CoAP Implementation Guidance
draft-ietf-lwig-coap-01

Abstract

The Constrained Application Protocol (CoAP) is designed for resource-constrained nodes and networks, e.g., 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 http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on January 5, 2015.
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. Duplicate Rejection .................................... 6
  2.4. Token Usage ........................................... 7
    2.4.1. Tokens for Observe ............................... 7
    2.4.2. Tokens for Blockwise Transfers ................ 8
  2.5. Transmission States .................................... 8
    2.5.1. Request/Response Layer .......................... 9
    2.5.2. Message Layer .................................. 10
  2.6. Out-of-band Information ................................ 11
  2.7. Programming Model ...................................... 11
    2.7.1. Client ........................................ 12
    2.7.2. Server ....................................... 12
3. Optimizations ................................................ 13
  3.1. Message Buffers ....................................... 13
  3.2. Retransmissions ....................................... 14
  3.3. Observable Resources .................................. 14
  3.4. Blockwise Transfers ................................... 15
  3.5. Deduplication with Sequential MIDs ................... 15
4. Alternative Configurations .................................. 18
  4.1. Transmission Parameters .............................. 18
  4.2. CoAP over IPv4 ...................................... 19
5. IANA considerations ......................................... 19
6. Security considerations ...................................... 19
7. References .................................................. 19
  7.1. Normative References ................................. 19
  7.2. Informative References ............................... 20
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 feature, 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 [I-D.ietf-core-observe]). Note that implementations on constrained platforms often not even support the full MTU. Larger messages must then use blockwise transfers [I-D.ietf-core-block], 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 with a 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 [I-D.ietf-core-observe] 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 in server role and can benefit from caching.
2.2. Message Processing

Apart from the required buffers, message processing is symmetric for clients and servers. First the 4-byte base header has to be parsed and thereby checked if it is a CoAP message. Since the encoding is very dense, only a wrong Version or a datagram size smaller than four bytes identify non-CoAP datagrams. These need to be silently ignored. Other message format errors, such as an incomplete datagram length 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). Next the Token is read based on the TKL field. For the following header options, there are two alternatives: Either process the header 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. In simpler cases, conditionals can preselect one of the repeated option values.

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).

The byte strings (e.g., the URI) are usually not required when generating the response. And since all important values were copied, this alternative facilitates using the buffer for incoming messages also for the assembly of outgoing messages – which can be the shared IP buffer provided by the OS.

Setting options for outgoing messages is also easier with an internal data structure. Application developers can set options independent from the option number, whose order is required for the delta encoding. The CoAP encoding is then 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 header options or restrictions for developers to set options in their correct order.

2.3. Duplicate Rejection

If CoAP is used directly on top of UDP (i.e., in NoSec mode), it needs to cope with the fact that the UDP datagram transport can reorder and duplicate messages. (In contrast to UDP, DTLS has its own duplicate detection.) CoAP has been designed with protocol functionality such that rejection of duplicate messages is always possible. It is at the discretion of the receiver if it actually wants to make use of this functionality. 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 unconstrained nodes that have throughput in the order of one hundred thousand requests per second (which might need 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 do the deduplication.

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. Duplicate rejection on the client side can be simplified by choosing clever Tokens and only filter based on this information (e.g., a list of Tokens currently in use or an obscured counter in the Token value).

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 even allows 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 [I-D.ietf-core-observe], a request will be answered with multiple notifications and it can become hard to determine the end of a Token lifetime. When establishing an Observe relationship, the Token is registered at the server. Hence, the client partially loses control of the used Token. A client can attempt 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 acknowledged; see [I-D.ietf-core-observe]
section 3.6). However, the client might never receive the ACK 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 not send CON notifications during that time. Alternative Observe lifetime models such as Stubbornness(tm) might also keep relationships alive for longer periods.

Thus, Observe requests should carefully chose the value (and 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. This is, however, critical for spoofing attacks in NoSec mode. 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.

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 1 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 1: 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 2, 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)                  |

+--------+                        RESPONSE--------+
|SERVING| / M_CMD(accept)             |
+--------+-----------------RR_EVT(rx)[is CON]-----+
```

Figure 2: CoAP Server Request/Response Layer FSM
2.5.2. Message Layer

Figure 3 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.

Figure 3: 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.bormann-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 [I-D.ietf-core-observe]. 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 [I-D.ietf-core-block] 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 [I-D.ietf-core-block], 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.

3.5. 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].

<table>
<thead>
<tr>
<th>MID base</th>
<th>K-bit bitfield</th>
<th>base time value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID_0</td>
<td>010010101001</td>
<td>t_0</td>
</tr>
<tr>
<td>MID_1</td>
<td>111101110111</td>
<td>t_1</td>
</tr>
<tr>
<td>... etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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_Lesire = 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.bormann-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.

5. IANA considerations

This document has no actions for IANA.

6. Security considerations

TBD

7. References

7.1. Normative References

[I-D.bormann-core-cocoa]

[I-D.ietf-core-block]
7.2. Informative References


Authors' Addresses
Energy Efficient Implementation of IETF Constrained Protocol Suite

draft-ietf-lwig-energy-efficient-02

Abstract

This document summarizes the problems and current practices of energy efficient protocol implementation on constrained devices, mostly about how to make the protocols within IETF scope behave energy friendly. This document also summarizes the impact of link layer protocol power saving behaviors to the upper layer protocols, so that they can coordinately make the system energy efficient.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 8, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

In many scenarios, the network systems comprise many battery-powered or energy-harvesting devices. For example, in an environmental monitoring system or a temperature and humidity monitoring system in a data center, there are no always-on and handy sustained power supplies for the potentially large number of constrained devices. In such deployment environments, it is necessary to optimize the energy
consumption of the entire system, including computing, application
layer behavior, and lower layer communication.

Significant research efforts have been spent on this "energy
efficiency" problem. Most of this research has focused on how to
optimize the system’s power consumption regarding a certain
deployment scenario or how could an existing network function such as
routing or security be more energy-efficient. Only few efforts were
spent 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 6LoWPAN (  
[RFC6282],[RFC6775],[RFC4944] ), RPL[RFC6550], and
CoAP[I-D.ietf-core-coap]. This document tries to summarize the
design considerations of making the IETF 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 a comprehensive overview of the
techniques used by the lower layers to save energy and how these may
impact on the upper layers.

After reviewing the energy-efficient design of each layer, an overall
conclusion is summarized. 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. Conventions used in this document

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

1.2. Terminology

The terminologies used in this document can be referred to [RFC7228].

2. Overview

The IETF has developed multiple protocols to enable end-to-end IP
communication between constrained nodes and fully capable nodes.
This work has witnessed 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, DECT ULE, MS/TP-BACnet and NFC.

![Diagram of network protocols](image)

**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 atom operations 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. [AN053]), we can see that the energy consumption of optimized transmission and reception may be in the same order. For IEEE 802.15.4 and UWB radios, transmitting may actually be even cheaper than receiving. Only for broadcast and non-synchronized communication transmissions become costly in terms of energy because they need to flood the medium for a long time.
### Figure 2: Power consumption of atom operations on the Tmote Sky with ContikiMAC

#### 3. MAC and Radio Duty Cycling

In low-power wireless networks, communication and power consumption are intertwined. The communication device is typically the most power-consuming component, but merely refraining from transmissions is not enough to attain a low power consumption: the radio may consume as much power in listen mode as when actively transmitting. This augments 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 consumes 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 in the order of years, whereas otherwise the battery may be exhausted in a few days or even hours. Duty-cycling is a technique generally exploited 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 MAC&RDC, all upper layer protocols, such as routing, RESTful communication, adaptation, and management flows, are all applications. Since the duty cycling algorithm is the key to energy-efficiency of the wireless medium, it synchronizes the transmission and/or reception request from the higher layer.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Energy (uJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast reception</td>
<td>178</td>
</tr>
<tr>
<td>Unicast reception</td>
<td>222</td>
</tr>
<tr>
<td>Broadcast transmission</td>
<td>1790</td>
</tr>
<tr>
<td>Non-synchronized unicast transmission</td>
<td>1090</td>
</tr>
<tr>
<td>Synchronized unicast transmission</td>
<td>120</td>
</tr>
<tr>
<td>Unicast TX to awake receiver</td>
<td>96</td>
</tr>
</tbody>
</table>
The MAC&RDC are not in the scope of the IETF, yet lower layer designers and chipset manufacturers take great care of the problem. For the IETF protocol designers, however, it is good to know the behaviors of lower layers so that the designed protocols can work perfectly with them.

Once again, the IETF protocols we are going to talk about in the following sections are the customers of the lower layers. If the different protocol layers want to get better service in a cooperative way, they should be considerate and understand each other.

3.1. Radio Duty Cycling techniques

This subsection describes the main three RDC techniques. Note that more than one of the presented 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.

b) Scheduled transmissions. This approach allows a device to know the instants in which it should be awake (during some time interval) in order to receive data units. Otherwise, the device may remain in sleep mode. The decision on the instants that will be used for communication is reached by means of some form of negotiation between the involved devices. Such negotiation may be performed per transmission or per session/connection. Bluetooth Low Energy 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 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 this kind of networks, such as over-the-air software updates or when off-line sensors accumulate measurements that have to be quickly transferred when there is a connectivity opportunity.

Since RDC introduces inactive intervals in energy-constrained devices, it reduces the throughput that can 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. 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 802.11v, Bluetooth Low Energy 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.

3.5.1. 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, traffic filtering. It would be useful if upper layer protocols knows such capabilities provided by the lower layer, so that they can coordinate with each other.

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) station (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 station (STA) that can reduce the number of frames sent to the non-AP STA by not forwarding individually addressed frames addressed to the non-AP STA that do not match traffic filters specified by the non-AP 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 Idle Period and FMS).

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

3.5.2. Power Save Services Provided by Bluetooth Low Energy

Bluetooth Low Energy (Bluetooth LE) is a wireless low-power communications technology that is the hallmark component of the Bluetooth 4.0 and Bluetooth 4.1 specifications [Bluetooth41]. BT-LE has been designed for the goal of ultra-low-power consumption. Currently, it is possible to run IPv6 over Bluetooth LE networks by using a 6LoWPAN variant adapted to BT-LE [I-D.ietf-6lowpan-btle].

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 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 nature. 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,
3.5.3. 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 PHY and MAC layer functionality.

IEEE 802.15.4 defines three roles called device, coordinator and 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 has mainly defined two 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 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 will thus consume energy. An underlying assumption is that when a message is sent to a coordinator, the radio of the latter 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 s.

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 has been recently 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, whereby 95 of these time slots are unscheduled and the time slot duration is 15 ms.

Other 802.15.4e modes, which are in fact designed for low energy, are the previously mentioned CSL and RIT.

4. IP Adaptation and Transport Layer

6LoWPAN is the adaption layer to run IPv6 over IEEE 802.15.4 MAC&PHY. It was born to fill the gap that the IPv6 layer does not support
fragmentation and assembly of <1280-byte packets while IEEE 802.15.4 only supports a MTU of 127 bytes.

IPv6 is the basis for the higher layer protocols, including both TCP/UDP transport and applications. So they are quite ignorant of the lower layers, and are almost neutral to the energy-efficiency problem.

What the network stack can optimize is to save the 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.

The 6LoWPAN contributes to the energy-efficiency problem in two ways. First of all, it swaps computing with communication. 6LoWPAN applies compression of the IPv6 header. This means less amount of data will be handled by the lower layer, but both the sender and receiver should spend more computing power on the compression and decompression of the packets over the air. Secondly, 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].

5. Routing Protocols

The routing protocol designed by the IETF for constrained environments is called RPL [RFC6550]. As a routing protocol, RPL has to exchange messages periodically and keep 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. It divides the routing protocol into control and data traffic. The control channel uses ICMP messages to establish and maintain the routing states. The data channel is any application that uses RPL for routing packets. The study has shown that the power consumption of the control traffic goes down over time and data traffic stays relatively constant. 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 algorithm parameters (i.e. Imin, Imax and k) to adequate 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] was 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.

The energy-efficient design is implicitly included in 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 roundtrip while keeping information exchange a ubiquitous service and, most importantly, it allows an energy-constrained server to 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.

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]. The different approaches 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. A more 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 a sleeping state [I-D.koster-core-coap-pubsub]. As of the writing, none of these proposals 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.

On the other hand, oneM2M defines a CoAP binding with an application layer mechanism for sleepy nodes.

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.
7. Summary

We find a summary section necessary although most IETF documents do not contain it. The points we would like to summarize are as follows.

a. All Internet protocols, which are in the scope of the IETF, are customers of the lower layers (PHY, MAC, and Duty-cycling). In order to get a better service, the designers of higher layers should know them better.

b. The IETF has developed multiple protocols for constrained networked devices. A lot of implicit energy efficient design principles have been used in these protocols. The latter should be fine-tuned to exploit the collaboration with the lower layer protocols. Layers should offer interfaces that can be exploited by other layers in order to optimize global protocol stack performance.

c. The power trace analysis of different protocol operations showed that for radio-duty-cycled networks broadcasts should be avoided. Saving unnecessary states maintenance is also an effective method to be energy-friendly.

8. Acknowledgments

Carles Gomez has been supported by Ministerio de Economia y Competitividad and FEDER through project TEC2012-32531.

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

The text of this document was improved based on IESG Document Editing session during IETF87. Thank Ted Lemon, Joel Jaeggli, and efforts to initiate this facilities.

9. IANA Considerations

This document has no IANA requests.

10. 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.
11. References

11.1. Normative References

[AN053] Selvig, B., "Measuring power consumption with CC2430 and Z-Stack", .


11.2. Informative References


Authors’ Addresses

Zhen Cao (Ed.)
Leibniz University of Hannover
P.R.China

Email: zhencao.ietf@gmail.com

Carles Gomez
Universitat Politecnica de Catalunya/i2CAT
C/Esteve Terradas, 7
Castelldefels  08860
Spain

Email: carlesgo@entel.upc.edu