CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2013

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
E. Dijk
Philips Research
October 22, 2012

Best Practices for HTTP-CoAP Mapping Implementation
draft-castellani-core-http-mapping-06

Abstract

This draft provides reference information for HTTP-CoAP proxy implementors focusing primarily on the reverse proxy case. It details deployment options, discusses possible approaches for URI mapping, and provides useful considerations related to protocol translation.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

CoAP [I-D.ietf-core-coap] 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 [RFC2616] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [I-D.ietf-core-coap] describes the fundamentals of the CoAP-HTTP (and vice-versa) 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 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 arbitrary (coincidental) variation in 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 different mapping approaches and the related proxy deployments;
- Section 3 discusses impact of the mapping on URI and describes notable options;
- Section 5 analyzes the mapping from HTTP to CoAP;
- Section 7 discusses possible security impact related to cross-protocol mapping.

2. Terminology

This document assumes readers are familiar with the terms Forward Proxy, Reverse Proxy and Interception Proxy as defined in [I-D.ietf-httpbis-p1-messaging] and [RFC3040].

Cross-Protocol Proxy (or Cross 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 [RFC3040].

Note: In this document we focus on the Reverse Proxy mode of the Cross-Protocol Proxy.

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", "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 means 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 is the first part of the URI, and it 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 has cross-protocol URIs referring to it.

HTTP clients typically only support 'http' and 'https' schemes. Therefore, they cannot directly access a CoAP server (which support 'coap' or 'coaps'). In this situation, communication is enabled by a Cross-Protocol Proxy, as shown in Figure 1, supporting URI mapping features. Such features are discussed in the following section.

4. HTTP to CoAP URI Mapping

Assume that a HTTP client wants to access a CoAP resource and indicates a target resource of "http://node.something.net/foobar" to a cross proxy. A possible URI mapping could be "coap://node.coap.something.net/foo".

As shown in the above example, if a cross-protocol URI exists, scheme, authority and path parts of the URI may change. The process of providing cross URIs may be complex, since a mechanism to statically or dynamically (discover) map the URI is needed.

Two simple static URI mapping solutions are proposed in the following subsections. Note that other mapping approaches are possible as well.

4.1. Homogeneous Mapping

In a homogeneous mapping approach, only the scheme portion of the URI needs to be mapped. The rest of the URI (i.e. authority, path, etc.) remains unchanged.

Example: The CoAP resource "//node.coap.something.net/foo" can be accessed by an HTTP client by requesting "http://node.coap.something.net/foo". The Cross-Protocol Proxy receiving the request is responsible to map the URI to "coap://node.coap.something.net/foo"

When homogeneous cross-protocol URIs are supported, HTTP to CoAP URI Mapping is easily implemented.

4.2. Embedded Mapping

In an embedded mapping approach, the HTTP URI has embedded inside it the authority and path part of the CoAP URI.

Example: The CoAP resource "//node.coap.something.net/foo" can be accessed by an HTTP client by inserting in the request "http://hc-proxy.something.net/coap/node.coap.something.net/foo". The Cross-Protocol Proxy then maps the URI to "coap://node.coap.something.net/foo"

When embedded mapping of URIs are supported, the complexity of a cross-protocol proxy is reduced.

4.3. Scheme Security Mapping

In general, regardless of the URI mapping scheme used in the Cross-Protocol Proxy, an "https" request SHOULD be translated to a "coaps" request. The exception case being cases where security on the CoAP side is not needed because the network is well enough protected already by other means (e.g. strong link-layer security, or the CoAP network runs inside a firewalled network, etc.).

5. HTTP-CoAP Reverse Cross-Protocol 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.

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 [I-D.ietf-core-coap]. 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 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 Cross-Protocol 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.

5.1. Cross-Protocol Proxy Placement

Typically, a Cross-Protocol Proxy is expected to be located at the edge of the constrained network. See Figure 1. The arguments supporting this placement are the following:

TCP/UDP: Translation between HTTP and CoAP requires also TCP/UDP translation; TCP is preferred over UDP on the global Internet given their relative performance. To minimize required retransmissions and maximize overall reliability, TCP/UDP conversion SHOULD be performed as close to the server as possible.

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.

Multicast: To support CoAPs use of local-multicast functionalities that MAY be available in a constrained network, the Cross-Protocol Proxy MAY require a network interface directly attached to the constrained network.

```
                            +------+
                            |      |
                            | DNS  |
                            |      |
                            +------+            Constrained Network
                                                --------------------
                                               /                    \
                                              /  /-----\     /-----\ \
                                             /     CoAP       CoAP    \
                                            /    server     server     \
                                           ||    \-----/     \-----/   ||
    +------+       HTTP Request      +----------+                      ||
    |HTTP  |------------------------>| HTTP/CoAP|  Req   /-----\       ||
    |Client|                         |  Cross-  |------->| CoAP        ||
    |      |<------------------------|  Proxy   |<-------|server       ||
    +------+       HTTP Response     +----------+  Resp  \-----/       ||
                                           ||                          ||
                                           ||   /-----\                ||
                                           ||    CoAP                  ||
                                            \    server                /
                                             \  \-----/               /
                                              \         /-----\      /
                                               \         CoAP       /
                                                \        server    /
                                                 \      \-----/   /
                                                  ----------------
```

Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

5.2. Comparison of Proxy Placement Scenarios

Table 1 shows three relevant Cross-Protocol Proxy deployment scenarios and notes the advantages ('+') and disadvantages ('-') related to each scenario.

Feature	HCF CS	HCR SS	HCI SS
TCP/UDP usage	-	+	+
Multicast support	-	+	+
Caching efficiency	-	+	+
Scalability/Availability	+	+/-	+
Configuration needs	-	-	+

Table 1: Comparison of relevant Cross-Protocol Proxy deployments

It can be seen that SS deployment is typically preferred above CS. Scalability and Availability are usually improved using multiple redundant proxies. For HCR SS, Scalability/Availability can be provided but there is configuration overhead involved. For HCI this overhead is minimal. For HCF CS, there is the issue that static configuration of multiple forward proxies is typically not feasible in existing (legacy) HTTP clients.

5.3. Response Code Translations

Table 2 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 [I-D.ietf-core-coap] and is intended to cover all possible cases. Multiple appearances of a CoAP response code in the first 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 2: HTTP-CoAP 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 [I-D.ietf-httpbis-p2-semantics] 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. [I-D.ietf-httpbis-p2-semantics] 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 here, as long as in CoAP there is no equivalent defined of the required WWW-Authenticate header (Section 3.1 of [I-D.ietf-httpbis-p7-auth]).

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 recognized 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.

5.4. Media Type Translations

A Cross-Protocol Proxy translates a media type string, carried in a HTTP Content-Type header in a request, to a CoAP Content-Format Option with the equivalent numeric value. The media types supported by CoAP are defined in the CoAP Content-Format Registry. Any HTTP request with a Content-Type for which the proxy does not know an equivalent CoAP Content-Format number, MUST lead to HTTP response 415 (Unsupported Media Type).

Also, a CoAP Content-Format value in a response is translated back to the equivalent HTTP Content-Type. If a proxy receives a CoAP Content-Format value that it does not recognize (e.g. because the value is IANA-registered after the proxy software was deployed), and is unable to look up the equivalent HTTP Content-Type on the fly, the proxy SHOULD return an HTTP entity (payload) without Content-Type header (complying to Section 3.1.1.5 of [I-D.ietf-httpbis-p2-semantics]).

5.5. Caching and Congestion Control

A Cross-Protocol 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 Cross-Protocol 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 Cross-Protocol Proxy (closing the TCP connection) after the HTTP request was made, a Cross-Protocol Proxy SHOULD wait for the associated CoAP response and cache it if possible. Further requests to the Cross-Protocol 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 [I-D.ietf-core-coap], 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 Cross-Protocol 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 Cross-Protocol 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 Cross-Protocol Proxy to keep their cached representation fresh while minimizing the number CoAP messages. See Section 5.6.

5.6. 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 [I-D.ietf-core-coap]. Such scenarios include, but are not limited to, sleeping 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.

5.7. Use of CoAP Blockwise Transfer

A Cross-Protocol 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 [I-D.ietf-core-coap] Section 4.6.

A Cross-Protocol 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 Cross-Protocol 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 Cross-Protocol 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 Cross-Protocol 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 cross 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.

5.8. 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.

5.9. 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 [I-D.ietf-httpbis-p1-messaging].

A cross 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 [I-D.ietf-core-coap].

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 [I-D.ietf-httpbis-p1-messaging]) MAY be supported by the proxy and is transparent to the CoAP network: the HC cross proxy will sequentially serve the pipelined requests by issuing different CoAP requests.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

The security concerns raised in Section 15.7 of [RFC2616] also apply to the cross proxy scenario. In fact, the cross proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the cross 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 cross 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 cross proxy module.

7.1. 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 5.5), 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 [I-D.ietf-core-coap].

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.

7.2. Handling Secured Exchanges

It is possible that the request from the client to the cross 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 cross 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 cross proxy deployment shown in Figure 1), the cross 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 4) a CoAP resource intended to be accessed only using HTTPS.

A secured connection that is terminated at the cross proxy, i.e. the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The cross 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. Acknowledgements

An initial version of the table found in Section 5.3 has been provided in revision -05 of [I-D.ietf-core-coap]. 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].

9. References

9.1. Normative References

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-08 (work in progress), February 2012.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-12 (work in progress), October 2012.

[I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-02 (work in progress), July 2012.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-06 (work in progress), September 2012.

[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-21 (work in progress), October 2012.

[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-21 (work in progress), October 2012.

[I-D.ietf-httpbis-p7-auth]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", draft-ietf-httpbis-p7-auth-21 (work in progress), October 2012.

[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.

9.2. 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.shelby-core-resource-directory] Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-04 (work in progress), July 2012.

[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.

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