CORE WG
Internet-Draft
Intended status: Standards Track
Expires: April 11, 2014

A. Rahman
InterDigital Communications, LLC

October 08, 2013

Enhanced Sleepy Node Support for CoAP
draft-rahman-core-sleepy-04

Abstract

CoAP is a specialized web transfer protocol for constrained devices. These devices typically have some combination of limited battery power, small memory footprint and low throughput links. It is expected that in CoAP networks there will be a certain portion of devices that are "sleepy" and which may occasionally go into a sleep mode (i.e. go into a low power state to conserve power) and temporarily suspend CoAP protocol communication. This document proposes a minimal and efficient mechanism building on the Resource Directory (RD) functionality to enhance sleepy node support in CoAP networks. The RD functionality may be incorporated as part of a CoAP Proxy.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on April 11, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminology and Conventions 2
2. Introduction 3
3. Proposal 3
3.1. RD Based Sleep Tracking 4
3.2. Example of Synchronous RD Based Sleep Tracking 4
3.3. Example of Asynchronous RD Based Sleep Tracking 7
3.4. RD Caching Proxy 11
4. Performance Results 11
4.1. Prototype Setup 11
4.2. Initial Results 13
4.3. Comparison with Observe 13
5. Acknowledgements 13
6. IANA Considerations 13
7. Security Considerations 13
8. References 14
8.1. Normative References 14
8.2. Informative References 14
Author's Address 14

1. Terminology and Conventions

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 RFC 2119 [RFC2119].

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap] and [RFC6690]. In addition, this document defines the following terminology:

Sleepy Node
: A sleepy node is a CoAP client or server that may sometimes go into a sleep mode (i.e. go into a low power state to conserve power) and temporarily suspend CoAP protocol communication. A sleepy node will otherwise remain in a fully powered on state where it has the capability to perform full CoAP protocol communication.

Non-Sleepy Node

A non-sleepy node is a CoAP client or server that always remains in a fully powered on state (i.e. always awake) where it has the capability to perform full CoAP protocol communication. The general operation of non-sleepy nodes is assumed to be well known and so are not explicitly spelled out in this document except where needed for clarity.

2. Introduction

The current CoAP approach provides a minimal support of sleepy nodes as follows:

- [I-D.ietf-core-coap] defines CoAP proxies which can cache and service requests for sleepy CoAP servers. A client explicitly sends a CoAP request (GET) to a forward proxy (identified by its IP address) while indicating the URI (of the resource of interest) associated to a sleepy CoAP origin server. If the proxy has a valid representation of the resource in its cache it can then respond directly to the client regardless of the current sleep state of the origin server. Otherwise the proxy has to attempt to retrieve (GET) the resource from the sleepy origin server. The attempt may or may not be successful depending on the sleep state of the origin server.

- [RFC6690] and [I-D.ietf-core-resource-directory] defines a Resource Directory (RD) mechanism where sleepy CoAP servers can register/update (POST/PUT to "/.well-known/core") their list of resources on a central (non-sleepy) RD server. This allows clients to discover the list of resources from the RD (GET /rd-lookup/...) for a sleepy server, regardless of its current sleep state. Unlike a proxy, the RD stores only the URIs for other nodes, and not the actual resource representation [RFC6690] . The client then may attempt to operate on (GET/PUT/POST/DELETE) the desired resource at the sleepy origin server. This attempt may or may not be successful depending on the sleep state of the origin server.

- Lower layer (i.e. below the IP layer) support for sleepy nodes exist in most wireless technologies (e.g. IEEE 802.11 (WiFi), and IEEE 802.15.4 (ZigBee)). For example, most wireless technologies support limited functionality such as packet scheduling to account for sleepy nodes in their physical and MAC layer protocols. This lower layer functionality is not aware of any specific timing or operational aspects of application layer protocols like CoAP.

3. Proposal

3.1. RD Based Sleep Tracking

The current CoAP approach to support sleepy nodes can be significantly improved by introducing RD based mechanisms for a CoAP client to determine whether:

- A targeted resource is located on a sleepy server.
- A sleepy server is currently in sleep mode or not.

We define the following new parameters to characterize a sleepy node:

- SleepState - Indicates whether the node is currently in sleep mode or not (i.e. Sleeping or Awake).
- SleepDuration - Indicates the maximum duration of time that the node stays in sleep mode.
- TimeSleeping - Indicates the length of time the node has been sleeping (i.e. if Sleep State = Sleeping).
- NextSleep - Indicates the next time the node will go to sleep (i.e. if Sleep State = Awake).

These parameters are all server (node) level and are new parameters added to the RD URI Template Variables defined in [I-D.ietf-core-resource-directory].

We also define a new lookup-type ("ss") for the RD lookup interface specified in [I-D.ietf-core-resource-directory]. This new lookup-type supports looking up the SleepState of a specified end-point.

The three time based parameters (SleepDuration, TimeSleeping, NextSleep) can be based on either an absolute network time (for a time synchronized network) or a relative local time (measured at the local node).

Following the approach of [RFC6690] and [I-D.ietf-core-resource-directory], sleep parameters for sleepy servers can be stored by the server in the RD and accessed by all interested clients. Examples of using these parameters in a synchronous or asynchronous manner are shown in the following sections.

3.2. Example of Synchronous RD Based Sleep Tracking

Figure 1 shows an example of using RD based sleep tracking in a synchronous fashion:

(1) SleepyNode-1 is awake and having previously discovered the local RD, stores its CORE link format in the RD (POST/rd) identified by its entry point (?ep=SleepyNode-1). The sleep parameters are also updated as part of this step.

(2)-(3) RD services the POST and stores the CORE link format and starts the sleep timers for this node.

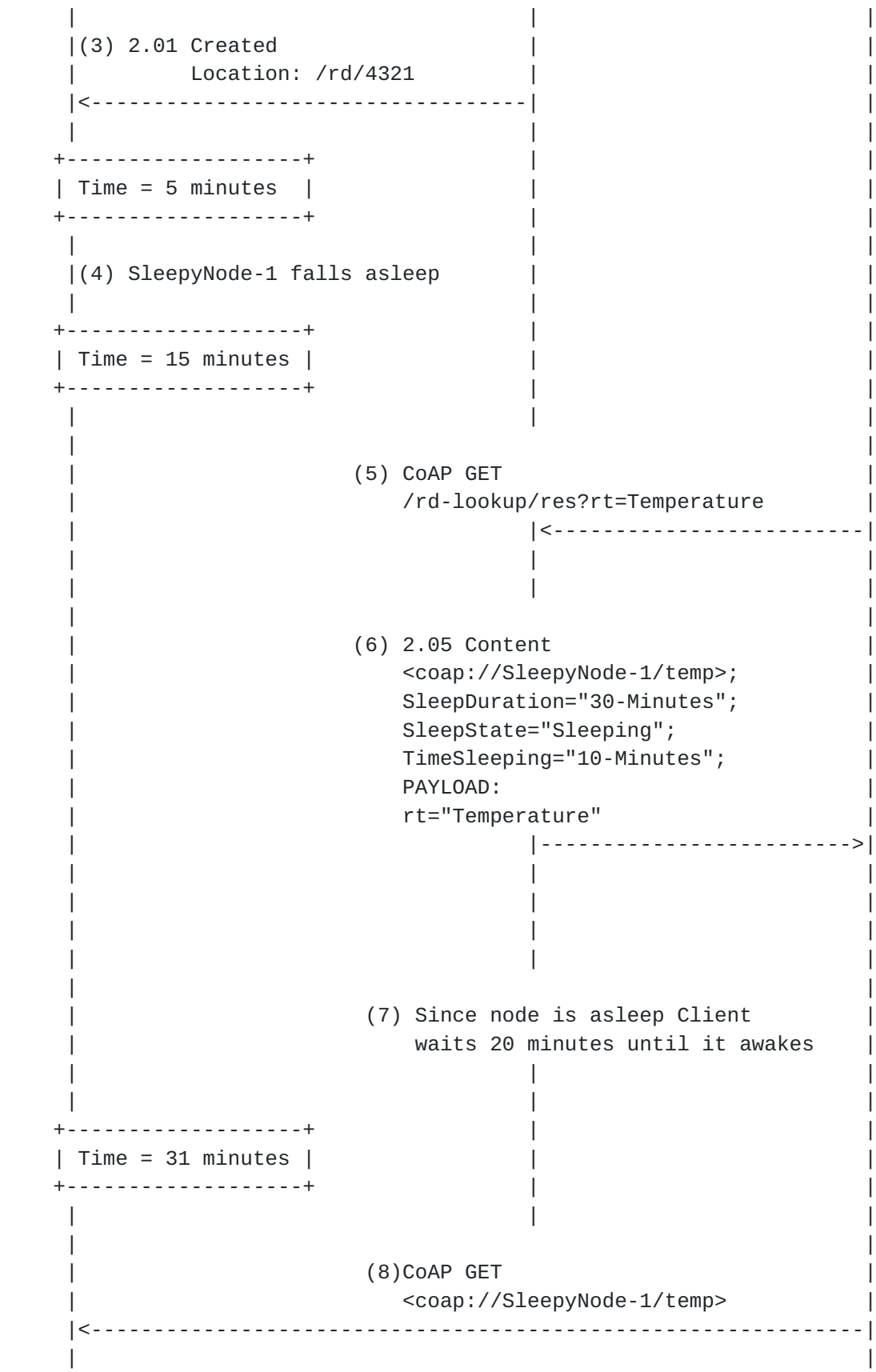
(4) SleepyNode-1 falls asleep.

(5) A client is interested in temperature sensors in this domain and does a lookup on the RD.

(6) The RD does a lookup and finds that SleepyNode-1 is the only node meeting the match and sends back the required info including the Sleep parameters.

(7) From the sleep parameters, the client knows that the node is currently asleep and so internally schedules to send the GET request when the node wakes up (plus a small safety hysteresis).

(8)-(9) Client sends GET request for temperature sensors and successfully receives the content as SleepyNode-1 is awake.


(3) 2.01 Created
Location: /rd/4321
Time = 5 minutes
(4) SleepyNode-1 falls asleep
Time = 15 minutes
(5) CoAP GET
/rd-lookup/res?rt=Temperature
(6) 2.05 Content
<coap://SleepyNode-1/temp>;
SleepDuration="30-Minutes";
SleepState="Sleeping";
TimeSleeping="10-Minutes";
PAYLOAD:
rt="Temperature"
(7) Since node is asleep Client
waits 20 minutes until it awakes
Time = 31 minutes
(8)CoAP GET
<coap://SleepyNode-1/temp>

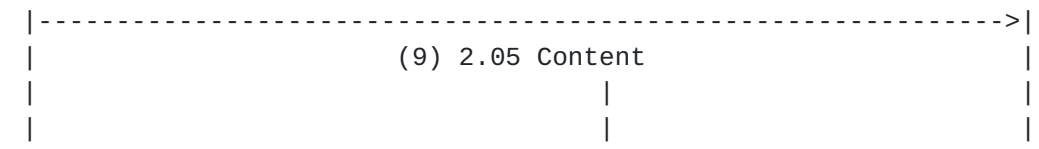


Figure 1: Synchronous Resource Directory Based Sleep Tracking

3.3. Example of Asynchronous RD Based Sleep Tracking

Figure 2 shows an example of using RD based sleep tracking in an asynchronous fashion:

(1) SleepyNode-1 is awake and having previously discovered the local RD, stores its CORE link format in the RD (POST/rd) identified by its entry point (?ep=SleepyNode-1).

(2)-(3) RD services the POST and stores the CORE link format.

(4) A client is interested in temperature sensors in this domain and does a lookup on the RD for all sensors that are currently awake.

(5) The RD does a lookup and finds that SleepyNode-1 is the only node meeting the match and sends back the required info.

(6)-(7) Using the sleep state lookup functionality (lookup-type := "ss"), the client adds itself to the list of observers to get SleepState updates from RD for SleepyNode-1 [I-D.ietf-core-observe].

(8)-(9) Client performs RD 'resource' lookup to find URI of temperature sensor of resource hosted on SleepyNode-1.

(10)-(13) SleepyNode-1 prepares to goes to sleep and updates the SleepState in the RD.

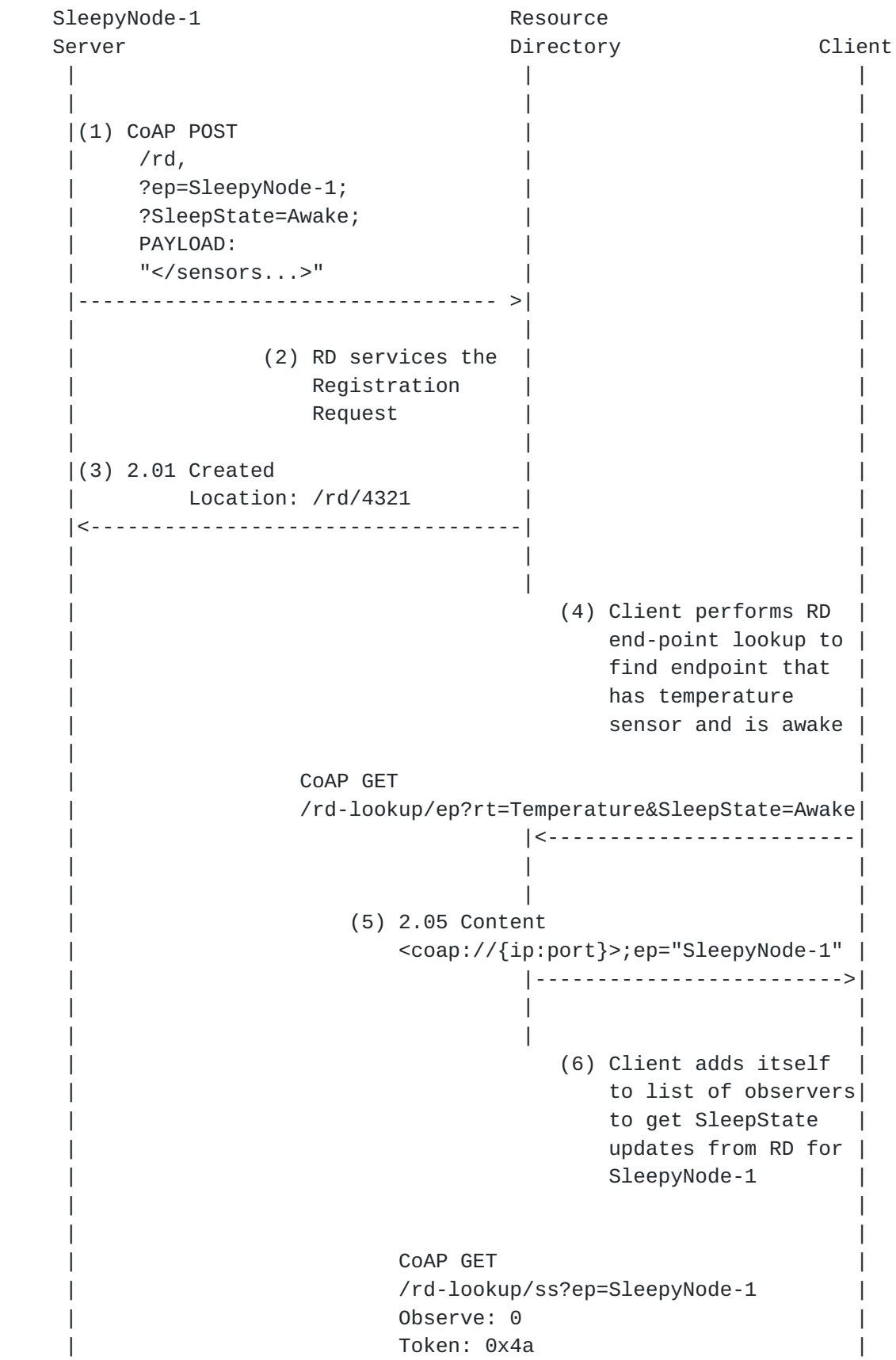
(14) RD notifies the client through previously established observe relationship.

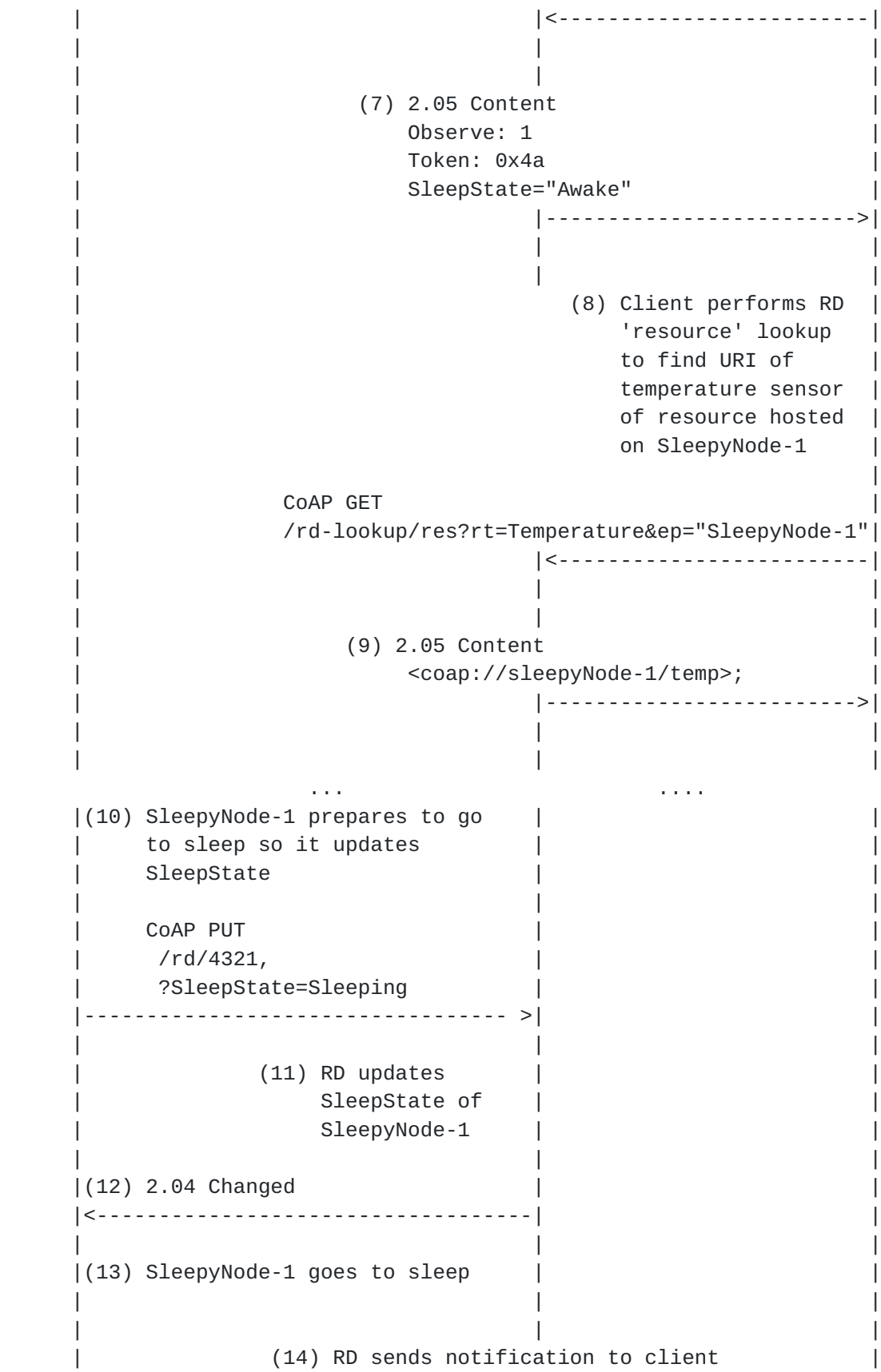
(15) Client application wants to get the temperature now but does not send the request as it knows SleepyNode-1 is currently sleeping.

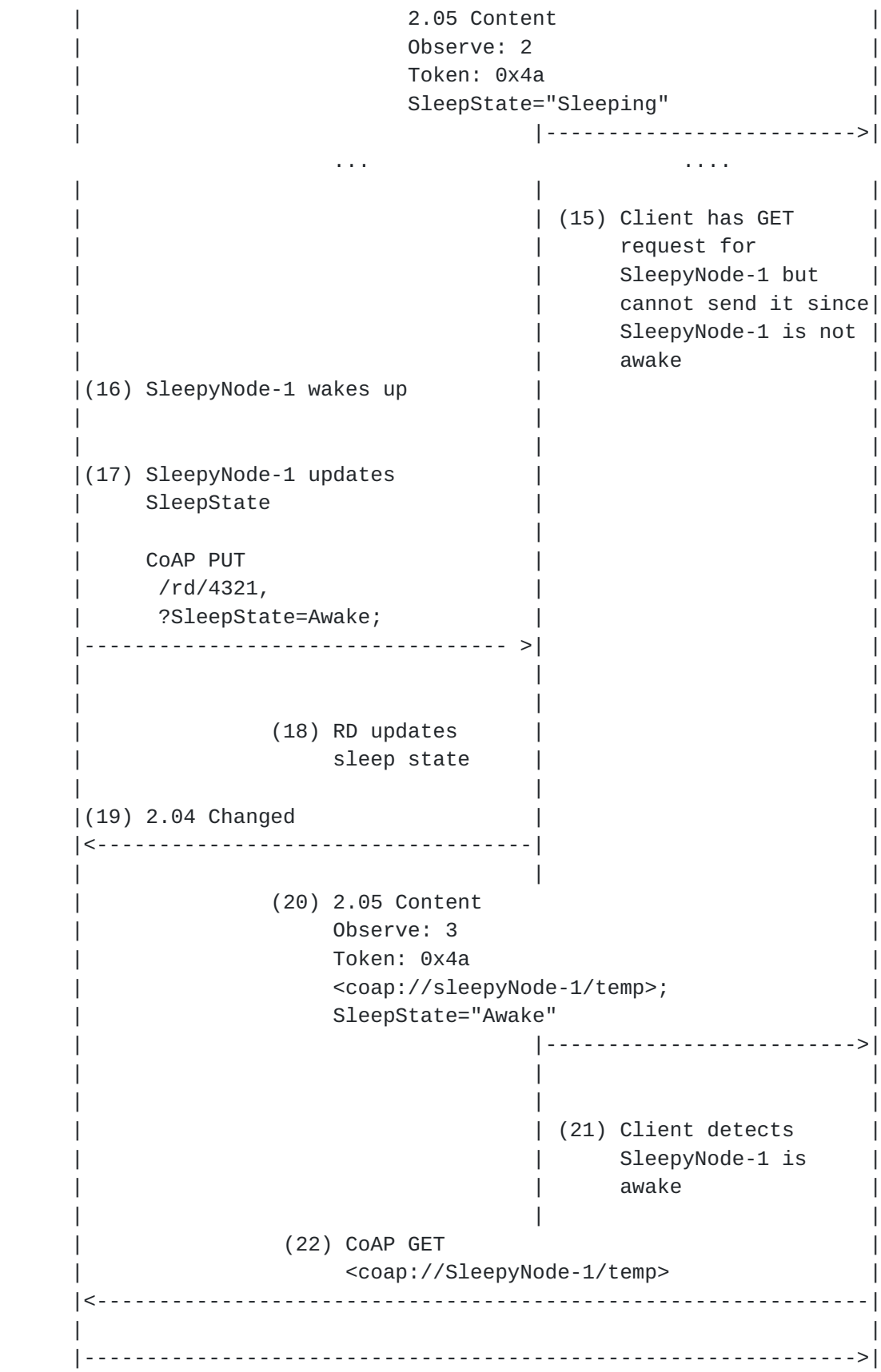
(16)-(19) SleepyNode-1 wakes up and updates the SleepState in the RD.

(20)-(21) RD notifies the client through previously established observe relationship.

(22)-(23) Client sends GET request for temperature sensors and successfully receives the content as SleepyNode-1 is awake.


SleepyNode-1
Server
Resource
Directory
Client
(1) CoAP POST
/rd,
?ep=SleepyNode-1;
?SleepState=Awake;
PAYLOAD:
"</sensors...>"
(2) RD services the
Registration
Request
(3) 2.01 Created
Location: /rd/4321
(4) Client performs RD
end-point lookup to
find endpoint that
has temperature
sensor and is awake
CoAP GET
/rd-lookup/ep?rt=Temperature&SleepState=Awake
(5) 2.05 Content
<coap://{ip:port}>;ep="SleepyNode-1"
(6) Client adds itself
to list of observers
to get SleepState
updates from RD for
SleepyNode-1
CoAP GET
/rd-lookup/ss?ep=SleepyNode-1
Observe: 0
Token: 0x4a


(7) 2.05 Content
Observe: 1
Token: 0x4a
SleepState="Awake"
(8) Client performs RD 'resource' lookup to find URI of temperature sensor of resource hosted on SleepyNode-1
CoAP GET
/rd-lookup/res?rt=Temperature&ep="SleepyNode-1"
(9) 2.05 Content
<coap://sleepyNode-1/temp>;
(10) SleepyNode-1 prepares to go to sleep so it updates SleepState
CoAP PUT
/rd/4321,
?SleepState=Sleeping
(11) RD updates SleepState of SleepyNode-1
(12) 2.04 Changed
(13) SleepyNode-1 goes to sleep
(14) RD sends notification to client


2.05 Content
Observe: 2
Token: 0x4a
SleepState="Sleeping"
(15) Client has GET request for SleepyNode-1 but cannot send it since SleepyNode-1 is not awake
(16) SleepyNode-1 wakes up
(17) SleepyNode-1 updates SleepState
CoAP PUT
/rd/4321,
?SleepState=Awake;
(18) RD updates sleep state
(19) 2.04 Changed
(20) 2.05 Content
Observe: 3
Token: 0x4a
<coap://sleepyNode-1/temp>;
SleepState="Awake"
(21) Client detects SleepyNode-1 is awake
(22) CoAP GET
<coap://SleepyNode-1/temp>

```
     |                (23) 2.05 Content                              |
     |                                                               |
```

Figure 2: Asynchronous Resource Directory Based Sleep Tracking

3.4. RD Caching Proxy

It would be useful for an RD to be able to indicate which proxy performs caching for Sleepy CoAP nodes (see Section 2). This would be done through a new RD "CachingProxy" attribute for each device (similiar to the attributes defined in Section 3.1):

- An RD may be co-located with a proxy that performs caching for CoAP nodes. In this case, the RD automatically adds itself to each CachingProxy entry.

- The sleepy node itself could suggest the CachingProxy if it is peered to a specific proxy.

This parameter would be added to the RD URI Template Variables defined in [I-D.ietf-core-resource-directory].

4. Performance Results

4.1. Prototype Setup

A simple prototype was implemented to validate certain aspects of the performance of the proposed CoAP sleepy node support protocol enhancement. The network for the prototype is shown in Figure 3.

Key aspects of the prototype are as follows:

- There are two modes of operation: "Caching Only" and "Caching and Sleep Aware"

- In "Caching Only" mode the Reverse Proxy will cache responses and service new requests according to the "Max-Age" parameter as defined in [I-D.ietf-core-coap].

- In "Caching and Sleep Aware" mode the Reverse Proxy will also cache responses and service new requests according to "Max-Age". However, if the cache is empty (e.g. exceeded Max-Age) then there will be extra sleep aware functionality. Specifically, the proxy will also be aware of the "SleepDuration", "NextSleep" and "SleepState" sleepy node parameters as defined in Section 3.1. Based on these sleep parameters:

  * For servers that are asleep but expected to wake up soon: The proxy will immediately send a non-piggy backed ACK to the client. Then store-and-forward the request to the server when it wakes up, and then return the non-piggybacked response to the client thereafter.

  * For servers that are asleep but not expected to wake up soon: The proxy will immediately send a "5.03 Service Unavailable (and retry after)" to the client.

- The key variables in the experiment are: (1) Number of clients; (2) Number of sleepy servers; (3) Delay between client requests; (4) MaxAge; (5) SleepDuration; (6) NextSleep; and (7) SleepState.

- The target of the experiment will be to measure the amount of traffic over the network in the two modes of operation (for the same input client requests profile). It is hypothesized that the "Caching and Sleep Aware" mode will have the minimal amount of network traffic indicating that the Sleep Aware network will be the most efficient.

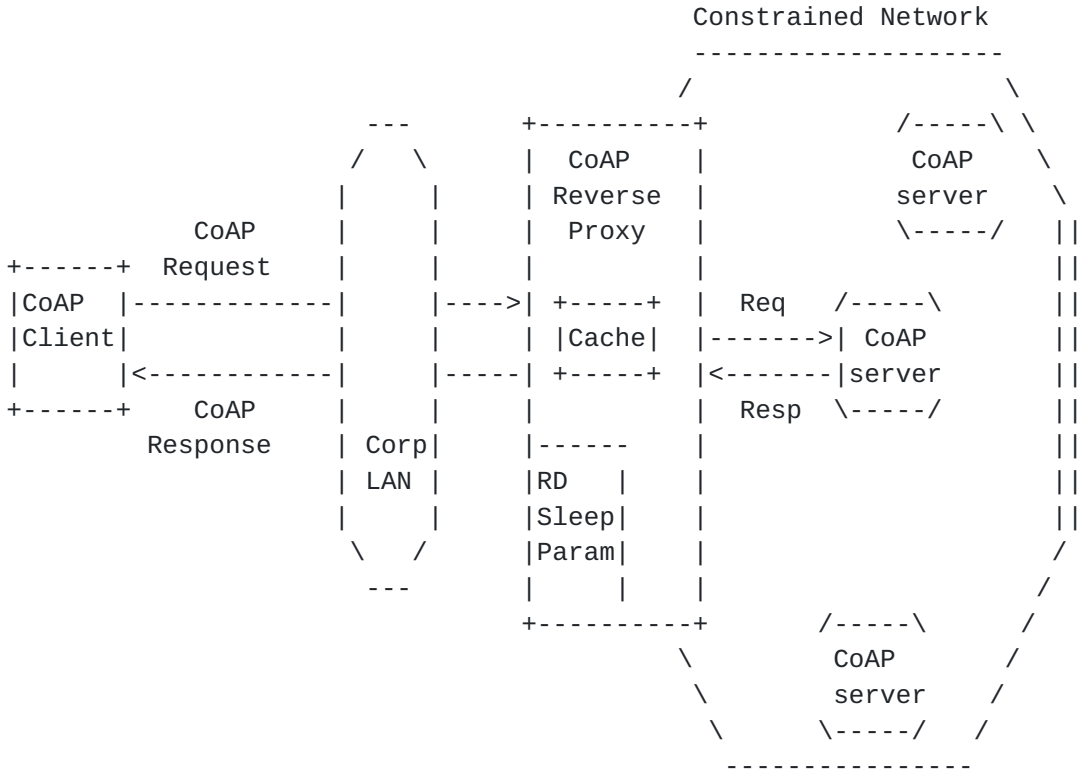


Figure 3: Prototype Configuration

4.2. Initial Results

Baseline performance results of A CoAP system with Sleep Aware Proxy support described in Section 4.1 is given in [IETF86Results] (see pg. 65-95). Key conclusions were:

- If only GETs are considered, a cache based system is efficient as long as the cache is valid (i.e before Max-Age).

- However, since the cache often expires, a Sleep Aware Proxy further minimizes the number of GET transactions. Also, the Sleep Aware Proxy reduces number of transactions for other methods (e.g. PUTs) where a cache does not help.

4.3. Comparison with Observe

A comparison of the performance of a CoAP system with Sleep Aware Proxy versus a CoAP system supporting Observe is given in [IETF87Results] (see pg. 150-182). Key conclusions were:

- If only GETs are considered, an Observe based system is quite efficient.

- However, Observe in combination with Sleep Aware Proxy further minimizes the number of GET transactions. Also, the Sleep Aware Proxy reduces number of transactions for other methods (e.g. PUTs) where Observe does not help.

5. Acknowledgements

Thanks to Carsten Bormann, Esko Dijk, Thomas Fossati, Salvatore Loreto, Dale Seed, and Zach Shelby for valuable discussions and feedback on this document.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

TBD. (All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.)

8. References

8.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-10 (work in progress), September 2013.

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

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

8.2. Informative References

[I-D.ietf-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-00 (work in progress), June 2013.

[IETF86Results]
Rahman, A., "IETF-86 Sleepy Node Performance Results", March 2013, <http://www.ietf.org/proceedings/86/slides/slides-86-core-1.pdf>.

[IETF87Results]
Rahman, A., "IETF-87 Sleepy Node Performance Results", July 2013, <http://www.ietf.org/proceedings/87/slides/slides-87-core-0.pdf>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

Author's Address

Akbar Rahman
InterDigital Communications, LLC
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com