CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

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

March 9, 2015

Sleepy CoAP Nodes
draft-zotti-core-sleepy-nodes-01

Abstract

6LoWPAN 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 6LoWPAN networks with the goal of guiding and stimulating the discussion on Sleepy Nodes support for CoAP in the CoRE WG.

Table of Contents

1. Introduction

6LoWPAN 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 6LoWPAN 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 6LoWPAN 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. As such it contributes to understanding how well the other proposals support the operation of the sleepy nodes in a building control context.

The issues that need to be addressed to provide support for Sleepy Nodes in 6LoWPAN networks are summarized in Section 1.1. Section 2 shows the communications patterns involving Sleepy Nodes in 6LoWPAN networks. Section 3 provides a set of use case descriptions that

illustrate how these communication patterns can be used in home and building control scenarios. For each of these scenarios, the behaviour of the Sleepy Node is explained in Section 5.

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 an event 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 6LoWPAN 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.

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

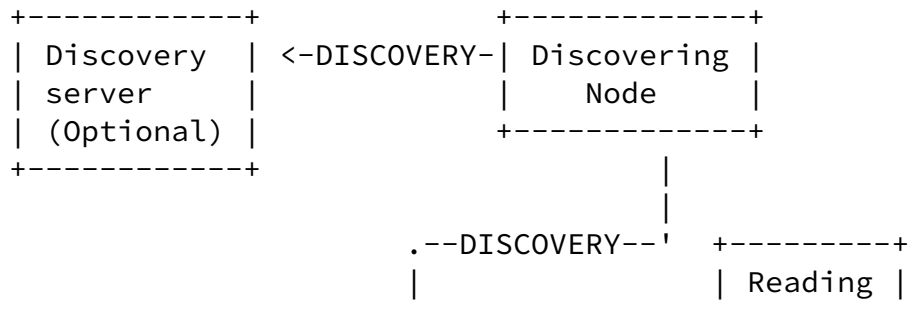
defined in [I-D.ietf-core-resource-directory] or a DNS-SD server as defined in [RFC6763].

2. Solution Architecture

The solution architecture described in this document makes use of a Proxy Node to which a Sleepy Node delegates part of its communication tasks during its sleeping periods. In particular, the solution is based on the set of functionalities described in [I-D.vial-core-mirror-server] according to which a Proxy Node hosts a 'delegated' version of the original CoAP resources of the Sleepy Node. [I-D.vial-core-mirror-server] provides the interface to register, update and remove proxied resources, along with the interface to read and update the proxied resources by both the Sleepy Node and Regular Nodes.

Figure 1 provides an overview of the communication interfaces required to support a Sleepy Node in a 6LoWPAN Network, highlighting the different types and roles of the Nodes (shown as blocks) along with the interactions between them. The interfaces are depicted as arrows. The arrows point from the Node taking the communication initiative to the target Node.

In some implementations, the roles of Proxy and Discovery Server could be implemented by a single node. Furthermore, a single Node could act in a combination of roles (e.g. it may play both the role of discovering node and Configuring Node).



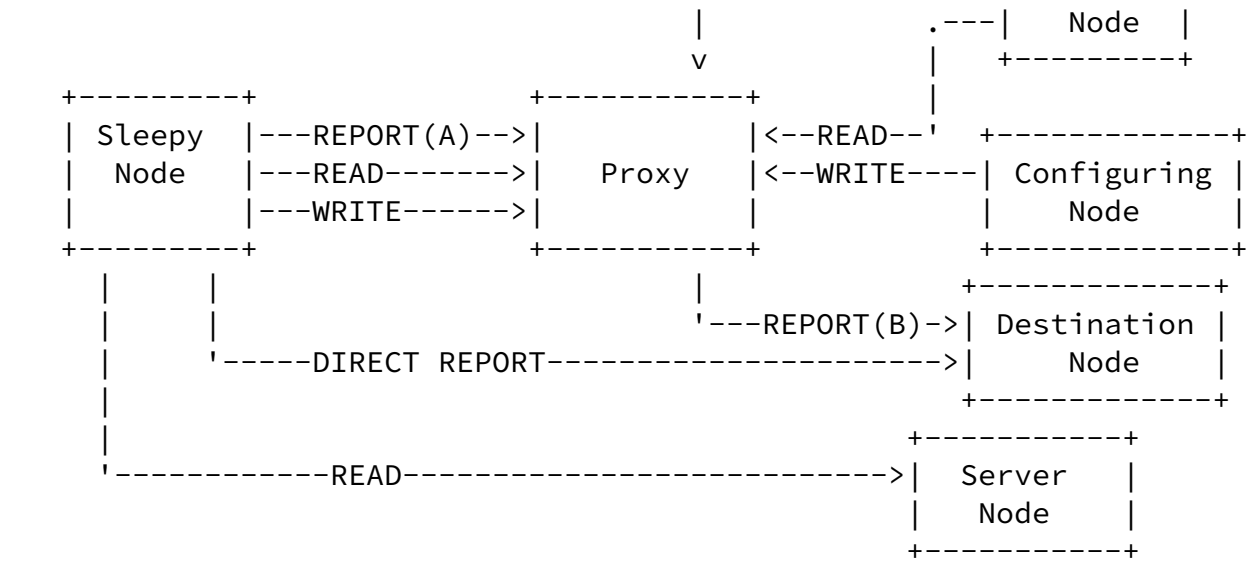


Figure 1: Interaction model for Sleepy Nodes in 6LowPAN networks

3. Use case scenarios

To describe the application viewpoint of the solution, we introduce some example scenarios for the various interface functions in Figure 1, assuming the Sleepy Node to be a sensor device in a home or a building control context.

Function 1: a Node DISCOVERs Sleepy Node(s) (via Proxy or Discovery Server); for example:

- A Node wants to discover given services related to a group of deployed sensors via multicast. It gets responses for the sleeping sensors from the Proxy nodes.

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

Function 2: Sleepy Node REPORTs event to other Node(s) (directly or via Proxy); for example:

- A battery-powered sensor sends an event "battery low" directly

to a designated reporting location Node.

- A battery-powered occupancy sensor detects an event "people present", switches on the radio and sends a request to one or a group of lights to turn on.

- A battery-powered temperature sensor reports periodically the room temperature to a designated Node that controls HVAC devices. The sensor reports also extra events when the temperature change deviates from a predefined range.

Function 3: Sleepy Node WRITEs information to the Proxy; for example:

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

Function 4: Sleepy Node READs from other Node(s) (directly or via Proxy); 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.

- 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).

Function 5: Node READs information from Sleepy Node(s) (via Proxy only); 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.

- An authorized 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.

Function 6: A Node WRITEs information to a Sleepy Node (via Proxy only); for example:

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

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

4. Initial operations

In order to become fully operational in a network and to communicate over the interfaces shown in Figure 1, a Sleepy Node needs first to perform some initial operations:

- Discovery of Proxy (directly or via Discovery Server)

- Registration of resources to delegate at a Proxy

- Initialization of its delegated resources at the Proxy

- Registration to a Discovery Server via Proxy (optional)

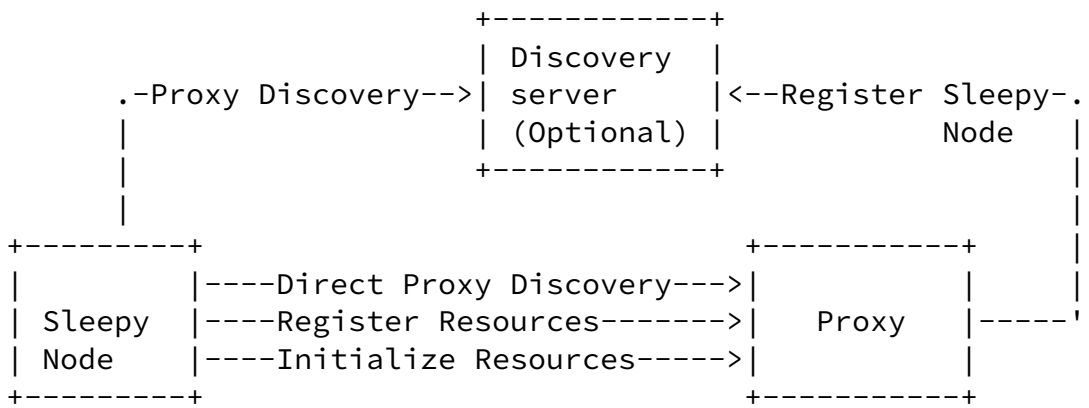


Figure 2: Overview of initial operations

4.1. Proxy Discovery

A Sleepy Node can find a Proxy implementing resource cache functionalities to which it can delegate its own resources by means of:

1. Discovery via Discovery Server: this interface is the default one supplied by the Discovery Server, e.g. CoRE Resource Directory [I-D.ietf-core-resource-directory] or DNS-SD [RFC6763].

2. Direct Discovery: a CoAP multicast GET request can be performed on the /.well-known/core resource as specified for CoAP in [RFC7390].

In both cases, a query can be done for the core.ms resource type, defined in [I-D.vial-core-mirror-server].

In a system, The Proxy discovery can be performed even in both ways (e.g. if Discovery via Discovery Server fails, the Sleepy Node can try Direct Discovery).

4.2. Registration at a Proxy

Once a Sleepy Node has discovered a Proxy by means of one of the procedures described above, 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. The description of the resource includes the Sleepy Node identifier, its domain and the lifetime of the registration. The Link Format description is identical to the /.well-known/core resource. At the moment of the registration at the Proxy, the Sleepy Node may specify the 'obs' attribute to indicate to the Proxy that a CoAP observation relationship between the delegated resource and a client is allowed and can be performed as described in IETF Draft CoRE Observe [I-D.ietf-core-observe]. Upon successful registration, the Proxy creates a new resource and returns its location.

4.3. Initialization of Delegated Resource

Once registration has been successfully performed, the Sleepy Node must initialize the delegated resource before it can be visible in Resource Discovery via the Proxy Node. 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 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.

4.4. Proxy registers at a Discovery Server on behalf of Sleepy Node

Once a Sleepy Node has registered itself to a Proxy, the Proxy has the responsibility to register the Sleepy Node to a Discovery Server and to keep this registration up-to-date. This interface, not to be confused with the interface in which the Sleepy Node registers its resources to a Proxy, is required whenever a Discovery Server is present in the network. There may be in fact deployments that do not have a Discovery Server. At run-time, the Proxy will try to find a Discovery Server and if such server is found it will register the Sleepy Node. The details of the interface are exactly according to the respective Discovery Server specification. A special case might be when Proxy and Discovery Server are embodied by the same node. In this case the registration occurs as an internal process within the Proxy Node itself, upon registration of the Sleepy Node at the Proxy.

5. Interfaces during operation

This section details the scope and behaviour of each interface function specified in the architecture in Figure 1.

5.1. Discovering Node DISCOVERs Sleepy Node via Discovery Server

Through this interface, a Discovering Node can discover one or more Sleepy Node(s) through a Discovery Server. The interface is the default one supplied by the Discovery Server, e.g. CoRE Resource Directory or DNS-SD.

5.2. Discovering Node DISCOVERs Sleepy Node via Proxy

Through this interface, a Discovering Node can discover one or more Sleepy Node(s) through a Proxy. In case a Discovery Server is not active in a system, 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].

5.3. 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.

5.4. Sleepy Node REPORTs event to Destination Node(s) via Proxy

This interface can be used by the Sleepy Node to communicate a sensor event report message to Proxy (REPORT A) which will further notify it to interested Destination Node(s) (REPORT B) that are not directly present in the subscribers list of the Sleepy Node itself. 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. The standard CoAP unicast POST can be used to report events to the Proxy (REPORT A), while the mechanism according to which the Proxy forwards the event to Destination Nodes (REPORT B) may be linked to a specific protocol (for example: CoAP, HTTP, or publish/subscribe as in MQTT). 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 event to its Proxy, it can perform a subscription at the Proxy to that specific resource. In this case, a standard CoAP GET with the CoAP Observe option on the delegated resource at the Proxy can be used, as described in [I-D.ietf-core-observe].

5.5. Sleepy Node WRITEs changed resource to Proxy

A Sleepy Node can update a proxy resource at the Proxy using a

standard CoAP PUT requests on the proxied resource. This interface is only needed when a resource can be changed on the Sleepy Node outside the knowledge of the Proxy, i.e. by an entity which is not the Proxy. For example, a resource can be changed by the Sleepy Node itself. It is good practice, to avoid write/write conflicts at the proxy side, to ensure that such frequently-updated resources are read-only, e.g. the sensed temperature value of a sensor can be read by external nodes but not written.

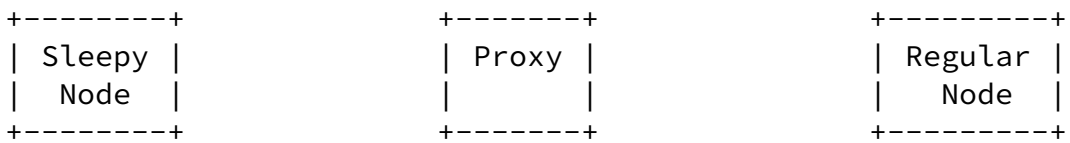
5.6. A Node WRITEs to Sleepy Node via Proxy

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 PS Node checks the Proxy for any modification of its delegated resources and reads those changed resources using CoAP GET requests, as shown in Figure 3. 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.

5.7. Sleepy Node READs resource updates from Proxy

This interface allows a Sleepy Node to retrieve a list of delegated resources that have been modified at the Proxy by other nodes. As in [I-D.vial-core-mirror-server], the path /ms is used to store the sleepy node resources in the proxy.

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 3 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.



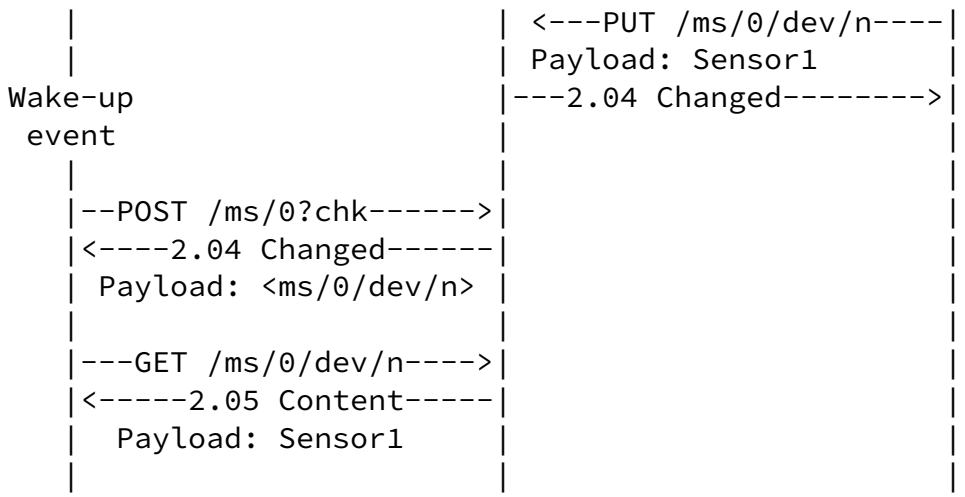


Figure 3: Example: A Sleepy Node READs resource updates from his Proxy

5.8. 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 on 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 5.6) 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.

5.9. 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.

6. Realization with PubSub server

The registration and discovery of the PubSub broker

[I-D.koster-core-coap-pubsub] is covered to the same extent as discussed in this document. Not covered is the direct interaction between sleepy node and destination nodes. The support from a server node to initialize resources or other information also represents an addition to PubSub broker.

In addition to the continuous updates provided by the PubSub broker, the ad-hoc query of values, the maintenance of operational parameters, the provision of direct update from sleepy node to a node, the reliability aspects of the update, and the concept of groups are equally important topics that need consideration.

7. Acknowledgements

TBD

8. IANA Considerations

The new Resource Type (rt=) Link Target Attribute, 'core.ms' needs to be registered in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry. This is not yet done by [I-D.vial-core-mirror-server].

9. Security Considerations

Layer 2 (MAC) security is used in all communication in the 6LoWPAN network. A Sleepy Node may obtain the Layer 2 network key using the bootstrapping mechanism described in [I-D.kumar-6lo-selective-bootstrap]. On top of this, DTLS and DTLS-multicast can be used for further transport-layer protection of messages between a Sleepy Node and other nodes; and also between a Proxy and other nodes. 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).

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[RFC7390] Rahman, A. and E. Dijk, "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, October 2014.

10.2. Informative References

[I-D.ietf-core-interfaces] Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-core-interfaces-02 (work in progress), November 2014.

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

[I-D.ietf-core-resource-directory] Shelby, Z. and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-02 (work in progress), November 2014.

[I-D.koster-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe in the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-00 (work in progress), October 2014.

[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, February 2013.

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
Kamperfoelie 8
Helmond 5708 DM
The Netherlands

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

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

Phone: +31 6 55408986

Email: esko.dijk@philips.com