CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2016

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

July 2, 2015

Advanced Guidelines for HTTP-CoAP Mapping Implementations
draft-castellani-core-advanced-http-mapping-06

Abstract

This draft describes advanced features for HTTP-CoAP proxy implementers. It details deployment options, discusses possible approaches for URI mapping, and provides useful 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 3, 2016.

Table of Contents

1. Terminology and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap] . In addition, this document defines the following terminology:

A device providing cross-protocol HTTP-CoAP mapping is called an HTTP-CoAP cross-protocol proxy (HC proxy).

At least two different kinds of HC proxies exist:

- One-way cross-protocol proxy (1-way proxy): This proxy translates from a client of a protocol to a server of another protocol but not vice-versa.

- Two-way (or bidirectional) cross-protocol proxy (2-way proxy): This proxy translates from a client of both protocols to a server supporting one protocol.

2. Introduction

RESTful protocols, such as HTTP [RFC2616] and CoAP [I-D.ietf-core-coap], can interoperate through an intermediary proxy which performs cross-protocol mapping.

A base reference for the mapping process is provided in [I-D.ietf-core-coap]. However, depending on the involved application, deployment scenario, or network topology, such mapping can be realized using a wide range of intermediaries.

Moreover, the process of implementing such a proxy can be complex, and details regarding its internal procedures and design choices deserve further discussion, which is provided in this document.

This draft itself is an evolution of the mapping features covered in [I-D.ietf-core-http-mapping].

3. Use Case: HTTP/IPv4-CoAP/IPv6 Proxy

This section covers the expected common use case regarding an HTTP/IPv4 client accessing a CoAP/IPv6 resource.

While HTTP and IPv4 are today widely adopted communication protocols in the Internet, a pervasive deployment of constrained nodes

exploiting the IPv6 address space is expected: enabling direct interoperability of such technologies is a valuable goal.

An HC proxy supporting IPv4/IPv6 mapping is said to be a v4/v6 proxy.

An HC v4/v6 proxy SHOULD always try to resolve the URI authority, and SHOULD prefer using the IPv6 resolution if available. The authority part of the URI is used internally by the HC proxy and SHOULD NOT be mapped to CoAP.

Figure 1 shows an HTTP client on IPv4 (C) accessing a CoAP server on IPv6 (S) through an HC proxy on IPv4/IPv6 (P). The DNS has an A record for "node.coap.something.net" resolving to the IPv4 address of the HC proxy, and an AAAA record with the IPv6 address of the CoAP server.

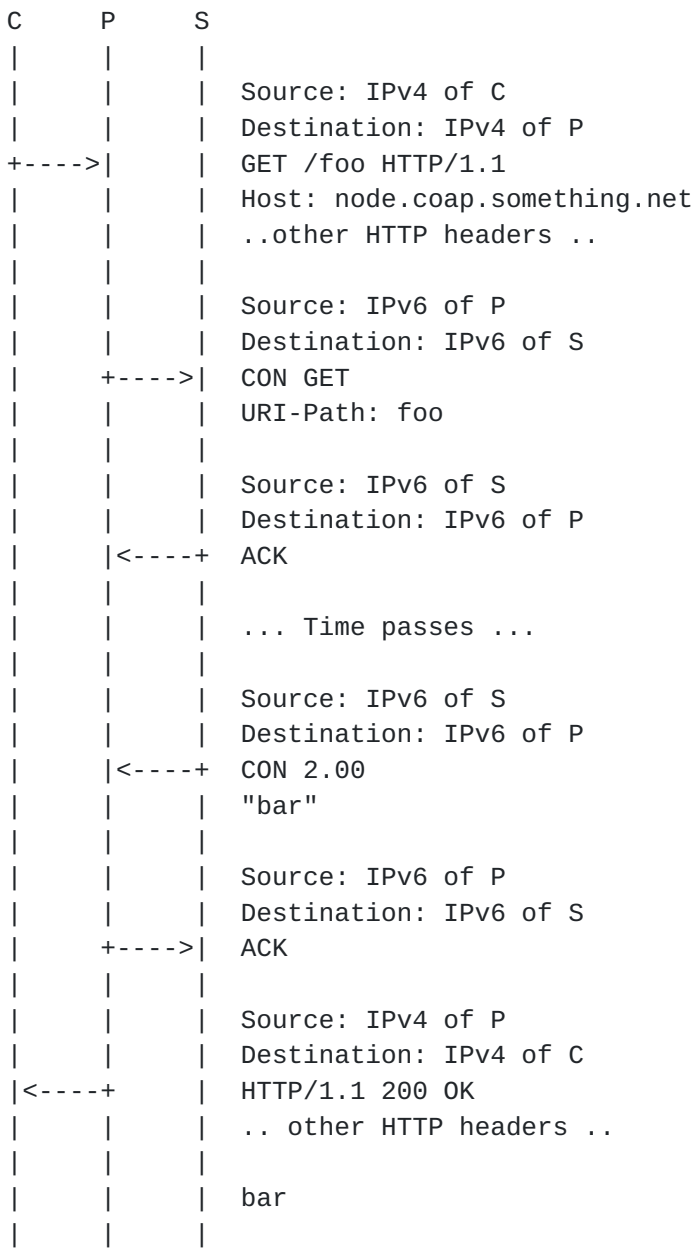

Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping

The proposed example shows the HC proxy operating also the mapping between IPv4 to IPv6 using the authority information available in any HTTP 1.1 request. This way, IPv6 connectivity is not required at the HTTP client when accessing a CoAP server over IPv6 only, which is a typical expected use case.

When P is an interception HC proxy, the CoAP request SHOULD have the IPv6 address of C as source (IPv4 can always be mapped into IPv6).

The described solution takes into account only the HTTP/IPv4 clients accessing CoAP/IPv6 servers; this solution does not provide a full fledged mapping from HTTP to CoAP.

In order to obtain a working deployment for HTTP/IPv6 clients, a different HC proxy access method may be required, or Internet AAAA records should not point to the node anymore (the HC proxy should use a different DNS database pointing to the node).

When an HC interception proxy deployment is used this solution is fully working even with HTTP/IPv6 clients.

4. URI Mapping via HTTP Cache Control Extensions

An advanced strategy for triggering the cross-proxy that a translation is needed can be done via the HTTP Cache Control Extensions described in Section 5.2.3 of [RFC7234]. Specifically two new extensions can be defined, i.e. cross-coap and cross-coaps, that when included in a request to an HC forward cross-proxy translate the request to coap or coaps.

5. Multiple Message Exchanges Mapping

This section discusses the mapping of the multicast and observe features of CoAP, which have no corresponding primitive in HTTP, and as such are not immediately translatable.

The mapping, which must be considered in both the arrow directions (H->C, C->H) may involve multi-part responses, as in the multicast use case, asynchronous delivery through HTTP bidirectional techniques, and HTTP Web Linking in order to reduce the semantics lost in the translation.

5.1. Relevant Features of Existing Standards

Various features provided by existing standards are useful to efficiently represent sessions involving multiple messages.

5.1.1. Multipart Messages

In particular, the "multipart/*" media type, defined in Section 5.1 of [RFC2046], is a suitable solution to deliver multiple CoAP responses within a single HTTP payload. Each part of a multipart entity SHOULD be represented using "message/http" media type containing the full mapping of a single CoAP response as previously described.

5.1.2. Immediate Message Delivery

An HC proxy may prefer to transfer each CoAP response immediately after its reception. This is possible thanks to the HTTP Transfer-Encoding "chunked", that enables transferring single responses without any further delay.

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [RFC6202]. Large delays between chunks can lead the HTTP session to timeout, more details on this issue can be found in [I-D.thomson-hybi-http-timeout].

An HC proxy MAY prefer (e.g. to avoid buffering) to transfer each response related to a multicast request as soon as it comes in from the server. One possible way to achieve this result is using the "chunked" Transfer-Encoding in the HTTP response, to push individual responses until some trigger is fired (timeout, max number of messages, etc.).

An example showing immediate delivery of CoAP responses using HTTP chunks will be provided in Section 5.4, while describing its application to an observe session.

5.1.3. Detailing Source Information

Under some circumstances, responses may come from different sources (i.e. responses to a multicast request); in this case details about the actual source of each CoAP response MAY be provided to the client. Source information can be represented using HTTP Web Linking as defined in [RFC5988], by adding the actual source URI into each response using Link option with "via" relation type.

5.2. Multicast Mapping

In order to establish a multicast communication such a feature should be offered either by the network (i.e. IP multicast, link-layer multicast, etc.) or by a gateway (i.e. the HC proxy). Rationale on the methods available to obtain such a feature is out-of-scope of this document, and extensive discussion of group communication techniques is available in [I-D.ietf-core-groupcomm].

Additional considerations related to handling multicast requests mapping are detailed in the following sections.

5.2.1. URI Identification and Mapping

In order to successfully handle a multicast request, the HC proxy MUST successfully perform the following tasks on the URI:

Identification: The HC proxy MUST understand whether the requested URI identifies a group of nodes.

Mapping: The HC proxy MUST know how to distribute the multicast request to involved servers; this process is specific of the group communication technology used.

When using IPv6 multicast paired with DNS, the mapping to IPv6 multicast is simply done using DNS resolution. If the group management is performed at the proxy, the URI or part of it (i.e. the authority) can be mapped using some static or dynamic table available at the HC proxy. In Section 3.5 of [I-D.ietf-core-groupcomm] discusses a method to build and maintain a local table of multicast authorities.

5.2.2. Request Handling

When the HC proxy receives a request to a URI that has been successfully identified and mapped to a group of nodes, it SHOULD start a multicast proxying operation, if supported by the proxy.

Multicast request handling consists of the following steps:

Multicast TX: The HC proxy sends out the request on the CoAP side by using the methods offered by the specific group communication technology used in the constrained network;

Collecting RXs: The HC proxy collects every response related to the request;

Timeout: The HC proxy has to pay special attention in multicast timing, detailed discussion about timing depends upon the particular group communication technology used;

Distributing RXs to the client: The HC proxy can distribute the responses in two different ways: batch delivering them at the end of the process or on timeout, or immediately delivering them as they are available. Batch requires more caching and introduces delays but may lead to lower TCP overhead and simpler processing. Immediate delivery is the converse. A trade-off solution of partial batch delivery may also be feasible and efficient in some circumstances.

5.2.3. Examples

Figure 2 shows an HTTP client (C) requesting the resource "/foo" to a group of CoAP servers (S1/S2/S3) through an HC proxy (P) which uses IP multicast to send the corresponding CoAP request.

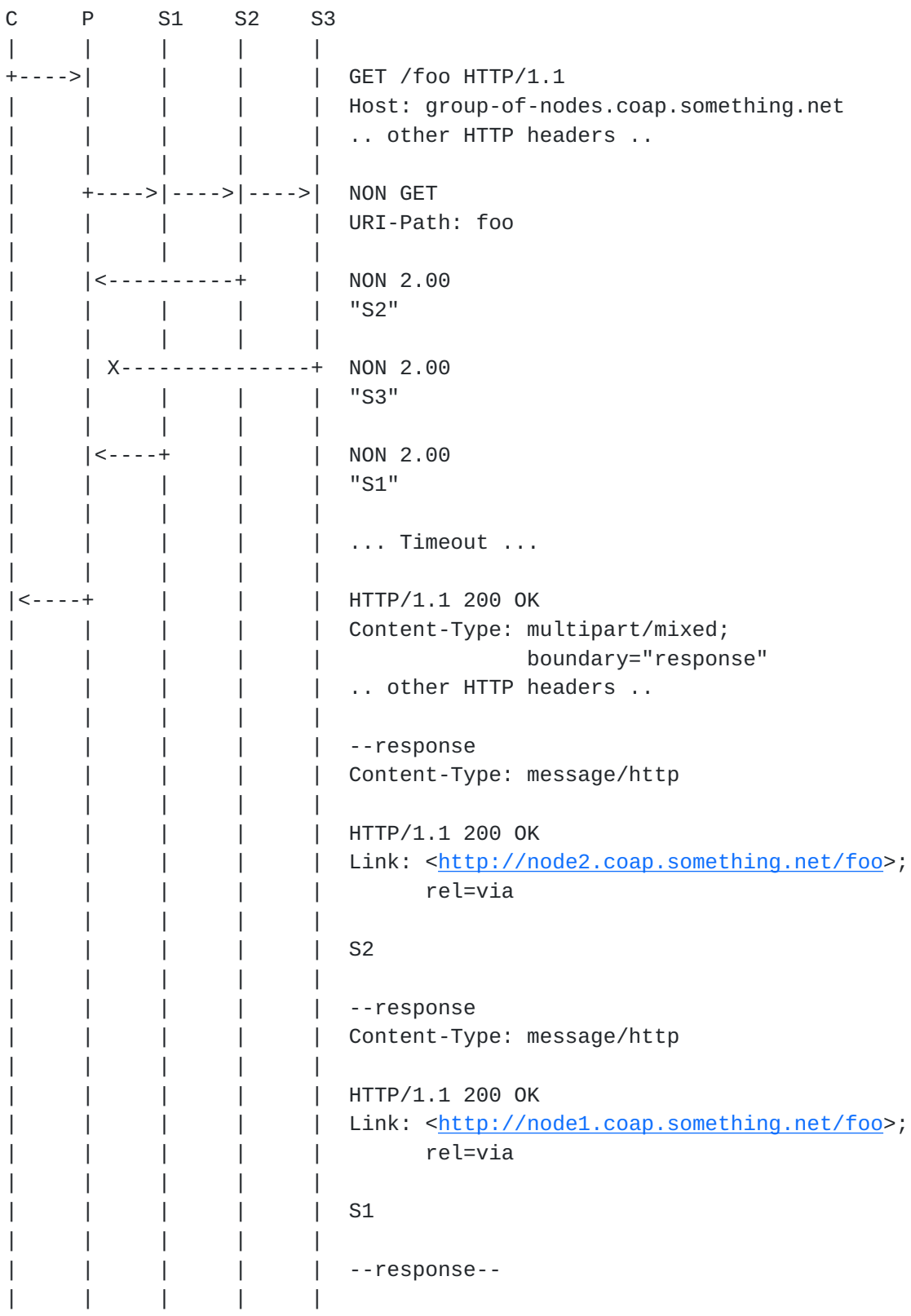

Figure 2: Unicast HTTP to Multicast CoAP Mapping

The example proposed in the above diagram does not make any assumption on which underlying group communication technology is available in the constrained network. Some detailed discussion is provided about it along the following lines.

C makes a GET request to group-of-nodes.coap.something.net. This domain name MAY either resolve to the address of P, or to the IPv6 multicast address of the nodes (if IP multicast is supported and P is an interception proxy), or the proxy P is specifically known by the client that sends this request to it.

To successfully start multicast proxying operation, the HC proxy MUST know that the destination URI involves a group of CoAP servers, e.g. the authority group-of-nodes.coap.something.net is known to identify a group of nodes either by using an internal lookup table, using DNS paired with IPv6 multicast, or by using some other special technique.

A specific implementation option is proposed to further explain the proposed example. Assume that DNS is configured such that all subdomain queries to coap.something.net, such as group-of-nodes.coap.something.net, resolve to the address of P. P performs the HC URI mapping by removing the 'coap' subdomain from the authority and by switching the scheme from 'http' to 'coap' (result: "coap://group-of-node.something.net/foo"); "group-of-nodes.something.net" is resolved to an IPv6 multicast address to which S1, S2 and S3 belong. The proxy handles this request as multicast and sends the request "GET /foo" to the multicast group .

5.3. Multicast Response Caching

We call perfect caching when the proxy uses only the cached representations to provide a response to the HTTP client. In the case of a multicast CoAP request, perfect caching is not adequate. This section updates the general caching and congestion control guidelines of with specific guidelines for the multicast use case.

Due to the inherent unreliable nature of the NON messages involved and since nodes may have dynamic membership in multicast groups, responding only with previously cached responses without issuing a new multicast request is not recommended. This perfect caching behaviour leads to miss responses of nodes that later joined the multicast group, and/or to repeatedly serve partial representations due to message losses. Therefore a multicast CoAP request SHOULD be sent by a HC proxy for each incoming request addressed to a multicast group.

Caching of multicast responses is still a valuable goal to pursue reduce network congestion, battery consumption and response latency.

Some considerations to be performed when adopting a multicast caching behaviour are outlined in the following paragraph.

Caching of multicast GET responses MAY be implemented by adopting some technique that takes into account either knowledge about dynamic characteristics of group membership (occurrence or frequency of group changes) or even better its full knowledge (list of nodes currently part of the group).

When using a technique exploiting this knowledge, valid cached responses SHOULD be served from cache.

5.4. Observe Mapping

By design, and certainly not without a good rationale, HTTP lacks a publish-subscriber facility. This implies that the mapping of the CoAP observe semantics has to be created ad hoc, perhaps by making use of one of the well-known HTTP techniques currently employed to establish an HTTP bidirectional connection with the target resource - as documented in [RFC6202].

In the following sections we will describe some of the approaches that can be used to identify an observable resource and to create the communication bridging needed to set up an end to end HTTP-CoAP observation.

5.4.1. Identification

In order to appropriately process an observe request, the HC proxy needs to know whether a given request is intended to establish an observation on the target resource, instead of triggering a regular request-response exchange.

At least two different approaches to identify such special requests exist, as discussed below.

5.4.1.1. Observable URI Mapping

An URI is said to be observable whenever every request to it implicitly requires the establishment of an HTTP bidirectional connection to the resource.

Such subscription to the resource is always paired, if possible, to a CoAP observe session to the actual resource being observed. In general, multiple connections that are active with a single observable resource at the same time, are multiplexed to the single observe session opened by the intermediary. Its notifications are then de-multiplexed by the HC proxy to every HTTP subscriber.

An intermediary MAY pair a couple of distinct HTTP URIs to a single CoAP observable resource: one providing the usual request-response mediated access to the resource, and the other that always triggers a CoAP observe session.

5.4.1.1.1. Discovery

As shown in Figure 3, in order to know whether an URI is observable, an HTTP UA MAY do a pre-flight request to the target resource using the HTTP OPTIONS method (see section 6.2 of [I-D.ietf-httpbis-p2-semantics]) to discover the communication options available for that resource.

If the resource supports observation, the proxy adds a Link Header [RFC5988] with the "obs" attribute as link-param (see Section 7 of [I-D.ietf-core-observe]).

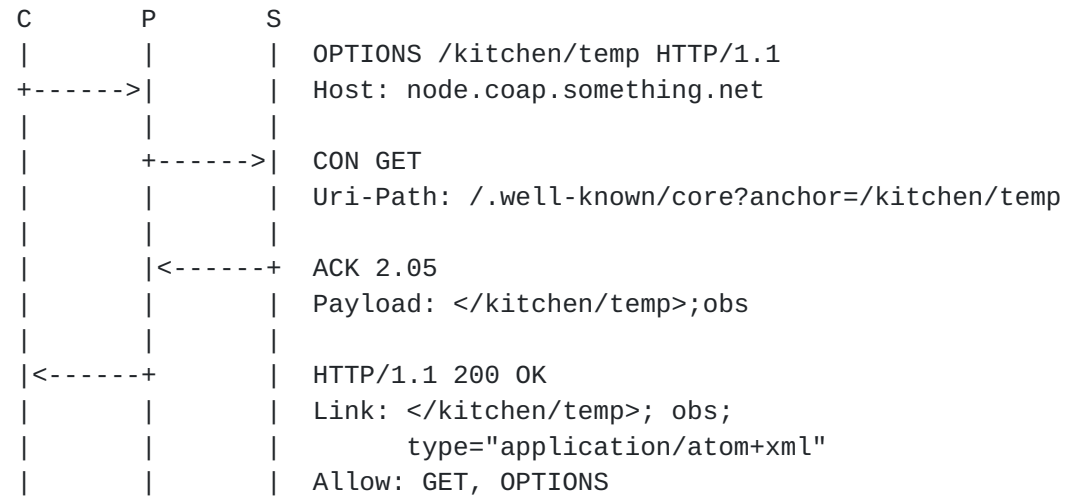

Figure 3: Discover Observability with HTTP OPTIONS

5.4.1.2. Differentiation Using HTTP Header

Discerning an observation request through in-protocol means, e.g. via the presence and values of some HTTP metadata, avoids introducing static "observable" URIs in the HC proxy namespace. Though ideally the former should be preferred, there seems to be no standard way to use one of the established HTTP headers to convey the observe semantics.

Standardizing such methods is out-of-scope of this document, so we just point out some possible approaches that in the future may be used to differentiate observation requests from regular requests.

5.4.1.2.1. Expect Header

The first method involves the use of the Expect header as defined in Section 9.3 of [I-D.ietf-httpbis-p2-semantics]. Whenever an HC proxy receives a request with a "206-partial-content" expectation, the proxy MUST fulfill this expectation by pairing this request to either a new or existing observe session to the resource.

If the proxy is unable to observe the resource, or if the observation establishment fails, the proxy MUST reply to the client with "417 Expectation Failed" status code.

Given that the Expect header is processed hop-by-hop, this method will fail immediately in case a proxy not supporting this expectation is traversed. For this reason, at present, the said approach can't be used in the public Internet.

5.4.1.2.2. Prefer Header

A second, very similar, approach involves the use of the Prefer header, defined in [I-D.snell-http-prefer]. The HTTP user agent expresses the preference to establish an observation with the target resource by including a "streaming" preference to request an HTTP Streaming session, or a "long-polling" preference to signal to the proxy its intended polling behaviour (see [RFC6202]).

A compliant HC proxy will try to fulfill the preference, and manifest observation establishment success by responding with a status code of "206 Partial Content". The observation request fails, falling back to a single response, whenever the status code is different from 206.

This approach will never fail immediately, differently from the previous one, even across a chain of unaware proxies; however, as documented in [RFC6202], caching intermediaries may interfere, delay or block the HTTP bidirectional connection, making this approach unacceptable when no weak consistency of the resource can be tolerated by the requesting UA.

5.4.2. Notification(s) Mapping

Multiplexing notifications using a single HTTP bidirectional session needs some further considerations about the selection of the media type that best fits this specific use case.

The usage of two different content-types that are suitable for carrying multiple notifications in a single session, is discussed in the following sections.

5.4.2.1. Multipart Messaging

As already discussed in Section 5.1.1 for multicasting, the "multipart/*" media type is a suitable solution to deliver multiple CoAP notifications within a single HTTP payload.

As in the multicast case, each part of the multipart entity MAY be represented using a "message/http" media type, containing the full mapping of the single CoAP notification mapped, so that CoAP envelope information are preserved (e.g. the response code).

A more sophisticated mapping could use multipart/mixed with native or translated media type.

5.4.2.2. Using ATOM Feeds

Popular observable resources with refresh rates higher than a couple of seconds may be treated as Atom feeds [RFC4287], especially with delay tolerant user agents and where persistence is required.

Figure 3 shows a resource supporting 'application/atom+xml' media-type. In such case clients can listen to update notification by regularly polling the resource via opportunely spaced GETs, i.e. driven by the advertised max-age value.

5.4.3. Examples

Figure 4 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Expect Header in the request; if P or S do not support this interaction, the request MUST fail with "417 Expectation Failed" return code. In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" return code.

At every notification corresponds the emission of a HTTP chunk containing a single part, which contains a "message/http" payload containing the full mapping of the notification. When the observation is dropped by the CoAP server, the HTTP streaming session is closed.

```
   |     |     |  Host: node.coap.something.net
   |     |     |  Expect: 206-partial-content
   |     |     |  Accept: multipart/mixed
   |     |     |
   |     +---->|  CON GET
   |     |     |  Uri-Path: temperature
   |     |     |  Observe: 0
   |     |     |
   |     |<----+  ACK 2.05
   |     |     |  Observe: 3482
   |     |     |  "22.1 C"
   |     |     |
   |<----+     |  HTTP/1.1 206 Partial Content
   |     |     |  Content-Type: multipart/mixed; boundary=notification
   |     |     |
   |     |     |  XX
   |     |     |  --notification
   |     |     |  Content-Type: message/http
   |     |     |
   |     |     |  HTTP/1.1 200 OK
   |     |     |
   |     |     |  22.1 C
   |     |     |
   |     |     |  ... about 60 seconds have passed ...
   |     |     |
   |     |<----+  NON 2.05
   |     |     |  Observe: 3542
   |     |     |  "21.6 C"
   |     |     |
   |<----+     |  YY
   |     |     |  --notification
   |     |     |  Content-Type: message/http
   |     |     |
   |     |     |  HTTP/1.1 200 OK
   |     |     |
   |     |     |  21.6 C
   |     |     |
   |     |     |  ... if the server drops the relationship ...
   |     |     |
   |     |<----+  NON 2.05
   |     |     |  "21.8 C"
   |     |     |
   |<----+     |  ZZ
   |     |     |  --notification
   |     |     |  Content-Type: message/http
   |     |     |
   |     |     |  HTTP/1.1 200 OK
   |     |     |
```

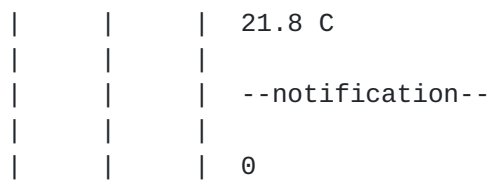

Figure 4: HTTP Streaming to CoAP Observe

Figure 5 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Prefer Header in the request; if P or S do not support this interaction, the request silently fails if a status code "200 OK" is returned, which means that no further notification is expected on that session.

In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" status code. At every notification a new response is sent to the pending client, always containing the "206 Partial Content" status code, to indicate that the observe session is still active, so that C can issue a new long-polling request immediately after this notification.

If the observation relationship is dropped by S, P notifies the last received content using the "200 OK" status code, indicating that no further notification is expected on this observe session.

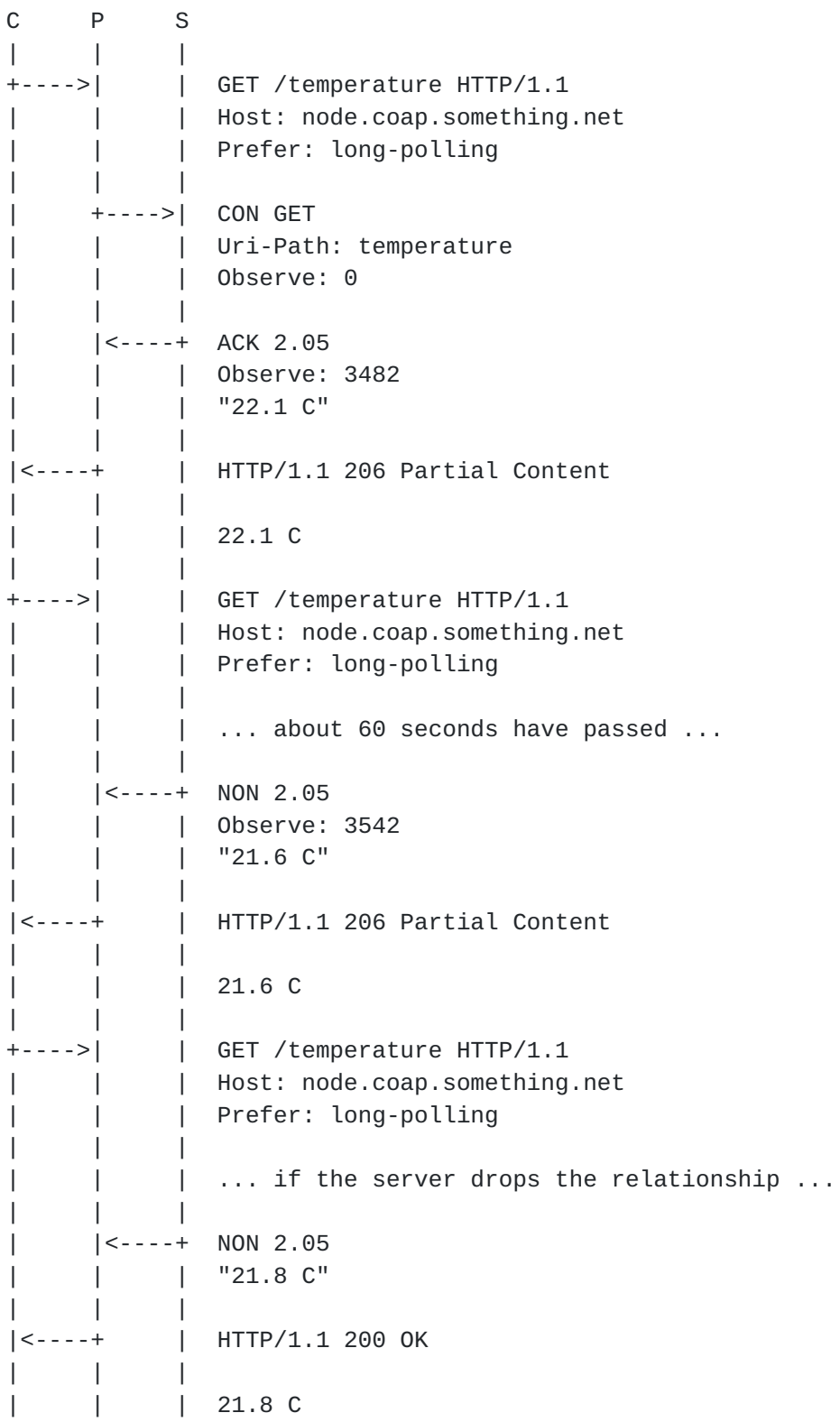

Figure 5: HTTP Long Polling to CoAP Observe

Figure 6 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "kitchen/temp" (T) available on S.

It is assumed that the HC proxy knows that the requested resource is observable (since perhaps being asked beforehand to discover its properties as described in Figure 3.) When asked by the HTTP client to retrieve the resource, it requests an observation - in case it weren't already in place - and then sends the collected data to the client as an Atom feed. The data coming through in the constrained network is stored locally on the proxy, and forwarded when further requests are received on the HTTP side. As already said, using the Atom format has two main advantages: first, there is always a "current" feed, but there may also be a complete log made available to HTTP clients; secondly, the HTTP intermediaries can play a substantial role in absorbing a fair amount of the load on the HC proxy. The latter is a very important property when the requested resource is or becomes very popular.

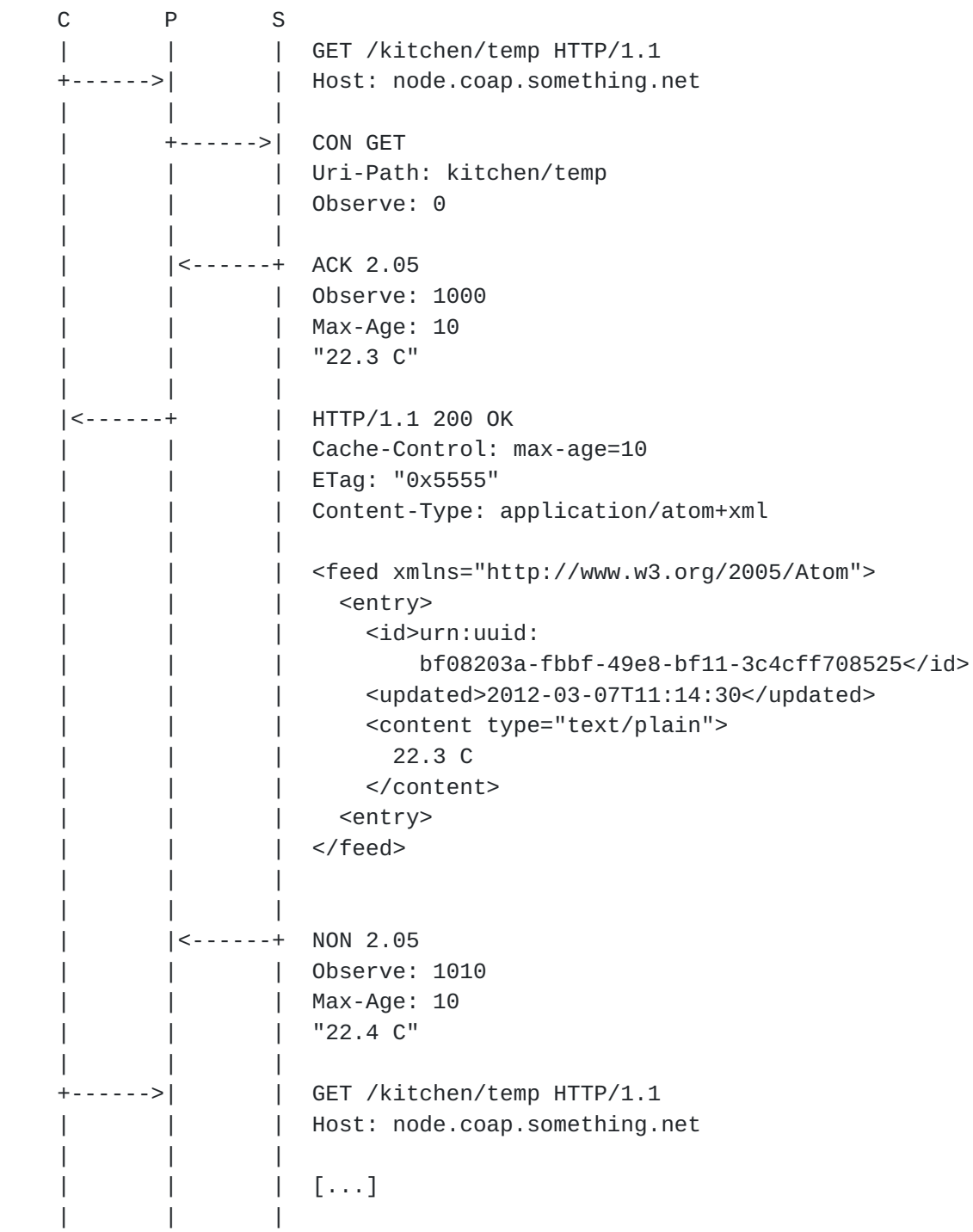

Figure 6: Observation via Atom feeds

6. HTML5 Scheme Handler Registration

The draft HTML5 standard offers a mechanism that allows an HTTP user agent to register a custom scheme handler through an HTML5 web page. This feature permits to an HC proxy to be registered as "handler" for URIs with the 'web+coap' or 'web+coaps' schemes using an HTML5 web page which embeds the custom scheme handler registration call registerProtocolHandler() described in Section 6.5.1.2 of [W3C.HTML5].

Example: the HTML5 homepage of a HC proxy at h2c.example.org could include the method call:

```
registerProtocolHandler('web+coap','proxy?url=%s','example HC proxy')
```

This registration call will prompt the HTTP user agent to ask for the user's permission to register the HC proxy as a handler for all 'web+coap' URIs. If the user accepts, whenever a 'web+coap' link is requested, the request will be fulfilled through the HC proxy: URI "web+coap://foo.org/a" will be transformed into URI "http://h2c.example.org/proxy?url=web+coap://foo.org/a".

7. Placement and Deployment

In typical scenarios, for communication from a CoAP client to an HTTP origin server, the HC proxy is expected to be located on the client-side (CS). Specifically, the HC proxy is expected to be deployed at the edge of the constrained network as shown in Figure 7.

The arguments supporting CS placement are as follows:

Client/Proxy/Network configuration overhead: CoAP clients require either static proxy configuration or proxy discovery support. This overhead is simplified if the proxy is placed on the same network domain of the client.

TCP/UDP: Translation between CoAP and HTTP requires also UDP to TCP mapping; UDP performance over the unconstrained Internet may not be adequate. In order to minimize the number of required retransmissions on the constrained part of the network and the overall reliability, TCP/UDP conversion SHOULD be performed as soon as possible in the network path.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, thus a CS placement, collecting all the traffic, is strategic for this need.

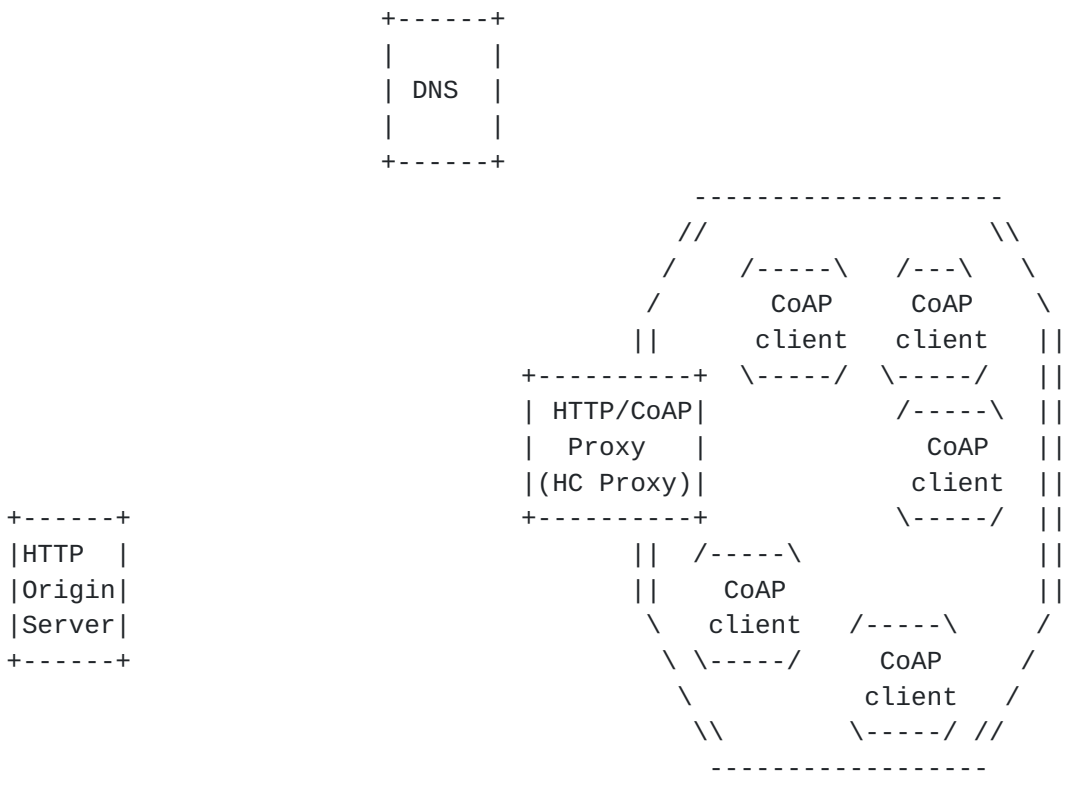

Figure 7: Client-side HC Proxy Deployment Scenario

8. Examples

Figure 8 shows an example implementation of a basic CoAP GET request with an HTTP URI as the value of a Proxy-URI option. The proxy retrieves a representation of the target resource from the HTTP origin server. It converts the payload to a UTF-8 charset, calculates the Max-Age Option from the Expires header field, and derives an entity-tag from the ETag header field.

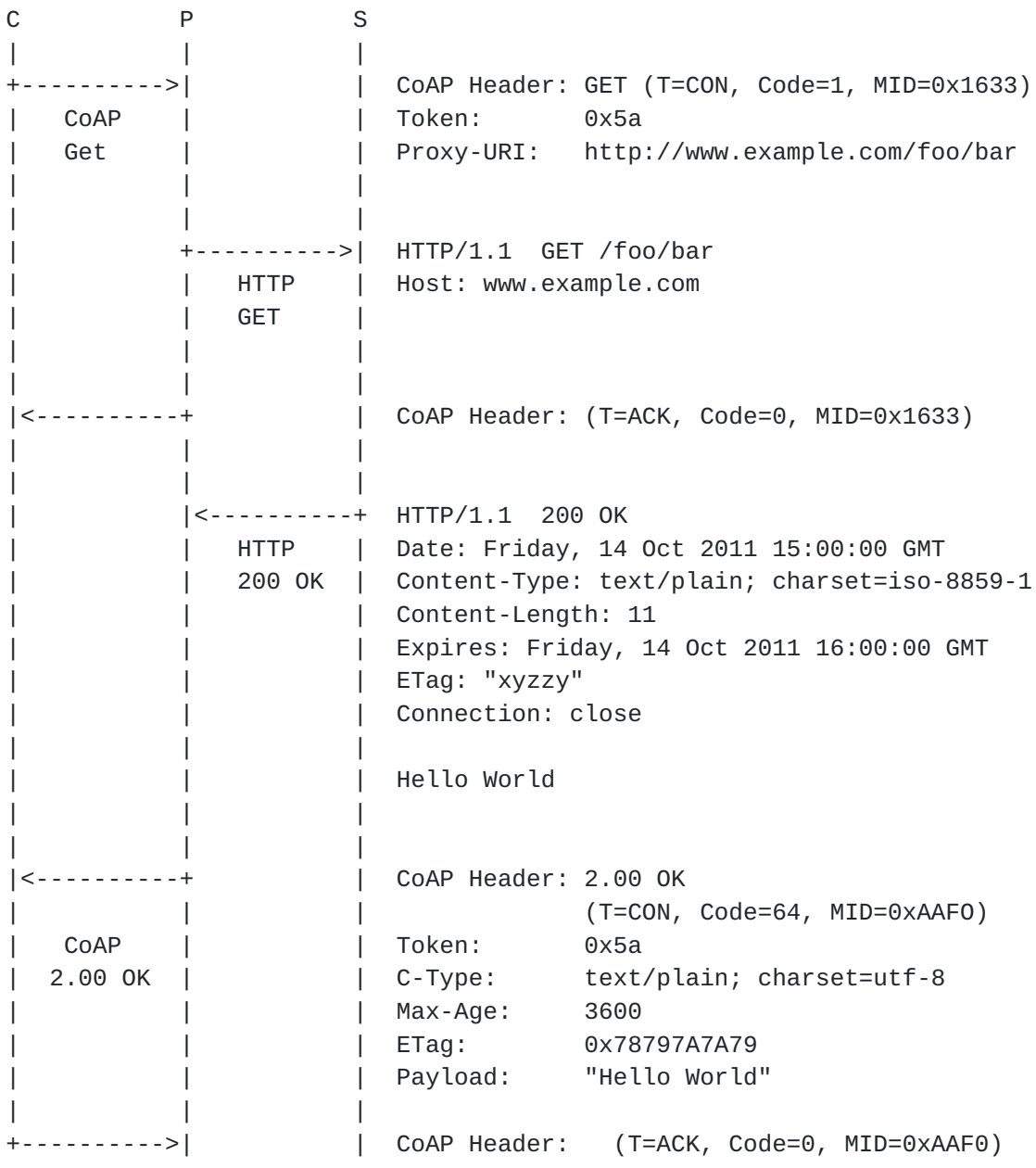

Figure 8: A Basic CoAP-HTTP GET Request

The example in Figure 9 builds on the previous example and shows an implementation of a GET request that includes a previously returned ETag Option. The proxy makes a Conditional Request to the HTTP origin server by including an If-None-Match header field in the HTTP GET Request. The CoAP response indicates that the response stored by the client is fresh. It includes a Max-Age Option calculated from the HTTP response's Expires header field.

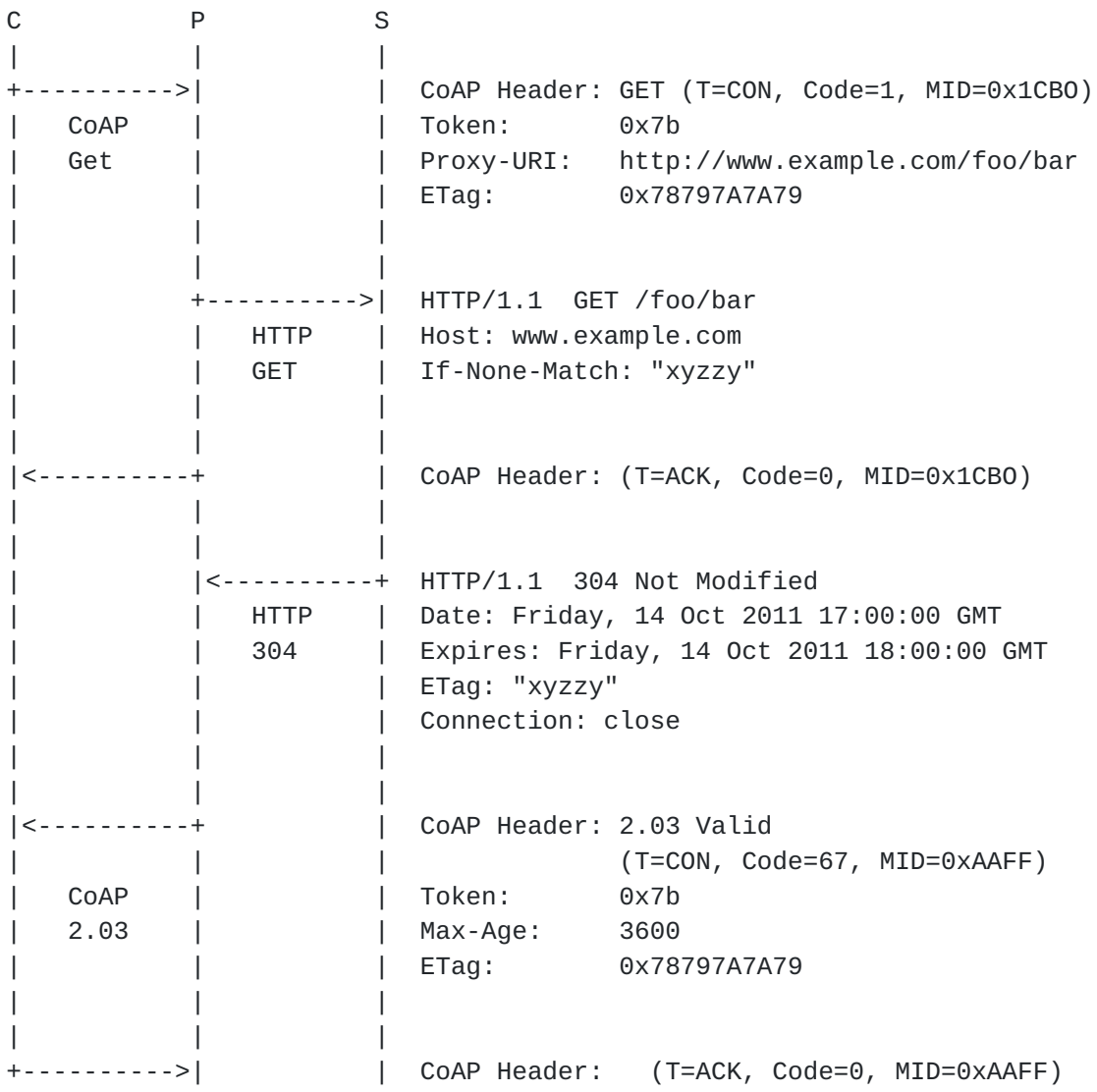

Figure 9: A CoAP-HTTP GET Request with an ETag Option

9. Acknowledgements

TBD.

10. IANA Considerations

This memo includes no request to IANA.

11. Security Considerations

11.1. Cross-protocol Security Policy Mapping

At the moment of this writing, CoAP and HTTP are missing any cross-protocol security policy mapping.

The HC proxy SHOULD flexibly support security policies between the two protocols, possibly as part of the HC URI mapping function, in order to statically map HTTP and CoAP security policies at the proxy (see Appendix A.2 for an example.)

11.2. Subscription

As noted in Section 7 of [I-D.ietf-core-observe], when using the observe pattern, an attacker could easily impose resource exhaustion on a naive server who's indiscriminately accepting observer relationships establishment from clients. The converse of this problem is also present, a malicious client may also target the HC proxy itself, by trying to exhaust the HTTP connection limit of the proxy by opening multiple subscriptions to some CoAP resource.

Effective strategies to reduce success of such a DoS on the HTTP side (by forcing prior identification of the HTTP client via usual web authentication mechanisms), must always be weighted against an acceptable level of usability of the exposed CoAP resources.

12. References

12.1. 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-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-09 (work in progress), May 2013.

[I-D.ietf-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-ietf-core-http-mapping-00 (work in progress), June 2013.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

[I-D.ietf-httpbis-p1-messaging]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-p1-messaging-22 (work in progress), February 2013.

[I-D.ietf-httpbis-p2-semantics]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", draft-ietf-httpbis-p2-semantics-22 (work in progress), February 2013.

[I-D.thomson-hybi-http-timeout]
Thomson, M., Loreto, S., and G. Wilkins, "Hypertext Transfer Protocol (HTTP) Keep-Alive Header", draft-thomson-hybi-http-timeout-03 (work in progress), July 2012.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

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

[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.

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

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

12.2. Informative References

[I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery", draft-bormann-core-simple-server-discovery-01 (work in progress), March 2012.

[I-D.ietf-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-00 (work in progress), June 2013.

[I-D.snell-http-prefer]
Snell, J., "Prefer Header for HTTP", draft-snell-http-prefer-18 (work in progress), January 2013.

[I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.

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

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

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

[RFC7234] Fielding, R., Nottingham, M., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014.

[W3C.HTML5]
Hickson, I., "HTML5", World Wide Web Consortium WD (work in progress) WD-html5-20111018, October 2011, <http://dev.w3.org/html5/spec/>.

Appendix A. Internal Mapping Functions (from an Implementer's Perspective)

At least three mapping functions have been identified, which take place at different stages of the HC proxy processing chain, involving the URL, Content-Type and Security Policy translation.

All these maps are required to have at least URL granularity so that, in principle, each and every requested URL may be treated as an independent mapping source.

In the following, the said map functions are characterized via their expected input and output, and a simple, yet sufficiently rich, configuration syntax is suggested.

In the spirit of a document providing implementation guidance, the specification of a map grammar aims at putting the basis for a reusable software component (e.g. a stand-alone C library) that many different proxy implementations can link to, and benefit from.

A.1. URL Map Algorithm

In case the HC proxy is a reverse proxy, i.e. it acts as the origin server in face of the served network, the URL of the resource requested by its clients (perhaps having an 'http' scheme) shall be mapped to the real resource origin (perhaps in the 'coap' scheme).

In case HC is a forward proxy, no URL translation is needed since the client already knows the "real name" of the resource.

An interception HC proxy, instead, MAY use the homogeneous mapping strategy to operate without any pre-configuration need.

As noted in Appendix B of [RFC3986] any correctly formatted URL can be matched by a POSIX regular expression. By leveraging on this property, we suggest a syntax that describes the URL mapping in terms of substituting the regex-matching portions of the requested URL into the mapped URL template.

E.g.: given the source regular expression '^http://example.com/coap/.*$' and destination template 'coap://$1' (where $1 stands for the first - and only in this specific case - substring matched by the regex pattern in the source), the input URL "http://example.com/coap/node1/resource2" translates to "coap://node1/resource2".

This is a well established technique used in many today's web components (e.g. Django URL dispatcher, Apache mod_rewrite, etc.), which provides a compact and powerful engine to implement what essentially is an URL rewrite function.

```
INPUT
    * requested URL

OUTPUT
    * target URL

SYNTAX
    url_map [rule name] {
        requested_url   <regex>
        mapped_url      <regex match subst template>
    }

EXAMPLE 1
    url_map homogeneous {
        requested_url   '^http://.*$'
        mapped_url      'coap//$1'
    }

EXAMPLE 2
    url_map embedded {
        requested_url   '^http://example.com/coap/.*$'
        mapped_url      'coap//$1'
    }
```

Note that many different url_map records may be given in order to build the whole mapping function. Each of these records can be queried (in some predefined order) by the HC proxy until a match is found, or the list is exhausted. In the latter case, depending on the mapping policy (only internal, internal then external, etc.) the original request can be refused, or the same mapping query is forwarded to one or more external URL mapping components.

A.2. Security Policy Map Algorithm

In case the "incoming" URL has been successfully translated, the HC proxy must lookup the security policy, if any, that needs to be applied to the request/response transaction carried on the "outgoing" leg.

```
INPUT
    * target URL (after URL map has been applied)
    * original requester identity (given by cookie, or IP address, or
      crypto credentials/security context, etc.)

OUTPUT
    * security context that will be applied to access the target URL

SYNTAX
    sec_map [rule name] {
        target_url      <regex>     -- one or more
        requester_id    <TBD>
                sec_context     <TBD>
    }

EXAMPLE
        <TBD>
```

A.3. Content-Type Map Algorithm

In case a set of destination URLs is known as being limited in handling a narrow subset of mime types, a content-type map can be configured in order to let the HC proxy transparently handle the compatible/lossless format translation.

```
INPUT
    * destination URL (after URL map has been applied)
    * original content-type

OUTPUT
    * mapped content-type

SYNTAX
    ct_map {
        target_url  <regex>                 -- one or more targetURLs
        ct_switch   <source_ct, dest_ct>    -- one or more CTs
    }

EXAMPLE
    ct_map {
        target_url  '^coap://class-1-device/.*$'
        ct_switch   */xml   application/exi
    }
```

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
KoanLogic
Via di Sabbiuno 11/5
Bologna 40136
Italy

Phone: +39 051 644 82 68
Email: tho@koanlogic.com

Esko Dijk
Philips Research

Email: esko.dijk@philips.com