LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

M. Kovatsch
ETH Zurich
O. Bergmann
C. Bormann, Ed.
Universitaet Bremen TZI

July 02, 2018

CoAP Implementation Guidance
draft-ietf-lwig-coap-06

Abstract

The Constrained Application Protocol (CoAP) is designed for resource-constrained nodes and networks such as sensor nodes in a low-power lossy network (LLN). Yet to implement this Internet protocol on Class 1 devices (as per RFC 7228, ~ 10 KiB of RAM and ~ 100 KiB of ROM) also lightweight implementation techniques are necessary. This document provides lessons learned from implementing CoAP for tiny, battery-operated networked embedded systems. In particular, it provides guidance on correct implementation of the CoAP specification RFC 7252, memory optimizations, and customized protocol parameters.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Table of Contents

1. Introduction 3
2. Protocol Implementation 4
2.1. Client/Server Model 4
2.2. Message Processing 5
2.2.1. On-the-fly Processing 5
2.2.2. Internal Data Structure 6
2.3. Message ID Usage 7
2.3.1. Duplicate Rejection 7
2.3.2. MID Namespaces 8
2.3.3. Relaxation on the Server 8
2.3.4. Relaxation on the Client 9
2.4. Token Usage 10
2.4.1. Tokens for Observe 11
2.4.2. Tokens for Blockwise Transfers 12
2.5. Transmission States 12
2.5.1. Request/Response Layer 12
2.5.2. Message Layer 13
2.6. Out-of-band Information 14
2.7. Programming Model 15
2.7.1. Client 16
2.7.2. Server 16
3. Optimizations 17
3.1. Message Buffers 17
3.2. Retransmissions 18
3.3. Observable Resources 18
3.4. Blockwise Transfers 19
3.4.1. Generic Proxying of Block Messages 19
3.4.2. Atomic Blockwise Operations 20
3.5. Deduplication with Sequential MIDs 20
4. Alternative Configurations 23
4.1. Transmission Parameters 23
4.2. CoAP over IPv4 24
5. Binding to specific lower-layer APIs 24
5.1. Berkeley Socket Interface 24
5.1.1. Responding from the right address 24
5.1.2. Handling ICMP errors 25
5.2. Java 25
5.3. Multicast detection 26
5.4. DTLS 26

6. CoAP on various transports 26
6.1. CoAP over reliable transports 27
6.2. Translating between transports 27
6.2.1. Transport translation by proxies 27
6.2.2. One-to-one Transport translation 28
7. IANA considerations 28
8. Security considerations 28
9. Acknowledgements 28
10. References 28
10.1. Normative References 28
10.2. Informative References 29
Authors' Addresses 30

1. Introduction

The Constrained Application Protocol [RFC7252] has been designed specifically for machine-to-machine communication in networks with very constrained nodes. Typical application scenarios therefore include building automation, process optimization, and the Internet of Things. The major design objectives have been set on small protocol overhead, robustness against packet loss, and against high latency induced by small bandwidth shares or slow request processing in end nodes. To leverage integration of constrained nodes with the world-wide Internet, the protocol design was led by the REST architectural style that accounts for the scalability and robustness of the Hypertext Transfer Protocol [RFC7230].

Lightweight implementations benefit from this design in many respects: First, the use of Uniform Resource Identifiers (URIs) for naming resources and the transparent forwarding of their representations in a server-stateless request/response protocol make protocol translation to HTTP a straightforward task. Second, the set of protocol elements that are unavoidable for the core protocol, and thus must be implemented on every node, has been kept very small, minimizing the unnecessary accumulation of "optional" features. Options that - when present - are critical for message processing are explicitly marked as such to force immediate rejection of messages with unknown critical options. Third, the syntax of protocol data units is easy to parse and is carefully defined to avoid creation of state in servers where possible.

Although these features enable lightweight implementations of the Constrained Application Protocol, there is still a tradeoff between robustness and latency of constrained nodes on one hand and resource demands on the other. For constrained nodes of Class 1 or even Class 2 [RFC7228], the most limiting factors usually are dynamic memory needs, static code size, and energy. Most implementations

therefore need to optimize internal buffer usage, omit idle protocol features, and maximize sleeping cycles.

The present document gives possible strategies to solve this tradeoff for very constrained nodes (i.e., Class 1). For this, it provides guidance on correct implementation of the CoAP specification [RFC7252], memory optimizations, and customized protocol parameters.

2. Protocol Implementation

In the programming styles supported by very simple operating systems as found on constrained nodes, preemptive multi-threading is not an option. Instead, all operations are triggered by an event loop system, e.g., in a send-receive-dispatch cycle. It is also common practice to allocate memory statically to ensure stable behavior, as no memory management unit (MMU) or other abstractions are available. For a CoAP node, the two key parameters for memory usage are the number of (re)transmission buffers and the maximum message size that must be supported by each buffer. Often the maximum message size is set far below the 1280-byte MTU of 6LoWPAN to allow more than one open Confirmable transmission at a time (in particular for parallel observe notifications [RFC7641]). Note that implementations on constrained platforms often not even support the full MTU. Larger messages must then use blockwise transfers [RFC7959], while a good tradeoff between 6LoWPAN fragmentation and CoAP header overhead must be found. Usually the amount of available free RAM dominates this decision. For Class 1 devices, the maximum message size is typically 128 or 256 bytes (blockwise) payload plus an estimate of the maximum header size for the worst case option setting.

2.1. Client/Server Model

In general, CoAP servers can be implemented more efficiently than clients. REST allows them to keep the communication stateless and piggy-backed responses are not stored for retransmission, saving buffer space. The use of idempotent requests also allows to relax deduplication, which further decreases memory usage. It is also easy to estimate the required maximum size of message buffers, since URI paths, supported options, and maximum payload sizes of the application are known at compile time. Hence, when the application is distributed over constrained and unconstrained nodes, the constrained ones should preferably have the server role.

HTTP-based applications have established an inverse model because of the need for simple push notifications: A constrained client uses POST requests to update resources on an unconstrained server whenever an event (e.g., a new sensor reading) is triggered. This requirement is solved by the Observe option [RFC7641] of CoAP. It allows servers

to initiate communication and send push notifications to interested client nodes. This allows a more efficient and also more natural model for CoAP-based applications, where the information source is an origin server, which can also benefit from caching. Publish-subscribe brokers [I-D.ietf-core-coap-pubsub] may be deployed to act in the server role on behalf of constrained clients.

2.2. Message Processing

Apart from the required buffers, message processing is symmetric for clients and servers. First the base header has to be parsed and thereby checked if it is a valid CoAP message. For UDP datagrams, the version identifier or a size smaller than four bytes identify non-CoAP data. These datagrams need to be silently ignored. Other message format errors, such as an incomplete datagram or the usage of reserved values, may need to be rejected with a Reset (RST) message (see Section 4.2 and 4.3 of [RFC7252] for details).

As CoAP over TCP has a different base header, the Length field must be parsed to determine the message size. As this field may have up to five bytes, it may be extend over TCP segment boundaries. For CoAP over WebSockets the actual message length is given by the WebSocket frame hence the Length field is always zero.

Next, the token length is read based on the TKL field which is for all transports contained in the four least significant bits of the first byte. The (possibly empty) Token itself is located immediately after the four-byte base header for UDP, while for TCP and WebSockets, it follows the variable Length field and Code byte.

For the options following the Token, there are two alternatives: either process them on the fly when an option is accessed or initially parse all values into an internal data structure.

2.2.1. On-the-fly Processing

The advantage of on-the-fly processing is that no additional memory needs to be allocated to store the option values, which are stored efficiently inline in the buffer for incoming messages. Once the message is accepted for further processing, the set of options contained in the received message must be decoded to check for unknown critical options. To avoid multiple passes through the option list, the option parser might maintain a bit-vector where each bit represents an option number that is present in the received request. With the wide and sparse range of option numbers, the number itself cannot be used to indicate the number of left-shift operations to mask the corresponding bit. Hence, an implementation-specific enum of supported options should be used to mask the present

options of a message in the bitmap. In addition, the byte index of every option (a direct pointer) can be added to a sparse list (e.g., a one-dimensional array) for fast retrieval.

This particularly enables efficient handling of options that might occur more than once such as Uri-Path. In this implementation strategy, the delta is zero for any subsequent path segment, hence the stored byte index for this option (e.g., 11 for Uri-Path) would be overwritten to hold a pointer to only the last occurrence of that option. The Uri-Path can be resolved on the fly, though, and a pointer to the targeted resource stored directly in the sparse list.

Once the option list has been processed, all known critical option and all elective options can be masked out in the bit-vector to determine if any unknown critical option was present. If this is the case, this information can be used to create a 4.02 response accordingly. Note that full processing must only be done up to the highest supported option number. Beyond that, only the least significant bit (Critical or Elective) needs to be checked. Otherwise, if all critical options are supported, the sparse list of option pointers is used for further handling of the message.

2.2.2. Internal Data Structure

Using an internal data structure for all parsed options has an advantage when working on the option values, as they are already in a variable of corresponding type (e.g., an integer in host byte order). The incoming payload and byte strings of the header can be accessed directly in the buffer for incoming messages using pointers (similar to on-the-fly processing). This approach also benefits from a bitmap. Otherwise special values must be reserved to encode an unset option, which might require a larger type than required for the actual value range (e.g., a 32-bit integer instead of 16-bit).

Many of the byte strings (e.g., the URI) are usually not required when generating the response. When all important values are copied (e.g., the Token, which needs to be mirrored), the internal data structure facilitates using the buffer for incoming messages also for the assembly of outgoing messages - which can be the shared IP buffer provided by the operating system.

Setting options for outgoing messages is also easier with an internal data structure. Application developers can set options independent from the option number and do not need to care about the order for the delta encoding. The CoAP encoding is applied in a serialization step before sending. In contrast, assembling outgoing messages with on-the-fly processing requires either extensive memmove operations to

insert new options, or restrictions for developers to set options in their correct order.

2.3. Message ID Usage

Many applications of CoAP use unreliable transports, in particular UDP, which can lose, reorder, and duplicate messages. Although DTLS's replay protection deals with duplication by the network, losses are addressed with DTLS retransmissions only for the handshake protocol and not for the application data protocol. Furthermore, CoAP implementations usually send CON retransmissions in new DTLS records, which are not considered duplicates at the DTLS layer.

2.3.1. Duplicate Rejection

CoAP's messaging sub-layer has been designed with protocol functionality such that rejection of duplicate messages is always possible. It is realized through the Message IDs (MIDs) and their lifetimes with regard to the message type.

Duplicate detection is under the discretion of the recipient (see Section 4.5 of [RFC7252], Section 2.3.3, Section 2.3.4). Where it is desired, the receiver needs to keep track of MIDs to filter the duplicates for at least NON_LIFETIME (145 s). This time also holds for CON messages, since it equals the possible reception window of MAX_TRANSMIT_SPAN + MAX_LATENCY.

On the sender side, MIDs of CON messages must not be re-used within the EXCHANGE_LIFETIME; MIDs of NONs respectively within the NON_LIFETIME. In typical scenarios, however, senders will re-use MIDs with intervals far larger than these lifetimes: with sequential assignment of MIDs, coming close to them would require 250 messages per second, much more than the bandwidth of constrained networks would usually allow for.

In cases where senders might come closer to the maximum message rate, it is recommended to use more conservative timings for the re-use of MIDs. Otherwise, opposite inaccuracies in the clocks of sender and recipient may lead to obscure message loss. If needed, higher rates can be achieved by using multiple endpoints for sending requests and managing the local MID per remote endpoint instead of a single counter per system (essentially extending the 16-bit message ID by a 16-bit port number and/or an 128-bit IP address). In controlled scenarios, such as real-time applications over industrial Ethernet, the protocol parameters can also be tweaked to achieve higher message rates (Section 4.1).

2.3.2. MID Namespaces

MIDs are assigned under the control of the originator of CON and NON messages, and they do not mix with the MIDs assigned by the peer for CON and NON in the opposite direction. Hence, CoAP implementors need to make sure to manage different namespaces for the MIDs used for deduplication. MIDs of outgoing CONs and NONs belong to the local endpoint; so do the MIDs of incoming ACKs and RSTs. Accordingly, MIDs of incoming CONs and NONs and outgoing ACKs and RSTs belong to the corresponding remote endpoint. Figure 1 depicts a scenario where mixing the namespaces would cause erroneous filtering.

```
                   Client              Server
                      |                  |
                      |   CON [0x1234]   |
                      +----------------->|
                      |                  |
                      |   ACK [0x1234]   |
                      |<-----------------+
                      |                  |
                      |   CON [0x4711]   |
                      |<-----------------+ Separate response
                      |                  |
                      |   ACK [0x4711]   |
                      +----------------->|
                      |                  |
  A request follows that uses the same MID as the last separate response
                      |                  |
                      |   CON [0x4711]   |
                      +----------------->|
  Response is filtered |                  |
    because MID 0x4711 |   ACK [0x4711]   |
       is still in the X<-----------------+ Piggy-backed response
    deduplication list |                  |
```

Figure 1: Deduplication must manage the MIDs in different namespace corresponding to their origin endpoints.

2.3.3. Relaxation on the Server

Using the de-duplication functionality is at the discretion of the receiver: Processing of duplicate messages comes at a cost, but so does the management of the state associated with duplicate rejection. The number of remote endpoints that need to be managed might be vast. This can be costly in particular for less constrained nodes that have throughput in the order of hundreds of thousands requests per second (which needs about 16 GiB of RAM just for duplicate rejection). Deduplication is also heavy for servers on Class 1 devices, as also

piggy-backed responses need to be stored for the case that the ACK message is lost. Hence, a receiver may have good reasons to decide not to perform deduplication. This behavior is possible when the application is designed with idempotent operations only and makes good use of the If-Match/If-None-Match options.

If duplicate rejection is indeed necessary (e.g., for non-idempotent requests) it is important to control the amount of state that needs to be stored. It can be reduced, for instance, by deduplication at resource level: Knowledge of the application and supported representations can minimize the amount of state that needs to be kept.

2.3.4. Relaxation on the Client

Duplicate rejection on the client side can be simplified by choosing clever Tokens that are virtually not re-used (e.g., through an obfuscated sequence number in the Token value) and only filter based on the list of open Tokens. If a client wants to re-use Tokens (e.g., the empty Token for optimizations), it requires strict duplicate rejection based on MIDs to avoid the scenario outlined in Figure 2.

```
                              Client              Server
                                 |                  |
                                 |   CON [0x7a10]   |
                                 |    GET /temp     |
                                 |   (Token 0x23)   |
                                 +----------------->|
                                 |                  |
                                 |   ACK [0x7a10]   |
                                 |<-----------------+
                                 |                  |
                                 ... Time Passes  ...
                                 |                  |
                                 |   CON [0x23bb]   |
                                 |  4.04 Not Found  |
                                 |   (Token 0x23)   |
                                 |<-----------------+
                                 |                  |
                                 |   ACK [0x23bb]   |
                                 +--------X         |
                                 |                  |
                                 |   CON [0x7a11]   |
                                 |   GET /resource  |
                                 |   (Token 0x23)   |
                                 +----------------->|
                                 |                  |
                                 |   CON [0x23bb]   |
          Causing an implicit    |  4.04 Not Found  |
           acknowledgement if    |   (Token 0x23)   |
       not filtered through      X<-----------------+ Retransmission
        duplicate rejection      |                  |
```

Figure 2: Re-using Tokens requires strict duplicate rejection.

2.4. Token Usage

Tokens are chosen by the client and help to identify request/response pairs that span several message exchanges (e.g., a separate response, which has a new MID). Servers do not generate Tokens and only mirror what they receive from the clients. Tokens must be unique within the namespace of a client throughout their lifetime. This begins when being assigned to a request and ends when the open request is closed by receiving and matching the final response. Neither empty ACKs nor notifications (i.e., responses carrying the Observe option) terminate the lifetime of a Token.

As already mentioned, a clever assignment of Tokens can help to simplify duplicate rejection. Yet this is also important for coping with client crashes. When a client restarts during an open request

and (unknowingly) re-uses the same Token, it might match the response from the previous request to the current one. Hence, when only the Token is used for matching, which is always the case for separate responses, randomized Tokens with enough entropy should be used. The 8-byte range for Tokens can even allow for one-time usage throughout the lifetime of a client node. When DTLS is used, client crashes/ restarts will lead to a new security handshake, thereby solving the problem of mismatching responses and/or notifications.

2.4.1. Tokens for Observe

In the case of Observe [RFC7641], a request will be answered with multiple notifications and it is important to continue keeping track of the Token that was used for the request - its lifetime will end much later. Upon establishing an Observe relationship, the Token is registered at the server. Hence, the client's use of that specific Token is now limited to controlling the Observation relationship. A client can use it to cancel the relationship, which frees the Token upon success (i.e., the message with an Observe Option with the value set to 'deregister' (1) is confirmed with a response; see [RFC7641] section 3.6). However, the client might never receive the response due to a temporary network outage or worse, a server crash. Although a network outage will also affect notifications so that the Observe garbage collection could apply, the server might simply happen not to send CON notifications during that time. Alternative Observe lifetime models such as Stubbornness(tm) might also keep relationships alive for longer periods.

Thus, it is best to carefully choose the Token value used with Observe requests. (The empty value will rarely be applicable.) One option is to assign and re-use dedicated Tokens for each Observe relationship the client will establish. The choice of Token values also is critical in NoSec mode, to limit the effectiveness of spoofing attacks. Here, the recommendation is to use randomized Tokens with a length of at least four bytes (see Section 5.3.1 of [RFC7252]). Thus, dedicated ranges within the 8-byte Token space should be used when in NoSec mode. This also solves the problem of mismatching notifications after a client crash/restart.

When the client wishes to reinforce its interest in a resource, maybe not really being sure whether the server has forgotten it or not, the Token value allocated to the Observe relationship is used to re-register that observation (see Section 3.3.1 of [RFC7641] for details): If the server is still aware of the relationship (an entry with a matching endpoint and token is already present in its list of observers for the resource), it will not add a new relationship but will replace or update the existing one (Section 4.1 of [RFC7641]).

If not, it will simply establish a new registration which of course also uses the Token value.

If the client sends an Observe request for the same resource with a new Token, this is not a protocol violation, because the specification allows the client to observe the same resource in a different Observe relationship if the cache-key is different (e.g., requesting a different Content-Format). If the cache-key is not different, though, an additional Observe relationship just wastes the server's resources, and is therefore not allowed; the server might rely on this for its housekeeping.

2.4.2. Tokens for Blockwise Transfers

In general, blockwise transfers are independent from the Token and are correlated through client endpoint address and server address and resource path (destination URI). Thus, each block may be transferred using a different Token. Still it can be beneficial to use the same Token (it is freed upon reception of a response block) for all blocks, e.g., to easily route received blocks to the same response handler.

When Block2 is combined with Observe, notifications only carry the first block and it is up to the client to retrieve the remaining ones. These GET requests do not carry the Observe option and need to use a different Token, since the Token from the notification is still in use.

2.5. Transmission States

CoAP endpoints must keep transmission state to manage open requests, to handle the different response modes, and to implement reliable delivery at the message layer. The following finite state machines (FSMs) model the transmissions of a CoAP exchange at the request/response layer and the message layer. These layers are linked through actions. The M_CMD() action triggers a corresponding transition at the message layer and the RR_EVT() action triggers a transition at the request/response layer. The FSMs also use guard conditions to distinguish between information that is only available through the other layer (e.g., whether a request was sent using a CON or NON message).

2.5.1. Request/Response Layer

Figure 3 depicts the two states at the request/response layer of a CoAP client. When a request is issued, a "reliable_send" or "unreliable_send" is triggered at the message layer. The WAITING state can be left through three transitions: Either the client

cancels the request and triggers cancellation of a CON transmission at the message layer, the client receives a failure event from the message layer, or a receive event containing a response.

```
         +------------CANCEL-------------------------------+
         |        / M_CMD(cancel)                           |
         |                                                  V
         |                                             +------+
     +-------+ -------RR_EVT(fail)--------------------> |      |
     |WAITING|                                          | IDLE |
     +-------+ -------RR_EVT(rx)[is Response]---------> |      |
         ^                / M_CMD(accept)               +------+
         |                                                  |
         +--------------------REQUEST----------------------+
                    / M_CMD((un)reliable_send)
```

Figure 3: CoAP Client Request/Response Layer FSM

A server resource can decide at the request/response layer whether to respond with a piggy-backed or a separate response. Thus, there are two busy states in Figure 4, SERVING and SEPARATE. An incoming receive event with a NON request directly triggers the transition to the SEPARATE state.

```
     +--------+ <----------RR_EVT(rx)[is NON]---------- +------+
     |SEPARATE|                                         |      |
     +--------+ ----------------RESPONSE--------------> | IDLE |
         ^            / M_CMD((un)reliable_send)        |      |
         |                                         +---> +------+
         |EMPTY_ACK                                |         |
         |/M_CMD(accept)                           |         |
         |                                         |         |
         |                                         |         |
     +--------+                                    |         |
     |SERVING | --------------RESPONSE------------+         |
     +--------+          / M_CMD(accept)                     |
         ^                                                   |
         +------------------------RR_EVT(rx)[is CON]--------+
```

Figure 4: CoAP Server Request/Response Layer FSM

2.5.2. Message Layer

Figure 5 shows the different states of a CoAP endpoint per message exchange. Besides the linking action RR_EVT(), the message layer has a TX action to send a message. For sending and receiving NONs, the endpoint remains in its CLOSED state. When sending a CON, the endpoint remains in RELIABLE_TX and keeps retransmitting until the

transmission times out, it receives a matching RST, the request/response layer cancels the transmission, or the endpoint receives an implicit acknowledgement through a matching NON or CON. Whenever the endpoint receives a CON, it transitions into the ACK_PENDING state, which can be left by sending the corresponding ACK.

```
+-----------+ <-------M_CMD(reliable_send)-----+
|           |            / TX(con)              \
|           |                                 +---------------+
|           | ---TIMEOUT(RETX_WINDOW)------> |               |
|RELIABLE_TX|     / RR_EVT(fail)              |               |
|           | ---------------------RX_RST--> |               | <----+
|           |                / RR_EVT(fail)   |               |      |
+-----------+ ----M_CMD(cancel)------------> |    CLOSED     |      |
 ^  |  |  \  \                               |               | --+  |
 |  |  |   \  +-------------------RX_ACK---> |               |   |  |
 +*1+  |    \                / RR_EVT(rx)    |               |   |  |
       |     +----RX_NON-------------------> +---------------+   |  |
       |       / RR_EVT(rx)                  ^ ^ ^ ^  | | | |   |  |
       |                                     | | | |  | | | |   |  |
       |                                     | | | +*2+ | | |   |  |
       |                                     | | +--*3--+ | |   |  |
       |                                     | +----*4----+ |   |  |
       |                                     +------*5------+   |  |
       |                +---------------+                       |  |
       |                |  ACK_PENDING  | <--RX_CON-------------+  |
       +----RX_CON----> |               |  / RR_EVT(rx)            |
         / RR_EVT(rx)   +---------------+ ---------M_CMD(accept)---+
                                                   / TX(ack)
```

*1: TIMEOUT(RETX_TIMEOUT) / TX(con)
*2: M_CMD(unreliable_send) / TX(non)
*3: RX_NON / RR_EVT(rx)
*4: RX_RST / REMOVE_OBSERVER
*5: RX_ACK

Figure 5: CoAP Message Layer FSM

T.B.D.: (i) Rejecting messages (can be triggered at message and request/response layer). (ii) ACKs can also be triggered at both layers.

2.6. Out-of-band Information

The CoAP implementation can also leverage out-of-band information, that might also trigger some of the transitions shown in Section 2.5. In particular ICMP messages can inform about unreachable remote endpoints or whole network outages. This information can be used to

pause or cancel ongoing transmission to conserve energy. Providing ICMP information to the CoAP implementation is easier in constrained environments, where developers usually can adapt the underlying OS (or firmware). This is not the case on general purpose platforms that have full-fledged OSes and make use of high-level programming frameworks.

The most important ICMP messages are host, network, port, or protocol unreachable errors. After appropriate vetting (cf. [RFC5927]), they should cause the cancellation of ongoing CON transmissions and clearing (or deferral) of Observe relationships. Requests to this destination should be paused for a sensible interval. In addition, the device could indicate of this error through a notification to a management endpoint or external status indicator, since the cause could be a misconfiguration or general unavailability of the required service. Problems reported through the Parameter Problem message are usually caused through a similar fundamental problem.

The CoAP specification recommends to ignore Source Quench and Time Exceeded ICMP messages, though. Source Quench messages were originally intended to inform the sender to reduce the rate of packets. However, this mechanism is deprecated through [RFC6633]. CoAP also comes with its own congestion control mechanism, which is already designed conservatively. One advanced mechanism that can be employed for better network utilization is CoCoA, [I-D.ietf-core-cocoa]. Time Exceeded messages often occur during transient routing loops (unless they are caused by a too small initial Hop Limit value).

2.7. Programming Model

The event-driven approach, which is common in event-loop-based firmware, has also proven very efficient for embedded operating systems [TinyOS], [Contiki]. Note that an OS is not necessarily required and a traditional firmware approach can suffice for Class 1 devices. Event-driven systems use split-phase operations (i.e., there are no blocking functions, but functions return and an event handler is called once a long-lasting operation completes) to enable cooperative multi-threading with a single stack.

Bringing a Web transfer protocol to constrained environments does not only change the networking of the corresponding systems, but also the programming model. The complexity of event-driven systems can be hidden through APIs that resemble classic RESTful Web service implementations.

2.7.1. Client

An API for asynchronous requests with response handler functions goes hand-in-hand with the event-driven approach. Synchronous requests with a blocking send function can facilitate applications that require strictly ordered, sequential request execution (e.g., to control a physical process) or other checkpointing (e.g., starting operation only after registration with the resource directory was successful). However, this can also be solved by triggering the next operation in the response handlers. Furthermore, as mentioned in Section 2.1, it is more like that complex control flow is done by more powerful devices and Class 1 devices predominantly run a CoAP server (which might include a minimal client to communicate with a resource directory).

2.7.2. Server

On CoAP servers, the event-driven nature can be hidden through resource handler abstractions as known from traditional REST frameworks. The following types of RESTful resources have proven useful to provide an intuitive API on constrained event-driven systems:

NORMAL A normal resource defined by a static Uri-Path and an associated resource handler function. Allowed methods could already be filtered by the implementation based on flags. This is the basis for all other resource types.

PARENT A parent resource manages several sub-resources under a given base path by programmatically evaluating the Uri-Path. Defining a URI template (see [RFC6570]) would be a convenient way to pre-parse arguments given in the Uri-Path.

PERIODIC A resource that has an additional handler function that is triggered periodically by the CoAP implementation with a resource-specific interval. It can be used to sample a sensor or perform similar periodic updates of its state. Usually, a periodic resource is observable and sends the notifications by triggering its normal resource handler from the periodic handler. These periodic tasks are quite common for sensor nodes, thus it makes sense to provide this functionality in the CoAP implementation and avoid redundant code in every resource.

EVENT An event resource is similar to an periodic resource, only that the second handler is called by an irregular event such as a button.

SEPARATE Separate responses are usually used when handling a request takes more time, e.g., due to a slow sensor or UART-based subsystems. To not fully block the system during this time, the handler should also employ split-phase execution: The resource handler returns as soon as possible and an event handler resumes responding when the result is ready. The separate resource type can abstract from the split-phase operation and take care of temporarily storing the request information that is required later in the result handler to send the response (e.g., source address and Token).

3. Optimizations

3.1. Message Buffers

The cooperative multi-threading of an event loop system allows to optimize memory usage through in-place processing and reuse of buffers, in particular the IP buffer provided by the OS or firmware.

CoAP servers can significantly benefit from in-place processing, as they can create responses directly in the incoming IP buffer. Note that an embedded OS usually only has a single buffer for incoming and outgoing IP packets. The first few bytes of the basic header are usually parsed into an internal data structure and can be overwritten without harm. Thus, empty ACKs and RST messages can promptly be assembled and sent using the IP buffer. Also when a CoAP server only sends piggy-backed or Non-confirmable responses, no additional buffer is required at the application layer. This, however, requires careful timing so that no incoming data is overwritten before it was processed. Because of cooperative multi-threading, this requirement is relaxed, though. Once the message is sent, the IP buffer can accept new messages again. This does not work for Confirmable messages, however. They need to be stored for retransmission and would block any further IP communication.

Depending on the number of requests that can be handled in parallel, an implementation might create a stub response filled with any option that has to be copied from the original request to the separate response, especially the Token option. The drawback of this technique is that the server must be prepared to receive retransmissions of the previous (Confirmable) request to which a new acknowledgement must be generated. If memory is an issue, a single buffer can be used for both tasks: Only the message type and code must be updated, changing the message id is optional. Once the resource representation is known, it is added as new payload at the end of the stub response. Acknowledgements still can be sent as described before as long as no additional options are required to describe the payload.

3.2. Retransmissions

CoAP's reliable transmissions require the before-mentioned retransmission buffers. Messages, such as the requests of a client, should be stored in serialized form. For servers, retransmissions apply for Confirmable separate responses and Confirmable notifications [RFC7641]. As separate responses stem from long-lasting resource handlers, the response should be stored for retransmission instead of re-dispatching a stored request (which would allow for updating the representation). For Confirmable notifications, please see Section 2.6, as simply storing the response can break the concept of eventual consistency.

String payloads such as JSON require a buffer to print to. By splitting the retransmission buffer into header and payload part, it can be reused. First to generate the payload and then storing the CoAP message by serializing into the same memory. Thus, providing a retransmission for any message type can save the need for a separate application buffer. This, however, requires an estimation about the maximum expected header size to split the buffer and a memmove to concatenate the two parts.

For platforms that disable clock tick interrupts in sleep states, the application must take into consideration the clock deviation that occurs during sleep (or ensure to remain in idle state until the message has been acknowledged or the maximum number of retransmissions is reached). Since CoAP allows up to four retransmissions with a binary exponential back-off it could take up to 45 seconds until the send operation is complete. Even in idle state, this means substantial energy consumption for low-power nodes. Implementers therefore might choose a two-step strategy: First, do one or two retransmissions and then, in the later phases of back-off, go to sleep until the next retransmission is due. In the meantime, the node could check for new messages including the acknowledgement for any Confirmable message to send.

3.3. Observable Resources

For each observer, the server needs to store at least address, port, token, and the last outgoing message ID. The latter is needed to match incoming RST messages and cancel the observe relationship.

It is favorable to have one retransmission buffer per observable resource that is shared among all observers. Each notification is serialized once into this buffer and only address, port, and token are changed when iterating over the observer list (note that different token lengths might require realignment). The advantage becomes clear for Confirmable notifications: Instead of one

retransmission buffer per observer, only one buffer and only individual retransmission counters and timers in the list entry need to be stored. When the notifications can be sent fast enough, even a single timer would suffice. Furthermore, per-resource buffers simplify the update with a new resource state during open deliveries.

3.4. Blockwise Transfers

Blockwise transfers have the main purpose of providing fragmentation at the application layer, where partial information can be processed. This is not possible at lower layers such as 6LoWPAN, as only assembled packets can be passed up the stack. While [RFC7959] also anticipates atomic handling of blocks, i.e., only fully received CoAP messages, this is not possible on Class 1 devices.

When receiving a blockwise transfer, each block is usually passed to a handler function that for instance performs stream processing or writes the blocks to external memory such as flash. Although there are no restrictions in [RFC7959], it is beneficial for Class 1 devices to only allow ordered transmission of blocks. Otherwise on-the-fly processing would not be possible.

When sending a blockwise transfer out of dynamically generated information, Class 1 devices usually do not have sufficient memory to print the full message into a buffer, and slice and send it in a second step. For instance, if the CoRE Link Format at /.well-known/core is dynamically generated, a generator function is required that generates slices of a large string with a specific offset length (a 'sonprintf()'). This functionality is required recurrently and should be included in a library.

3.4.1. Generic Proxying of Block Messages

Proxies cannot ignore the Block options by specification, because the options Block1 and Block2 are not safe-to-forward. The rationale behind this design decision is that servers might not be able to distinguish blocks originating from different senders once they have been forwarded by a CoAP proxy. For atomic operations where all blocks are assembled before actually executing the desired operation, this could lead to inconsistent state on the server side.

To ensure that this does not happen, a proxy can add the Request-Tag option (see [I-D.ietf-core-echo-request-tag]) containing data that uniquely identifies the originating endpoint in the proxy namespace.

3.4.2. Atomic Blockwise Operations

When an implementation needs to assemble blocks from block-wise transfers, applications need to create an identifier to group messages that belong together. This "Block Key" at least contains:

- The source endpoint (e.g., IP address and port in the UDP case),
- the destination endpoint,
- the Cache-Key (as updated in [RFC7252]), and
- all options that are proxy unsafe and not explicitly described as safe for block-wise assembly.

The only known options safe for block-wise assembly are the options Block1 and Block2 [RFC7959].

For the Block1 phase, the request payload is excluded from the identifier generation as it is just being assembled.

If a message is received that is not the start of a block-wise operation has a Block Key that is not known, and the implementation needs to act atomically on a request body, it must answer 4.08 (Request Entity Incomplete).

Conversely, clients should be aware that requests whose Block Key matches can be interpreted by the server atomically. This especially affects proxies (see Section 3.4.1).

3.5. Deduplication with Sequential MIDs

CoAP's duplicate rejection functionality can be straightforwardly implemented in a CoAP endpoint by storing, for each remote CoAP endpoint ("peer") that it communicates with, a list of recently received CoAP Message IDs (MIDs) along with some timing information. A CoAP message from a peer with a MID that is in the list for that peer can simply be discarded.

The timing information in the list can then be used to time out entries that are older than the _expected extent of the re-ordering_, an upper bound for which can be estimated by adding the _potential retransmission window_ ([RFC7252] section "Reliable Messages") and the time packets can stay alive in the network.

Such a straightforward implementation is suitable in case other CoAP endpoints generate random MIDs. However, this storage method may consume substantial RAM in specific cases, such as:

- many clients are making periodic, non-idempotent requests to a single CoAP server;

- one client makes periodic requests to a large number of CoAP servers and/or requests a large number of resources; where servers happen to mostly generate separate CoAP responses (not piggy-backed);

For example, consider the first case where the expected extent of re-ordering is 50 seconds, and N clients are sending periodic POST requests to a single CoAP server during a period of high system activity, each on average sending one client request per second. The server would need 100 * N bytes of RAM to store the MIDs only. This amount of RAM may be significant on a RAM-constrained platform. On a number of platforms, it may be easier to allocate some extra program memory (e.g. Flash or ROM) to the CoAP protocol handler process than to allocate extra RAM. Therefore, one may try to reduce RAM usage of a CoAP implementation at the cost of some additional program memory usage and implementation complexity.

Some CoAP clients generate MID values by a using a Message ID variable [RFC7252] that is incremented by one each time a new MID needs to be generated. (After the maximum value 65535 it wraps back to 0.) We call this behavior "sequential" MIDs. One approach to reduce RAM use exploits the redundancy in sequential MIDs for a more efficient MID storage in CoAP servers.

Naturally such an approach requires, in order to actually reduce RAM usage in an implementation, that a large part of the peers follow the sequential MID behavior. To realize this optimization, the authors therefore RECOMMEND that CoAP endpoint implementers employ the "sequential MID" scheme if there are no reasons to prefer another scheme, such as randomly generated MID values.

Security considerations might call for a choice for (pseudo)randomized MIDs. Note however that with truly randomly generated MIDs the probability of MID collision is rather high in use cases as mentioned before, following from the Birthday Paradox. For example, in a sequence of 52 randomly drawn 16-bit values the probability of finding at least two identical values is about 2 percent.

From here on we consider efficient storage implementations for MIDs in CoAP endpoints, that are optimized to store "sequential" MIDs. Because CoAP messages may be lost or arrive out-of-order, a solution has to take into account that received MIDs of CoAP messages are not actually arriving in a sequential fashion, due to lost or reordered messages. Also a peer might reset and lose its MID counter(s) state.

In addition, a peer may have a single Message ID variable used in messages to many CoAP endpoints it communicates with, which partly breaks sequentiality from the receiving CoAP endpoint's perspective. Finally, some peers might use a randomly generated MID values approach. Due to these specific conditions, existing sliding window bitfield implementations for storing received sequence numbers are typically not directly suitable for efficiently storing MIDs.

Table 1 shows one example for a per-peer MID storage design: a table with a bitfield of a defined length _K_ per entry to store received MIDs (one per bit) that have a value in the range [MID_i + 1 , MID_i + K].

MID base	K-bit bitfield	base time value
MID_0	010010101001	t_0
MID_1	111101110111	t_1
... etc.		

Table 1: A per-peer table for storing MIDs based on MID_i

The presence of a table row with base MID_i (regardless of the bitfield values) indicates that a value MID_i has been received at a time t_i. Subsequently, each bitfield bit k (0...K-1) in a row i corresponds to a received MID value of MID_i + k + 1. If a bit k is 0, it means a message with corresponding MID has not yet been received. A bit 1 indicates such a message has been received already at approximately time t_i. This storage structure allows e.g. with k=64 to store in best case up to 130 MID values using 20 bytes, as opposed to 260 bytes that would be needed for a non-sequential storage scheme.

The time values t_i are used for removing rows from the table after a preset timeout period, to keep the MID store small in size and enable these MIDs to be safely re-used in future communications. (Note that the table only stores one time value per row, which therefore needs to be updated on receipt of another MID that is stored as a single bit in this row. As a consequence of only storing one time value per row, older MID entries typically time out later than with a simple per-MID time value storage scheme. The endpoint therefore needs to ensure that this additional delay before MID entries are removed from the table is much smaller than the time period after which a peer starts to re-use MID values due to wrap-around of a peer's MID variable. One solution is to check that a value t_i in a table row

is still recent enough, before using the row and updating the value t_i to current time. If not recent enough, e.g. older than N seconds, a new row with an empty bitfield is created.) [Clearly, these optimizations would benefit if the peer were much more conservative about re-using MIDs than currently required in the protocol specification.]

The optimization described is less efficient for storing randomized MIDs that a CoAP endpoint may encounter from certain peers. To solve this, a storage algorithm may start in a simple MID storage mode, first assuming that the peer produces non-sequential MIDs. While storing MIDs, a heuristic is then applied based on monitoring some "hit rate", for example, the number of MIDs received that have a Most Significant Byte equal to that of the previous MID divided by the total number of MIDs received. If the hit rate tends towards 1 over a period of time, the MID store may decide that this particular CoAP endpoint uses sequential MIDs and in response improve efficiency by switching its mode to the bitfield based storage.

4. Alternative Configurations

4.1. Transmission Parameters

When a constrained network of CoAP nodes is not communicating over the Internet, for instance because it is shielded by a proxy or a closed deployment, alternative transmission parameters can be used. Consequently, the derived time values provided in [RFC7252] section 4.8.2 will also need to be adjusted, since most implementations will encode their absolute values.

Static adjustments require a fixed deployment with a constant number or upper bound for the number of nodes, number of hops, and expected concurrent transmissions. Furthermore, the stability of the wireless links should be evaluated. ACK_TIMEOUT should be chosen above the xx% percentile of the round-trip time distribution. ACK_RANDOM_FACTOR depends on the number of nodes on the network. MAX_RETRANSMIT should be chosen suitable for the targeted application. A lower bound for LEISURE can be calculated as

lb_Leisure = S * G / R

where S is the estimated response size, G the group size, and R the target data transfer rate (see [RFC7252] section 8.2). NSTART and PROBING_RATE depend on estimated network utilization. If the main cause for loss are weak links, higher values can be chosen.

Dynamic adjustments will be performed by advanced congestion control mechanisms such as [I-D.ietf-core-cocoa]. They are required if the

main cause for message loss is network or endpoint congestion. Semi-dynamic adjustments could be implemented by disseminating new static transmission parameters to all nodes when the network configuration changes (e.g., new nodes are added or long-lasting interference is detected).

4.2. CoAP over IPv4

CoAP was designed for the properties of IPv6, which is dominating in constrained environments because of the 6LoWPAN adaption layer [RFC6282]. In particular, the size limitations of CoAP are tailored to the minimal MTU of 1280 bytes. Until the transition towards IPv6 converges, CoAP nodes might also communicate over IPv4, though. Sections 4.2 and 4.6 of the base specification [RFC7252] already provide guidance and implementation notes to handle the smaller minimal MTUs of IPv4.

Another deployment issue in legacy IPv4 deployments is caused by Network Address Translators (NATs). The session timeouts are unpredictable and NATs may close UDP sessions with timeout as short as 60 seconds. This makes CoAP endpoints behind NATs practically unreachable, even when they contact the remote endpoint with a public IP address first. Incorrect behavior may also arise when the NAT session heuristic changes the external port between two successive CoAP messages. For the remote endpoint, this will look like two different CoAP endpoints on the same IP address. Such behavior can be fatal for the resource directory registration interface.

5. Binding to specific lower-layer APIs

Implementing CoAP on specific lower-layer APIs appears to consistently bring up certain less-known aspects of these APIs. This section is intended to alert implementers to such aspects.

5.1. Berkeley Socket Interface

5.1.1. Responding from the right address

In order for a client to recognize a reply (response or acknowledgement) as coming from the endpoint to which the initiating packet was addressed, the source IPv6 address of the reply needs to match the destination address of the initiating packet.

Implementers that have previously written TCP-based applications are used to binding their server sockets to INADDR_ANY. Any TCP connection received over such a socket is then more specifically bound to the source address from which the TCP connection setup was received; no programmer action is needed for this.

For stateless UDP sockets, more manual work is required. Simply receiving a packet from a UDP socket bound to INADDR_ANY loses the information about the destination address; replying to it through the same socket will use the default address established by the kernel. Two strategies are available:

- o Only use sockets bound to a specific address (not INADDR_ANY). A system with multiple interfaces (or addresses) will thus need to bind multiple sockets and send replies back on the same socket the initiating packet was received on.

- o Use IPV6_RECVPKTINFO [RFC3542] to configure the socket, and mirror back the IPV6_PKTINFO information for the reply (see also Section 5.1.1.1).

5.1.1.1. Managing interfaces

For some applications, it may further be relevant what interface is chosen to send to an endpoint, beyond the kernel choosing one that has a routing table entry for the destination address. E.g., it may be natural to send out a response or acknowledgment on the same interface that the packet prompting it was received. The end of the introduction to section 6 of [RFC3542] describes a simple technique for this, where that RFC's API (IPV6_PKTINFO) is available. The same data structure can be used for indicating an interface to send a packet that is initiating an exchange. (Choosing that interface is too application-specific to be in scope for the present document.)

5.1.2. Handling ICMP errors

Sockets that use the connect and send functions usually receive ICMP errors in the form of error codes, sockets that use sendto or sendmsg do not receive them at all.

Neither is sufficient to implement the guidance in Section 2.6, as the vetting of the message requires access to the CoAP headers in the ICMP error. The necessary information can be obtained by using the IPV6_RECVERR option.

5.2. Java

Java provides a wildcard address (0.0.0.0) to bind a socket to all network interface. This is useful when a server is supposed to listen on any available interface including the lookback address. For UDP, and hence CoAP this poses a problem, however, because the DatagramPacket class does not provide the information to which address it was sent. When replying through the wildcard socket, the JVM will pick the default address, which can break the correlation of

messages when the remote endpoint did not send the message to the default address. This is in particular precarious for IPv6 where it is common to have multiple IP addresses per network interface. Thus, it is recommended to bind to all adresses explicitly and manage the destination address of incoming messages within the CoAP implementation.

5.3. Multicast detection

Similar to the considerations above, Section 8 of [RFC7252] requires a node to detect whether a packet that it is going to reply to was sent to a unicast or to a multicast address. On most platforms, binding a UDP socket to a unicast address ensures that it only receives packets addressed to that address. Programmers relying on this property should ensure that it indeed applies to the platform they are using. If it does not, IPV6_PKTINFO may, again, help for Berkeley Socket Interfaces. For Java, explicit management of different sockets (in this case a MulticastSocket) is required.

5.4. DTLS

CoAPS implementations require access to the authenticated user/device prinicipal to realize access control for resources. How this information can be accessed heavily depends on the DTLS implementation used. Generic and portable CoAP implementations might want to provide an abstraction layer that can be used by application developers that implement resource handlers. It is recommended to keep the API of such an application layer close to popular HTTPS solutions that are available for the targeted platform, for instance, mod_ssl or the Java Servlet API.

6. CoAP on various transports

As specified in [RFC7252], CoAP is defined for two underlying transports: UDP and DTLS. These transports are relatively similar in terms of the properties they expose to their users. (The main difference, apart from the increased security, is that DTLS provides an abstraction of a connection, into which the endpoint abstraction is placed; in contrast, the UDP endpoint abstraction is based on four-tuples of IP addresses and ports.)

Recently, the need to carry CoAP over other transports [I-D.silverajan-core-coap-alternative-transports] has led to specifications such as CoAP over TLS or TCP or WebSockets[RFC8323], or even over non-IP transports such as SMS [I-D.becker-core-coap-sms-gprs]. This section discusses considerations that arise when handling these different transports in an implementation.

6.1. CoAP over reliable transports

To cope with transports without reliable delivery (such as UDP and DTLS), CoAP defines its own message layer, with acknowledgments, timers, and retransmission. When CoAP is run over a transport that provides its own reliability (such as TCP or TLS), running this machinery would be redundant. Worse, keeping the machinery in place is likely to lead to interoperability problems as it is unlikely to be tested as well as on unreliable transports. Therefore, [I-D.silverajan-core-coap-alternative-transports] was defined by removing the message layer from CoAP and just running the request/response layer directly on top of the reliable transport. This also leads to a reduced (from the UDP/DTLS 4-byte header) header format.

Conversely, where reliable transports provide a byte stream abstraction, some form of message delimiting had to be added, which now needs to be handled in the CoAP implementation. The use of reliable transports may reduce the disincentive for using messages larger than optimal link layer packet sizes. Where different message sizes are chosen by an application for reliable and for unreliable transports, this can pose additional challenges for translators (Section 6.2).

Where existing CoAP APIs expose details of the the message layer (e.g., CON vs. NON, or assigning application layer semantics to ACKs), using a reliable transport may require additional adjustments.

6.2. Translating between transports

One obvious way to convey CoAP exchanges between different transports is to run a CoAP proxy that supports both transports. The usual considerations for proxies apply. Section 6.2.1 discusses some additional considerations.

Where not much of the functionality of CoAP proxies (such as caching) is required, a simpler 1:1 translation may be possible, as discussed in Section 6.2.2.

6.2.1. Transport translation by proxies

(TBD. In particular, point out the obvious: fan-in/fan-out means that separate message ID and token spaces need to be maintained at the ends of the proxy.)

One more CoAP specific function of a transport translator proxy may be to convert between different block sizes, e.g. between a TCP connection that can tolerate large blocks and UDP over a constrained node network.

6.2.2. One-to-one Transport translation

A translator with reduced requirements for state maintenance can be constructed when no fan-in or fan-out is required, and when the namespace lifetimes of the two sides can be made to coincide. For this one-to-one translation, there is no need to manage message-ID and Token value spaces for both sides separately. So, a simple UDP-to-UDP one-to-one translator could simply copy the messages (among other applications, this might be useful for translation between IPv4 and IPv6 spaces). Similarly, a DTLS-to-TCP translator could be built that executes the message layer (deduplication, retransmission) on the DTLS side, and repackages the CoAP header (add/remove the length information, and remove/add the message ID and message type) between the DTLS and the TCP side.

By definition, such a simple one-to-one translator needs to shut down the connection on one side when the connection on the other side terminates. However, a UDP-to-TCP one-to-one translator cannot simply shut down the UDP endpoint when the TCP endpoint vanishes because the TCP connection closes, so some additional management of state will be necessary.

7. IANA considerations

This document has no actions for IANA.

8. Security considerations

TBD

9. Acknowledgements

Esko Dijk contributed the sequential MID optimization. Xuan He provided help creating and improved the state machine charts. Christian Amsuess provided input on forwarding block messages by proxies and usage of the Request-Tag option.

10. References

10.1. Normative References

[I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-03 (work in progress), February 2018.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, DOI 10.17487/RFC6633, May 2012, <https://www.rfc-editor.org/info/rfc6633>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

10.2. Informative References

[Contiki] Dunkels, A., Groenvall, B., and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors", Proceedings of the First IEEE Workshop on Embedded Networked Sensors, November 2004.

[I-D.becker-core-coap-sms-gprs] Kuladinithi, K., Becker, M., Li, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-06 (work in progress), February 2017.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-04 (work in progress), March 2018.

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-02 (work in progress), June 2018.

[I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-11 (work in progress), March 2018.

[RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, DOI 10.17487/RFC3542, May 2003, <https://www.rfc-editor.org/info/rfc3542>.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010, <https://www.rfc-editor.org/info/rfc5927>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

[TinyOS] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Woo, A., Hill, J., Welsh, M., Brewer, E., and D. Culler, "TinyOS: An Operating System for Sensor Networks", Ambient intelligence, Springer (Berlin Heidelberg), ISBN 978-3-540-27139-0, 2005.

Authors' Addresses

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
CH-8092 Zurich
Switzerland

Email: kovatsch@inf.ethz.ch

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Email: bergmann@tzi.org

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 24, 2018

C. Gomez
Universitat Politecnica de Catalunya
M. Kovatsch
ETH Zurich
H. Tian
China Academy of Telecommunication Research
Z. Cao, Ed.
Huawei Technologies

October 21, 2017

Energy-Efficient Features of Internet of Things Protocols
draft-ietf-lwig-energy-efficient-08

Abstract

This document describes the challenges for energy-efficient protocol operation on constrained devices and the current practices used to overcome those challenges. It summarizes the main link-layer techniques used for energy-efficient networking, and it highlights the impact of such techniques on the upper layer protocols so that they can together achieve an energy efficient behavior. The document also provides an overview of energy-efficient mechanisms available at each layer of the IETF protocol suite specified for constrained node networks.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 24, 2018.

Table of Contents

1. Introduction 2
  1.1. Terminology 3
2. Overview 3
3. Medium Access Control and Radio Duty Cycling 5
  3.1. Radio Duty Cycling techniques 6
  3.2. Latency and buffering 7
  3.3. Throughput 7
  3.4. Radio interface tuning 8
  3.5. Packet bundling 8
  3.6. Power save services available in example low-power radios 8
    3.6.1. Power Save Services Provided by IEEE 802.11 8
    3.6.2. Power Save Services Provided by Bluetooth LE 9
    3.6.3. Power Save Services in IEEE 802.15.4 10
    3.6.4. Power Save Services in DECT ULE 12
4. IP Adaptation and Transport Layer 14
5. Routing Protocols 15
6. Application Layer 16
  6.1. Energy efficient features in CoAP 16
  6.2. Sleepy node support 16
  6.3. CoAP timers 17
  6.4. Data compression 17
7. Summary and Conclusions 18
8. Contributors 18
9. Acknowledgments 18
10. IANA Considerations 19
11. Security Considerations 19
12. References 19
  12.1. Normative References 19
  12.2. Informative References 21
Authors' Addresses 23

1. Introduction

Network systems for physical world monitoring contain many battery-powered or energy-harvesting devices. For example, in an environmental monitoring system, or a temperature and humidity

monitoring system, there may not be always-on and sustained power supplies for the potentially large number of constrained devices. In such deployment scenarios, it is necessary to optimize the energy consumption of the constrained devices. In this document we describe techniques that are in common use at Layer 2 and at Layer 3, and we indicate the need for higher-layer awareness of lower-layer features.

Many research efforts have studied this "energy efficiency" problem. Most of this research has focused on how to optimize the system's power consumption in certain deployment scenarios, or how an existing network function such as routing or security could be more energy-efficient. Only few efforts have focused on energy-efficient designs for IETF protocols and standardized network stacks for such constrained devices [I-D.kovatsch-lwig-class1-coap].

The IETF has developed a suite of Internet protocols suitable for such constrained devices, including IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [RFC6282],[RFC6775],[RFC4944], the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [RFC6550], and the Constrained Application Protocol (CoAP) [RFC7252]. This document tries to summarize the design considerations for making the IETF constrained protocol suite as energy-efficient as possible. While this document does not provide detailed and systematic solutions to the energy efficiency problem, it summarizes the design efforts and analyzes the design space of this problem. In particular, it provides an overview of the techniques used by the lower layers to save energy and how these may impact on the upper layers. Cross-layer interaction is therefore considered in this document from this specific point of view. Providing further design recommendations that go beyond the layered protocol architecture is out of the scope of this document.

After reviewing the energy-efficient designs of each layer, we summarize the document by presenting some overall conclusions. Though the lower layer communication optimization is the key part of energy efficient design, the protocol design at the upper layers is also important to make the device energy-efficient.

1.1. Terminology

Terms used in this document are defined in [RFC7228] [I-D.bormann-lwig-7228bis].

2. Overview

The IETF has developed protocols to enable end-to-end IP communication between constrained nodes and fully capable nodes. This work has expedited the evolution of the traditional Internet

protocol stack to a light-weight Internet protocol stack. As shown in Figure 1 below, the IETF has developed CoAP as the application layer and 6LoWPAN as the adaption layer to run IPv6 over IEEE 802.15.4 and Bluetooth Low-Energy, with the support of routing by RPL and efficient neighbor discovery by 6LoWPAN-ND. 6LoWPAN is currently being adapted by the 6lo working group to support IPv6 over various other technologies, such as ITU-T G.9959 [G9959], DECT ULE [TS102], MS/TP-BACnet [MSTP], and Near Field Communication (NFC) [NFC].

```
+-----+   +-----+    +-----+                +------+
|HTTP |   | FTP |    |SNMP |                | CoAP |
+-----+   +-----+    +-----+                +------+
      \    /           /                   /        \
     +-----+     +-----+              +-----+      +-----+
     | TCP |     | UDP |              | TCP |      | UDP |
     +-----+     +-----+       ===>   +-----+      +-----+
            \   /                           \        /
 +-----+  +------+  +-------+              +------+   +-----+
 | RTG |--| IPv6 |--|ICMP/ND|              | IPv6 |---| RTG |
 +-----+  +------+  +-------+              +------+   +-----+
             |                                 |
          +-------+                        +-------+  +----------+
          |MAC/PHY|                        |  6Lo  |--|6LoWPAN-ND|
          +-------+                        +-------+  +----------+
                                                |
                                           +-------+
                                           |MAC/PHY|
                                           +-------+
```

Figure 1: Traditional and Light-weight Internet Protocol Stack

There are numerous published studies reporting comprehensive measurements of wireless communication platforms [Powertrace]. As an example, below we list the energy consumption profile of the most common operations involved in communication on a prevalent sensor node platform. The measurement was based on the Tmote Sky with ContikiMAC [ContikiMAC] as the radio duty cycling algorithm. From this and many other measurement reports (e.g.[AN079]), we can see that the energy consumption of optimized transmission and reception are in the same order. For IEEE 802.15.4 and Ultra WideBand (UWB) links, transmitting may actually be even cheaper than receiving. It also shows that broadcast and non-synchronized communication transmissions are energy costly because they need to acquire the medium for a long time.

Activity	Energy (uJ)
Broadcast reception	178
Unicast reception	222
Broadcast transmission	1790
Non-synchronized unicast transmission	1090
Synchronized unicast transmission	120
Unicast TX to awake receiver	96
Listening (for 1000 ms)	63000

Figure 2: Power consumption of common operations involved in communication on the Tmote Sky with ContikiMAC

At the Physical layer, one approach that may allow reducing energy consumption of a device that uses a wireless interface is based on reducing the device transmit power level as long as the intended next hop(s) are still within range of the device. In some cases, if node A has to transmit a message to node B, a solution to reduce node A transmit power is to leverage an intermediate device, e.g. node C as a message forwarder. Let d be the distance between node A and node B. Assuming free-space propagation, where path loss is proportional to d^2, if node C is placed right in the middle of the path between A and B (that is, at a distance d/2 from both node A and node B), the minimum transmit power to be used by node A (and by node C) is reduced by a factor of 4. However, this solution requires additional devices, it requires a routing solution, and it also increases transmission delay between A and B.

3. Medium Access Control and Radio Duty Cycling

In networks, communication and power consumption are interdependent. The communication device is typically the most power-consuming component, but merely refraining from transmissions is not enough to achieve a low power consumption: the radio may consume as much power in listen mode as when actively transmitting. This illustrates the key problem known as idle listening, whereby the radio of a device may be in receive mode (ready to receive any message), even if no message is being transmitted to that device. Idle listening can consume a huge amount of energy unnecessarily. To reduce power consumption, the radio must be switched completely off -- duty-cycled

-- as much as possible. By applying duty-cycling, the lifetime of a device operating on a common button battery may be on the order of years, whereas otherwise the battery may be exhausted in a few days or even hours. Duty-cycling is a technique generally employed by devices that use the P1 strategy [RFC7228], which need to be able to communicate on a relatively frequent basis. Note that a more aggressive approach to save energy relies on the P0, Normally-off strategy, whereby devices sleep for very long periods and communicate infrequently, even though they spend energy in network reattachment procedures.

From the perspective of Medium Access Control (MAC) and Radio Duty Cycling (RDC), all upper layer protocols, such as routing, RESTful communication, adaptation, and management flows, are applications. Since the duty cycling algorithm is the key to energy-efficiency of the wireless medium, it synchronizes transmission and/or reception requests from the higher layers.

MAC and RDC are not in the scope of the IETF, yet lower layer designers and chipset manufacturers take great care to save energy. By knowing the behaviors of these lower layers, IETF engineers can design protocols that work well with them. The IETF protocols to be discussed in the following sections are the customers of the lower layers.

3.1. Radio Duty Cycling techniques

This subsection describes three main three RDC techniques. Note that more than one of these techniques may be available or can even be combined in a specific radio technology:

a) Channel sampling. In this solution, the radio interface of a device periodically monitors the channel for very short time intervals (i.e. with a low duty cycle) with the aim of detecting incoming transmissions. In order to make sure that a receiver can correctly receive a transmitted data unit, the sender may prepend a preamble of a duration at least the sampling period to the data unit to be sent. Another option for the sender is to repeatedly transmit the data unit, instead of sending a preamble before the data unit. Once a transmission is detected by a receiver, the receiver may stay awake until the complete reception of the data unit. Examples of radio technologies that use preamble sampling include ContikiMAC, the Coordinated Sampled Listening (CSL) mode of IEEE 802.15.4e, and the Frequently Listening (FL) mode of ITU-T G.9959 [G9959].

b) Scheduled transmissions. This approach allows a device to know the particular time at which it should be awake (during some time interval) in order to receive data. Otherwise, the device may remain

in sleep mode. The decision on the times at which communication is attempted relies on some form of negotation between the involved devices. Such negotiation may be performed per transmission or per session/connection. Bluetooth Low Energy (Bluetooth LE) is an example of a radio technology based on this mechanism.

c) Listen after send. This technique allows a node to remain in sleep mode by default, wake up and poll a sender (which must be ready to receive a poll message) for pending transmissions. After sending the poll message, the node remains in receive mode, ready for a potential incoming transmission. After a certain time interval, the node may go back to sleep. For example, the Receiver Initiated Transmission (RIT) mode of 802.15.4e, and the transmission of data between a coordinator and a device in IEEE 802.15.4-2003 use this technique.

3.2. Latency and buffering

The latency of a data unit transmission to a duty-cycled device is equal to or greater than the latency of transmitting to an always-on device. Therefore, duty-cycling leads to a trade-off between energy consumption and latency. Note that in addition to a latency increase, RDC may introduce latency variance, since the latency increase is a random variable (which is uniformly distributed if duty-cycling follows a periodical behavior).

On the other hand, due to the latency increase of duty-cycling, a sender waiting for a transmission opportunity may need to store subsequent outgoing packets in a buffer, increasing memory requirements and potentially incurring queuing waiting time that contributes to the packet's overall delay and increases the probability of buffer overflow, leading to losses.

3.3. Throughput

Although throughput is not typically a key concern in constrained node network applications, it is indeed important in some services in such networks, such as over-the-air software updates or when off-line sensors accumulate measurements that have to be quickly transferred when there is an opportunity for connectivity.

Since RDC introduces inactive intervals in energy-constrained devices, it reduces the throughput that can be achieved when communicating with such devices. There exists a trade-off between the achievable throughput and energy consumption.

3.4. Radio interface tuning

The parameters controlling the radio duty cycle have to be carefully tuned to achieve the intended application and/or network requirements. On the other hand, upper layers should take into account the expected latency and/or throughput behavior due to RDC. The next subsection provides details on key parameters controlling RDC mechanisms, and thus fundamental trade-offs, for various examples of relevant low-power radio technologies.

3.5. Packet bundling

Another technique that may be useful to increase communication energy efficiency is packet bundling. This technique, which is available in several radio interfaces (e.g. LTE and some 802.11 variants), allows to aggregate several small packets into a single large packet. Header and communication overhead is therefore reduced.

3.6. Power save services available in example low-power radios

This subsection presents power save services and techniques used in a few relevant examples of wireless low-power radios: IEEE 802.11, Bluetooth LE and IEEE 802.15.4. For a more detailed overview of each technology, the reader may refer to the literature or to the corresponding specifications.

3.6.1. Power Save Services Provided by IEEE 802.11

IEEE 802.11 defines the Power Save Mode (PSM) whereby a station may indicate to an Access Point (AP) that it will enter a sleep mode state. While the station is sleeping, the AP buffers any frames that should be sent to the sleeping station. The station wakes up every Listen Interval (which can be a multiple of the Beacon Interval) in order to receive beacons. The AP signals in the beacon whether there is data pending for the station or not. If there are not frames to be sent to the station, the latter may get back to sleep mode. Otherwise, the station may send a message requesting the transmission of the buffered data and stay awake in receive mode.

IEEE 802.11v [IEEE80211v] further defines mechanisms and services for power save of stations/nodes that include flexible multicast service (FMS), proxy ARP advertisement, extended sleep modes, and traffic filtering. Upper layer protocols knowledge of such capabilities provided by the lower layer enables better interworking.

These services include:

Proxy ARP: The Proxy ARP capability enables an Access Point (AP) to indicate that the non-AP station (STA) will not receive ARP frames. The Proxy ARP capability enables the non-AP STA to remain in power-save for longer periods of time.

Basic Service Set (BSS) Max Idle Period management enables an AP to indicate a time period during which the AP does not disassociate a STA due to non-receipt of frames from the STA. This supports improved STA power saving and AP resource management.

FMS: A service in which a non-access point (non-AP) STA can request a multicast delivery interval longer than the delivery traffic indication message (DTIM) interval for the purposes of lengthening the period of time a STA may be in a power save state.

Traffic Filtering Service (TFS): A service provided by an access point (AP) to a non-AP STA that can reduce the number of frames sent to the STA by dropping individually addressed frames that do not match traffic filters specified by the STA.

Using the above services provided by the lower layer, the constrained nodes can achieve either client initiated power save (via TFS) or network assisted power save (Proxy-ARP, BSS Max Idel Period and FMS).

Upper layer protocols should synchronize with the parameters such as FMS interval and BSS MAX Idle Period, so that the wireless transmissions are not triggered periodically.

3.6.2. Power Save Services Provided by Bluetooth LE

Bluetooth LE is a wireless low-power communications technology that is the hallmark component of the Bluetooth 4.0, 4.1, and 4.2 specifications [Bluetooth42]. BT-LE has been designed for the goal of ultra-low-power consumption. IPv6 can be run IPv6 over Bluetooth LE networks by using a 6LoWPAN variant adapted to BT-LE [RFC7668].

Bluetooth LE networks comprise a master and one or more slaves which are connected to the master. The Bluetooth LE master is assumed to be a relatively powerful device, whereas a slave is typically a constrained device (e.g. a class 1 device).

Medium access in Bluetooth LE is based on a Time Division Multiple Access (TDMA) scheme which is coordinated by the master. This device determines the start of connection events, in which communication between the master and a slave takes place. At the beginning of a connection event, the master sends a poll message, which may encapsulate data, to the slave. The latter must send a response, which may also contain data. The master and the slave may continue

exchanging data until the end of the connection event. The next opportunity for communication between the master and the slave will be in the next connection event scheduled for the slave.

The time between consecutive connection events is defined by the connInterval parameter, which may range between 7.5 ms and 4 s. The slave may remain in sleep mode since the end of its last connection event until the beginning of its next connection event. Therefore, Bluetooth LE is duty-cycled by design. Furthermore, after having replied to the master, a slave is not required to listen to the master (and thus may keep the radio in sleep mode) for connSlaveLatency consecutive connection events. connSlaveLatency is an integer parameter between 0 and 499 which should not cause link inactivity for more than connSupervisionTimeout time. The connSupervisionTimeout parameter is in the range between 100 ms and 32 s.

Upper layer protocols should take into account the medium access and duty-cycling behavior of Bluetooth LE. In particular, connInterval, connSlaveLatency and connSupervisionTimeout determine the time between two consecutive connection events for a given slave. The upper layer packet generation pattern and rate should be consistent with the settings of the aforementioned parameters (and vice versa). For example, assume connInterval=4 seconds, connSlaveLatency=7 and connSupervisionTimeout=32 seconds. With these settings, communication opportunities between a master and a slave will occur during a given interval every 32 seconds. Duration of the interval will depend on several factors, including number of connected slaves, amount of data to be transmitted, etc. In the worst case, only one data unit can be sent from master to slave and vice versa every 32 seconds.

3.6.3. Power Save Services in IEEE 802.15.4

IEEE 802.15.4 is a family of standard radio interfaces for low-rate, low-power wireless networking [fifteendotfour]. Since the publication of its first version in 2003, IEEE 802.15.4 has become the de-facto choice for a wide range of constrained node network application domains and has been a primary target technology of various IETF working groups such as 6LoWPAN [RFC6282], [RFC6775], [RFC4944] and 6TiSCH [I-D.ietf-6tisch-architecture]. IEEE 802.15.4 specifies a variety of related PHY and MAC layer functionalites.

IEEE 802.15.4 defines three roles called device, coordinator and Personal Area Network (PAN) coordinator. The device role is adequate for nodes that do not implement the complete IEEE 802.15.4 functionality, and is mainly targeted for constrained nodes with a limited energy source. The coordinator role includes synchronization

capabilities and is suitable for nodes that do not suffer severe constraints (e.g. a mains-powered node). The PAN coordinator is a special type of coordinator that acts as a principal controller in an IEEE 802.15.4 network.

IEEE 802.15.4 defines two main types of networks depending on their configuration: beacon-enabled and nonbeacon-enabled networks. In the first network type, coordinators periodically transmit beacons. The time between beacons is divided in three main parts: the Contention Access Period (CAP), the Contention Free Period (CFP) and an inactive period. In the first period, nodes use slotted Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) for data communication. In the second one, a TDMA scheme controls medium access. During the idle period, communication does not take place, thus the inactive period is a good opportunity for nodes to turn the radio off and save energy. The coordinator announces in each beacon the list of nodes for which data will be sent in the subsequent period. Therefore, devices may remain in sleep mode by default and wake up periodically to listen to the beacons sent by their coordinator. If a device wants to transmit data, or learns from a beacon that it is an intended destination, then it will exchange messages with the coordinator (and thus consume energy). An underlying assumption is that when a message is sent to a coordinator, the radio of the coordinator will be ready to receive the message.

The beacon interval and the duration of the beacon interval active portion (i.e. the CAP and the CFP), and thus the duty cycle, can be configured. The parameters that control these times are called macBeaconOrder and macSuperframeOrder, respectively. As an example, when IEEE 802.15.4 operates in the 2.4 GHz PHY, both times can be (independently) set to values in the range between 15.36 ms and 251.6 seconds.

In the beaconless mode, nodes use unslotted CSMA/CA for data transmission. The device may be in sleep mode by default and may activate its radio to either i) request to the coordinator whether there is pending data for the device, or ii) to transmit data to the coordinator. The wake-up pattern of the device, if any, is out of the scope of IEEE 802.15.4.

Communication between the two ends of an IEEE 802.15.4 link may also take place in a peer-to-peer configuration, whereby both link ends assume the same role. In this case, data transmission can happen at any moment. Nodes must have their radio in receive mode, and be ready to listen to the medium by default (which for battery-enabled nodes may lead to a quick battery depletion), or apply

synchronization techniques. The latter are out of the scope of IEEE 802.15.4.

The main MAC layer IEEE 802.15.4 amendment to date is IEEE 802.15.4e. This amendment includes various new MAC layer modes, some of which include mechanisms for low energy consumption. Among these, the Time-Slotted Channel Hopping (TSCH) is an outstanding mode which offers robust features for industrial environments, among others. In order to provide the functionality needed to enable IPv6 over TSCH, the 6TiSCH working group was created. TSCH is based on a TDMA schedule whereby a set of time slots are used for frame transmission and reception, and other time slots are unscheduled. The latter time slots may be used by a dynamic scheduling mechanism, otherwise nodes may keep the radio off during the unscheduled time slots, thus saving energy. The minimal schedule configuration specified in [I-D.ietf-6tisch-minimal] comprises 101 time slots; 95 of these time slots are unscheduled and the time slot duration is 15 ms.

The previously mentioned CSL and RIT are also 802.15.4e modes designed for low energy.

3.6.4. Power Save Services in DECT ULE

DECT Ultra Low Energy (DECT ULE) is a wireless technology building on the key fundamentals of traditional DECT / CAT-iq [EN300] but with specific changes to significantly reduce the power consumption at the expense of data throughput [TS102]. DECT ULE devices typically operate on special power optimized silicon, but can connect to a DECT Gateway supporting traditional DECT / CAT-iq for cordless telephony and data as well as the DECT ULE extensions. IPv6 can be run over DECT ULE by using a 6LoWPAN variant [I-D.ietf-6lo-dect-ule].

DECT defines two major roles: the Portable Part (PP) is the power constrained device, while the Fixed Part (FP) is the Gateway or base station in a star topology. DECT operates in license free and reserved frequency bands based on TDMA/FDMA and TDD using dynamic channel allocation for interference avoidance. It provides good indoor (~50 m) and outdoor (~300 m) coverage. It uses a frame length of 10 ms divided into 24 timeslots, and it supports connection oriented, packet data and connection-less services.

The FP usually transmits a so-called dummy bearer (beacon) that is used to broadcast synchronization, system and paging information. The slot/carrier position of this dummy bearer can automatically be reallocated in order to avoid mutual interference with other DECT signals.

At the MAC level DECT ULE communications between FP and PP are initiated by the PP. A FP can initiate communication indirectly by sending paging signal to a PP. The PP determines the timeslot and frequency on which the communication between FP and PP takes place. The PP verifies the radio timeslot/frequency position is unoccupied before it initiates its transmitter. An access-request message, which usually carries data, is sent to the FP. The FP sends a confirm message, which also may carry data. More data can be sent in subsequent frames. A MAC level automatic retransmission scheme significantly improves data transfer reliability. A segmentation and reassembly scheme supports transfer of larger higher layer SDUs and provides data integrity check. The DECT ULE packet data service ensures data integrity, proper sequencing, duplicate protection, but not guaranteed delivery. Higher layers protocols have to take this into consideration.

The FP may send paging information to PPs to trigger connection setup and indicate the required service type. The interval between paging information to a specific PP can be defined in range 10 ms to 327 seconds. The PP may enter sleep mode to save power. The listening interval is defined by the PP application. For short sleep intervals (below ~10 seconds) the PP may be able to retain synchronization to the FP dummy bearer and only turn on the receiver during the expected timeslot. For longer sleep intervals the PP can’t keep synchronization and has to search for and resynchronize to the FP dummybearer. Hence, longer sleep interval reduces the average energy consumption, but adds a energy consumption penalty for acquiring synchronization to the FP dummy bearer. The PP can obtain all information to determine paging and acquire synchronization information in a single reception of one full timeslot.

Packet data latency is normally 30 ms for short packets (below or equal to 32 octets), however if retry and back-off scenarios occur, the latency is increased. The latency can actually be reduced to about 10 ms by doing energy consuming RSSI scanning in advance. In the direction from FP to PP the latency is usually increased by the used paging interval and the sleep interval. The MAC layer can piggyback commands to improve efficiency (reduce latency) of higher layer protocols. Such commands can instruct the PP to initiate a new packet transfer in N frames without the need for resynchronization and listening to paging or instruct the PP to stay in a higher duty cycle paging detection mode.

The DECT ULE technology allows per PP configuration of paging interval, MTU size, reassembly window size and higher layer service negotiation and protocol.

4. IP Adaptation and Transport Layer

6LoWPAN provides an adaptation layer designed to support IPv6 over IEEE 802.15.4. 6LoWPAN affects the energy-efficiency problem in three aspects, as follows.

First, 6LoWPAN provides one fragmentation and reassembly mechanism which is aimed at solving the packet size issue in IPv6 and could also affect energy-efficiency. IPv6 requires that every link in the internet have an MTU of 1280 octets or greater. On any link that cannot convey a 1280-octet packet in one piece, link-specific fragmentation and reassembly must be provided at a layer below IPv6 [RFC2460]. 6LoWPAN provides fragmentation and reassembly below the IP layer to solve the problem. One of the benefits from placing fragmentation at a lower layer such as the 6LoWPAN layer is that it can avoid the presence of more IP headers, because fragmentation at the IP layer will produce more IP packets, each one carrying its own IP header. However, performance can be severely affected if, after IP layer fragmentation, then 6LoWPAN fragmentation happens as well (e.g. when the upper layer is not aware of the existence of the fragmentation at the 6LoWPAN layer). One solution is to require higher layers awareness of lower layer features and generate small enough packets to avoid fragmentation. In this regard, the Block option in CoAP can be useful when CoAP is used at the application layer [RFC 7959].

Secondly, 6LoWPAN swaps computing with communication. 6LoWPAN applies compression of the IPv6 header. Subject to the packet size limit of IEEE 802.15.4, 40 octets long IPv6 header and 8 octets or 20 octets long UDP and TCP header will consume even more packet space than the data itself. 6LoWPAN provides IPv6 and UDP header compression at the adaptation layer. Therefore, a lower amount of data will be handled by the lower layers, whereas both the sender and receiver will spend more computing power on the compression and decompression of the packets over the air. Compression can also be performed at higher layers (see Section 6.4).

Finally, the 6LoWPAN working group developed the energy-efficient Neighbor Discovery called 6LoWPAN-ND, which is an energy efficient replacement of the IPv6 ND in constrained environments. IPv6 Neighbor Discovery was not designed for non-transitive wireless links, as its heavy use of multicast makes it inefficient and sometimes impractical in a low-power and lossy network. 6LoWPAN-ND describes simple optimizations to IPv6 Neighbor Discovery, its addressing mechanisms, and duplicate address detection for Low-power Wireless Personal Area Networks and similar networks. However, 6LoWPAN ND does not modify Neighbor Unreachability Detection (NUD) timeouts, which are very short (by default three transmissions spaced

one second apart). NUD timeout settings should be tuned taking into account the latency that may be introduced by duty-cycled mechanisms at the link layer, or alternative, less impatient NUD algorithms should be considered [I-D.ietf-6man-impatient-nud].

IPv6 underlies the higher layer protocols, including both TCP/UDP transport and applications. By design, the higher-layer protocols do not typically have specific information about the lower layers, and thus cannot solve the energy-efficiency problem.

The network stack can be designed to save computing power. For example the Contiki implementation has multiple cross layer optimizations for buffers and energy management, e.g., the computing and validation of UDP/TCP checksums without the need of reading IP headers from a different layer. These optimizations are software implementation techniques, and out of the scope of IETF and the LWIG working group.

5. Routing Protocols

RPL [RFC6550] is a routing protocol designed by the IETF for constrained environments. RPL exchanges messages periodically and keeps routing states for each destination. RPL is optimized for the many-to-one communication pattern, where network nodes primarily send data towards the border router, but has provisions for any-to-any routing as well.

The authors of the Powertrace tool [Powertrace] studied the power profile of RPL. Their analysis divides the routing protocol into control and data traffic. The control plane carries ICMP messages to establish and maintain the routing states. The data plane carries any application that uses RPL for routing packets. The study has shown that the power consumption of the control traffic goes down over time in a relatively stable network. The study also reflects that the routing protocol should keep the control traffic as low as possible to make it energy-friendly. The amount of RPL control traffic can be tuned by setting the Trickle [RFC6206] algorithm parameters (i.e. Imin, Imax and k) to appropriate values. However, there exists a trade-off between energy consumption and other performance parameters such as network convergence time and robustness.

RFC 6551 [RFC6551] defines routing metrics and constraints to be used by RPL in route computation. Among others, RFC 6551 specifies a Node Energy object that allows to provide information related to node energy, such as the energy source type or the estimated percentage of remaining energy. Appropriate use of energy-based routing metrics

may help to balance energy consumption of network nodes, minimize network partitioning and increase network lifetime.

6. Application Layer

6.1. Energy efficient features in CoAP

CoAP [RFC7252] is designed as a RESTful application protocol, connecting the services of smart devices to the World Wide Web. CoAP is not a chatty protocol. It provides basic communication services such as service discovery and GET/POST/PUT/DELETE methods with a binary header.

Energy efficiency is part of the CoAP protocol design. CoAP uses a fixed-length binary header of only four bytes that may be followed by binary options. To reduce regular and frequent queries of the resources, CoAP provides an observe mode, in which the requester registers its interest of a certain resource and the responder will report the value whenever it was updated. This reduces the request response round trips while keeping information exchange a ubiquitous service; an energy-constrained server can remain in sleep mode during the period between observe notification transmissions.

Furthermore, [RFC7252] defines CoAP proxies which can cache resource representations previously provided by sleepy CoAP servers. The proxies themselves may respond to client requests if the corresponding server is sleeping and the resource representation is recent enough. Otherwise, a proxy may attempt to obtain the resource from the sleepy server.

CoAP proxy and cache functionality may also be used to perform data aggregation. This technique allows a node to receive data messages (e.g. carrying sensor readings) from other nodes in the network, perform an operation based on the content in those messages, and transmit the result of the operation. Such operation may simply be intended to use one packet to carry the readings transported in several packets (which reduces header and transmission overhead), or it may be a more sophisticated operation, possibly based on mathematical, logical or filtering principles (which reduces the payload size to be transmitted).

6.2. Sleepy node support

Beyond these features of CoAP, there have been a number of proposals to further support sleepy nodes at the application layer by leveraging CoAP mechanisms. A good summary of such proposals can be found in [I-D.rahman-core-sleepy-nodes-do-we-need], while an example application (in the context of illustrating several security

mechanisms) in a scenario with sleepy devices has been described [I-D.ietf-lwig-crypto-sensors]. Approaches to support sleepy nodes include exploiting the use of proxies, leveraging the Resource Directory [I-D.ietf-core-resource-directory] or signaling when a node is awake to the interested nodes. Recent work defines publish-subscribe and message queuing extensions to CoAP and the Resource Directory in order to support devices that spend most of their time in asleep [I-D.ietf-core-coap-pubsub]. Notably, this work has been adopted by the CoRE Working Group.

In addition to the work within the scope of CoAP to support sleepy nodes, other specifications define application layer functionality for the same purpose. The Lightweight Machine-to-Machine (LWM2M) specification from the Open Mobile Alliance (OMA) defines a Queue Mode whereby an LWM2M Server queues requests to an LWM2M Client until the latter (which may often stay in sleep mode) is online. LWM2M functionality operates on top of CoAP.

oneM2M defines a CoAP binding with an application layer mechanism for sleepy nodes [oneM2M].

6.3. CoAP timers

CoAP offers mechanisms for reliable communication between two CoAP endpoints. A CoAP message may be signaled as a confirmable (CON) message, and an acknowledgment (ACK) is issued by the receiver if the CON message is correctly received. The sender starts a Retransmission TimeOut (RTO) for every CON message sent. The initial RTO value is chosen randomly between 2 and 3 s. If an RTO expires, the new RTO value is doubled (unless a limit on the number of retransmissions has been reached). Since duty-cycling at the link layer may lead to long latency (i.e. even greater than the initial RTO value), CoAP RTO parameters should be tuned accordingly in order to avoid spurious RTOs which would unnecessarily waste node energy and other resources. On the other hand, note that CoAP can also run on top of TCP [I-D.ietf-core-coap-tcp-tls]. In that case, similar guidance applies to TCP timers, albeit with greater motivation to carefully configure TCP RTO parameters, since [RFC6298] reduced the default initial TCP RTO to 1 second, which may interact more negatively with duty-cycled links than default CoAP RTO values.

6.4. Data compression

Another method intended to reduce the size of the data units to be communicated in constrained-node networks is data compression, which allows to encode data using less bits than the original data representation. Data compression is more efficient at higher layers, particularly before encryption is used. In fact, encryption

mechanisms may generate an output that does not contain redundancy, making it almost impossible to reduce the data representation size. In CoAP, messages may be encrypted by using DTLS (or TLS when CoAP over TCP is used), which is the default mechanism for securing CoAP exchanges.

7. Summary and Conclusions

We summarize the key takeaways in this document:

a. Internet protocols designed by IETF can be considered as the customer of the lower layers (PHY, MAC, and Duty-cycling). To reduce power consumption, it is recommended that Layer 3 designs should operate based on awareness of lower-level parameters rather than treating the lower layer as a black box (Sections 4, 5 and 6).

b. It is always useful to compress the protocol headers in order to reduce the transmission/reception power. This design principle has been employed by many protocols in 6Lo and CoRE working group (Sections 4 and 6).

c. Broadcast and non-synchronized transmissions consume more than other TX/RX operations. If protocols must use these ways to collect information, reduction of their usage by aggregating similar messages together will be helpful in saving power (Sections 2 and 6.1).

d. Saving power by sleeping as much as possible is used widely (Section 3).

8. Contributors

Jens T. Petersen, RTX, contributed the section on power save services in DECT ULE.

9. Acknowledgments

Carles Gomez has been supported by the Spanish Government, FEDER and the ERDF through projects TEC2012-32531 and TEC2016-79988-P.

Authors would like to thank the review and feedback from a number of experts in this area: Carsten Bormann, Ari Keranen, Hannes Tschofenig, Dominique Barthel, Bernie Volz and Charlie Perkins.

The text of this document was improved based on IESG Document Editing session during IETF87. Thanks to Ted Lemon and Joel Jaeglli for initiating and facilitating this editing session.

10. IANA Considerations

This document has no IANA requests.

11. Security Considerations

This document discusses the energy efficient protocol design, and does not incur any changes or challenges on security issues besides what the protocol specifications have analyzed.

12. References

12.1. Normative References

[Bluetooth42]
Bluetooth Special Interest Group, "Bluetooth Core Specification Version 4.2", December 2014, <https://www.bluetooth.org/en-us/specification/adopted-specifications>.

[EN300] ETSI, "Digital Enhanced Cordless Telecommunications (DECT); Common Interface (CI)", March 2015, <https://www.etsi.org/deliver/etsi_en/300100_300199/30017501/02.06.01_60/en_30017501v020601p.pdf>.

[fifteendotfour]
IEEE Computer Society, "IEEE Std. 802.15.4-2015 IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)", 2015, <https://standards.ieee.org/findstds/standard/802.15.4-2015.html>.

[G9959] International Telecommunication Union, "Short range narrow-band digital radiocommunication transceivers - PHY and MAC layer specifications, ITU-T Recommendation G.9959", January 2015, <http://www.itu.int/rec/T-REC-G.9959>.

[IEEE80211v]
IEEE, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Amendment 8: IEEE 802.11 Wireless Network Management.", February 2012.

[MSTP] ANSI/ASHRAE, "Addenda: BACnet -- A Data Communication Protocol for Building Automation and Control Networks, ANSI/ASHRAE Addenda an, at, au, av, aw, ax, and az to ANSI/ASHRAE Standard 135-2012", July 2014, <https://www.ashrae.org/File%20Library/docLib/StdsAddenda/07-31-2014_135_2012_an_at_au_av_aw_ax_az_Final.pdf>.

[NFC] NFC Forum, "NFC Logical Link Control Protocol version 1.3, NFC Forum Technical Specification", March 2016.

[oneM2M] oneM2M, "oneM2M specifications", <http://www.onem2m.org/technical/published-documents>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC6206] Levis, P., Clausen, T., Hui, J., Gnawali, O., and J. Ko, "The Trickle Algorithm", RFC 6206, DOI 10.17487/RFC6206, March 2011, <https://www.rfc-editor.org/info/rfc6206>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <https://www.rfc-editor.org/info/rfc6551>.

[RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/info/rfc6775>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

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

[RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <https://www.rfc-editor.org/info/rfc7668>.

[TS102] ETSI, "Digital Enhanced Cordless Telecommunications (DECT); Ultra Low Energy (ULE); Machine to Machine Communications; Part 2: Home Automation Network (phase 2", March 2015, <https://www.etsi.org/deliver/etsi_ts/102900_102999/10293902/01.01.01_60/ts_10293902v010101p.pdf>.

12.2. Informative References

[AN079] Kim, C., "Measuring Power Consumption of CC2530 With Z-Stack", September 2012, <http://www.ti.com/lit/an/swra292/swra292.pdf>.

[ContikiMAC] Dunkels, A., "The ContikiMAC Radio Duty Cycling Protocol, SICS Technical Report T2011:13", December 2011, <https://www.mysciencework.com/publication/download/2f406d3c4cc1eda32a234f7a1ad2cc3b/7eb199e4f8b00857e21af2b7d2b31c0d>.

[I-D.bormann-lwig-7228bis] Bormann, C., Ersue, M., Keranen, A., and C. Gomez, "Terminology for Constrained-Node Networks", draft-bormann-lwig-7228bis-01 (work in progress), May 2017.

[I-D.ietf-6lo-dect-ule]
Mariager, P., Petersen, J., Shelby, Z., Logt, M., and D. Barthel, "Transmission of IPv6 Packets over DECT Ultra Low Energy", draft-ietf-6lo-dect-ule-09 (work in progress), December 2016.

[I-D.ietf-6man-impatient-nud]
Nordmark, E. and I. Gashinsky, "Neighbor Unreachability Detection is too impatient", draft-ietf-6man-impatient-nud-07 (work in progress), October 2013.

[I-D.ietf-6tisch-architecture]
Thubert, P., "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4", draft-ietf-6tisch-architecture-12 (work in progress), August 2017.

[I-D.ietf-6tisch-minimal]
Vilajosana, X., Pister, K., and T. Watteyne, "Minimal 6TiSCH Configuration", draft-ietf-6tisch-minimal-21 (work in progress), February 2017.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-02 (work in progress), July 2017.

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-09 (work in progress), May 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-11 (work in progress), July 2017.

[I-D.ietf-lwig-crypto-sensors]
Sethi, M., Arkko, J., Keranen, A., and H. Back, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks", draft-ietf-lwig-crypto-sensors-04 (work in progress), August 2017.

[I-D.kovatsch-lwig-class1-coap]
Kovatsch, M., "Implementing CoAP for Class 1 Devices", draft-kovatsch-lwig-class1-coap-00 (work in progress), October 2012.

[I-D.rahman-core-sleepy-nodes-do-we-need]
Rahman, A., "Sleepy Devices: Do we need to Support them in CORE?", draft-rahman-core-sleepy-nodes-do-we-need-01 (work in progress), February 2014.

[Powertrace]
Dunkels, Eriksson, Finne, and Tsiftes, "Powertrace: Network-level Power Profiling for Low-power Wireless Networks", March 2011, <https://core.ac.uk/download/pdf/11435067.pdf?repositoryId=362>.

Authors' Addresses

Carles Gomez
Universitat Politecnica de Catalunya
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
Zurich, CH-8092
Switzerland

Email: kovatsch@inf.ethz.ch

Hui Tian
China Academy of Telecommunication Research
Huayuanbeilu No.52
Beijing, Haidian District 100191
China

Email: tianhui@ritt.cn

Zhen Cao (editor)
Huawei Technologies
China

Email: zhencao.ietf@gmail.com