CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: March 31, 2016

T. Zotti
Philips Research
P. van der Stok
Consultant
E. Dijk
Philips Research

September 28, 2015

Sleepy CoAP Nodes
draft-zotti-core-sleepy-nodes-04

Abstract

Control networks rely on application protocols like CoAP to enable RESTful communications in constrained environments. Many of these networks make use of "Sleepy Nodes": battery powered devices that switch off their (radio) interface during most of the time to conserve battery energy. As a result of this, Sleepy Nodes cannot be reached most of the time. This fact prevents using normal communication patterns as specified in the CoRE group, since the server-model is not applicable to these devices. This document discusses and specifies an architecture to support Sleepy Nodes such as battery-powered sensors in mesh networks with the goal of proposing a standardisation solution for Sleepy Node proxies.

Table of Contents

1. Introduction

Control networks rely on application protocols such as CoAP to enable RESTful communications in constrained environments. Many of these networks feature "Sleepy Nodes": battery-powered nodes which switch on/off their communication interface to conserve battery energy. As a result of this, Sleepy Nodes cannot be reached most of the time. This fact prevents using normal communication patterns as specified by the CoRE group, since the server model is clearly not applicable to the most energy constrained devices.

This document discusses and specifies an architecture to support Sleepy Nodes such as battery-powered sensors in wireless networks. The proposed solution makes use of a Proxy Node to which a Sleepy Node delegates part of its communication tasks while it is not accessible in the wireless network. Direct interactions between Sleepy Nodes and non-Sleepy Nodes are only possible, when the Sleepy Node initiates the communication.

Earlier related documents treating the Sleepy Node subject are the CoRE mirror server [I-D.vial-core-mirror-server] and the Publish-Subscribe in the Constrained Application Protocol (CoAP) [I-D.koster-core-coap-pubsub]. Both documents describe the interfaces to the proxy accompanying the Sleepy Node. Both make use of the observe option discussed in [I-D.ietf-core-observe]. This document describes the roles of the nodes communicating with the Sleepy Node and/or its proxy. The draft describes the differences between the concepts supporting the Sleepy Node, and the concepts underlying the PubSub paradigm.

The draft relies heavily on the concepts introduced by the Resource Directory [I-D.ietf-core-resource-directory], and describes how the Sleepy Node profits of the introduction of a Resource Directory into the network.

The issues that need to be addressed to provide support for Sleepy Nodes in Control networks are summarized in Section 1.1. Section 2 provides a set of use case descriptions that introduce communication patterns to be used in home and building control scenarios. Section 4, Section 5,Section 6, and Section 7 specify interfaces to support each of these scenarios. Many interface specifications and examples are taken over from [I-D.vial-core-mirror-server].

1.1. Problem statement

During typical operation, a Sleepy Node has its radio disabled and the CPU may be in a sleeping state. If an external event occurs (e.g. person walks into the room activating a presence sensor), the

CPU and radio are powered back on and they send out a message to another node, or to a group of nodes. After sending this message, the radio and CPU are powered off again, and the Sleepy Node sleeps until the next external event or until a predefined time period has passed. The main problems when introducing Sleepy Nodes into a wireless network are as follows:

Problem 1: How to contact a Sleepy Node that has its radio turned off most of the time for:

- Writing configuration settings.

- Reading out sensor data, settings or log data.

- Configuring additional event destination nodes or node groups.

Problem 2: How to discover a Sleepy Node and its services, while the node is asleep:

- Direct node discovery (CoAP GET /.well-known/core as defined in [RFC7252]) does not find the node with high probability.

- Mechanisms may be needed to provide, as the result of node discovery, the IP address of a Proxy instead of the IP address of the node directly.

Problem 3: How a Sleepy Node can convey data to a node or groups of nodes, with good reliability and minimal energy consumption.

1.2. Assumptions

The solution architecture specified here assumes that a Sleepy Node has enough energy to perform bidirectional communication during its normal operational state. This solution may be applicable also to extreme low-power devices such as solar powered sensors as long as they have enough energy to perform commissioning and the initial registration steps. These installation operations may require, in some cases, an additional source of power. Since a Sleepy Node is unreachable for relatively long periods of times, the data exchanges in the interaction model are always initiated by a Sleepy Node when its sleep period ends.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document assumes readers are familiar with the terms and concepts discussed in [RFC7252],[RFC5988], [I-D.ietf-core-resource-directory], [I-D.ietf-core-interfaces],[I-D.ietf-core-observe] and [I-D.vial-core-mirror-server].

In addition, this document makes use of the following additional terminology:

Sleepy Node: a battery-powered node which does the on/off switching of its communication interface with the purpose of conserving battery energy

Sleeping/Asleep: A Sleepy Node being in a "sleeping state" i.e. its network interface is switched off and a Sleepy Node is not able to send or receive messages.

Awake/Not Sleeping: A Sleepy Node being in an "awake state" i.e. its network interface is switched on and the Sleepy Node is able to send or receive messages.

Wake up reporting duration: the duration between a wake up from a Sleepy Node and the next wake up and report of the same Node.

Proxy: any node that is configured to, or selected to, perform communication tasks on behalf of one or more Sleepy Nodes.

Regular Node: any node in the network which is not a Proxy or a Sleepy Node.

1.4. Acronyms

This Internet-Draft contains the following acronyms:

DTLS: Datagram Transport Layer Security

EP: Endpoint

MC: Multicast

RD: Resource Directory

2. Use cases and architecture

To describe the application viewpoint of the solution, we introduce some example scenarios for the various interactions shown in Figure 1. The figure assigns the following roles taken up by a regular node:

- Reading Node: any regular node that reads information from the Sleepy Node.

- Configuring Node: any regular node that writes information/ configuration into Sleepy Node(s). Examples of configuration are new thresholds for a sensor or a new value for the wake-up cycle time.

- Discovering Node: any regular node that performs discovery of the nodes in a network, including Sleepy Nodes.

- Destination Node: any regular node or node in a group that receives a message that is generated by the Sleepy Node.

- Server Node: an optional server that the Sleepy Node knows about, or is told about, which is used to fetch information/configuration/firmware updates/etc.

- Discovery Server: an optional server that enables nodes to discover all the devices in the network, including Sleepy Nodes, and query their capabilities. For example, a Resource Directory server as defined in [I-D.ietf-core-resource-directory] or a DNS-SD server as defined in [RFC6763]. For the rest of this document the discovery server is a Resource Directory. Specifically, the functionalities of the Resource Directory related to the architecture presented in this Internet-Draft are described in more details in Section 4.

- Delegated resource is the copy at the Proxy of a resource present in the Sleepy Node.

2.1. Node interactions and use cases

```
                +------------+             +-------------+
                | Discovery  | <-DISCOVERY-| Discovering |
                | server     |             |    Node     |
                | (Optional) |             +-------------+
                +------------+                      |
                                                    |
                                      .--DISCOVERY--'  +---------+
                                      |                | Reading |
                                      |            .---|   Node  |
                                      v            |   +---------+
   +---------+               +-----------+         |
   | Sleepy  |---REPORT(A)-->|           |<--READ--' +-------------+
   |  Node   |---READ------->|   Proxy   |<--WRITE---| Configuring |
   |         |---WRITE------>|           |           |    Node     |
   +---------+               +-----------+           +-------------+
     |     |                          |              +-------------+
     |     |                          '---REPORT(B)->| Destination |
     |     '-----DIRECT REPORT---------------------->|     Node    |
     |                                               +-------------+
     |                                            +-----------+
     '------------READ--------------------------->|  Server   |
                                                  |   Node    |
                                                  +-----------+
```

Figure 1: Interaction model for Sleepy Nodes in IP-based networks

The interactions visualized in Figure 1 are discussed and motivated with their use cases. The arrows in the figure indicate that the initiative for an interaction is taken by the source of the arrow.

DISCOVERY Interaction: a Discovering Node discovers Sleepy Node(s)via Proxy or Discovery Server; for example:

- A Discovering Node wants to discover given services related to a group of deployed sensors by sending a multicast to /.well-known/core. It gets responses for the sleeping sensors from the Proxy nodes.

- During commissioning phase, a discovering node queries a Discovery Server to find all the proxies providing a given service.

REPORT Interaction: On request of a Destination Node or because of configuration settings which have instructed the Node to do so, a Node sends a sequence of event notifications to destination Node(s), (A) directly or (B) via Proxy; for example:

- A battery-powered sensor sends a notification with "battery low" event directly to a designated Destination Node (REPORT(A)).

- A battery-powered occupancy sensor detects an event "people present", switches on the radio and multicasts an "ON" command to a group of lights (REPORT(A)).

- A battery-powered temperature sensor reports periodically the room temperature to a proxy Node (REPORT(A)). The proxy node reports to all associated HVAC destination nodes when the temperature change deviates from a predefined range (REPORT(B)).

WRITE Interaction: A node sends a request to a proxy to set a value.

o A Sleepy Node WRITES to the proxy; for example:

  - A battery-powered sensor wants to extend the registration lifetime of its delegated resource at the Proxy.

o A configuring Node WRITEs information to a Proxy; for example:

  - A configuring Node changes the reporting frequency of a deployed sensor by contacting the Proxy node to which the sensor is registered.

  - Sensor firmware is upgraded. A configuring Node pushes firmware data blocks to the Proxy, which pushes the blocks to the Sleepy Node.

  - A configuring Node adds a new subscription to an operational sensor via the Proxy. From that moment on, the new Node receives also the sensor events and status updates from the sensor.

READ Interaction: A node sends a read request to a node that returns a value.

o Sleepy Node sends a read request to a server Node; for example:

  - A sensor (periodically) updates internal data tables by fetching it from a predetermined remote node.

  - A sensor (periodically) checks for new firmware with a remote node. If new firmware is found, the sensor switches to a non-sleepy operation mode, and fetches the data.

o A Sleepy Node sends a read request to its proxy; for example:

- A sensor (periodically) checks with his Proxy availability of configuration updates or changes of its delegated resources (e.g. a sensor may detect in this way that a configuring Node has changed its name or modified its reporting frequency).

o A reading Node sends a read request to a proxy; for example:

- A Node (e.g. in the backend) requests the status of a deployed sensor, e.g. asking the sensor state and/or firmware version and/or battery status and/or its error log. The Proxy returns this information.

- A Node requests a Proxy when a Sleepy sensor was 'last active' (i.e. identified as being awake) in the network.

2.2. Architecture

The architecture associated with the support of Sleepy Nodes is illustrated in Figure 2. Three High level interfaces are shown.

```
       direct        synchronize      delegate
          |                |                |
+----+    |   +---------+  |   +--------+   |   +----+
| EP |---|---| sleepy |---|---| proxy |---|---| EP |
+----+    |   +---------+  |   +--------+   |   +----+
          |                |                |
```

Figure 2: Architecture of Sleepy Node support

o Direct interface: it allows the Sleepy Node to communicate directly to endpoints (i.e. for sending or reading information). The operations performed via this interface are always initiated by the Sleepy Node when its sleep period ends.

o Delegate interface: via this interface the Proxy exposes the values of delegated resources to interested endpoints on behalf of the Sleepy Node. The same interface is used by endpoints which want to communicate with the Sleepy Node (e.g. for reading or writing information).

o Synchronize interface: used by Sleepy Node and Proxy to synchronize values of delegated resources. Through this interface operations as discovery of the Proxy, registration, initialization and update of resources at the Proxy are performed, along with a de-registration operation to explicitly remove resources already registered to the Proxy.

The interfaces consist of a set of functions which together realize the interactions described in Section 2.1.

Endpoints and the proxy communicate with a Resource Directory (RD) to discover resources of the Sleepy Node and delegated resources on the proxy (not shown in the Figure 2).

2.3. Example contents

The examples presented in this specification make use of a smart temperature sensor the resources of which are defined below using Link Format [RFC6690]. Three resources are dedicated to the Device Description (manufacturer, model, name) and one contains the current temperature in degree Celsius.

```
</dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s"
```

3. Design motivation

The Sleepy Node stack features a CoAP interface, to make the Sleepy Node part of the IP-based network. Adding CoAP with a transport protocol increases the possibilities to configure the Sleepy Node within the network. The increased energy consumption coming from the overhead of the CoAP and IP headers can be acceptable in many cases.

The proxy and Sleepy Node make use of the /.well-known/core resource to handle discovery during network initialization. Using the Resource Directory during operation of the Sleepy Node reduces its participation in the discovery traffic.

A Sleepy Node delegates its resources to a proxy. The proxy functionality extends the functionality of the RD, because the proxy handles the value of the resource, and the RD does not. A proxy may support multiple Sleepy Nodes. A Sleepy Node may also delegate its resources to multiple proxies. A node can select a proxy that handles the resource of the Sleepy Node of choice.

The complexity of the discovery and delegation interfaces is minimized by reusing the RD interface as much as possible.

4. Interactions involving Resource Directory

It is assumed that the Proxy has a resource type rt="core.sp", where sp stands for sleepy proxy.

In order to become fully operational in a network and to communicate over the functional interfaces shown in Figure 2, a Sleepy Node and the Proxy need to perform operations via the Registration interface of the RD:

- Discovery of Proxy via RD. The Sleepy Node MAY discover the Proxy by sending a request to the RD to return all EP with rt=core.sp.

- Register existence of Proxy. When a RD is present and a Sleepy Node has registered itself to a Proxy (see Section 5.2), the Proxy MUST register the Sleepy Node at the RD and MUST keep this registration up-to-date.

- Register delegated resources. When a RD is present, the Proxy MUST register the delegated resources at the RD and keep them up-to date.

A Configuring Endpoint (often part of a so-called Commissioning Tool) registers the services that are reported directly by the Sleepy Node in the resource directory, by registering the resource type and the multicast address. The multicast address can be associated with a group as described in [I-D.ietf-core-resource-directory].

A discovering Endpoint can discover one or more Sleepy Node resources via the Resource Directory.

```
   +-------------+                          +-----------------+
   | Configuring |                          | Discovering     |---.
   | Endpoint    |                          | Endpoint        |   |
   +-------------+                          +-----------------+   |
         |                                                        |
         |                   +------------+                       |
         .-Register MC------>|            |<--Discover resources -.
                             | Resource   |
                             | Directory  |<--Register Proxy -----.
         .-Proxy Discovery-->|            |<--Register resources -.
         |                   +------------+                       |
         |                                                        |
   +---------+                              +-----------+         |
   | Sleepy  |                              |   Proxy   |---------'
   | Node    |                              |           |
   +---------+                              +-----------+
```

Figure 3: Interactions involving Resource Directory

5. Synchronize interface

The functions of the synchronize interface implemented by the Proxy are described in this section.

5.1. Sleepy Node discovers proxy

A Sleepy Node can discover the proxy in two ways:

- via the CoAP interface [RFC7390] by sending a multicast message to discover an endpoint with rt=core.sp.

- via RD as already described in Section 4.

The following example shows a sleeping endpoint discovering a proxy using this interface, thus learning that the base Proxy resource, where the Sleepy Node resources are registered, is at /sp.

```
Sleepy                                                Proxy
  |                                                     |
  | ----- GET /.well-known/core?rt=core.sp ------>      |
  |                                                     |
  |                                                     |
  | <---- 2.05 Content "</sp>; rt="core.sp" ------      |
  |                                                     |
```

```
Req: GET coap://[ff02::1]/.well-known/core?rt=core.sp
Res: 2.05 Content
</sp>;rt="core.sp"
```

The use of /sp is recommended and not compulsory.

5.2. Registration at a Proxy

Once a Sleepy Node has discovered a Proxy by means of one of the procedures described in Section 5.1, the registration step can be performed. To perform registration, a Sleepy Node sends to the Proxy Node a CoAP POST request containing a description of the resources to be delegated to the Proxy as the message payload in the CoRE Link Format [RFC6690]. The description of the resource includes the Sleepy Node identifier, its domain and the lifetime of the registration.

Upon successful registration a Proxy creates a new delegated resource or updates an existing delegated resource and returns its location. The resources specified by the Sleepy Node during registration are created with path that has as prefix the base Proxy resource path (e.g. /sp). The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameter does not create multiple delegated resources. The delegated resource SHOULD implement the Interface Type CoRE Link List defined in [I-D.ietf-core-interfaces]. A GET request on this resource MUST return the list of delegated resources for the corresponding Sleepy Node.

After successful registration, a Proxy SHOULD enable resource discovery for the new resources by updating its "/.well-known/core" resource. A Proxy MUST wait for the initial representation of a resource before it can be visible during resource discovery. The top level delegated resource MUST be published in "/.well-known/core" to enable the discovery of the resources via RD as described in Section 4. Resources of a delegated container SHOULD be discoverable either directly in "/.well-known/core" or indirectly through gradual reveal from the delegated resource. The Web Link of a delegated resource MUST contain an "ep" attribute with the value of the End-Point parameter received during registration.

A Proxy MAY be configured to register the Sleepy Node's resources in a RD. In this case, a Sleepy Node MUST NOT register the resources in a RD by itself since it is the responsibility of the Proxy to perform the registration in the RD on behalf of the Sleepy Node. Since each Sleepy Node may register resources with different lifetimes, a Proxy MUST register the resources of a given Sleepy Node in a dedicated path of the RD.

In case a Sleepy Node delegates its own resources to more than one Proxy and each Proxy registers the Sleepy Node's resource in a RD, the RD entries from the different Proxies for the same Sleepy Node risk to overlap.

To avoid this problem, a Proxy MUST create its own resource path to register the resources of a Sleepy Node on the RD.

The new path name is typically formed by concatenating the Proxy's endpoint identifier with the path in use. This precaution ensures that the ep identifier of a Sleepy Node is unique for each resource path in the RD.

Implementation note: It is not recommended to reuse the value of the ep parameter in the URI of the delegated resource. This parameter may be a relatively long identifier to guarantee global uniqueness (e.g. EUI64) and would generate inefficient URIs on the Proxy where only a local handler is necessary.

The following example shows a Sleepy Node registering with a Proxy.

```
Sleepy                                                      Proxy
  |                                                           |
  | --- POST /sp?ep=0224e8fffe925dcf;rt=sensor "</dev..."---> |
  |                                                           |
  |                                                           |
  | <-- 2.01 Created Location: /sp/0 ----------------------   |
  |                                                           |
```

```
Req: POST coap://sp.example.org/sp?ep=0224e8fffe925dcf;rt=sensor
Etag: 0x3f
Payload:
</dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s"

Res: 2.01 Created
Location: /sp/0
```

The delegated resource has been created with path /sp/0 on the Proxy in the example above. The path to the ep can be discovered as shown below:

```
Req: GET coap://sp.example.org/.well-known/core
Res: 2.05 Content
</sp>;rt="core.sp",
</sp/0>;ep="0224e8fffe925dcf";rt="sensor"
```

A node can discover the delegated resources of the ep as shown below:

```
Req: GET coap://sp.example.org/sp/0
Res: 2.05 Content
Payload:
</sp/0/dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</sp/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</sp/0/dev/n>;rt="ipso.dev.n";if="core.p",
</sp/0/sen/temp>;rt="ucum.Cel";if="core.s"
```

Once the resources are registered in the Proxy, the Proxy registers the delegated resources in the RD.

```
Proxy                                                       RD
  |                                                          |
  | --- POST /rd?ep=0224e8fffe925dcf "</sp/0..." -------->   |
  |                                                          |
  |                                                          |
  | <-- 2.01 Created Location: /rd/6534 -------------------  |
  |                                                          |

Req: POST coap://rd.example.org/rd?ep=0224e8fffe925dcf
Etag: 0x6a
Payload:
</sp/0/dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</sp/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</sp/0/dev/n>;rt="ipso.dev.n";if="core.p",
</sp/0/sen/temp>;rt="ucum.Cel";if="core.s"

Res: 2.01 Created
Location: /rd/6534
```

5.3. De-registration at a Proxy

Sleepy Node resources in the Proxy are kept active for the period indicated by the lifetime parameter. The Sleepy Node is responsible for refreshing the delegated resource within this period using either the registration or update function (see Section 5.5 of the Synchronize interface). Once a delegated resource has expired, the Proxy deletes all resources associated to that resource and updates its "/.well-known/core" resource. When the Proxy resources are also registered in a RD, the RD and delegated resources are supposed to have the same lifetime. Consequently, when the delegated resource expires, a Proxy MAY let the RD resource expire too instead of explicitly deleting it. When the delegated resource is deleted by means of explicit de-registration operation then also the RD resource MUST be explicitly removed.

A Proxy could lose or delete the delegated resource associated to a Sleepy Node without sending an explicit notification (e.g. after reboot). A Sleepy Node SHOULD be able to detect this situation by processing the response code while using the Sleepy Node Operation or Update interface. Especially an error code "4.04 Not Found" SHOULD cause the Sleepy Node to register again. A Sleepy Node MAY also register with multiple proxies to alleviate the risk of interruption of service.

5.4. Initialization of delegated resource

Once registration has been successfully performed, the Sleepy Node must initialize the delegated resource. To send the initial contents (e.g. values, device name, manufacturer name) of the delegated resources to the Proxy, the Sleepy Node uses CoAP PUT repeatedly.

The basic interface is specified as follows:

Interaction: Sleepy -> Proxy

Method: PUT

URI Template: /{+location}{+resource}{?lt}

URI Template Variables:

location := This is the Location path returned by the Proxy as a result of a successful registration.

resource := This is the relative path to a delegated resource managed by the registered Sleepy Node.

lt := Lifetime (optional). The number of seconds by which the lifetime of the whole delegated resource is extended. Range of 1-4294967295. If no lifetime is included, the current remaining lifetime stays unchanged.

Request Content-Type: Defined at registration

Response Content-Type: Defined at registration for GET method. application/link-format for PUT method if at least one of the mutable resources has been updated since the last PUT request.

Etag: The Etag option MAY be included to allow clients to validate a resource on multiple Proxies.

Success: 2.01 "Created", the request MUST include the initial representation of the delegated resource.

Success: 2.04 "Changed", the request MUST include the new representation of the delegated resource.

Success: 2.05 "Content", the response MUST include the current representation of the delegated resource.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example describes how a Sleepy Node can initialize the resource containing its manufacturer name just after registration.

```
Sleepy                                              Proxy
 |                                                    |
 | --- PUT /sp/0/dev/mfg "acme" ---------------->     |
 |                                                    |
 |                                                    |
 | <-- 2.01 Created -----------------------------     |
 |                                                    |
```

Req: PUT /sp/0/dev/mfg
Payload: acme
Res: 2.01 Created

The example below shows how a Sleepy Node can indicate that it is supposed to send a temperature value at least every hour to keep its delegated resource active.

```
Sleepy                                              Proxy
 |                                                    |
 | --- PUT /sp/0/sen/temp?lt=3600 "22" -------->      |
 |                                                    |
 |                                                    |
 | <-- 2.04 Changed -----------------------------     |
 |                                                    |
```

Req: PUT /sp/0/sen/temp?lt=3600
Payload: 22
Res: 2.04 Changed

The use of repeated CoAP PUT can be avoided by writing all relevant resources into the Proxy in one operation by means of the Batch interface described in [I-D.ietf-core-interfaces]. After successful initialization, a Proxy SHOULD enable resource discovery for the new delegated resources by updating its /.well-known/core resource.

5.5. Sleepy Node updates delegated resource at Proxy

A Sleepy Node can update a delegated resource at the Proxy (REPORT A) using standard CoAP PUT requests on the delegated resource as shown in Section 5.4.

When a Sleepy Node sends a PUT request to update its resources, the response MAY contain a link-format payload. The payload does not directly relate to the target resource of the PUT request. Instead, it is a list of web links to resources that have been modified by clients since either the last PUT request or the last call to the modification check interface (see Section 5.6).

5.6. Sleepy Node READs resource updates from Proxy

This function allows a Sleepy Node to retrieve a list of delegated resources that have been modified at the Proxy by other nodes. The interface format for GET is the same as the one specified for PUT in Section 5.4.

A configuring Node (EP) can update a resource in the Proxy. The Sleepy Node receives an indication of the changed resources as specified in Section 5.5.

The Sleepy Node can send GET requests to its Proxy on each delegated resource in order to receive their updated representation. The example in Figure 4 shows a configuration node which changes the name of a Sleepy Node at the Proxy. The Sleepy Node can then check and read the modification in its resource.

```
     Sleepy                   Proxy                    EP
       |                        | <---PUT /sp/0/dev/n----|
       |                        | Payload: Sensor1       |
    Wake-up                     |---2.04 Changed-------->|
     event                      |                        |
       |                        |                        |
       |--POST /sp/0/dev/.. --> |                        |
       |<----2.04 Changed------|                         |
       | Payload: <sp/0/dev/n> |                         |
       |                        |                        |
       |---GET /sp/0/dev/n---->|                         |
       |<-----2.05 Content-----|                         |
       |  Payload: Sensor1      |                        |
       |                        |                        |
```

Figure 4: Example: A Sleepy Node READs resource updates from his Proxy

6. Delegate Interface

This section details the functions belonging to the delegate interface.

6.1. Discovering Endpoint discovers Sleepy Node at Proxy

Through this function, a Discovering Endpoint can discover one or more Sleepy Node(s) at a Proxy. In case a Resource Directory is not present, this is the only way to discover Sleepy Nodes. A CoAP client discovers resources owned by the Sleepy Node but hosted on the Proxy using typical mechanisms such as one or more GETs on the resource /.well-known/core [RFC6690].

Resource discovery between an Endpoint and a proxy or an Endpoint and a RD needs special care to take into account the fact that resources from a Sleepy Node might appear duplicated. EPs SHOULD employ 2-step resource discovery by looking up Sleepy Nodes AND resource types to detect duplicate resources. EPs MAY use single-step resource discovery only if the Sleepy Node can register with no more than one Proxy. An EP can use the "ep" link attribute as a filter on the "/.well-known/core" resource to retrieve a list of endpoints and detect duplicate Sleepy Nodes registered on multiple proxies. An EP can use the "ep" type of lookup to do the same on a RD. The result of endpoint discovery is then used to filter out duplicate resources returned from simple resource discovery.

The following example shows a client discovering the Sleepy Nodes and learning that the Sleepy Node 0224e8fffe925dcf is registered on two Proxies.

```
EP                                        proxy1   proxy2
 |                                           |        |
 | ----- GET /.well-known/core?ep=* ------->|------>|
 |                                           |        |
 |                                           |        |
 | <---- 2.05 Content "</sp/0>..."  --------|        |
 |                                           |        |
 | <---- 2.05 Content "</sp/0>..."  --------|-------|
```

```
Req: GET coap://[ff02::1]/.well-known/core?ep=*
Res: 2.05 Content
</sp/0>;ep="0224e8fffe925dcf"
Res: 2.05 Content
</sp/0>;ep="02004cfffe4f4f50"
</sp/1>;ep="0224e8fffe925dcf"
```

From the previous exchange and the next resource discovery request, the EP can infer that the resources coap://sp1/sp/0/sen/temp and coap://sp2/sp/1/sen/temp actually come from the same Sleepy Node with ep=0224e8fffe925dcf.

```
EP                                          proxy1  proxy2
 |                                             |       |
 | - GET /.well-known/core?rt=ipso:ucum.Cel ->|------>|
 |                                             |       |
 |                                             |       |
 | <---- 2.05 Content "</sp/0>..."  ----------|       |
 |                                             |       |
 | <---- 2.05 Content "</sp/1>..."  ----------|-------|

Req: GET coap://[ff02::1]/.well-known/core?rt=ucum.Cel
                                        &ep=0224e8fffe925dcf
Res: 2.05 Content
</sp/0/sen/temp;rt="ucum.Cel"
Res: 2.05 Content
</sp/1/sen/temp>;rt="ucum.Cel"
```

6.2. Proxy REPORTs events to Endpoint

This interface can be used by the Endpoint to receive event report message to Proxy (REPORT A) which further notifies it to interested Destination Endpoint(s)(REPORT B). This indirect reporting is useful for a scalable solution, e.g. there may be many interested subscribers but the Sleepy Node itself can only support a limited number of subscribers given its limits on battery energy. A client interested in the events related with a specific resource may send a CoAP GET to the Proxy, to obtain the last published state. If a Reading node is interested in receiving updates whenever the Sleepy Node reports new event to its Proxy, it can use observe [I-D.ietf-core-observe] at the Proxy for that specific resource.

A proxy using the CoAP protocol [RFC7252] SHOULD accept to establish a CoAP observation relationship between the delegated resource and a client as defined in [I-D.ietf-core-observe].

A Sleepy Node may stop updating its delegated resources without explicitly removing its delegated resource (e.g. transition to another proxy after network unreachability detection). An Endpoint can detect this situation when the corresponding delegated resource has expired. Upon receipt of a response with error code 4.04 "Not Found", an Endpoint SHOULD restart resource discovery to determine if the resources are now delegated to another proxy.

The interface function is specified as follows:

Interaction: EP -> Proxy

Method: Defined at registration

URI Template: /{+location}{+resource}

URI Template Variables:

location := This is the Location path returned by the Proxy as a result of a successful registration.

resource := This is the relative path to a delegated resource managed by a Sleepy Node.

Content-Type: Defined at registration

In the example below an EP observes the changes of temperature through the Proxy.

```
Sleepy                        Proxy                       EP
 |                              |                          |
 |                              | <- GET /sp/0/sen/temp -  |
 |                              |          (observe)       |
 |                              |                          |
 |                              | -- 2.05 Content "22" ->  |
 |                              |                          |
 | - PUT /sp/0/sen/temp "23" -> |                          |
 |                              |                          |
 | <- 2.04 Changed ------------ |                          |
 |                              |                          |
 |                              | -- 2.05 Content "23" ->  |
```

6.3. A Node WRITEs to Sleepy Node via Proxy

A Configuring Node uses CoAP PUT to write information (such as configuration data) to the Proxy, where the information is destined for a Sleepy Node. Upon change of a delegated resource, an internal flag is set in the Proxy that the specific resource has changed. Next time the Sleepy Node wakes up, the Sleepy Node checks the Proxy for any modification of its delegated resources and reads those changed resources using CoAP GET requests, as shown in Figure 4. The allowed resources that a Configuring Node can write to, and the CoAP Content-Format of those CoAP resources, is determined in the initial registration phase.

The following example shows a commissioning tool (EP) changing the name of a Sleepy Node through a Proxy. The Sleepy Node detects this change right after updating its current temperature.

```
   Sleepy                     Proxy                       EP
     |                          |                          |
     |                          | <-- PUT /sp/0/dev/n ---  |
     |                          |                          |
     |                          | -- 2.04 Changed ------>  |
     |                          |                          |
     | - PUT /sp/0/sen/temp --->|                          |
     | <- 2.04 Changed ---------|                          |
     | Payload: <sp/0/dev/n> ---|                          |
     |                          |                          |
     | - GET /sp/0/dev/n ------>|                          |
     |                          |                          |
     | <- 2.05 Content ---------|                          |
     |                          |                          |
```

```
   Req: PUT /sp/0/dev/n
   Payload: "sensor-1"
   Res: 2.04 Changed

   Req: PUT /sp/0/sen/temp
   Payload: "24"
   Res: 2.04 Changed, Content-Type: application/link-format
   Payload: "</sp/0/dev/n>"

   Req: GET /sp/0/dev/n
   Res: 2.05 Content
   Payload: "sensor-1"
```

6.4. A Node READs information from Sleepy Node via Proxy

A Reading Node uses standard CoAP GET to read information of a Sleepy Node via a Proxy. However, not all information/resources from the Sleepy Node may be copied to the Proxy. In that case, the Reading Node cannot get direct access to resources that are not delegated to the Proxy. The strategy to follow in that case is to first WRITE to the Sleepy Node (via the Proxy, Section 6.3) a request for reporting this missing information; where the request can be fulfilled by the Sleepy Node the next time the Sleepy Node wakes up.

7. Direct Interface

This section details the functions belonging to the direct interface.

7.1. Sleepy Node REPORTs events directly to Destination Node

When the Sleepy Node needs to report an event to Destination nodes or groups of Destination nodes present in the subscribers list, it becomes Awake and then it can use standard CoAP POST unicast or multicast requests to report the event.

TODO: MC example

7.2. A Sleepy Node READs information from a Server Node

A Sleepy Node while Awake uses standard CoAP GET to read any information from a Server Node. While the Sleepy Node awaits a CoAP response containing the requested information, it remains awake. To increase battery life of Sleepy Nodes, such an operation should not be performed frequently.

8. Realization with PubSub broker

The PubSub broker [I-D.koster-core-coap-pubsub] can be used to implement the REPORT function of the Sleepy Node proxy specified in this document. However, there are some differences to be taken into account:

- The PubSub broker handles topics. In the case of the proxy the topics must be equated to resources.

- Clients publish anonymously updates to a topic. In the case of the proxy, a delegated resource is bound to one given node that is allowed to update it. For the same functionality, the PubSub broker must restrict topic updates to one client only. The client linked to the topic must be visible to the clients which subscribe to the topic.

In addition, some other functionality needs to be added to the PubSub broker to satisfy the interaction model shown in Figure 1:

- the READ function from Sleepy Node to proxy is not covered by the PubSub broker. The PubSub broker needs to piggy-back a "check topic" on the confirmation of a publication by the proxy. The proxy can then perform a Read on the signalled topic.

- The interaction "register resources" from proxy to Resource Directory, shown in Figure 3, is not part of the PubSub broker.

9. IANA Considerations

The new Resource Type (rt=) Link Target Attribute, ’core.sp’ needs to be registered in the "Resource Type (rt=) Link Target Attribute Values" sub registry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

10. Security Considerations

For the communication between Sleepy Node and Proxy it MAY be sufficient to use Layer 2 (MAC) security without the recommended use of DTLS. However, it must be ascertained that the Sleepy Node can communicate only with a given secured Proxy. A Sleepy Node may obtain the Layer 2 network key using the bootstrapping mechanism described in [I-D.kumar-6lo-selective-bootstrap]. DTLS MUST be used over link-layer security for further transport-layer protection of messages between Regular Nodes and Proxies in the network. There are no special adaptations needed of the DTLS handshake to support Sleepy Nodes. During the whole handshake, Sleepy Nodes are required to remain awake to avoid that, in case of small retransmission timers, the other node may think the handshake message was lost and starts retransmitting. In view of this, the only key point, therefore, is that DTLS handshakes are not performed frequently to save on battery power. Based on the DTLS authentication, also an authorization method could be implemented so that only authorized nodes can e.g.

- Act as a Proxy for a Sleepy Node. (The Proxy shall be a trusted device given its important role of storing values of parameters for the delegated resources);

- READ data from Sleepy Nodes;

- WRITE data to Sleepy Nodes (via the Proxy);

- Receive REPORTs from Sleepy Nodes (direct or via Proxy).

11. Acknowledgements

Much of the text and examples in this document are copied from [I-D.vial-core-mirror-server]. Matthieu Vial has generously authorized us to use his text. Rahman Akbar has pointed out the CoAP dependency of earlier versions.

12. Changelog

RFC editor, please delete this section before publication.

From version 2 to version 3:

Introduced interfaces and copied examples and text from mirror server draft.

From version 3 to version 4:

Comparison with PubSub Broker completed.

Mistakes in examples removed.

Less dependence on 6LowPAN networks.

Added Design motivation section.

13. References

13.1. Normative References

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-16 (work in progress), December 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

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

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

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

13.2. Informative References

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., and M. Koster, "CoRE Interfaces", draft-ietf-core-interfaces-03 (work in progress), July 2015.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-04 (work in progress), July 2015.

[I-D.koster-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-02 (work in progress), July 2015.

[I-D.kumar-6lo-selective-bootstrap]
Kumar, S. and P. Stok, "Security Bootstrapping over IEEE 802.15.4 in selective order", draft-kumar-6lo-selective-bootstrap-00 (work in progress), March 2015.

[I-D.vial-core-mirror-server]
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.

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

Authors' Addresses

Teresa Zotti
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Phone: +31 6 21175346
Email: teresa.zotti@philips.com

Peter van der Stok
Consultant

Phone: +31 492474673
Email: consultancy@vanderstok.org

Esko Dijk
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
The Netherlands

Phone: +31 6 55408986
Email: esko.dijk@philips.com