Workgroup: CoRE
Internet-Draft:
draft-amsuess-core-resource-directory-
extensions-10
Published: 4 March 2024
Intended Status: Experimental
Expires: 5 September 2024
Authors: C. Amsüss

CoRE Resource Directory Extensions

Abstract

A collection of extensions to the Resource Directory [rfc9176] that can stand on their own, and have no clear future in specification yet.

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/resource-directory-extensions.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document pools some extensions to the Resource Directory [rfc9176] that might be useful but have no place in the original document.

They might become individual documents for IETF submission, simple registrations in the RD Parameter Registry at IANA, or grow into a shape where they can be submitted as a collection of tools.

At its current state, this draft is a collection of ideas.

2. Reverse Proxy requests

When a registrant registers at a Resource Directory, it might not have a suitable address it can use as a base address. Typical reasons include being inside a NAT without control over port forwarding, or only being able to open outgoing connections (as program running inside a web browser utilizing CoAP over WebSocket [RFC8323] might be).

[rfc9176] suggests (in the Cellular M2M use case) that proxy access to such endpoints can be provided, it gives no concrete mechanism to do that; this is such a mechanism.

This mechanism is intended to be a last-resort option to provide connectivity. Where possible, direct connections are preferred. Before registering for proxying, the registrant should attempt to obtain a publicly available port, for example using PCP ([RFC6887]).

The same mechanism can also be employed by registrants that want to conceal their network address from its clients.

A deployed application where this is implicitly done is LwM2M [citation missing]. Notable differences are that the protocol used between the client and the proxying RD is not CoAP but application specific, and that the RD (depending on some configuration) eagerly populates its proxy caches by sending requests and starting observations at registration time.

2.1. Discovery

An RD that provides proxying functionality advertises it by announcing the additional resource type "TBD1" on its directory resource.

2.2. Registration

A client passes the "proxy=yes" or "proxy=ondemand" query parameter in addition to (but typically instead of) a "base" query parameter.

A server that receives a "proxy=yes" query parameter in a registration (or receives "proxy=ondemand" and decides it needs to proxy) MUST come up with a "Proxy URL" on which it accepts requests, and which it uses as a Registration Base URI for lookups on the present registration.

The Proxy URL SHOULD have no path component, as acting as a reverse proxy in such a scenario means that any relative references in all representations that are proxied must be recognized and possibly rewritten.

The RD MAY accept connections also on alternative Registration Base URIs using different protocols; it can advertise them using the mechanisms of [I-D.ietf-core-transport-indication].

The registrant is not informed of the chosen public name by the RD. (Section 8 discusses means how to change that).

This mechanism is applicable to all transports that can be used to register. If proxying is active, the restrictions on when the base parameter needs to be present ([rfc9176] Registration template) are relaxed: The base parameter may also be absent if the connection originates from an ephemeral port, as long as the underlying protocol supports role reversal, and link-local IPv6 addresses may be used without any concerns of expressibility.

If the client uses the role reversal rule relaxation, both it and the server keep that connection open for as long as it wants to be reachable. When the connection terminates, the RD SHOULD treat the registration as having timed out (even if its lifetime has not been exceeded) and MAY eventually remove the registration. It is yet to be decided whether the RD's announced ability to do proxying should imply that infinite lifetimes are necessarily supported for such registrations; at least, it is RECOMMENDED.

2.2.1. Registration updates

The "proxy" query parameter can not be changed or repeated in a registration update; RD servers MUST answer 4.00 Bad Request to any registration update that has a "proxy" query parameter.

As always, registration updates can explicitly or implicitly update the Registration Base URI. In proxied registrations, those changes are not propagated to lookup, but do change the forwarding address of the proxy.

For example, if a registration is established over TCP, an update can come along in a new TCP connection. Starting then, proxied requests are forwarded along that new connection.

2.3. Proxy behavior

The RD operates as a reverse-proxy as described in [RFC7252] Section 5.7.3 at the announced Proxy URL(s), where it decides based on the requested host and port to which registrant endpoint to forward the request.

The address the incoming request are forwarded to is the base address of the registration. If an explicit "base" paremter is given, the RD will forward requests to that location. Otherwise, it forwards to the registration's source address (which is the implied base parameter).

When an implicit base is used, the requests forwarded by the RD to the EP contain no Uri-Host option. EPs that want to run multiple parallel registrations (especially gateway-like devices) can either open up separate connections, or use an additional (to-be-specified) mechanism to set the "virtual host name" for that registration in a separate argument.

2.3.1. Limitations from using a reverse proxy

The registrant requesting the reverse proxying needs to ensure that all services it provides are compatible with being operated behind a reverse proxy with an unknown name. In particular, this rules out all applications that refer to local resources by a full URI (as opposed to relative references without scheme and host). Applications behind a reverse proxy can not use role reversal.

Some of these limitations can be mitigated if the application knows its advertised address. The mechanisms of Section 8 might be used to change that.

2.4. On-Demand proxying

If an endpoint is deployed in an unknown network, it might not know whether it is behind a NAT that would require it to configure an explicit base address, and ask the RD to assist by proxying if necessary by registering with the "proxy=ondemand" query parameter.

A server receiving that SHOULD use a different IP address to try to access the registrant's .well-known/core file using a GET request under the Registration Base URI. If that succeeds, it may assume that no NAT is present, and ignore the proxying request. Otherwise, it configures proxying as if "proxy=yes" were requested.

Note that this is only a heuristic [and not tested in deployments yet].

2.5. Multiple upstreams

When a proxying RD is operating behind a router that has uplinks with multiple provisioning domains (see [RFC7556]) or a similar setup, it MAY mint multiple addresses that are reachable on the respective provisioning domains. When possible, it is preferred to keep the number of URIs handed out low (avoiding URI aliasing); this can be achieved by announcing both the proxy's public addresses under the same wildcard name.

If RDs are announced by the uplinks using RDAO, the proxy may use the methods of [I-D.amsuess-core-rd-replication] to distribute its registrations to all the announced upstream RDs.

In such setups, the router can forward the upstream RDs using the PvD option ([RFC8801]) to PvD-aware hosts and only announce the local RD to PvD-unaware ones (which then forwards their registrations). It can be expected that PvD-aware endpoints are capable of registering with multiple RDs simultaneously.

2.6. Examples

2.6.1. Registration through a firewall

```
Req from [2001:db8:42::9876]:5683:
POST coap://rd.example.net/rd?ep=node9876&proxy=ondemand
</some-resource>;rt="example.x"

Req from other-address.rd.example.net:
GET coap://[2001:db8:42::9876]/.well-known/core

Request blocked by stateful firewall around [2001:db8:42::]

RD decides that proxying is necessary

Res: 2.04 Created
Location: /reg/abcd
```

Later, lookup of that registration might say:

```
Req: GET coap://rd.example.net/lookup/res?rt=example.x

Res: 2.05 Content
<coap://node987.rd.example.net/some-resource>;rt="example.x
```

A request to that resource will end up at an IP address of the RD, which will forward it using its the IP and port on which the registrant had registered as source port, thus reaching the registrant through the stateful firewall.

2.6.2. Registration from a browser context

```
Req: POST coaps+ws://rd.example.net/rd?ep=node1234&proxy=yes
</gyroscope>;rt="core.s"

Res: 2.04 Created
Location: /reg/123
```

The gyroscope can now not only be looked up in the RD, but also be reached:

```
Req: GET coap://rd.example.net/lookup/res?rt=core.s

Res: 2.05 Content
<coap://[2001:db8:1::1]:10123/gyroscope>;rt="core.s"
```

In this example, the RD has chosen to do port-based rather than host-based virtual hosting and announces its literal IP address as that allows clients to not send the lengthy Uri-Host option with all requests.

2.7. Notes on stability and maturity

Using this with UDP can be quite fragile; the author only draws on own experience that this can work across cell-phone NATs and does not claim that this will work over generic firewalls.

[It may make sense to have the example as TCP right away.]

2.8. Security considerations

An RD MAY impose additional restrictions on which endpoints can register for proxying, and thus respond 4.01 Unauthorized to request that would pass had they not requested proxying.

Attackers could do third party registrations with an attacked device's address as base URI, though the RD would probably not amplify any attacks in that case.

The RD MUST NOT reveal the address at which it reaches the registrant except for adaequately authenticated and authorized debugging purposes, as that address could reveal sensitive location data the registrant may wish to hide by using a proxy.

Usual caveats for proxies apply.

2.9. Alternatives to be explored

With the mechanisms of [I-D.ietf-core-transport-indication], an RD could also operate as a forward proxy, and indicate its availability

for that purpose in a has-proxy link it creates on its own, and which it makes discoverable through its lookup interfaces.

How a registrant opts in to that behavior, how it selects a suitable public address (using the base attribute is tempting, but conflicts with the currently prescribed proxy behavior) and for which scenarios this is preferable is a topic being explored.

As with the reverse proxy address, the registrant is not informed of the public addresses (though again, Section 8 can be used to change that). Knowing these addresses can be relevant when the endpoint advertises its services out of band (e.g. by showing a QR code or exposing links through NFC), but also when the mechanism of [I-D.ietf-core-transport-indication] Appendix D is used.

3. Infinite lifetime

An RD can indicate support for infinite lifetimes by adding the resoruce type "TBD2" to its list of resource types.

A registrant that wishes to keep its registration alive indefinitely can set the lifetime value as "lt=inf".

Registrations with infinite lifetimes never time out. Unlike regular registrations, they are not "soft state"; the registrant can expect the RD to persist the registrations across network changes, reboots, softare updates and that like.

Typical use cases for infinite life times are:

* Commissioning tools (CTs) that do not return to the deployment site, and thus can not refresh the soft state

* Proxy registrations whose lifetime is limited by a connection that is kept alive

3.1. Example

Had the example of Section 2.6.2 discovered support for infinite lifetimes during lookup like this:

```
Req: GET coaps+ws://rd.example.net/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd TBD1 TBD2";ct=40
```

it could register like that:

```
Req: POST coaps+ws://rd.example.net/rd?ep=node1234&proxy=yes&lt=inf
</gyroscope>;rt="core.s"

Res: 2.04 Created
Location: /reg/123
```

and never need to update the registration for as long as the websocket connection is open.

(When it gets terminated, it could try renewing the registration, but needs to be prepared for the RD to already have removed the original registration.)

4. Limited lifetimes

Even if an RD supports infinite lifetimes, it may not accept them from just any registrant. Even more, an RD may have policies in place that require a certain frequency of updates and thus place an upper limit on lt lower than the technical limit of 136 years.

This document does not define any means of communicating lifetime limits, but explores a few options:

* Administrative channels.

  An RD that sees registrations with unreasonably long lifetimes can flag them for its operator to take further measures.

  While sounding tediously manual, this captures the observation that different components are configured in a softly incompatible way, and need operator intervention (because if there were automatic means, they obviously failed).

* General advertisement of preferred lifetimes.

  When the limitations on the lifetimes are not from authorization but from general setup, an RD could advertise that property in a to-be-created link target attribute of its registration resource. Different attributes could express preference or hard limits.

  This information is also available easily for registrants, which may then heed the advice if supported, and may notify their operators that they just started spending more resources than they were configured to.

  It is also available to tools that provision endpoints with their RD address (and parameters), as they can use the same lookup interface.

* Per-registration information.

For soft limits, the RD can offer the endpoint additional metadata if it queries them post-registration. That query can use the endpoint lookup interface, or the extension of Section 8. This may require additional round-trips on the part of endpoint.

*Hard limits informed by error codes.

An RD can reject registrations with overly long lifetimes if the endpoint is not authorized to use such long lifetimes with a 4.01 Unauthorized error. The mechanisms of [RFC9290], with a to-be-defined error detail on the permissible lifetime, can be used to propagate information back to then endpoint.

This behavior is explicitly NOT RECOMMENDED, because devices may crucially depend on the RD's services -- this rejection may even be the reason why the device is not configured with the new settings that would contain a shorter lifetime.

5. Zone identifier introspection

The 'split-horizon' mechanism of [rfc9176] (that registrations with link-local bases can only be read from the zone they registered on) reduces the usability of the endpoint lookup interface for debugging purposes.

To allow an administrator to read out the "show-zone-id" query parameter for endpoint and resource lookup is introduced.

A Resource Directory that understands this parameter MUST NOT limit lookup results to registrations from the lookup's zone, and MUST use [RFC6874] zone identifiers to annotate which zone those registrations are valid on.

The RD MUST limit such requests to authenticated and authorized debugging requests, as registrants may rely on the RD to keep their presence secret from other links.

5.1. Example

```
Req: GET /rd-lookup/ep?show-zone-id&et=printer

Res: 2.05 Content
</reg/1>;base="coap://[2001:db8::1]";et=printer;ep="bigprinter",
</reg/2>;base="coap://[fe80::99%wlan0]";et=printer;ep="localprinter-1234
</reg/3>;base="coap://[fe80::99%eth2]";et=printer;ep="localprinter-5678"
```

6. Proxying multicast requests

Multicast requests are hard to forward at a proxy: Even if a media type is used in which multiple responses can be aggregated

transparently, the proxy can not reliably know when all responses have come in. [RFC7390] Section 2.9 describes the difficulties in more detail.

Note that [I-D.tiloca-core-groupcomm-proxy] provides a mechanism that *does* allow the forwarding of multicast requests. It is yet to be determined what the respective pros and cons are. Conversely, that lookup mechanism may also serve as an alternative to resource lookup on an RD.

A proxy MAY expose an interface compatible with the RD lookup interface, which SHOULD be advertised by a link to it that indicates the resource types core.rd-lookup-res and TBD4.

The proxy sends multicast requests to All CoAP Nodes ([RFC7252] Section 12.8) requesting their .well-known/core files either eagerly (ie. in regular intervals independent of queries) or on demand (in which case it SHOULD limit the results by applying [RFC6690] query filtering; if it has received multiple query parameters it should forward the one it deems most likely to limit the results, as .well-known/core only supports a single query parameter).

In comparison to classical RD operation, this RD behaves roughly as if it had received a simple registration with a All CoAP Nodes address as the source address, if such behavior were specified. The individual registrations that result from this neither have an explicit registration resource nor an explicit endpoint name; given that the endpoint lookup interface is not present on such proxies, neither can be queried.

Clients that would intend to do run a multicast discovery operation behind the proxy can then instead query that resource lookup interface. They SHOULD use observation on lookups, as an on-demand implementation MAY return the first result before others have arrived, or MAY even return an empty link set immediately.

6.1. Example

```
Req: GET coap+ws://gateway.example.com/.well-known/core?rt=TBD4

Res: 2.05 Content
</discover>;rt="core.rd-lookup-res TBD4";ct=40

Req: GET coap+ws://gateway.example.com/discover?rt=core.s
Observe: 0

Res: 2.05 Content
Observe: 0
Content-Format: 40
(empty payload)
```

At the same time, the proxy sends out multicast requests on its interfaces:

```
Req: GET coap://ff05::fd/.well-known/core?rt=core.s

Res (from [2001:db8::1]:5683): 2.05 Content
</temp>;ct="0 112";rt="core.s"

Res (from [2001:db8::2]:5683): 2.05 Content
</light>;ct="0 112";rt="core.s"
```

upon receipt of which it sends out a notification to the websocket client:

```
Res: 2.05 Content
Observe: 1
Content-Format: 40
<coap://[2001:db8::1]/temp>;ct="0 112";rt="core.s";anchor="coap://[2001:
<coap://[2001:db8::2]/light>;ct="0 112";rt="core.s";anchro="coap://[2001
```

7. Registrations that update DNS records

An RD that is provisioned with means to update a DNS zone and that has a known mapping from registrants to host names could use registrations to populate DNS records from registration base addresses.

When combined with Section 2, these records point to the RD's built-in proxy rather than to the base address.

This mechanism is not described in further detail yet as it does not interact well yet with how the base registration attribute interacts with the proxy announcements of [I-D.ietf-core-transport-indication].

8. Propagating server generated registration information

The RD can populate some data into the registration: The RD may pick the sector and endpoint name based on the endpoint's credentials, or (as introduced in this documents) reverse proxy names and soft lifetime limits can be added.

With the exception of sector and endpoint name, the registrant can query those properties through the endpoint lookup interface. However, this is cumbersome as it requires it to use both the registration and the lookup interface.

The architecture of [I-D.ietf-core-coap-pubsub] offers a different architectural setup: Applied to the RD, the registration would generate both a registration metadata resource (at which the

registrant can set or query its registration's metadata) and a registration link resource (which contains all the links the registrant provides). Such a setup would make it easier for registrants to query or update registration metadata, including querying for an implicitly assigned endpoint name or sector.

Extending the RD specification to allow this style of operation would be possible without altering its client facing interfaces. Alternatively, using a new media type for operations on the registration resource and/or the FETCH and PATCH methods would enable such operations in a less intrusive way. While it would be tempting to add an Accept option to the registration request to solicit immediate information on the registration that was just created, the Accept option's criticality would render this incompatible with existing servers. The option can still be set if the new content format is advertised by the RD.

Without any media type suggested so far, this is what a registration could look like if the RD advertised that it provided content format TBD6 on the registration interface:

```
Req from [2001:db8::1]:5683:
POST coap://rd.example.net/rd
Accept: TBD6
Payload:
</some-resource>;rt="example.x"

Res: 2.04 Created
Location: /reg/abcd
Content-Format: TBD6
Payload:
Soft lifetime limit 3600, please update your registration in time.
Forward proxy services are offered at coaps+ws://rd.example.net and
coaps+tcp://rd.example.net.
```

9. Combining simple registration with EDHOC and ACE

For very constrained devices, starting a simple registration may be the only occasion at which they use the CoAP client role. If they exclusively send piggybacked responses (Section 5.2.1 of [RFC7252]) and handle only idempotent requests, they can completely avoid the need for handling retransmissions.

This section presents some patterns in which an endpoint can register securely without implementing more CoAP features.

9.1. Generic EDHOC in reverse flow

When such a endpoints uses EDHOC [I-D.ietf-lake-edhoc], it can follow this flow:

The endpoints requests simple registration with its RD. This request is unencrypted. Both for privacy reasons and to reduce configuration effort, the endpoints elides the ep registration parameter -- it will be established during authentication anyway.

TBD: Can we establish the correctness of the parameters somewhere later? (It could if it repeated any parameters in EAD3, at least as an empty item indicating that it's a simple registration).

The RD initiates an EDHOC exchange as part of its query for the endpoints's /.well-known/core resource. As the endpoints has the more vulnerable identity, EDHOC is performed in reverse message flow. After the RD has received message 3, it sends the GET request for the endpoints's /.well-known/core resource to complete the registration and responds to the original unencrypted registration.

TBD: Can the endpoints obtain confirmation that it is now registered? (It could, if the EAD3 item above is critical: then, the presence of OSCORE traffic with that key material implies acceptance of that EAD3.)

9.2. ACE roles

In an ACE context ([RFC9200]), the endpoints is the Resource Server (RS), and the RD is the client (C). This is aligned with the endpoints' capabilities: The RD is in a position to talk to the ACE Authorization Server (AS) and obtain a token to enable communication with the endpoints, and the endpoints does not need to perform any additional communication.

The token's audience may be "any endpoint eligible for RD registration" or a particular endpoint. Its subject is the RD. Its scope is to read the /.well-known/core resource.

In some scenarios it may make sense to operate the AS and the RD in a single system, in which case communication between those parties is cut short.

9.3. ACE EDHOC profile

When the ACE EDHOC profile [I-D.ietf-ace-edhoc-oscore-profile] is used, the RD needs to upload its token to the endpoints's authz-info endpoint (which is a term from ACE, using "endpoint" for a resource path and not for a host as the RD specification does) before executing EDHOC, because sending a token is only supported in EDHOC messages 1 and 3 (whereas the RD sends message 2). The posted token needs to be issued for the audience group of all eligible endpoints, as the RD does not know the identity of the endpoint at this stage. When the endpoint sends its credentials, the RD will know from matching the endpoint's credentials against the rs_cnf2 and aud2

list it obtained with the token (defined in Section 3.2 of [I-D.ietf-ace-workflow-and-params]) which endpoint is being registered.

Both parties can use kid as their ID_CRED_x to keep messages small. The endpoint receives the full ID_CRED of the RD as part of the signed token; the RD can look up the ID_CRED of the endpoint in its rs_cnf2 data.

Hypothetically, if a token were permitted to be sent in message 2, the RS could do that, and save the extra round-trip for POSTing the token.

Alternatively, the RD could initiate EDHOC in forward message flow. By the time it receives the endpoint's credentials (eg. kid) in message 2, it can ask the AS for a token suitable for that particular endpoint, or find a suitable token in a list it has obtained earlier (TBD: how?). This workflow has the downside of revealing the endpoint's ID_CRED_x to active attackers. This may be acceptable, especially when the endpoint is only sending a kid, and more so if the AS has a means of updating that ID.

9.4. ACE OSCORE profile

In the ACE OSCORE profile [RFC9203], a token is used that contains symmetric key material.

The message flow is similar to EDHOC and OSCORE: The endpoint sends an unencrypted registration request, but the endpoint needs to publicly reveal its identity by sending the ep registration parameter.

Then, the RD can obtain a token for this particular endpoint. Like in ACE EDHOC, it POSTs it to the endpoint's authz-info endpoint; in addition, it sends a random nonce and receives one in the response.

Without any further steps, it can then derive an OSCORE context from the token and the nonces, and send an OSCORE request for the endpoint's /.well-known/core resource.

This mode of operation is only recommended if the endpoint already makes its identity public for other reasons.

9.5. ACE OSCORE profile without ACE

When assigning the reversed ACE roles to the participants, there is a mode of operation that enables the ACE OSCORE profile while preserving privacy of the endpoint:

If the endpoint is provisioned with a public key of the AS in addition to the symmetric material it shares with it in the ACE OSCORE profile, the device can generate a token containing a secret key, symmstrically encrypt (or MAC it) for the AS, and asymmetrically encrypt it for the AS (eg. using Direct Key Agreement). It then initiates the ACE OSCORE profile with the RD, which needs to introspect the token at the AS to obtain the secret key material within.

This token's roles are different: Its subject is an endpoint, its audience is the RD, and its scope is to register with a particular endpoint name. The AS verifies during introspection whether the endpoint is actually eligible to do this.

It is unsure whether this whole process provides complexity benefits over the EDHOC based workflow, given that it does necessitate an asymmetric operation.

(Note that while it is well possible to perform ACE OSCORE profile without the asymmetrical step, for example by just symmetrically encrypting the token created by the endpoint, by provisioning the endpoint with a single token response containing a token encrypted to the RS, or with one containing an abbreviated token which the RS can introspect at the AS, this adds little compared to Section 9.4, because the endpoint's initial message will always contain identical parts that allow identification. The endpoint creating a random token and encrypting it symmetrically to the AS is almost viable and privacy preserving, but decrypting the token at the AS without any information identifying the symmetric ke would scale badly.)

10. References

10.1. Normative References

[I-D.amsuess-core-rd-replication]
Amsüss, C., "Resource Directory Replication", Work in Progress, Internet-Draft, draft-amsuess-core-rd-replication-02, 11 March 2019, <https://datatracker.ietf.org/doc/html/draft-amsuess-core-rd-replication-02>.

[RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/rfc/rfc6874>.

[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>.

[rfc9176]
Amsüss, C., Ed., Shelby, Z., Koster, M., Bormann, C., and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, DOI 10.17487/RFC9176, April 2022, <https://www.rfc-editor.org/rfc/rfc9176>.

10.2. Informative References

[I-D.ietf-ace-edhoc-oscore-profile] Selander, G., Mattsson, J. P., Tiloca, M., and R. Höglund, "Ephemeral Diffie-Hellman Over COSE (EDHOC) and Object Security for Constrained Environments (OSCORE) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-edhoc-oscore-profile-03, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-edhoc-oscore-profile-03>.

[I-D.ietf-ace-workflow-and-params] Tiloca, M. and G. Selander, "Alternative Workflow and OAuth Parameters for the Authentication and Authorization for Constrained Environments (ACE) Framework", Work in Progress, Internet-Draft, draft-ietf-ace-workflow-and-params-00, 2 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-workflow-and-params-00>.

[I-D.ietf-core-coap-pubsub] Jimenez, J., Koster, M., and A. Keränen, "A publish-subscribe architecture for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-13, 20 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-13>.

[I-D.ietf-core-coral] Amsüss, C. and T. Fossati, "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-06, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-coral-06>.

[I-D.ietf-core-transport-indication]
Amsüss, C., "CoAP Protocol Indication", Work in Progress, Internet-Draft, draft-ietf-core-transport-indication-03, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-transport-indication-03>.

[I-D.ietf-lake-edhoc] Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-

edhoc-23, 22 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-23>.

[I-D.tiloca-core-groupcomm-proxy] Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", Work in Progress, Internet-Draft, draft-tiloca-core-groupcomm-proxy-09, 31 August 2023, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-groupcomm-proxy-09>.

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

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, DOI 10.17487/RFC6887, April 2013, <https://www.rfc-editor.org/rfc/rfc6887>.

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

[RFC7556] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", RFC 7556, DOI 10.17487/RFC7556, June 2015, <https://www.rfc-editor.org/rfc/rfc7556>.

[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>.

[RFC8801] Pfister, P., Vyncke, É., Pauly, T., Schinazi, D., and W. Shao, "Discovering Provisioning Domain Names and Data", RFC 8801, DOI 10.17487/RFC8801, July 2020, <https://www.rfc-editor.org/rfc/rfc8801>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

[RFC9203] Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "The Object Security for Constrained RESTful Environments (OSCORE) Profile of the Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9203, DOI 10.17487/RFC9203, August 2022, <https://www.rfc-editor.org/rfc/rfc9203>.

[RFC9290]
Fossati, T. and C. Bormann, "Concise Problem Details for Constrained Application Protocol (CoAP) APIs", RFC 9290, DOI 10.17487/RFC9290, October 2022, <https://www.rfc-editor.org/rfc/rfc9290>.

Appendix A. Attic

Several extensions to the RD have been proposed in earlier versions of this document and were removed; this section summarizes them, lists where to look up the latest version, and gives reasons for their removal:

* Opportunistic RD (until -10)

  Describes how moderately capable devices can automatically configure and advertise themselves as an RD while no administratively configured RD is present.

  Removed due to large complexity and lack of real use cases.

* Lifetime age (until -10)

  References Section 5.2 of [I-D.amsuess-core-rd-replication] to allow administrators to see how much of a registration's lifetime has expired.

  Removed in favor of more generic provenance mechanisms described in Section 5.1 of [I-D.amsuess-core-rd-replication], and for lack of use cases.

* Lookup across link relations (until -10)

  Describes how a lookup may be combined ahead of time with requests for following more link relations.

  Removed in favor of utilizing [I-D.ietf-core-coral] FETCH requests.

Appendix B. Change log

Since -09:

* Added section on use with EDHOC and ACE security

* Moved Opportunistic RD, Lifetime age and Lookup across link relations into the newly created attic.

Since -08:

* Add section on propagating server generated information.

* Reference transport-indication appendix as one reason why propagation can be relevant.

Since -07:

* Update references.

Since -06:

* Add sketch for DNS updates.

* Add sketch for forward proxying.

* Fix erroneous section numbers.

Since -05:

* Add section on Limited Lifetimes.

* Point out limitations to applications that use reverse proxying.

* Minor reference and bugfix updates.

Since -04:

* Minor adjustments:

  - Mention LwM2M and how it is already doing RD proxying.

  - Tie proxying in with infinite lifetimes.

  - Remove note on not being able to switch protocols: RDs that support some future protocol negotiation can do that.

  - Point out that there is no Uri-Host from the RD proxy to the EP, but there could be.

  - Infinite lifetimes: Take up CTs more explicitly from RD discussion.

  - Start exploring interactions with groupcomm-proxy.

Since -03:

* Added interaction with PvD (Provisioning Domains)

Since -02:

* Added abstract

* Added example of CoRAL FETCH to Lookup across link relations section

Since -01:

* Added section on Opportunistic RDs

Since -00:

* Add multicast proxy usage pattern

* ondemand proxying: Probing queries must be sent from a different address

* proxying: Point to RFC7252 to describe how the actual proxying happens

* proxying: Describe this as a last-resort options and suggest attempting PCP first

Appendix C. Acknowledgements

[Reviews from: Jaime Jimenez]

Section 4 was inspired by Ben Kaduk's comments from reviewing [rfc9176].

Author's Address

Christian Amsüss

Email: christian@amsuess.com