)

2. The HTTP code is 200 or 204, respectively, for the case where a CoAP server returns a payload or not. [RFC7231], Section 6.3 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 MUST 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 running a local cache. An unconditional HTTP GET that produces a cache-hit could trigger a revalidation (i.e., a conditional GET) on the CoAP side. The proxy receiving 2.03 updates the freshness of its cached representation and returns it to the HTTP client.

5. An HTTP 401 Unauthorized (Section 3.1 of [RFC7235]) response is not applicable because there is no equivalent of WWW-Authenticate in CoAP, which is mandatory in an HTTP 401 response.

6. If the proxy has a way to determine that the Bad Option is due to the straightforward mapping of a client request header into a CoAP option, then returning HTTP 400 (Bad Request) is appropriate. In all other cases, the proxy MUST return HTTP 500 (Internal Server Error) stating its inability to provide a suitable translation to the client’s request.

7. A CoAP 4.05 (Method Not Allowed) response SHOULD normally be mapped to an HTTP 400 (Bad Request) 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 an HTTP reason-phrase in the HTTP status line that starts with the string "CoAP server returned 4.05" 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 ([CoRE-RD])), then it MAY send back an 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.

10. Only used in CoAP block-wise transfer [RFC7959] between HC Proxy and CoAP server; never translated into an HTTP response.

11. Only returned to the HTTP client if the HC Proxy was unable to successfully complete the request by retrying it with CoAP block-wise transfer; see Section 8.3.

8. Additional Mapping Guidelines

8.1. Caching and Congestion Control

An HC Proxy should cache CoAP responses and reply whenever applicable with a cached representation of the requested resource.

If the HTTP client drops the 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 requests to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the HC Proxy should also put a limit on the number of concurrent CoAP requests pending on the same constrained network; further incoming requests may either be queued or be dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior should be configurable.

Highly volatile resources that are being frequently requested may be observed [RFC7641] by the HC Proxy to keep their cached representation fresh while minimizing the amount 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 [RFC7641] to handle proxy cache refresh is preferable to the validation mechanism based on the entity-tag (ETag) as defined in [RFC7252]. Such scenarios include 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. The idea being that the HC Proxy proactively installs an observation on a "popular enough" resource and actively monitors:

a. Its update pattern on the CoAP side

b. The request pattern on the HTTP side

and uses the formula below to determine whether the observation should be kept alive or shut down.

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 if and only if T_R < 2*T_C with respect to using ETag validation, that is, if and only if 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 Block-Wise Transfer

An HC Proxy SHOULD support CoAP block-wise transfers [RFC7959] 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 endpoints as described in [RFC7252], Section 4.6.

An HC Proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with block-wise 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 block-wise 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 endpoints, 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 endpoints not supporting block-wise transfers. This can be done by checking for a 4.02 (Bad Option) response returned by an endpoint 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 block-wise transfers to CoAP servers that do not support it.

8.4. 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. If the 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 proxy scenario, an 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.

For further considerations related to the handling of multicast requests, see Section 10.1.

8.5. Timeouts

If the CoAP server takes a long time in responding, the HTTP client or any other proxy in between may timeout. Further discussion of timeouts in HTTP is available in Section 6.5 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 uses confirmable CoAP requests, such timeout value T SHOULD be

T = MAX_RTT + MAX_SERVER_RESPONSE_DELAY

where MAX_RTT is defined in [RFC7252] and MAX_SERVER_RESPONSE_DELAY is defined as the worst-case expected response delay of the CoAP server. If unknown, a default value of 250 seconds can be used for MAX_SERVER_RESPONSE_DELAY as in Section 2.5 of [RFC7390].

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 5.5 of RFC 8075.

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 an integer in range 0-65535 corresponding to a CoAP Content-Format parameter ([RFC7252], Section 12.3). The parameter "cf" 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, as defined by the "CoAP Content-Formats" subregistry that is part of the "Constrained RESTful Environments (CoRE) Parameters" registry).

Optional parameters: None

Encoding considerations: Common use is BINARY. The specific CoAP content-format encoding considerations for the selected Content-Format ("cf" parameter) apply. The encoding can vary based on the value of the "cf" parameter.

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

Interoperability considerations: This media type can never be used directly in CoAP messages because there are no means available to encode the mandatory "cf" parameter in CoAP.

Published specification: RFC 8075

Applications that use this media type: HTTP-to-CoAP proxies.

Fragment identifier considerations: CoAP does not support URI fragments; therefore, a CoAP payload fragment cannot be identified. Fragments are not applicable for this media type.

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: No

10. Security Considerations

The security considerations in Section 9.2 of [RFC7230] apply in full to the HC Proxy. This section discusses security aspects and requirements that are specific to the deployment and operation of an HC Proxy.

An HC Proxy located 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. In particular, its quality of implementation and operation -- i.e., use of current software development practices, careful selection of third-party libraries, sane configuration defaults, and an expedited way to upgrade a running instance -- are all essential attributes of the HC Proxy.

The correctness of request parsing in general (including any content transcoding), and of URI translation in particular, is essential to the security of the HC Proxy function. This is especially true when the constrained network hosts devices with genuinely limited capabilities. For this purpose, see also Sections 9.3, 9.4, 9.5 and 9.6 of [RFC7230] for well-known issues related to HTTP request parsing and Section 11.1 of [RFC7252] for an overview of CoAP-specific concerns related to URI processing -- in particular, the potential impact on access control mechanisms that are based on URIs.

An HC Proxy MUST implement Transport Layer Security (TLS) with a Pre-Shared Key (PSK) [RFC4279] and SHOULD implement TLS [RFC5246] with support for client authentication using X.509 certificates. A prerequisite of the latter is the availability of a Certification Authority (CA) to issue suitable certificates. Although this can be a challenging requirement in certain application scenarios, it is worth noting that there exist open-source tools (e.g., [OpenSSL]) that can be used to set up and operate an application-specific CA.

By default, the HC Proxy MUST authenticate all incoming requests prior to forwarding them to the CoAP server. This default behavior MAY be explicitly disabled by an administrator.

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

10.1. Multicast

Multicast requests impose a non-trivial cost on the constrained network and endpoints and might be exploited as a DoS attack vector (see also Section 10.2). From a privacy perspective, they can be used to gather detailed information about the resources hosted in the constrained network. For example, an outsider that is able to successfully query the "/.well-known/core" resource could obtain a comprehensive list of the target's home appliances and devices. From a security perspective, they can be used to carry out a network reconnaissance attack to gather information about possible vulnerabilities that could be exploited at a later point in time. For these reasons, it is RECOMMENDED that requests to multicast resources are access controlled with a default-deny policy. It is RECOMMENDED that the requestor of a multicast resource be strongly authenticated. If privacy and/or security are first class requirements, for example, whenever the HTTP request transits through the public Internet, the request SHOULD be transported over a mutually authenticated and encrypted TLS connection.

10.2. 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 an inefficient proxy implementation 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 that is mapped to a CoAP multicast resource, as discussed 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, which disrupts availability of the targeted CoAP server, can be limited by access controlling the exposed multicast resources, so that only known/authorized users can access such URIs.

10.3. 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 rules for security context translations. For example, all 'https' unicast requests are translated to 'coaps' requests, or 'https' requests are translated to unsecured 'coap' requests. Another rule could specify the security policy and parameters used for Datagram Transport Layer Security (DTLS) sessions [RFC7925]. Such rules will largely depend on the application and network context in which the HC Proxy operates. These rules should be configurable.

It is RECOMMENDED that, by default, accessing a 'coaps' URI is only allowed from a corresponding 'https' URI.

By default, an HC Proxy SHOULD reject any secured CoAP client request (i.e., one with a 'coaps' scheme) 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 means. Assuming that CoAP nodes are isolated behind a firewall as in the HC Proxy deployment shown in Figure 1, the HC Proxy may be configured to translate the incoming HTTPS request using plain CoAP (NoSec mode).

10.4. URI Mapping

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

DoS attack on the constrained/CoAP network.
Mitigation: by default, deny any Target CoAP URI whose authority is (or maps to) a multicast address. Then explicitly whitelist multicast resources/authorities that are allowed to be dereferenced. See also Section 8.4.

Leaking information on the constrained/CoAP network resources and topology.
Mitigation: by default, deny any Target CoAP URI (especially "/.well-known/core" is a resource to be protected), and then explicitly whitelist resources that are allowed to be seen by clients outside the constrained network.

The CoAP target resource is totally transparent from outside the constrained network.
Mitigation: implement an HTTPS-only interface, which makes the Target CoAP URI totally opaque to a passive attacker outside the constrained network.

11. References

11.1. 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, <http://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, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <http://www.rfc-editor.org/info/rfc4279>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <http://www.rfc-editor.org/info/rfc7232>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <http://www.rfc-editor.org/info/rfc7959>.

11.2. Informative References

[CoRE-JSON-CBOR] Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", Work in Progress, draft-ietf-core-links-json-06, July 2016.

[CoRE-RD] Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", Work in Progress, draft-ietf-core-resource-directory-09, October 2016.

[Fielding] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", PhD Dissertation, University of California, Irvine, ISBN 0-599-87118-0, 2000.

[OpenSSL] The OpenSSL Project, , "ca - sample minimal CA application", 2000-2016, <https://www.openssl.org/docs/manmaster/man1/ca.html>.

[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, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, DOI 10.17487/RFC2663, August 1999, <http://www.rfc-editor.org/info/rfc2663>.

[RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, DOI 10.17487/RFC3040, January 2001, <http://www.rfc-editor.org/info/rfc3040>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <http://www.rfc-editor.org/info/rfc4732>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <http://www.rfc-editor.org/info/rfc6454>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <http://www.rfc-editor.org/info/rfc7390>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <http://www.rfc-editor.org/info/rfc7925>.

[W3C.REC-html5-20141028]
Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O’Connor, E., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, October 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

Appendix A. Media Type Mapping Source Code

```
#!/usr/bin/env python

import unittest
import re

class CoAPContentFormatRegistry(object):
    """Map an Internet media type (and optional inherent encoding) to a
       CoAP Content-Format.
    """
    TEXT_PLAIN = 0
    LINK_FORMAT = 40
    XML = 41
    OCTET_STREAM = 42
    EXI = 47
    JSON = 50
    CBOR = 60
    GROUP_JSON = 256

# http://www.iana.org/assignments/core-parameters
# as of 2016/10/24.
    LOOKUP_TABLE = {
        ("text/plain;charset=utf-8", None): TEXT_PLAIN,
        ("application/link-format", None): LINK_FORMAT,
        ("application/xml", None): XML,
        ("application/octet-stream", None): OCTET_STREAM,
        ("application/exi", None): EXI,
        ("application/json", None): JSON,
        ("application/cbor", None): CBOR,
        ("application/coap-group+json", "utf-8"): GROUP_JSON,
    }

    def lookup(self, media_type, encoding):
        """Return the CoAP Content-Format matching the supplied
           media type (and optional encoding), or None if no
           match can be found."""
        return CoAPContentFormatRegistry.LOOKUP_TABLE.get(
            (media_type, encoding), None)
```

```
class LooseMediaTypeMapper(object):
    # Order matters in this table: more specific types have higher rank
    # compared to less specific types.
    # This code only performs a shallow validation of acceptable
    # characters and assumes overall validation of the media type and
    # subtype has been done beforehand.
    LOOKUP_TABLE = [
        (re.compile("application/.+\+xml$"), "application/xml"),
        (re.compile("application/.+\+json$"), "application/json"),
        (re.compile("application/.+\+cbor$"), "application/cbor"),
        (re.compile("text/xml$"), "application/xml"),
        (re.compile("text/[a-z\.\-\+]+$"), "text/plain;charset=utf-8"),
        (re.compile("[a-z]+/[a-z\.\-\+]+$"), "application/octet-stream")
    ]

    def lookup(self, media_type):
        """Return the best loose media type match available using
           the contents of LOOKUP_TABLE."""
        for entry in LooseMediaTypeMapper.LOOKUP_TABLE:
            if entry[0].match(media_type) is not None:
                return entry[1]
        return None


def mt2cf(media_type, encoding=None,
          coap_cf_registry=CoAPContentFormatRegistry(),
          loose_mapper=None):
    """Return a CoAP Content-Format given an Internet media type and
       its optional encoding.  The current (as of 2016/10/24) "CoAP
       Content-Formats" registry is supplied by default.  An optional
       'loose-mapping' implementation can be supplied by the caller."""
    assert media_type is not None
    assert coap_cf_registry is not None

    # Lookup the "CoAP Content-Formats" registry
    content_format = coap_cf_registry.lookup(media_type, encoding)

    # If an exact match is not found and a loose mapper has been
    # supplied, try to use it to get a media type with which to
    # retry the "CoAP Content-Formats" registry lookup.
    if content_format is None and loose_mapper is not None:
        content_format = coap_cf_registry.lookup(
            loose_mapper.lookup(media_type), encoding)

    return content_format
```

```
class TestMT2CF(unittest.TestCase):

    def testMissingContentType(self):
        with self.assertRaises(AssertionError):
            mt2cf(None)

    def testMissingContentFormatRegistry(self):
        with self.assertRaises(AssertionError):
            mt2cf(None, coap_cf_registry=None)

    def testTextPlain(self):
        self.assertEqual(mt2cf("text/plain;charset=utf-8"),
                         CoAPContentFormatRegistry.TEXT_PLAIN)

    def testLinkFormat(self):
        self.assertEqual(mt2cf("application/link-format"),
                         CoAPContentFormatRegistry.LINK_FORMAT)

    def testXML(self):
        self.assertEqual(mt2cf("application/xml"),
                         CoAPContentFormatRegistry.XML)

    def testOctetStream(self):
        self.assertEqual(mt2cf("application/octet-stream"),
                         CoAPContentFormatRegistry.OCTET_STREAM)

    def testEXI(self):
        self.assertEqual(mt2cf("application/exi"),
                         CoAPContentFormatRegistry.EXI)

    def testJSON(self):
        self.assertEqual(mt2cf("application/json"),
                         CoAPContentFormatRegistry.JSON)

    def testCBOR(self):
        self.assertEqual(mt2cf("application/cbor"),
                         CoAPContentFormatRegistry.CBOR)

    def testCoAPGroupJSON(self):
        self.assertEqual(mt2cf("application/coap-group+json",
                               "utf-8"),
                         CoAPContentFormatRegistry.GROUP_JSON)

    def testUnknownMediaType(self):
        self.assertFalse(mt2cf("unknown/media-type"))
    def testLooseXML1(self):
        self.assertEqual(
            mt2cf(
                "application/somesubtype+xml",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.XML)

    def testLooseXML2(self):
        self.assertEqual(
            mt2cf(
                "text/xml",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.XML)

    def testLooseJSON(self):
        self.assertEqual(
            mt2cf(
                "application/somesubtype+json",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.JSON)

    def testLooseCBOR(self):
        self.assertEqual(
            mt2cf(
                "application/somesubtype+cbor",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.CBOR)

    def testLooseText(self):
        self.assertEqual(
            mt2cf(
                "text/somesubtype",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.TEXT_PLAIN)

    def testLooseUnknown(self):
        self.assertEqual(
            mt2cf(
                "application/somesubtype-of-some-sort+format",
                loose_mapper=LooseMediaTypeMapper()),
            CoAPContentFormatRegistry.OCTET_STREAM)

    def testLooseInvalidStartsWithNonAlpha(self):
        self.assertFalse(
            mt2cf(
                " application/somesubtype",
                loose_mapper=LooseMediaTypeMapper()))
    def testLooseInvalidEndsWithUnexpectedChar(self):
        self.assertFalse(
            mt2cf(
                "application/somesubtype ",
                loose_mapper=LooseMediaTypeMapper()))

    def testLooseInvalidUnexpectedCharInTheMiddle(self):
        self.assertFalse(
            mt2cf(
                "application /somesubtype",
                loose_mapper=LooseMediaTypeMapper()))

    def testLooseInvalidNoSubType1(self):
        self.assertFalse(
            mt2cf(
                "application",
                loose_mapper=LooseMediaTypeMapper()))

    def testLooseInvalidNoSubType2(self):
        self.assertFalse(
            mt2cf(
                "application/",
                loose_mapper=LooseMediaTypeMapper()))

if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Acknowledgments

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 Abhijan Bhattacharyya, Alexey Melnikov, Brian Frank, Carsten Bormann, Christian Amsuess, Christian Groves, Cullen Jennings, Dorothy Gellert, Francesco Corazza, Francis Dupont, Hannes Tschofenig, Jaime Jimenez, Kathleen Moriarty, Kepeng Li, Kerry Lynn, Klaus Hartke, Larry Masinter, Linyi Tian, Michele Rossi, Michele Zorzi, Nicola Bui, Peter Saint-Andre, Sean Leonard, Spencer Dawkins, Stephen Farrell, Suresh Krishnan, and Zach Shelby 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.

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
Nokia
3 Ely Road
Milton, Cambridge CB24 6DD
United Kingdom

Email: thomas.fossati@nokia.com

Esko Dijk
Philips Lighting
High Tech Campus 7
Eindhoven 5656 AE
The Netherlands

Email: esko.dijk@philips.com