CoRE
Internet-Draft
Intended status: Informational
Expires: August 4, 2018

C. Amsuess
January 31, 2018

Resource Directory Replication
draft-amsuess-core-rd-replication-00

Abstract

Discovery of endpoints and resources in M2M applications over large networks is enabled by Resource Directories, but no special consideration has been given to how such directories can scale beyond what can be managed by a single device.

This document explores different ways in which Resource Directories can be scaled up from single network to enterprise and global scale. It does not attempt to standardize any of those methods, but only to demonstrate the feasibility of such extensions and to provide terminology and exploratory groundwork for later documents.

Table of Contents

1. Introduction

[See abstract for now.]

This document is being developed in a git based workflow. Please see https://github.com/chrysn/resource-directory-replication [1] for more details and easy ways to contribute.

2. Terminology

This document assumes familiarity with [RFC7252] and [I-D.ietf-core-resource-directory] and uses terminology from those documents.

Examples in which URI paths like "/rd" or "/rd-lookup/res" are used assume that those URIs have been obtained before by an RD Discovery process; these paths are only examples, and no implementation should make assumptions based on the literal paths.

3. Goals of upscaling

The following sections outline different reasons why a Resource Directory should be scaled beyond a singe device. Not all of them will necessarily apply to all use cases, and not all solution approaches might be suitable for all goals.

3.1. Large numbers of registrations

Even at 1kB of link data per registration, modern server hardware can easily keep the data of millions of registrations in RAM simultaneously. Thus, the mere size of registration data is not expected to be a factor that requires scaling to multiple nodes.

The traffic produced when millions of nodes with the default 24h lifetime amounts to dozens of exchanges per second, which is doable with equal ease at central network equipment.

However, if the directory has additional interaction with its registered nodes, for example because it provides proxying to registered endpoints, resources like file descriptors can be exhausted earlier, and the traffic load on the registration server grows with the traffic it is proxying for the endpoint.

3.2. Large number of requests

Not all approaches to constrained restful communication use the Resource Directory only in the setup stage; some are might also utilize a Resource Directory in more day-to-day operation.

[TODO: get some numbers on how many requests a single RD can deal with.]

3.3. Redundancy

With the RD as a central part of CoRE infrastructures, outages can affect a large number of users.

A decentralized RD should be able to deal both with scheduled downtimes of hosts as well as unexpected outages of hosts or parts of the network, especially with network splits between the individual parts of the directory.

4. Approaches

In this section, two independent chains of approaches are presented. The "shared authority" approach (using anycast or DNS aliases), and proxy-based caching (in stages from using generic proxies to RD replication that only bears little resemblance to proxies).

Elements from those chains can be mixed.

4.1. Shared authority

With this approach, a single host and port (or "authority" component in the generic URI syntax) is used for all interactions with the RD.

This can be implemented using a host name pointing to different IP addresses simultaneously or depending on the requester's location, using IP anycast addresses or both.

From the client's or proxy's point of view, all interaction happens with same Origin Server.

In this setup, the replication is hidden from the REST interactions, and takes place inside the RD server implementation or its database backend.

Compared to the other approaches, this is more complex to set up when it involves managing anycast addresses: Running an IPv4 anycast network on Internet scale requires running an Autonomous System. In either variation, all server instances are tightly coupled; they need shared administration and probably need to run the same software.

The replication characteristics are laregly inherited from the underlying backend.

As registering endpoints only store the URI constructed from the Location-Path option to their registration request, registration updates can end up at any instance of the server, though they are likely to reach the same one as before most of the time.

Spontaneous failure of individual nodes can interrupt endpoints' registrations in scenarious that do not use anycast addresses until the unusable addresses have left DNS caches.

4.2. Plain caching

Caching reverse proxies that are not particularly aware of a Resource Directory can be used to mitigate the effect of large numbers of requests on a single RD server. In this approach, there exists a single central RD server instance, but proxies are placed in front of it to reduce its load.

Caching is applicable only to the lookup interfaces; the POST request used in registration and renewal are not cacheable.

A prerequisite for successful caching is that fresh copies exist in the cache; this is likely to happen only if there are many alike requests to the Resource Directory. The proxy can than serve cached copies, and might find it advantageous to observe frequent queries.

The simplest way to set up such proxying is to have the proxies forward all requests to the central RD and to advertise only the proxies' addresses.

Due to the discovery process of the RD, operators can also limit the proxies to the lookup interfaces and advertise the central server for registration purposes. A sample exchange between a node and its 6LoWPAN border router could be:

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

Res: 2.05 Content
<coap://central-rd.example.com/rd>;rt="core.rd",
<coap://europe3.proxy.rd.example.com/rd-lookup/res>;rt="core.rd-lookup-res",
<coap://europe3.proxy.rd.example.com/rd-lookup/ep>;rt="core.rd-lookup-ep"
```

Special care should be taken when a reverse proxy is not accessed by the client under the same address as the origin server, as relative references change their meaning when served from there. This can be ignored completely on the resource lookup interface (as long as the provenance extension is not used); ignoring it on the endpoint lookup interface gives the client "wrong" results, though that is likely to only matter to applications that use both the lookup and the registration interface, like Commissioning Tools could do. Proxies can be configured to do content transcoding (cf. [RFC8075] Section 6.5.2) to preserve the lookup responses' original meanings.

This approach does not help at all with large numbers of registrations. It can mitigate issues with large numbers of lookup requests, provided that many identical requests arrive at the proxy. The effect on the redundancy goal is negligible: The proxy can provide lookup results only for as long as the cache is fresh during a central server outage, which is 60 seconds unless the RD server says otherwise.

This approach can be run with off-the-shelf RD servers and proxies. The only configuration required is for the proxy to have a forwarding address, and for the RD (or its announcer) tho know which lookup addresses to advertise.

4.3. RD-aware caching

Similar to the above, specialized proxies can be employed that are aware that their target is an RD lookup address.

The "plain caching" approach is limited in that it requires a small set of lookups to be frequently performed. A proxy that is aware that the address it is forwarding to is of the Resource Type "core.rd-lookup-*" can utilize knowledge of how an RD works to serve more specialized requests as well from fresh generic content.

For example, assume that the proxy frequently receives requests of the shape

```
Req: GET /rd-lookup/res?rt=core.s&rt=ex.temperature&ex.building=8341&title=X
```

for arbitrary values of X. Then it can use the following request to keep a fresh cache:

```
Req: GET coap://rd.example.com/rd-lookup/res?rt=core.s&rt=ex.temperature
    &ex.building=8341
Observe: 1
```

and from that serve filtered responses to individual requests.

This method shares the advantages of plain caching, with reduced limitations but requiring specialized proxying software. The software does not necessarily need more configuration: A general-purpose proxy is free to explore the origin server's ".well-known/core" information, and can decide to enable RD optimizations after discovering that the frequently accesses resources are of resource type "core.rd-lookup-*".

4.3.1. Potential for improvement

Observing a large lookup result is relatively inefficient as the complete document needs to be transferred when a change happens. Serializations of web links that are suitable for expressing small deltas are expected to be developed for PATCH operations on registration resources. If those formats are compatible with observation, they can be applied directly. Otherwise, the proxy can try to establish a "push" dynamic link ([I-D.ietf-core-dynlink]) to receive continuous PATCH updates on its resource.

The applicability of the RD-aware approach is further limited to query parameters of which the proxy knows that they are not subject to lookup filtering on other entities than the queried one. In the example above, were the variable part the "d" attribute (of endpoints, as opposed to the "title" of resources), the proxy could not do the filtering on its own becaus it would not have the required information. Even the above example does not allow for fully accurate replication, as the endpoint _might_ register with a "title" endpoint attribute, even though no such attribute is specified right now. It might be worth considering to be more explicit about filtering "up" and "down" in the hiearchy in the RD specification. Also, annotating the links in the endpoint lookup with information about which registration they belong to would help the proxy keep all the data around to solve more complex queries. The provenance extension is proposed for that purpose.

In its extreme form, the proxy can observe the complete lookup resources of the Resource Directory. It can then answer all queries on its own based on the continuously fresh state transferred in the observations. That form requires the RD to support the provenance extension.

For such proxies, it can be suitable to configure them to use stale cache values for extended periods of time when the RD becomes intermittently unavailable.

4.4. Distinct registration points

Caching proxies that are aware of RD semantics could be extended to gather information from more than one Resource Directory.

When executing queries, they would consider candidates from all configured upstream servers and report the union of the respective query results. At this stage, it is highly recommended that content transcoding takes place.

With this approach, many distinct registration URIs can be advertised, for example due to geographic proximity.

Unlike the other proxying approaches, this helps with the "large number of registrations" goal. If that number is unmanageable for single devices, proxies need not keep full copies of all the RDs' states but rather send out queries to all of their upstreams, behaving more like the "plain caching" proxies. This multiplies the lookup traffic, but allows for huge numbers of registrations. The problems of "too many lookups" versus "too many registrations" can be traded off against each other if the proxies keep parts of the RDs' states locally at hand while forwarding more exotic requests to all RDs.

4.4.1. Redundancy and handover

This approach also tackles the redundancy goal. When an endpoint registeres at its RD, the RD updates its endpoint and resource lookup results and includes the registration data until further notice (for correct operation, the "Lifetime Age" extension is useful).

If at some point in time that RD server becomes unavailable, the proxies can keep the cached information around. Before the lifetime expires, the endpoint will attempt to renew its registration and find that the RD is unavailable. It will then go through discovery again, find the most recently advertised registration URI or pick another one out of a set (see seciton on Recommendations) and start a new registration there.

If the lookup proxies do not evict the old (and soon-to-time-out) registration when the new one on a different RD with the same endpoint name and domain arrives, at worst there will be the same information twice from two registration resources available for lookup.

4.4.2. Loops between RDs and proxies

In this configuration, it can be tempting to run a Resource Directory and a lookup proxy (aimed at multiple resource directories) on the same host.

[It might be easier to recommend simply using different hosts, at least host names, in those cases, or anything else that allows direct and not publically advertised access to the real RDs' lookups.]

In such a setup, other aggregating lookup proxies must take care to only select locally registered entries. With the current filtering

rules, observing the resources "/rd-lookup/ep?href=/*" and "/rd-lookup/res?provenance=/*" crudely provides that kind of filtering.

5. Recommendations to RD

- Explicitly allow "foreign" URIs in discovery and endpoint lookup
  - This is already being done for group memberships.
  - This doesn't change a thing about there not being a "Location-Host" - the registration is still with the server the registration was sent to.
- Say something about what to do on registration or renewal failure: When should discovery be restarted?
  - "Retry when Max-Age is reached on 5.03 up to N times, and then (or on other errors) restart discovery and round-robin through choices"?
- Reconsider the filtering rules, make hierarchy traversal explicit.
- Allow empty submissions (see global example)

6. Proposed RD extensions

6.1. Provenance

In order for an RD-aware proxy to serve resource lookup requests that filter on endpoint parameters, the proxy needs a way to tell which endpoint registration submitted that link. That information might also be useful for other purposes.

This introduces a new link attribute "provenance". Its value is a URI reference as described by [RFC3986] Section 4.1. The URI is to be interpreted by the same rules that apply to the "anchor" attribute, namely by resolving the reference relative to the requested document's URI. The attribute should not be repeated, and in presence of multiple attributes, only the last should be considered.

[TODO: If a something link-format-ish comes up during the development of this document which allows setting base-hrefs in-line, evaluate whether it really makes sense to inherit anchor's rules or whether it's better to phrase it in a way that the requested base URI always counts.]

The URI given in the "provenance" attribute describes where the information in the link was obtained from. An aggregator of links can thus declare its sources for each link.

It is recommended that a Resource Directory adds the URI of the registration resource to resource lookups. Thus, if an endpoint registers as

```
Req: POST /rd?ep=node1
Payload:
</sensors/temp>;if="core.s"

Res: 2.01 Created
Location: /reg/1234
```

then a lookup will add a provenance attribute:

```
Req: GET /rd-lookup/res?if=core.s

Res: 2.05 Content
Payload:
<coap://.../sensors/temp>;if="core.s";anchor="coap://...";
    provenance="/reg/1234"
```

This is not an IANA consideration as there is no established registry of link attributes.

By itself, the provenance attribute does not need to be registered in the RD Parameters Registry because it is just another link attribute. If it is desired that provenance information is only shown on request (eg. by RD-aware proxies), a parameter can be introduced there:

- Full name: Link provenance
- short: provenance
- Validity: URI
- Use: Resource lookup only
- Description: If "provenance" or any string starting with "provenance=" is given as one of the ampersand-delimited query arguments, the RD is instructed to add the provenance attribute to all looked up links; otherwise, the RD will not present them. The filtering rules still apply: If there is a "=" sign in the query argument, only links with matching provenance will be reported.

6.2. Lifetime Age

The result of an endpoint lookup as a whole has inhomogenous cache properties that would determine its Max-Age:

- The document can change at any time when a new endpoint registers.
- The document can change at any time when an endpoint deregisters.
- Each record can be expected to not change until its lifetime has expired.

As currently specified, a lookup client has no way to tell where in its lifetime an endpoint is. Therefore, a new link attribute is suggested that allows the RD to share that information:

The new link attribute Lifetime Age (lt-age) is described for use in RD Endpoint Lookups. Valid values are integers from 0 to the lifetime of the registration. The value indicates how many seconds have passed since the endpoint last renewed its registration.

Care has to be taken when replicating this value in caches, as the caching agent might be unaware of the attribute's semantics and not update it. (This is unlike the Max-Age attribute, which a caching agent needs to understand and reduce accordingly when serving from the cache). It should therefore only be used with responses that carry the default Max-Age of 60 or less.

Clients that use the lookup interface (especially RD-aware proxies) are free to treat that record and its corresponding resource records as fresh until after the difference of lt and lt-age seconds have passed since the endpoint lookup result was obtained, especially if the origin server has become unavailable.

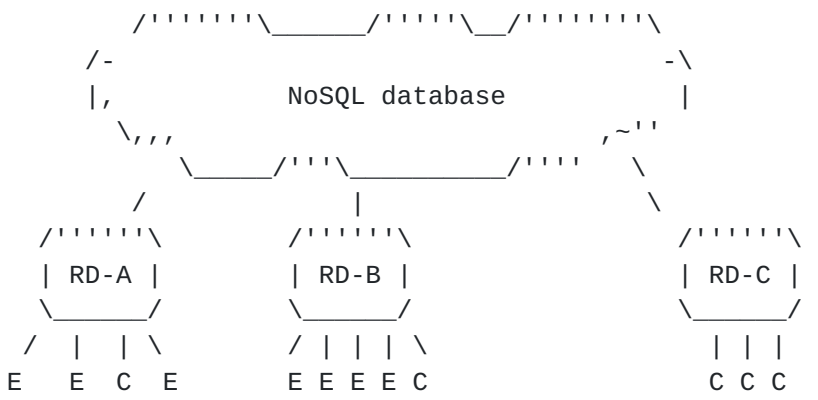
Security considerations: Given that this leaks information about the endpoint's communication patterns, it may be prudent for an RD only to reveal this information on a need-to-know basis.

7. Example scenarios

7.1. Redundant and replicated resource lookup (anycast)

This scenario describes a setup where millions of devices register in a company-wide Resource Directory.

The directory is scaled using the shared authority / anycast approach, and the RD implementation is backed by a NoSQL-style distributed database.



("E" and "C" represent endpoints and lookup clients, respectively)

Both endloints and lookup clients receive the RD address "2001:db8::an1:ca57" is announced to all devices on the network using the RDAO option in IPv6 Neighbor Discovery. Any packages to that addresses are routed by the network to the closest of the three RD instances A, B and C. Discovery invariably looks like this:

```
Req: GET coap://[2001:db8::an1:ca57/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd",
</rd-lookup/res>;rt="core.rd-lookup-res",
</rd-lookup/ep>;rt="core.rd-lookup-ep"
```

An endpoint close to B would therefore register with

```
Req: POST coap://[2001:db8::an1:ca57]/rd?ep=endpoint1&
    d=facility23.eu.example.com
Payload:
</sensors/temp>;if="core.s"

Res: 2.01 Created
Location: /reg/123e4567-e89b-12d3-a456-426655440000
```

Any client could immediately see that the endpoint is registered by issuing

```
Req: GET coap://[2001:db8::an1:ca57]/rd-lookup/ep?ep=endpoint1&
    d=facility23.eu.example.com

Res: 2.05 Content
Payload:
</reg/123e4567-e89b-12d3-a456-426655440000>;ep="endpoint1";
    d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"
```

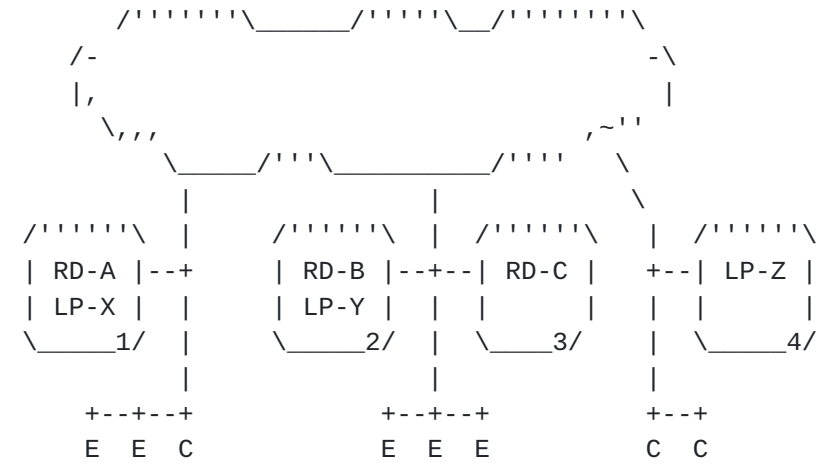
If at any point in time the RD instance B becomes unavailable, the registering endpoint's renewal requests will be routed to the next available instance, for example A. That instance can update the shared database with renewed lifetime just as B would have done.

How this performs under a net split depends on the database backend. Registration resources based on UUIDs were chosen in this example because those would allow the system to keep accepting new registrations even in a netsplit situation; the risk of the registration request not being idempotent towards a node that switches sides during such a split is considered acceptable.

7.2. Redundant and replicated resource lookup (distinct registration points)

This scenario takes place in the same environment as the previous one.

Rather than a shared database, distinct registration points are advertised. The advertised registration points are called RD-A to RD-C; independent of them are lookup proxies LP-X to LP-Z. Some of them run on the same hosts.



The lookup proxies in this scenario are constantly observing the "/rd-lookup/ep?href=/*" and "/rd-lookup/res?provenance=/*" resources of known RDs on other hosts, and might get updated internally with state from a co-hosted RD or observe that using an internal interface. As there is really suitable content format and observation mechanism for those yet, the exchanges are partially described in words here.

RDAO announcements point to the nearest host (whose IP address ends with the numbers of the respective box in the figure), and hosts that do not serve both functions provide lookup as follows:

```
   Req: GET coap://[2001:db8:23::3]/.well-known/core?rt=core.rd*

   Res: 2.05 Content
   Payload:
   </rd>;rt="core.rd",
   <coap://[2001:db8:23::2]/rd-lookup/ep>;rt="core.rd-lookup-ep",
   <coap://[2001:db8:23::2]/rd-lookup/res>;rt="core.rd-lookup-res"

   When a client then registers as

   Req: POST coap://[2001:db8:23::3]/rd?ep=endpoint1&
       d=facility23.eu.example.com
   Payload:
   </sensors/temp>;if="core.s"

   Res: 2.01 Created
   Location: /reg/42

   the RD at 3 sends notifications to the observing lookup proxies X, Y
   and Z:

 Res: Patch Result
 Add one record: </reg/42>;ep="endpoint1";d="facility23.eu.example.com";
     con="coap://[2001:db8:23::1]";lt-age=0

   As soon as that is processed, clients can query LP-Z

   Req: GET coap://[2001:db8:4::4]/rd-lookup/ep?ep=endpoint1&
       d=facility23.eu.example.com

   Res: 2.05 Content
   Payload:
   <coap://[2001:db8:23::3]/reg/42>;ep="endpoint1";
       d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"

   (Note that lt-age is elided to the client as per the security
   considerations for that information).

   When a net split happens that cuts LP-Z's site off the rest, it keeps
   that information available until the lt-age runs out.

   When RD-C unexpectedly becomes unavailable, endpoint1 fails to renew
   its registration.  It then starts the RD discovery process again,
   picks the next available RD (this time B) and gets a new registration
   from that.

   RD-B then sends an update to the proxies:
```

```
Res: Patch Result
Add one record: </reg/11>;ep="endpoint1";d="facility23.eu.example.com";
    con="coap://[2001:db8:23::1]";lt-age=0
```

The proxies remove C's registration "/reg/42" based on the duplicate name and now answer requests like this:

```
Req: GET /rd-lookup/ep?ep=endpoint1&d=facility23.eu.example.com

Res: 2.05 Content
Payload:
<coap://[2001:db8:23::2]/reg/11>;ep="endpoint1";
    d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"

Req: GET /rd-lookup/res?if=core.s&d=facility23.eu.example.com

Res: 2.05 Content
Payload:
<coap://[2001:db8:23::1]/sensors/temp>;if="core.s";
    anchor="coap://[2001:db8:23::1]/sensors/temp";
    provenance="coap://[2001:db8:23:2]/reg/11",
...
```

7.2.1. Variation: Large number of registrations, localized queries

If the lookup proxies are not capable of keeping track of all the registered data, they can opt to forward requests to all the RDs instead. In this example, queries are often localized (queries within a building are often limited to the same building), so LP-Y could decide to only keep two particular observations active to each RD:

- "/rd-lookup/ep?href=/*&d=facility23.eu.example.com"

- "/rd-lookup/res?provenance=/*&d=facility23.eu.example.com"

With those observed, it could still accurately respond to the above queries without calling out to the other RDs.

If a query came in as "/rd-lookup/res?if=core.s", it would still need to forward that query to all RDs to build an overview of all sensors in the network for the requester.

7.2.2. Variation: Combination with anycast

In a variation of this, all the RDs and LPs can use a shared anycast address. They would be then advertised as in the anycast/NoSQL example.

All RDs would need to be configured such that they encode their host name in their path (eg. "/reg/rd-c/42"). Nodes must then have proxy forwarding rules set up such that

- "/rd" is served from the local RD if there is one, or forwarded to any (the closest) RD
- "/reg/*" requests are served if hosted locally, otherwise forwarded to the appropriate RD, or respond with a 5.04 Gateway timeout if that is not available any more
- Lookup request are served from the local lookup proxy, or forwarded to the closest one on RD-only hosts.

Such a setup is easier if all hosts provide both registration and lookup functionality.

7.3. Anonymous global endpoint lookup

This scenario describes a way to provide connectivity into devices in difficult network situations based on identifiers of their cryptographic keys, the KID context identifiers of OSCORE. A global network of untrusted Resource Directory servers is built, and the individual servers provide network relaying services for endpoints that operate behind NAT or firewalls.

It assumes the existance of two other hypothetical mechanisms:

- The RD Parameter named "proxy".

  An endpoint can ask the RD to act as a reverse proxy for it by adding the "proxy" registration parameter; an RD that does proxying disregards the implicit "con" parameter and announces a name of its own instead.

- A URI scheme called "oscore".

  A URI of the form "oscore://VGhh...2aWNl/sensor/temp" refers to a resource "/sensor/temp/" on any OSCORE capable host with which the client has a key established under the KID context given by the base64 string in the authority component.

  To resolve the URI to a concrete protocol and socket, a form of Resource Directory assisted protocol negotiation is used.

RD servers join a global pool of servers using a protocol that is not further described here, but could conceivably be based on distributed hash tables (DHTs).

Endpoints register only with a key derived name, and usually do not provide any links because those would be accessible only to authenticated requesters.

They register at any of a set of preconfigured DNS names for finding a Resource Directory. Those names resolve to any of the currently active RD servers, where geographic proximity could play a role in the choice of address returned.

When the endpoint discovers the registration URI (for which it uses coap+tcp to make later proxying more stable), the server returns links to its explicit IP address:

```
<coap+tcp://[2001:db8:1:2::3]/rd>;rt="core.rd",
<coap+tcp://[2001:db8:1:2::3]/rd-lookup/ep>;rt="core.rd-lookup-ep"
```

(This avoids conflict when the DNS assignment flips and a different host (on which the registration resource is unknown) is returned. Alternatively, the servers could use a unified scheme of registration resource naming like "/reg/${name}" or a UUID-based scheme.)

The endpoint then registers:

```
Req: POST coap+tcp://[2001:db8:1:2::3]/rd?proxy&ep=VGhhdCdzIHRoZSB\
    LZXlJZENvbnRleHQgdXNlZCB3aXRoIHRoaXMgZGV2aWNl
Payload: empty

Res: 2.01 Created
Location: /reg/123
```

When a client wants to talk to that registered server, its RD discovery process will yield another instance, which it then queries:

```
Req: GET coap://[2001:db8:4:5::6]/rd-lookup/ep?ep=VGhhdCdzIHRoZSBL\
    ZXlJZENvbnRleHQgdXNlZCB3aXRoIHRoaXMgZGV2aWNl
```

The server will look up the given ep name in the backing DHT, and forward the request right to the (precisely: any) RD server that has announced that ep value, which then answers:

```
Res: 2.05 Created
Payload:
<coap+tcp://[2001:db8:1:2::3]/reg/123>;ep="VGhh...2aWNl";
    con="coap://[2001:db8:1:2::3]:10123";
    at="coap+tcp://[2001:db8:1:2::3]:10123"
```

(This particular server uses multiple ports to tell traffic for different endpoints apart; it could just as well use a catch-all DNS

record, do name based virtual hosting and announce "con="coap://reg123.server3.example.com" instead.)

The client will then use the discovered address to direct its OSCORE requests to, and the RD server will proxy for it.

Note that while this setup _can_ serve as a generic RD and answer resource requests as well, it is doubtful whether there would be any interest in it given the data becomes public, and is limited by the necessity to have an "ep=" filter in all requests lest the network be flooded with requests.

8. References

8.1. Informative References

[I-D.ietf-core-dynlink]
Shelby, Z., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-04 (work in progress), September 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-12 (work in progress), October 2017.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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>.

[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/info/rfc8075>.

8.2. URIs

[1] https://github.com/chrysn/resource-directory-replication

Author's Address

Christian Amsuess
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639