CoRE
Internet-Draft
Intended status: Standards Track
Expires: 4 September 2022

C. Amsüss
3 March 2022

CoAP Protocol Indication
draft-amsuess-core-transport-indication-03

Abstract

The Constrained Application Protocol (CoAP, [RFC7252]) is available over different transports (UDP, DTLS, TCP, TLS, WebSockets), but lacks a way to unify these addresses. This document provides terminology and provisions based on Web Linking [RFC8288] to express alternative transports available to a device, and to optimize exchanges using these.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/.

Source for this draft and an issue tracker can be found at https://gitlab.com/chrysn/transport-indication.

Status of This Memo

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) provides transports mechanisms (UDP and DTLS since [RFC7252], TCP, TLS and WebSockets since [RFC8323]), with some additional being used in LwM2M [lwm2m] and even more being explored ([I-D.bormann-t2trg-slipmux], [I-D.amsuess-core-coap-over-gatt]). These are mutually incompatible on the wire, but CoAP implementations commonly support several of them, and proxies can translate between them.

CoAP currently lacks a way to indicate which transports are available for a given resource, and to indicate that a device is prepared to serve as a proxy; this document solves both by introducing the "has-proxy" terminology to Web Linking [RFC8288] that expresses the former through the latter. The additional "has-unique-proxy" term is introduced to negate any per-request overhead that would otherwise be introduced in the course of this.

CoAP also lacks a unified scheme to label a resource in a transport-independent way. This document does _not_ attempt to introduce any new scheme here, or raise a scheme to be the canonical one. Instead, each host or application can pick a canonical address for its resources, and advertise other transports in addition.

1.1. Terminology

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252] and link format ([RFC6690] (or, equivalently, web links as described in [RFC8288]).

Same-host proxy: A CoAP server that accepts forward proxy requests (i.e., requests carrying the Proxy-Scheme option) exclusively for URIs that it is also the authoritative server for is defined as a "same-host proxy".

The distinction between a same-host and any other proxy is only relevant on a practical, server-implementation and illustrative level; this specification does not use the distinction in normative requirements, and clients need not make the distinction at all.

hosts: The verb "to host" is used here in the sense of the link relation of the same name defined in [RFC6690].

For resources discovered via CoAP's discovery interface, a hosting statement is typically provided by the defaults implied by [RFC6690] where a link like </sensor/temp> is implied to have the relation "hosts" and the anchor /, such that a statement "coap://hostname hosts coap://hostname/sensor/temp" is implied in the link.

The link relation has been occasionally used with different interpretations, which ascribe more meaning to the term than it has in its definition. In particular,

* the "hosts" relation can not be inferred merely by two URIs having the same scheme, host and port (and vice versa), and

* the "hosts" relation on its own does not make any statement about the physical devices that hold the resource's representation.

[TBD: The former could probably still be used without too many ill effects; but things might also get weird when a dynamic resource created with one transport from use with another transport unless explicitly cleared.]

When talking of proxy requests, this document only talks of the Proxy-Scheme option. Given that all URIs this is usable with can be expressed in decomposed CoAP URIs, the need for using the Proxy-URI option should never arise. The Proxy-URI option is still equivalent to the decomposed options, and can be used if the server supports it.

1.1.1. Using URIs to identify protocol endpoints

The URI coap://device.example.com identifies a particular resource, possibly a "welcome" text. It is, colloquially, also used to identify the combination of a host (identified through a name), the default port, and the CoAP method of sending requests to the host.

For precision, this document uses the term "the transport address indicated by (a URI)" to refer to the host / port / protocol combination, but otherwise no big deal is made of it.

For the CoAP schemes (coap, coaps, coap+tcp, coaps+tcp, coap+ws, coaps+ws), URIs indicating a transport are always given with an empty path (which under their URI normalization rules is equivalent to a path containing a single slash). For the coap and coap+tcp schemes, URIs with different host names can indicate the same transport as long as the names resolve to the same addresses. For the other protocols, the given host name informs the name set in TLS's Server Name Indication (SNI) and/or the host sent in the "Host" header of the underlying HTTP request.

If an update to this document extends the list, for new schemes it might be allowed to have paths, queries or fragment identifiers present in the URI indicating the transport address. No guidance can be given here for these, as no realistic example is known. (Note that while the coap+ws scheme does use the well-known path /.well-known/coap internally, that is used purely on the HTTP side, and not part of the CoAP URI, not even for indicating the transport address).

URIs indicating a transport are especially useful when talking about proxies; this use is aligned with the way they are exprssed in the conventional environment variables http_proxy etc. [cite https://about.gitlab.com/blog/2021/01/27/we-need-to-talk-no-proxy/]. Furthermore, URIs processing is widespread in CoAP systems, and when that changes (e.g. through the introduction of [I-D.ietf-core-href]), URIs indicating a transport will still be efficient to encode. And last but not least, it lines up well with the colloquial identity mentioned above. (An alternative would be using a dedicated naming scheme, say, transport:coap:device.example.com:port, but that would needlessly introduce implementation complexity).

Note that this mechanism can only used with proxies that use CoAP's native address indication mechanisms. Proxies that perform URI mapping (as described in Section 5 of [RFC8075], especially using URI templates) are not supported in this document.

[TBD: Do we want to extend this to HTTP proxies? Probably just not, and if so, only to those that can just take coap://... for a URI.]

1.2. Goals

This document introduces provisions for the seamless use of different transport mechanisms for CoAP. Combined, these provide:

* Enablement: Inform clients of the availability of other transports of servers.

* No Aliasing: Any URI aliasing must be opt-in by the server. Any defined mechanisms must allow applications to keep working on the canonical URIs given by the server.

* Optimization: Do not incur per-request overhead from switching protocols. This may depend on the server's willingness to create aliased URIs.

* Proxy usability: All information provided must be usable by aware proxies to reduce the need for duplicate cache entries.

* Proxy announcement: Allow third parties to announce that they provide alternative transports to a host.

For all these functions, security policies must be described that allow the client to use them as securely as the original transport.

This document will not concern itself with changes in transport availability over time, neither in causing them ("Please take up your TCP interface, I'm going to send a firmware update") nor in advertising their availability in advance. Hosts whose transport's availability changes over time can utilize any suitable mechanism to keep client updated, such as placing a suitable Max-Age value on their resources or having them observable.

2. Indicating alternative transports

While CoAP can set the authority component of the requested URI in all requests (by means of Uri-Host and Uri-Port), setting the scheme of a requested URI (by means of Proxy-Scheme) makes the request implicitly a proxy request. However, this needs to be of only little practical concern: Any device can serve as a proxy for itself (a "same-host proxy") by accepting requests that carry the Proxy-Scheme option. If it is to be a well-behaved proxy, the device should then check whether it recognizes the name set in Uri-Host as one of its own (as it should if no Proxy-Scheme option accompanied it). If the name is not recognized, it should reject the request with 5.05 (Proxying Not Supported) -- unless, of course, it implements forward proxy functionality exceeding the same-host proxy. If the name is recognized, it should process the request as it would process a request coming in on the given protocol (which, for many hosts, is the same as if the option were absent completely).

A server can advertise a recommended proxy by serving a Web Link with the "has-proxy" relation to a URI indicating its transport address. In particular (and that is a typical case), it can indicate its own transport address on an alternative transport when implementing same-host proxy functionality.

The semantics of a link from S to P with relations has-proxy ("S has-proxy P", <P>;rel=has-proxy;anchor="S") are that for any resource R hosted on S ("S hosts R"), the proxy with the transport address indicated by P can be used to obtain R.

2.1. Example

A constrained device at the address 2001:db8::1 that supports CoAP over TCP in addition to CoAP can self-describe like this:

```
Req: to [ff02::fd]:5683 on UDP
Code: GET
Uri-Path: /.well-known/core
Uri-Query: if=tag:example.com,sensor

Res: from [2001:db8::1]:5683
Content-Format: application/link-format
Payload:
</sensors/temp>;if="tag:example.com,sensor",
<coap+tcp://[2001:db8::1]>;rel=has-proxy;anchor="/"


Req: to [2001:db8::1]:5683 on TCP
Code: GET
Proxy-Scheme: coap
Uri-Path: /sensors/temp
Observe: 0

Res: 2.05 Content
Observe: 0
Payload:
39.1°C
```

Figure 1: Discovery and follow-up request through a has-proxy relation

Note that generating this discovery file needs to be dynamic based on its available addresses; only if queried using a link-local source address, the server may also respond with a link-local address in the authority component of the proxy URI.

Unless the device makes resources discoverable at coap+tcp://[2001:db8::1]/.well-known/core or another discovery mechanism, clients may not assume that coap+tcp://[2001:db8::1]/sensors/temp is a valid resource (let alone is equivalent to the other resource on the same path). The server advertising itself like this may reject any request on CoAP-over-TCP unless it contains a Proxy-Scheme option.

Clients that want to access the device using CoAP-over-TCP would send a request by connecting to 2001:db8::1 TCP port 5683 and sending a GET with the options Proxy-Scheme: coap, no Uri-Host or -Port options (utilizing their default values), and the Uri-Paths "sensors" and "temp".

2.2. Security context propagation

If the originally requested URI R or the application requirements demand a security mechanism is used, the client MUST only use the proxy P if the proxy can provide suitable credentials. (The hosting URI S is immaterial to these considerations).

For example, if the application uses the host name and a public key infrastructure and R is coap://example.com/ the proxy accessed as coap+tcp://[2001:db8::1] still needs to provide a certificate chain for the name example.com to one of the system's trust anchors. If, on the other hand, the application is doing a firmware update and requires any certificate from its configured firmware update issuer, the proxy needs to provide such a firmware update certificate.

Some applications have requirements exceeding the requirements of a secure connection, e.g., (explicitly or implicitly) requiring that name resolution happen through a secure process and packets are only routed into networks where it trusts that they will not be intercepted on the path to the server. Such applications need to extend their requirements to the source of the has-proxy statement; a sufficient (but maybe needlessly strict) requirement is to only follow has-proxy statements that are part of the same resource that advertises the link currently being followed. Section Section 6.2 adds further considerations.

2.3. Choice of transports

It is up to the client whether to use an advertised proxy transport, or (if multiple are provided) which to pick.

Links to proxies may be annotated with additional metadata that may help guide such a choice; defining such metadata is out of scope for this document.

Clients MAY switch between advertised transports as long as the document describing them is fresh; they may even do so per request. (For example, they may perform individual requests using CoAP-over-UDP, but choose CoAP-over-TCP for requests with large expected responses). When the describing document approaches expiry, the client can use the representation's ETag to efficiently renew its justification for using the alternative transport.

2.4. Selection of a canonical origin

While a server is at liberty to provide the same resource independently on different transports (i.e. to create aliases), it may make sense for it to pick a single scheme and authority under which it announces its resources. Using only one address helps proxies keep their caches efficient, and makes it easier for clients to avoid exploring the same server twice from different angles.

When there is a predominant scheme and authority through which an existing service is discovered, it makes sense to use these for the canonical addresses.

Otherwise, it is suggested to use the coap or coaps scheme (given that these are the most basic and widespread ones), and the most stable usable name the host has.

2.5. Advertisement through a Resource Directory

In the Resource Directory specification [I-D.ietf-core-resource-directory], protocol negotiation was anticipated to use multiple base values. This approach was abandoned since then, as it would incur heavy URI aliasing.

Instead, devices can submit their has-proxy links to the Resource Directory like all their other metadata.

A client performing resource lookup can ask the RD to provide available (same-host-)proxies in a follow-up request by asking for ?anchor=<the-discovered-host>&rel=has-proxy. The RD may also volunteer that information during resource lookups even though the has-proxy link itself does not match the search criteria.

[It may be useful to define RD parameters for use with lookup here, which'd guide which available proxies to include. For example, asking ?if=tag:example.com,sensor&proxy-links=tcp could give as a result: <coap://[2001:db8::1]/s>;rt=tag:example.com,sensor,<coap+tcp://[2001:db8::1]/>;rel=has-proxy;anchor="coap://[2001:db8::1]/"]

3. Elision of Proxy-Scheme and Uri-Host

A CoAP server may publish and accept multiple URIs for the same resource, for example when it accepts requests on different IP addresses that do not carry a Uri-Host option, or when it accepts requests both with and without the Uri-Host option carrying a registered name. Likewise, the server may serve the same resources on different transports. This makes for efficient requests (with no Proxy-Scheme or Uri-Host option), but in general is discouraged

[aliases].

To make efficient requests possible without creating URI aliases that propagate, the "has-unique-proxy" specialization of the has-proxy relation is defined.

If a proxy is unique, it means that requests arriving at the proxy are treated the same no matter whether the scheme, authority and port of the link context are set in the Proxy-Scheme, Uri-Host and Uri-Port options, respectively, or whether all of them are absent.

[The following two paragraphs are both true but follow different approaches to explaining the observable and implementable behavior; it may later be decided to focus on one or the other in this document.]

While this creates URI aliasing in the requests as they are sent over the network, applications that discover a proxy this way should not "think" in terms of these URIs, but retain the originally discovered URIs (which, because Cool URIs Don't Change[cooluris], should be long-term usable). They use the proxy for as long as they have fresh knowledge of the has-(unique-)proxy statement.

In a way, advertising has-unique-proxy can be viewed as a description of the link target in terms of SCHC [I-D.ietf-lpwan-coap-static-context-hc]: In requests to that target, the link source's scheme and host are implicitly present.

While applications retain knowledge of the originally requested URI (even if it is not expressed in full on the wire), the original URI is not accessible to caches both within the host and on the network (for the latter, see Section 5). Thus, cached responses to the canonical and any aliased URI are mutually interchangeable as long as both the response and the proxy statement are fresh.

A client MAY use a unique-proxy like a proxy and still send the Proxy-Scheme and Uri-Host option; such a client needs to recognize both relation types, as relations of the has-unique-proxy type are a specialization of has-proxy and typically don't carry the latter (redundant) annotation. [To be evaluated -- on one hand, supporting it this way means that the server needs to identify all of its addresses and reject others. Then again, is a server that (like many now do) fully ignore any set Uri-Host correct at all?]

Example:

```
Req: to [ff02::fd]:5683 on UDP
Code: GET
Uri-Path: /.well-known/core
Uri-Query: if=tag:example.com,sensor

Res: from [2001:db8::1]:5683
Content-Format: application/link-format
Payload:
</sensors/>;if="tag:example.com,collection",
<coaps+ws://[2001:db8::1]>;rel=has-unique-proxy;anchor="/"


Req: to [2001:db8::1] via WebSockets over HTTPS
Code: GET
Uri-Path: /sensors/

Res: 2.05 Content
Content-Format: application/link-format
Payload:
</sensors/temperature>;if="tag:example.com,sensor"
```

Figure 2: Follow-up request through a has-unique-proxy relation. Compared to the last example, 5 bytes of scheme indication are saved during the follow-up request.

It is noteworthy that when the URI reference /sensors/temperature is resolved, the base URI is coap://device0815.example.com and not its coaps+ws counterpart -- as the request is still for that URI, which both the client and the server are aware of. However, this detail is of little practical importance: A simplistic client that uses coaps+ws://device0815.proxy.rd.example.com as a base URI will still arrive at an identical follow-up request with no ill effect, as long as it only uses the wrongly assembled URI for dereferencing resources, the security context is the same, the state is kept no longer than the has-unique-proxy statement is fresh, and it does not (for example) pass the URI on to other devices.

3.1. Impact on caches

[This section is written with the "there is implied URI aliasing" mindset; it should be possible to write it with the "compression" mindset as well (but there is no point in having both around in the document at this time).

It is also slightly duplicating, but also more detailed than, the brief note on the topic in Section 5]

When a node that performs caching learns of a has-unique-proxy statement, it can utilize the information about the implied URI aliasing: Requests to resources hosted by S can be answered with cached entries from P (because by the rules of has-unique-proxy a request can be crafted that is sent to P for which a fresh response is available). The inverse direction (serving resources whose URI "starts with" P from a cached request that was sent to S) is harder to serve because it additionaly requires a fresh statement that "S hosts R" for the matching resource R.

3.2. Using unique proxies securely

[This section is work in progress, it is more a flow of considerations turning back on each other. This is all made a bit trickier by not applying to OSCORE which is usually the author's go-to example, because OSCORE's requirements already preclude all these troubles.]

The use of unique proxies requires slightly more care in terms of security.

No requirements are necessary on the client side; those of {#secctx-propagation} suffice. (In particular, it is not necessary for the statement to originate from the original server unless that were already a requirement without the uniqueness property).

The extra care is necessary on the side of servers that are commissioned with wide ranging authorization [or is it?]: These may now be tricked into serving a resource of which the client assumes a different name. For example, if the desired resource is coaps://high-security.example.org/configuration, and there exists a "home page" style service for employees with patterns of coaps+tcp://user-${username}.example.org/ at which they can store files, and the server operating that service is commissioned with a wild-card certificate "*.example.org", then a device that receives the (malicious) information <coaps+tcp://user-mave.example.org>;rel=has-unique-proxy;anchor="coaps://high-security.example.org" might use this statement to contact the transport address indicated by coaps+tcp://user-mave.example.org and ask for /config (which, to the server, is indistinguishable from coaps+tcp://user-mave.example.org/config) and obtain a malicious configuration.

In a non-unique proxy situation, the error would have been caught by the server, which would have seen the request for coaps://high-security.example.com and refused to serve a request containing critical options it can not adaequately process.

In the unique proxy situation, ... [TBD: now whose fault is it? Can only be the client's ... because it looked at the wildcard certificate rather than whether the host-name it was narrowing it down to is authorized to speak for high-security.example.com? The server (operator) can barely be blamed, for while the certificate is needlessly wide, to the server it did look precisely like a good request.]

4. Third party proxy services

A server that is aware of a suitable cross proxy may use the has-proxy relation to advertise that proxy. If the protocol used towards the proxy provides name indication (as CoAP over TLS or WebSockets does), or by using a large number of addresses or ports, it can even advertise a (more efficient) has-unique-proxy relation. This is particularly interesting when the advertisements are made available across transports, for example in a Resource Directory.

How the server can discover and trust such a proxy is out of scope for this document, but generally involves the same kind of links. In particular, a server may obtain a link to a third party proxy from an administrator as part of its configuration.

The proxy may advertise itself without the origin server's involvement; in that case, the client needs to take additional care (see Section 6.2).

```
Req: GET http://rd.example.com/rd-lookup?if=tag:example.com,sensor

Res:
Content-Format: application/link-format
Payload:
<coap://device0815.example.com/sensors/>;if="tag:example.com,collection",
<coap+wss://device0815.proxy.rd.example.com>;rel=has-unique-proxy;anchor="coap://
device0815.example.com/"


Req: to device0815.proxy.rd.example.com on WebSocket
Host (indicated during upgrade): device0815.proxy.rd.example.com
Code: GET
Uri-Path: /sensors/

Res: 2.05 Content
Content-Format: application/link-format
Payload:
</sensors/temperature>;if="tag:example.com,sensor"
```

Figure 3: HTTP based discovery and CoAP-over-WS request to a CoAP resource through a has-unique-proxy relation

4.1. Generic proxy advertisements

A third party proxy may advertise its availability to act as a proxy for arbitrary CoAP requests. This use is not directly related to the protocol indication in other parts of this document, but sufficiently similar to warrant being described in the same document.

The resource type "TBDcore.proxy" can be used to describe such a proxy. The link target attribute "proxy-schemes" can be used to indicate the scheme(s) supported by the proxy, separated by the space character.

```
Req: GET coap://[fe80::1]/.well-known/core?rt=TBDcore.proxy

Res:
Content-Format: application/link-format
Payload:
<>;rt=TBDcore.proxy;proxy-schemes="coap coap+tcp coap+ws http"

Req: to [fe80::1] via CoAP
Code: GET
Proxy-Scheme: http
Uri-Host: example.com
Uri-Path: /motd
Accept: text/plain

Res: 2.05 Content
Content-Format: text/plain
Payload:
On Monday, October 25th 2021, there is no special message of the day.
```

Figure 4: A CoAP client discovers that its border router can also serve as a proxy, and uses that to access a resource on an HTTP server.

The considerations of Section 6.2 apply here.

A generic advertised proxy is always a forward proxy, and can not be advertised as a "unique" proxy as it would lack information about where to forward. (A proxy limited to a single outbound protocol might in theory work as a unique proxy when using a transport in which the full default Uri-Host value is configured at setup time, but these are considered impractical and thus not assigned a resource type here.)

The use of a generic proxy can be limited to a set of devices that have permission to use it. Clients can be allowed by their network address if they can be verified, or by using explicit client authentication using the methods of [I-D.tiloca-core-oscore-capable-proxies].

5. Client picked proxies

This section is purely informative, and serves to illustrate that the mechanisms introduced in this document do not hinder the continued use of existing proxies.

When a resource is accessed through an "actual" proxy (i.e., a host between the client and the server, which itself may have a same-host proxy in addition to that), the proxy's choice of the upstream server is originally (i.e., without the mechanisms of this document) either configured (as in a "chain" of proxies) or determined by the request URI (where a proxy picks CoAP over TCP and resolves the given name for a request aimed at a coap+tcp URI).

A proxy that has learned, by active solicitation of the information or by consulting links in its cache, that the requested URI is available through a (possibly same-host) proxy, may use that information in choosing the upstream transport, to correct the URI associated with a cached response, and to use responses obtained through one transport to satisfy requests on another.

For example, if a host at coap://h1.example.com has advertised </res>,<coap+tcp://h1.example.com>;rel=has-proxy;anchor="/", then a proxy that has an active CoAP-over-TCP connection to h1.example.com can forward an incoming request for coap://h1.example.com/res through that CoAP-over-TCP connection with a suitable Proxy-Scheme on that connection.

If the host had marked the proxy point as <coap+tcp://h1.example.com>;rel=has-unique-proxy instead, then the proxy could elide the Proxy-Scheme and Uri-Host options, and would (from the original CoAP caching rules) also be allowed to use any fresh cache representation of coap+tcp://h1.example.com/res to satisfy requests for coap://h1.example.com/res.

A client that uses a forward proxy and learns of a different proxy advertised to access a particular resource will not change its behavior if its original proxy is part of its configuration. If the forward proxy was only used out of necessity (e.g., to access a resource on the protocol not supported by the client) it can be practical for the client to use the advertised proxy instead.

6. Security considerations

6.1. Security context propagation

Clients need to strictly enforce the rules of Section 2.2. Failure to do so, in particular using a thusly announced proxy based on a certificate that attests the proxy's name, would allow attackers to circumvent the client's security expectation.

When security is terminated at proxies (as is in DTLS and TLS), a third party proxy can usually not satisfy this requirement; these transports are limited to same-host proxies.

6.2. Traffic misdirection

Accepting arbitrary proxies, even with security context propagation performed properly, would allow attackers to redirect traffic through systems under their control. Not only does that impact availability, it also allows an attacker to observe traffic patterns.

This affects both OSCORE and (D)TLS, as neither protect the participants' network addresses.

Other than the security context propagation rules, there are no hard and general rules about when an advertised proxy is a suitable candidate. Aspects for consideration are:

* When no direct connection is possible (e.g. because the resource to be accessed is served as coap+tcp and TCP is not implemented in the client, or because the resource's host is available on IPv6 while the client has no default IPv6 route), using a proxy is necessary if complete service disruption is to be avoided.

  While an adversary can cause such a situation (e.g. by manipulating routing or DNS entries), such an adversary is usually already in a position to observe traffic patterns.

* A proxy advertised by the device hosting the resource to be accessed is less risky to use than one advertised by a third party.

  Note that in some applications, servers produce representations based on unverified user input. In such cases, and more so when multiple applications share a security context, the advertisements' provenance may need to be considered.

6.3. Protecting the proxy

A widely published statement about a host's availability as a proxy can cause many clients to attempt to use it.

This is mitigated in well-behaved clients by observing the rate limits of [RFC7252], and by ceasing attempts to reach a proxy for the Max-Age of received errors.

Operators can further limit ill-effects by ensuring that their client systems do not needlessly use proxies advertised in an unsecured way, and by providing own proxies when their clients need them.

7. IANA considerations

7.1. Link Relation Types

IANA is asked to add two entries into the Link Relation Type Registry last updated in [RFC8288]:

Relation Name	Description	Reference
has-proxy	The link target can be used as a proxy to reach the link context.	RFCthis
has-unique-proxy	Like has-proxy, and using this proxy implies scheme and host of the target.	RFCthis

Table 1: New Link Relation types

7.2. Resource Types

IANA is asked to add an entry into the "Resource Type (rt=) Link Target Attribute Values" registry under the Constrained RESTful Environments (CoRE) Parameters:

[The RFC Editor is asked to replace any occurrence of TBDcore.proxy with the actually registered attribute value.]

Attribute Value: core.proxy

Description: Forward proxying services

Reference: [this document]

Notes: The schemes for which the proxy is usable may be indicated using the proxy-schemes target attribute as per Section 4.1 of [this document].

8. References

8.1. Normative References

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

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

8.2. Informative References

[aliases] W3C, "Architecture of the World Wide Web, Section 2.3.1 URI aliases", n.d., <https://www.w3.org/TR/webarch/#uri-aliases>.

[cooluris] BL, T., "Cool URIs don’t change", n.d., <https://www.w3.org/Provider/Style/URI>.

[I-D.amsuess-core-coap-over-gatt] Amsüss, C., "CoAP over GATT (Bluetooth Low Energy Generic Attributes)", Work in Progress, Internet-Draft, draft-amsuess-core-coap-over-gatt-01, 2 November 2020, <https://datatracker.ietf.org/doc/html/draft-amsuess-core-coap-over-gatt-01>.

[I-D.amsuess-t2trg-rdlink] Amsüss, C., "rdlink: Robust distributed links to constrained devices", Work in Progress, Internet-Draft, draft-amsuess-t2trg-rdlink-01, 23 September 2019, <https://datatracker.ietf.org/doc/html/draft-amsuess-t2trg-rdlink-01>.

[I-D.bormann-t2trg-slipmux] Bormann, C. and T. Kaupat, "Slipmux: Using an UART interface for diagnostics, configuration, and packet transfer", Work in Progress, Internet-Draft, draft-bormann-t2trg-slipmux-03, 4 November 2019, <https://datatracker.ietf.org/doc/html/draft-bormann-t2trg-slipmux-03>.

[I-D.ietf-core-href]
Bormann, C. and H. Birkholz, "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-09, 15 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-core-href-09>.

[I-D.ietf-core-resource-directory]
Amsüss, C., Shelby, Z., Koster, M., Bormann, C., and P. V. D. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-28, 7 March 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-core-resource-directory-28>.

[I-D.ietf-lpwan-coap-static-context-hc]
Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-lpwan-coap-static-context-hc-19, 8 March 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-lpwan-coap-static-context-hc-19>.

[I-D.silverajan-core-coap-protocol-negotiation]
Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", Work in Progress, Internet-Draft, draft-silverajan-core-coap-protocol-negotiation-09, 2 July 2018, <https://datatracker.ietf.org/doc/html/draft-silverajan-core-coap-protocol-negotiation-09>.

[I-D.tiloca-core-oscore-capable-proxies]
Tiloca, M. and R. Höglund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-capable-proxies-01, 25 October 2021, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-capable-proxies-01>.

[lwm2m] OMA SpecWorks, "White Paper Lightweight M2M 1.1", n.d., <https://omaspecworks.org/white-paper-lightweight-m2m-1-1/>.

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

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/rfc/rfc6763>.

[RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/rfc/rfc7838>.

[RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <https://www.rfc-editor.org/rfc/rfc8075>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/rfc/rfc8323>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

Appendix A. Change log

Since -02 (mainly processing reviews from Marco and Klaus):

* Acknowledge that 'coap://hostname/' is not the proxy but a URI that (in a particular phrasing) is used to stand in for the proxy's address (while it regularly identifies a resurce on the server)

* Security: Referencing traffic misdirection already in the first security block.

* Security: Add (incomplete) considerations for unique-proxy case.

* Narrow down "unique" proxy semantics to those properties used by the client, allowing unique proxies to be co-hosted with forward proxies.

* "Client picked proxies" clarified to merely illustrate how this is compatible with them.

* Use of "hosts" relation sharpened.

* Precision on how this does and does not consider changing transports.

* "Related work" section demoted to appendix.

* Add note on DTLS session resumption.

* Variable renaming.

* Various editorial fixes.

Since -01:

* Removed suggestion for generally trusted proxies; now stating that with (D)TLS, "a third party proxy can usually not satisfy [the security context propagation requirement]".

* State more clearly that valid cache entries for resources aliased through has-unique-proxy can be used.

* Added considerations for Multipath TCP.

* Added concrete suggestion and example for advertisement of general proxies.

* Added concrete suggestion for RD lookup extension that provides proxies.

* Minor editorial and example changes.

Since -00:

* Added introduction

* Added examples

* Added SCHC analogy

* Expanded security considerations

* Added guidance on choice of transport, and canonical addresses

* Added subsection on interaction with a Resource Directory

* Added comparisons with related work, including rdlink and DNS-SD sketches

* Added IANA considerations

* Added section on open questions

Appendix B. Related work and applicability to related fields

B.1. On HTTP

The mechanisms introduced here are similar to the Alt-Svc header of [RFC7838] in that they do not create different application-visible addresses, but provide dispatch through lower transport implementations.

Unlike in HTTP, the variations of CoAP protocols each come with their unique URI schemes and thus enable the "transport address indicated by a URI" concept. Thus, there is no need for a distinction between protocol-id and scheme.

To accommodate the message size constraints typical of CoRE environments, and accounting for the differences between HTTP headers and CoAP options, information is delivered once at discovery time.

Using the has-proxy and has-unique-proxy with HTTP URIs as the context is NOT RECOMMENDED; the HTTP provisions of the Alt-Svc header and ALPN are preferred.

B.2. Using DNS

As pointed out in [RFC7838], DNS can already serve some of the applications of Alt-Svc and has-unique-proxy by providing different CNAME records. These cover cases of multiple addresses, but not different ports or protocols.

While not specified for CoAP yet (and neither being specified here),

[which is an open discussion point for CoRE -- should we? Here? In a separate DNS-SD document?]

DNS SRV records (possibly in combination with DNS Service Discovery [RFC6763]) can provide records that could be considered equivalent to has-unique-proxy relations. If _coap._tcp, _coaps._tcp, _coap._udp, _coap+ws._tcp etc. were defined with suitable semantics, these can be equivalent:

```
_coap._udp.device.example.com SRV 0 0 device.example.com 61616
device.example.com AAAA 2001:db8::1

<coap://[2001:db8::1]>;rel=has-unique-proxy;anchor="coap://device.example.com"
```

It would be up to such a specification to give details on what the link's context is; unlike the link based discovery of this document, it would either need to pick one distinguished context scheme for which these records are looked up, or would introduce aliasing on its own.

B.3. Using names outside regular DNS

Names that are resolved through different mechanisms than DNS, or names which are defined within the scope of DNS but have no universally valid answers to A/AAAA requests, can be advertised using the relation types defined here and CoAP discovery.

In Figure 5, a server using a cryptographic name as described in [I-D.amsuess-t2trg-rdlink] is discovered and used.

```
Req: to [ff02::fd]:5683 on UDP
Code: GET
Uri-Path: /.well-known/core
Uri-Query: rel=has-proxy
Uri-Query: anchor=coap://nbswy3dpo5xxe3denbswy3dpo5xxe3de.ab.rdlink.arpa

Res: from [2001:db8::1]:5683
Content-Format: application/link-format
Payload:
<coap+tcp://[2001:db8::1]>;rel=has-unique-proxy;
  anchor="coap://nbswy3dpo5xxe3denbswy3dpo5xxe3de.ab.rdlink.arpa"


Req: to [2001:db8::1]:5683 on TCP
Code: GET
OSCORE: ...
Uri-Path: /sensors/temp
Observe: 0

Res: 2.05 Content
OSCORE: ...
Observe: 0
Payload:
39.1°C
```

Figure 5: Obtaining a sensor value from a local device with a global name

B.4. Multipath TCP

When CoAP-over-TCP is used over Multipath TCP and no Uri-Host option is sent, the implicit assumption is that there is aliasing between URIs containing any of the endpoints' addresses.

As these are negotiated within MPTCP, this works independently of this document's mechanisms. As long as all the server's addresses are equally reachable, there is no need to advertise multiple addresses that can later be discovered through MPTCP anyway. When

advertisements are long-lived and there is no single more stable address, several available addresses can be advertised (independently of whether MPTCP is involved or not). If a client uses an address that is merely a proxy address (and not a unique proxy address), but during MPTCP finds out that the network location being accessed is actually an MPTCP alternative address of the used one, the client MAY forego sending of the Proxy-Scheme and Uri-Path option.

[This follows from multiple addresses being valid for that TCP connection; at some point we may want to say something about what that means for the default value of the Uri-Host option -- maybe something like "has the default value of any of the associated addresses, but the server may only enable MPTCP if there is implicit aliasing between all of them" (similar to OSCORE's statement)?]

[TBD: Do we need a section analog to this that deals with (D)TLS session resumption in absence of SNI?]

Appendix C. Open Questions / further ideas

* OSCORE interaction: [RFC8613] Section 4.1.3.2 requirements place OSCORE use in a weird category between has-proxy and has-unique-proxy (because if routing still works, the result will be correct). Not sure how to write this down properly, or whether it's actionable at all.

  Possibly there is an inbetween category of "The host needs the Uri-Host etc. when accessed through CoAP, but because the host does not use the same OSCORE KID across different virtual hosts, it's has-unique-proxy as soon as you talk OSCORE".

* Self-uniqueness:

  A host that wants to indicate that it doesn't care about Uri-Host can probably publish something like </>;rel=has-unique-proxy to do so.

  This'd help applications justify when they can elide the Uri-Host, even when no different protocols are involved.

* Advertising under a stable name:

  If a host wants to advertise its host name rather than its IP address during multicast, how does it best do that?

  Options, when answering from 2001:db8::1 to a request to ff02::fd are:

```
<coap://myhostname/foo>,...,
<coap://[2001:db8::1]>;rel=has-unique-proxy;anchor="coap://myhostname"
```

which is verbose but formally clear, and

```
</foo>,...,
<coap://[2001:db8::1]>;rel=has-unique-proxy;anchor="coap://myhostname"
```

which is compatible with unaware clients, but its correctness with respect to canonical URIs needs to be argued by the client, in this sequence

- understanding the has-unique-proxy line,
- understanding that the request that went to 2001:db8::1 was really a Proxy-Scheme/Uri-Host-elided version of a request to coap://myhostname, and then
- processing any relative reference with this new base in mind.

(Not that it'd need to happen in software in that sequence, but that's the sequence needed to understand how the /foo here is really coap://myhostname/foo).

If CoRAL is used during discovery, a base directive or reverse relation to has-unique-proxy would make this easier.

Appendix D. Acknowledgements

This document heavily builds on concepts explored by Bill Silverajan and Mert Ocak in [I-D.silverajan-core-coap-protocol-negotiation], and together with Ines Robles and Klaus Hartke inside T2TRG.

[TBD: reviewers Marco Klaus]

Author's Address

Christian Amsüss
Austria
Email: christian@amsuess.com