CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2015

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

July 4, 2014

Guidelines for HTTP-CoAP Mapping Implementations
draft-ietf-core-http-mapping-04

Abstract

This draft provides reference information for HTTP-CoAP protocol translation proxy implementation, focusing on the reverse proxy case. It details deployment options, defines a template for URI mapping, and provides a set of guidelines and considerations related to protocol translation.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 5, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

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 implement proxies that correctly inter-work with other CoAP and HTTP client/server implementations that adhere to the HTTP and CoAP specifications.

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 of adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability. (As an example use case, 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 draft is organized as follows:

- Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- Section 3 discusses how URIs refer to resources independent of access protocols;
- Section 4 briefly 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 the properties of the HTTP-to-CoAP reverse proxy;
- Section 8 discusses possible security impact related to HTTP-CoAP protocol mapping.

2. Terminology

This document assumes readers are familiar with the terms Reverse Proxy as defined in [RFC7230] and Interception Proxy as defined in [RFC3040]. In addition, the following terms are defined:

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

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/dereferencing agent for a predefined subset of the URI space.

Reverse Proxy: a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. It 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" [RFC7230] as a request-target URI.

Reverse and Forward proxies are technically very similar, with main differences being that the former appears to a client as an origin server while the latter does not, and that clients may be unaware they are communicating with a proxy.

Placement terms: a server-side (SS) proxy is placed in the same network domain as the server; conversely a client-side (CS) proxy is in the same network domain as the client. In any other case than SS or CS, the proxy is said to be External (E).

The key words "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].

3. Cross-Protocol Usage of URIs

A Uniform Resource Identifier (URI) provides a simple and extensible method for identifying a resource. It enables uniform identification of resources via a separately defined extensible set of naming schemes [RFC3986].

URIs are formed of at least three components: scheme, authority and path. The scheme often corresponds to the protocol used to access the resource. However, as noted in Section 1.2.2 of [RFC3986] the scheme does not imply that a particular protocol is used to access the resource. So, we can define the same resource to be accessible by different protocols i.e. the resource can have cross-protocol URIs referring to it.

HTTP clients only support 'http' and 'https' schemes and cannot directly access CoAP servers (which support 'coap' and/or 'coaps'). In this situation, communication is enabled by a HC Proxy, as shown in Figure 1, supporting URI mapping features. Such features are discussed in Section 5.

4. Use Cases

To illustrate in which situations HTTP to CoAP request mapping may be used, three use cases are briefly described.

1. Smartphone and home sensor: Any smartphone can access directly a home sensor using an authenticated 'https' request, if its home router contains a HTTP-CoAP proxy. For this use-case an HTML5 application can be built providing a friendlier UI to the user.

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

3. Making sensor data available to 3rd parties: For demonstration or public interest purposes, a HTTP-CoAP proxy may be configured to expose the contents of a sensor to the world via the web (HTTP and/or HTTPS). The sensor can only handle secure 'coaps' requests, therefore the proxy is configured to translate any request to a 'coaps' secured request. The proxy is furthermore configured to only pass through GET requests. 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 a HC Proxy, the reality is that all major web browsers and command line tools do not allow making HTTP requests using URIs with a scheme different from "http" or "https".

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 in to 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 [RFC6690] through which clients of a 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, it has a scheme of "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 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 in an unambiguous way from the Hosting HTTP URI. Also 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 a HC Proxy, unless there are valid reasons, e.g. application specific, to use a different mapping function.

5.2.1. Optional scheme

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 used by a 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 template's to take care of the expansion of values that are allowed to include reserved URI characters.

5.3.1. Simple Form

The simple form MUST be used for mappings where the Target CoAP URI is going to be copied verbatim 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 done in Section 5.2.1 apply.

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.

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

5.3.2. 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
```

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}{+q}

   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 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" MAY be returned in a "core.hc" link to provide the associated URI Mapping Template. 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 an "htc" attribute is present in the returned link, then a compliant client MUST use it to create the Hosting HTTP URI.

Discovery 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. 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 serialised 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:

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

6. HTTP-CoAP Reverse Proxy

A HTTP-CoAP Reverse Cross-Protocol Proxy is accessed by web clients only supporting HTTP, and handles their requests by mapping these to CoAP requests, which are forwarded to CoAP servers; and mapping back the received CoAP responses to HTTP. 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" [RFC7230] as a Request Target.

Normative requirements on the translation of HTTP requests to CoAP 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 HTTP-CoAP Forward Cross-Protocol 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. The present section provides guidelines and more details for the implementation of a Reverse Cross-Protocol Proxy, which MAY be followed in addition to the normative requirements.

Translation of unicast HTTP requests into multicast CoAP requests is currently out of scope since in a reverse proxy scenario a HTTP client typically expects to receive a single response, not multiple. However a HC Proxy MAY include custom application-specific functions to generate a multicast CoAP request based on a unicast HTTP request and aggregate multiple CoAP responses into a single HTTP response.

Note that the guidelines in this section also apply to an HTTP-CoAP Intercepting Cross-Protocol Proxy.

6.1. Proxy Placement

Typically, a Cross-Protocol Proxy is located at the edge of the constrained network. See Figure 1. The arguments supporting server-side (SS) placement are the following:

Caching: Efficient caching requires that all request traffic to a CoAP server is handled by the same proxy which receives HTTP requests from multiple source locations. This maximally reduces the load on (constrained) CoAP servers.

Multicast: To support CoAPs use of local-multicast functionality available in a constrained network, the Cross-Protocol Proxy requires a network interface directly attached to the constrained network.

TCP/UDP: Translation between HTTP and CoAP requires also TCP/UDP translation; TCP may be the preferred way for communicating with the constrained network due to its reliability or due to intermediate gateways configured to block UDP traffic.

Arguments against SS placement, in favor of client-side (CS), are:

Scalability: A solution where a single SS proxy has to manage numerous open TCP/IP connections to a large number of HTTP clients is not scalable. (Unless multiple SS proxies are employed with a load-balancing mechanism, which adds complexity.)

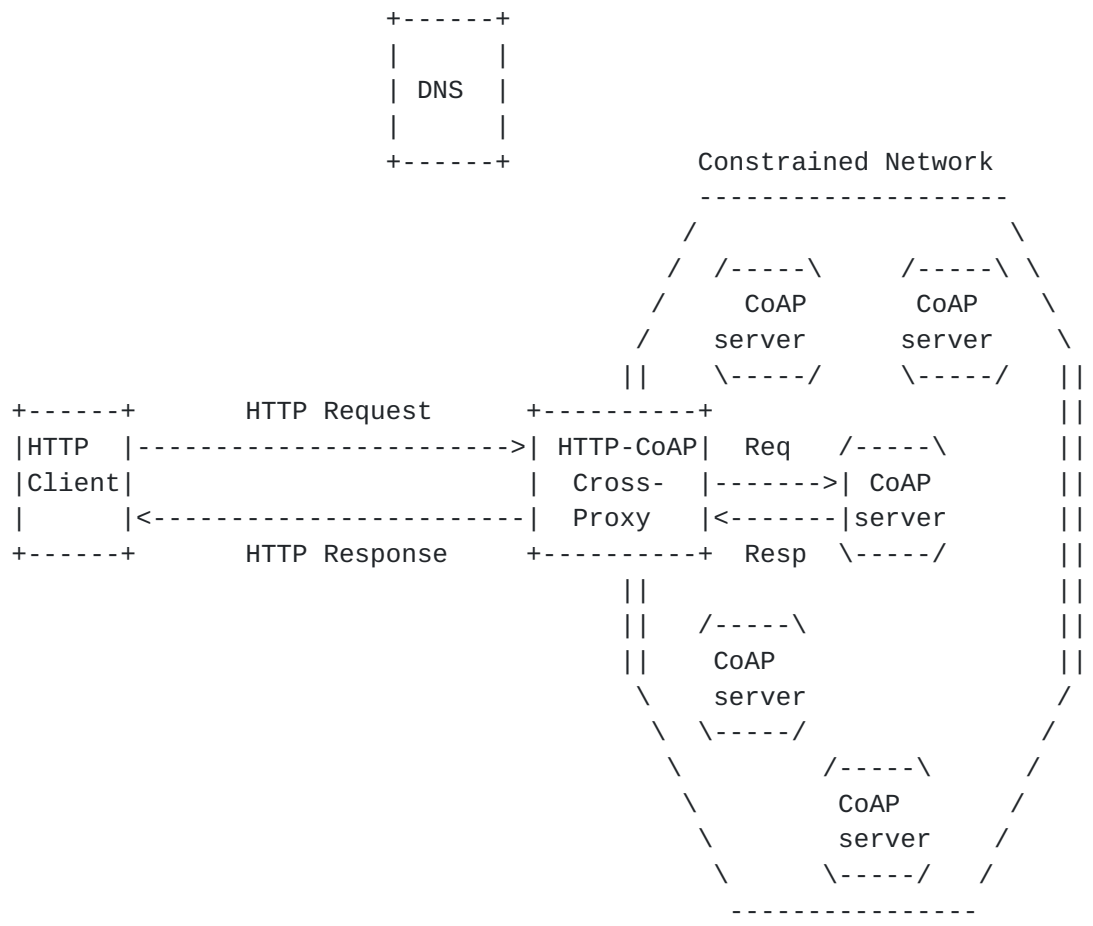

Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

6.2. Response Code Translations

Table 1 defines all possible CoAP responses along with the HTTP response to which each CoAP response SHOULD be translated. This table complies with the Section 10.2 requirements 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, depending on the conditions cited in the Notes (third column).

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	400 Bad Request	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 Entity 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 1: HTTP-CoAP Response Mapping

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 payload. (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 and PUT and code 204 if not. Hence, a proxy SHOULD transfer any CoAP payload contained in a 2.02 response to the HTTP client in a 200 OK response.

3. A CoAP 2.03 (Valid) response only (1) confirms that the request ETag is valid and (2) provides a new Max-Age value. HTTP 304 (Not Modified) also updates some header fields of a stored response. A non-caching proxy may not have enough information to fill in the required values in the HTTP 304 (Not Modified) response, so it may not be advisable for a non-caching proxy to provoke the 2.03 (Valid) response by forwarding an ETag. A caching proxy will fill the information out of the cache.

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

5. The HTTP code 401 Unauthorized MUST NOT be used, as long as in CoAP there is no equivalent defined of the required WWW-Authenticate header (Section 3.1 of [RFC7235]).

6. In some cases a proxy receiving 4.02 may retry the request with less CoAP Options in the hope that the server will understand the newly formulated request. For example, if the proxy tried using a Block Option which was not recognised by the CoAP server it may retry without that Block Option.

7. The HTTP code "405 Method Not Allowed" MUST NOT be used since CoAP does not provide enough information to determine a value for the required "Allow" response-header field.

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 to execute some, or all, of its CoAP requests.

6.3. Media Type mapping

A HC Proxy translates HTTP media types (Section 3.1.1.1 of [RFC7231]) and content encodings (Section 3.1.2.1 of [RFC7231]) into CoAP content formats (Section 12.3 of [RFC7252]).

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 the Content-Type and Content-Encoding HTTP headers into a CoAP Content-Format option, whereas GET needs to map Accept and Accept-Encoding HTTP

headers into a CoAP Accept option. On the way back, the CoAP Content-Format option is renormalised into 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.

If the HC Proxy receives a CoAP response with a Content-Format that it does not recognise (for example because the value has been registered after the proxy has been deployed), then it is allowed to either return a HTTP entity without a Content-Type header, or examine the data to determine its type on the fly.

On the content negotiation side, failing to map Accept and Accept-Encoding headers SHOULD be silently ignored: the HC Proxy SHOULD therefore forward the request with no Accept option.

While the CoAP to HTTP direction has always a well defined mapping, 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 cases, the "loose" media-type mapping detailed in Section 6.3.1 MAY be implemented.

The latter grants unconstrained 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 boundary of the communication path. Therefore, when applied, other forms of access control must be set in place to avoid unauthorised users to deplete or abuse systems and network resources.

6.3.1. 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 specialisations of a more generic media-type can be aliased to their super-class and then mapped (if possible) to one of 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 fall back when no better alias is found for a specific media-type.

Table 2 defines the default lookup table for the "loose" media-type mapping. Given an input media-type, the table returns its best generalised media-type using longest prefix match.

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

Table 2: Media type generalisation

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 generalisations allowed.

6.3.2. Internet Media Type to Content Format Mapping Algorithm

This section defines the algorithm used to map an 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 6.3.1 can be used if the implementation chooses so.

Note that the algorithm may have side effects on the associated representation (see also Section 6.3.3).

In the following:

- C-T, C-E, and C-F stand for the values of the Content-Type (or Accept), Content-Encoding (or Accept-Encoding) HTTP headers, 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 generalised 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

6.3.3. Content Transcoding

As noted in Section 6.3.2, the process of mapping the type of the resource 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 optimised version of a specific format exists. For example an XML-encoded resource could be transcoded to EXI, or a JSON-encoded resource into CBOR [RFC7049], effectively achieving compression without losing any information.

Payload transcoding (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.

6.4. Caching and Congestion Control

A 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 a HC Proxy to the same CoAP resource SHOULD in general be avoided, by duplexing the response to the 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, a HC Proxy SHOULD wait for the associated CoAP response and cache it if possible. Further 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 session.

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 efficiently apply this congestion control, the HC Proxy SHOULD be SS 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 CoAP messages. See Section 6.5.

6.5. 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 nodes -- with possibly high variance in requests' distribution -- which would greatly benefit from a server driven cache update mechanism. Ideal candidates would also be 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 the refresh of 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 F_R be the freshness lifetime of R representation, and let M_R be the total number of messages exchanged towards 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*F_R with respect to using ETag validation, that is iff the mean arrival time of requests for resource R is greater than half the refresh rate of R.

When using observations M_R is always upper bounded by 2*F_R: in the constrained network no more than 2*F_R messages will be generated towards resource R.

6.6. Use of CoAP Blockwise Transfer

A 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 LLNs, and to cope with small datagram buffers in CoAP end-points as described in [RFC7252] Section 4.6.

A 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. A HC Proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than a value BLOCKWISE_THRESHOLD. The value of BLOCKWISE_THRESHOLD MAY be implementation-specific, for example calculated based on a known or typical UDP datagram buffer size for CoAP end-points, or set to N times the size of a link-layer frame 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 parameters from which it is calculated SHOULD be configurable in a proxy implementation.

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. 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 413 (Request Entity Too Large) to the HTTP client.

For improved latency a 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.

6.7. Security Translation

A HC proxy SHOULD implement explicit rules for security context translations. 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. 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 is applied. To enable widest possible use of a proxy implementation, these rules SHOULD be configurable in a HC proxy.

If a policy for access to 'coaps' URIs is configurable in a 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.

6.8. Other guidelines

For long delays of a CoAP server, 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].

A HC Proxy MUST define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request. The timeout value SHOULD be approximately less than or equal to MAX_RTT defined in [RFC7252].

When the DNS protocol is not used between CoAP nodes 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 a HC proxy.

HTTP connection pipelining (section 6.2.2.1 of [RFC7230]) MAY be supported by the proxy and is transparent to the CoAP network: the HC Proxy will sequentially serve the pipelined requests by issuing different CoAP requests.

It is expected that the HC function will often be implemented in software on the proxy. Many different software approaches are possible, including using CGI [RFC3875] as an interface between the HTTP layer and the protocol translation engine.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

The security concerns raised in Section 15.7 of [RFC2616] 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. Even if we had a blind, bi-directional, end-to-end, tunneling facility like the one provided by the CONNECT method in HTTP, and also assuming the existence of a DTLS-TLS transparent mapping, the two tunneled ends should be speaking the same application protocol, which is not the case. Basically, the protocol translation function is a core duty of the HC Proxy that can't be removed, and makes it 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, a special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it could have fewer limitations than the constrained devices it is serving.

The following sub paragraphs categorize and argue about a set of specific security issues related to the translation, caching and forwarding functionality exposed by a HC Proxy module.

8.1. Traffic overflow

Due to the typically constrained nature of CoAP nodes, particular attention SHOULD be posed in the implementation of traffic reduction mechanisms (see Section 6.4), because inefficient 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 mapped to a CoAP multicast resource, as considered in Section TBD of [RFC7252].

The impact 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 resource, so that only known/authorized users access such URIs.

8.2. Handling Secured Exchanges

It is possible that the request from the client to the HC Proxy is sent over a secured connection. However, there may or may not exist a secure connection mapping to the other protocol. For example, a secure distribution method for multicast traffic is complex and MAY not be implemented (see [I-D.ietf-core-groupcomm]).

By default, a 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 the SS HC proxy deployment shown in Figure 1), the HC Proxy may be configured to translate the incoming HTTPS request using plain CoAP (i.e. NoSec mode.)

The HC URI mapping MUST NOT map to HTTP (see Section 5) a CoAP resource intended to be accessed only using HTTPS.

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.

8.3. URI Mapping

The following risks related to the URI mapping described in Section 5 have been identified:

DoS attack on the internal network.
: Default deny any Target CoAP URI whose authority is (or maps to) a multicast address. Then explicitly whitelist multicast resources that are allowed to be de-referenced.

Leaking information on the internal network resources and topology.
: Default deny any Target CoAP URI (especially /.well-known/core is the resource to be protected), and then explicit whitelist 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 implementing a HTTPS-only interface makes the Target CoAP URI totally opaque to a passive attacker.

9. Acknowledgements

An initial version of the table found in Section 6.2 has been provided in revision -05 of [RFC7252]. 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].

10. References

10.1. Normative References

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 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.

[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.

[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.

10.2. Informative References

[I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-19 (work in progress), June 2014.

[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.

[RFC3875] Robinson, D. and K. Coar, "The Common Gateway Interface (CGI) Version 1.1", RFC 3875, October 2004.

[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.

Appendix A. Change Log

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

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 HC 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