

CoRE
Internet-Draft
Intended status: Experimental
Expires: 26 August 2021

C. Amsüss
22 February 2021

CoRE Resource Directory Extensions
draft-amsuess-core-resource-directory-extensions-05

Abstract

A collection of extensions to the Resource Directory [[I-D.ietf-core-resource-directory](#)] that can stand on their own, and have no clear future in specification yet.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on 26 August 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Reverse Proxy requests	3
2.1.	Discovery	3
2.2.	Registration	3
2.2.1.	Registration updates	4
2.2.2.	Proxy behavior	5
2.2.3.	On-Demand proxying	5
2.2.4.	Multiple upstreams	5
2.2.5.	Examples	6
2.2.6.	Notes on stability and maturity	7
2.2.7.	Security considerations	7
3.	Infinite lifetime	7
3.1.	Example	8
4.	Lookup across link relations	8
4.1.	Example	9
5.	Lifetime Age	10
6.	Zone identifier introspection	10
6.1.	Example	11
7.	Proxying multicast requests	11
7.1.	Example	12
8.	Opportunistic RD	12
8.1.	Applications	15
9.	References	15
9.1.	Normative References	15
9.2.	Informative References	16
Appendix A.	Change log	17
Appendix B.	Acknowledgements	18
	Author's Address	18

[1.](#) Introduction

This document pools some extensions to the Resource Directory [[I-D.ietf-core-resource-directory](#)] 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.

[This document is being developed at <https://gitlab.com/chrysn/resource-directory-extensions> (<https://gitlab.com/chrysn/resource-directory-extensions>).]

[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).

[I-D.ietf-core-resource-directory] 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 clients 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 mint several alternative Registration Base URIs using different protocols to make the proxied content available; [\[I-D.silverajan-core-coap-protocol-negotiation\]](#) can be used to advertise them.

The registrant is not informed of the chosen public name by the RD.

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 ([\[I-D.ietf-core-resource-directory\]](#) 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.

[2.2.2.](#) 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" parameter 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.

[2.2.3.](#) 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].

[2.2.4.](#) 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 RDA0, 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.

[2.2.5.](#) Examples

[2.2.5.1.](#) 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.

[2.2.5.2.](#) 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.

[2.2.6.](#) 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.]

[2.2.7.](#) 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 adequately 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.

[3.](#) Infinite lifetime

An RD can indicate support for infinite lifetimes by adding the resource 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, software 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 2.2.5.2](#) discovered support for infinite lifetimes during lookup like this:

```
Req: GET coaps+ws://rd.example.net/.well-known/coer?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.](#) Lookup across link relations

Resource lookup occasionally needs execute multiple queries to follow links.

An RD server (or any other server that supports [\[RFC6690\]](#) compatible lookup), can announce support for following links in resource lookups by announcing support for the TBD3 interface type on its resource lookup.

A client can the query that server to not only provide the matched links, but also links that are reachable over relations given in "follow" query parameters.

[4.1.](#) Example

Assume a node presents the following data in its `<.well-known/core>` resource (and submitted the same to the RD):

```
</temp>;if="core.s";rt="example.temperature",  
</t-prot>;rel="calibration-protocol";anchor="/temp",  
<http://vendor.example.com/temp9000>;rel="describedby";anchor="/temp",  
</hum>;if="core.s";rt="example.humidity",  
</h-prot>;rel="calibration-protocol";anchor="/hum",
```

A lookup client can, in one query, find the temperature sensor and its relevant metadata:

Req: GET `/rd-lookup/res?rt=example.temperature&follow=calibration-protocol&follow=describedby`

```
<coap://node1/temp>;if="core.s";rt="example.temperature";anchor="coap://node1",  
<coap://node1/t-prot>;rel="calibration-protocol";anchor="coap://node1/temp",  
<http://vendor.example.com/temp9000>;rel="describedby";anchor="coap://node1/temp"
```

[There is a better example (<https://github.com/ace-wg/ace-oauth/issues/120#issuecomment-407997786>) in an earlier stage of [\[I-D.tiloca-core-oscore-discovery\]](#)]

Given the likelihood of a CoRAL based successor to [\[RFC6690\]](#), this lookup variant might easily be superseded by a CoRAL FETCH format; it might look like this there:

```
Req: FETCH /reef-lookup
Content-Format: application/template-coral+cbor
Payload:
#using core = <...>
#using reef = <...>
reef:content ?x {
  core:rt "example.temperature"
  calibration-protocol ?y {
    core:describedby ?z
  }
}

Res: 2.01 Content
Content-Format: application/coral+cbor
Payload:
reef:content <coap://node1/temp> {
  core:rt "example.temperature"
  calibration-protocol <coap://node1/t-prot> {
    core:describedby <http://vendor.example.com/temp9000>
  }
}
```

5. Lifetime Age

This extension is described in [[I-D.amsuess-core-rd-replication](#)] [Section 5.2](#).

The "provenance" extension in [Section 5.1](#) of the same document should probably be expressed differently to avoid using non-target link attributes.

6. Zone identifier introspection

The 'split-horizon' mechanism introduced in [[I-D.ietf-core-resource-directory](#)] (-19) (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.

[6.1.](#) 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",
```

[7.](#) 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.

[7.1.](#) 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:db8::1]"

<coap://[2001:db8::2]/light>;ct="0 112";rt="core.s";anchro="coap://[2001:db8::2]"

8. Opportunistic RD

An application that wants to advertise its resources in Resource Directory can find itself in a network that has no RD deployed. It may be able to start an RD on its own to fill in that gap until an explicitly configured one gets installed.

This bears the risk of having competing RDs on the same network, where resources registered at one can not be discovered on the other. To mitigate that, such Opportunistic Resource Directories should follow those steps:

- * The RD chooses its own Opportunistic Capability value. That integer number is an estimate of number of target attributes it expects to be able to store, where in absence of better estimates one would assume that a registration contains 16 links, and each links contains three target attributes each with an eight byte key and a 16 byte value.

The Opportunistic Capability value is advertised as a TBD5-cap= target attribute on the registration resource.

- * The RD chooses its own Opportunistic Tie-Break value. That integer number needs no other properties than being likely to be different even for two instances of the same device being started;

numeric forms of MAC addresses or random numbers make good candidates.

The Opportunistic Capability value is advertised as a TBD5-tie= target attribute on the registration resource.

- * The Opportunistic RD, before advertising its resources, performs RD discovery itself, using at least all the discovery paths it may become discoverable on itself.
- * If the Opportunistic RD finds no other RD, or if the RD it finds is less capable than itself, it can start advertising itself as a Resource Directory.

An RD is called more capable than another if its TBD5-cap value is greater than the other's, or if its TBD5-tie value is greater than the other's if the former results in a tie. Absent or unparsable attributes are considered greater than any present attribute.

In case an RD observes a tie even after evaluating the tie breaker, it may change its Opportunistic Tie-Break value if that was picked randomly, or reevaluate its life choices if it uses its own MAC address.

- * A running Opportunistic RD needs to perform discovery for other RDs repeatedly. If it discovers a more capable RD, it stops advertising its own resources. It should continue to serve lookup requests, but refuse any new registration or registration updates (which will trigger the registrant endpoints to look for a new RD).

An inactive Opportunistic RD will be notified of the higher capability RD's shutdown by the expiry of whatever it may be started to advertise that was now advertised there; see below for possible improvements.

- * An RD that discovers an Opportunistic RD of lower capability may speed up the transition process by (not mutually exclusive) two ways
 - It can register its own (registration) resource(s) into the lower capability directory. That RD can take that as having

discovered a higher capability RD and stop advertising.

- It can expose resources and registrations of the lower capability directory using the methods described in [[I-D.amsuess-core-rd-replication](#)].
- * An Opportunistic RD that yields to a more capable RD may ease the transition by attempting to register its active registrations at the more capable RD, taking the role of a CT. The lifetimes picked for those must not exceed the remaining lifetime of its registrations, and it must not renew those registrations.

Future iterations of this document may want to cut down on the possibilities listed above.

Some ideas are around for making the shutdown of a commissioned or otherwise high-capability RD more graceful, but they still have some problems

- * Setting a commissioned or high-capability RD's Capability to zero in preparation of the shutdown may create loops in any distributed lookups.
- * Asking the lower capability RD to register its registration resource (even though not otherwise advertised) at the higher capability RD still creates a situation where clients may find two RDs running simultaneously, and we can't expect clients to make any decisions based on TBD5 values.
- * Asking the higher capability RD to register its registration resource with the lower capability RD contradicts the current recommendation for the passive Opportunistic RD to not accept registrations / renewals. Also, the deployed RD may not know that Opportunistic RDs are a thing.

- * Advertising an almost-but-not-quite `rt=` value on passive registration resources may be an option, but needs to be thought through thoroughly.

Installations of Opportunistic RDs are at special risk of resource exhaustion because they are not sized with their actual deployment in mind, but rely on defaults set by the application that starts the RD. Opportunistic RDs should only be started if the application's administrator can be informed in a timely fashion when the RD's resources are nearing exhaustion; guidance towards installing a more capable RD on the network should be provided in that case.

[8.1.](#) Applications

- * Group managers using [[I-D.tiloca-core-oscore-discovery](#)] can ship its own low-priority Opportunistic RD to announce its join resources. This provides benefits over announcing them on multicast discovery if the network can efficiently route requests to the All CoAP Resource Directories multicast address (so group members get a response back from an early focused request to all RDs rather than falling back to multicasting All CoAP Nodes for "?rt=osc.j&..."), or if discovery of the group's multicast address is used.
- * Administrative tools that try to provide a broad overview of a network's CoAP devices could offer to open an Opportunistic RD if they find no active RD on the network (but should ask the user in interactive scenarios).

That allows them to see devices that newly join the network quickly (by observing their own or the found RD), rather than relying periodic multicasts.

[9.](#) References

[9.1.](#) Normative References

[I-D.amsuess-core-rd-replication]

Amsuess, C., "Resource Directory Replication", Work in Progress, Internet-Draft, [draft-amsuess-core-rd-replication-02](#), 11 March 2019, <<http://www.ietf.org/internet-drafts/draft-amsuess-core-rd-replication-02.txt>>.

[I-D.ietf-core-resource-directory]

Amsuess, C., Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, [draft-ietf-core-resource-directory-26](#), 2 November 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-core-resource-directory-26.txt>>.

[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/info/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/info/rfc7252>>.

[9.2](#). Informative References

[I-D.silverajan-core-coap-protocol-negotiation]

Silverajan, B. and M. Ocaik, "CoAP Protocol Negotiation", Work in Progress, Internet-Draft, [draft-silverajan-core-coap-protocol-negotiation-09](#), 2 July 2018, <<http://www.ietf.org/internet-drafts/draft-silverajan-core-coap-protocol-negotiation-09.txt>>.

[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-02](#), 2 November 2020, <<http://www.ietf.org/internet-drafts/draft-tiloca-core-groupcomm-proxy-02.txt>>.

[I-D.tiloca-core-oscore-discovery]

Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, [draft-tiloca-core-oscore-discovery-07](#), 2 November 2020, <<http://www.ietf.org/internet-drafts/draft-tiloca-core-oscore-discovery-07.txt>>.

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

Internet-Draft

CoRE Resource Directory Extensions

February 2021

- [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/info/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/info/rfc7390>>.
- [RFC7556] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", [RFC 7556](#), DOI 10.17487/RFC7556, June 2015, <<https://www.rfc-editor.org/info/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/info/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/info/rfc8801>>.

[Appendix A](#). Change log

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:

Amsüss

Expires 26 August 2021

[Page 17]

Internet-Draft

CoRE Resource Directory Extensions

February 2021

- * 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 B](#). Acknowledgements

[Reviews from: Jaime Jimenez]

Author's Address

Christian Amsüss

Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: christian@amsuess.com