

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 11, 2014

Y. Hong
ETRI
J. Youn
DONG-EUI Univ.
November 07, 2013

Sleep node control mechanism for Constrained networks
draft-hong-lwig-sleepmode-control-00

Abstract

This document describes the mechanism of sleep mode control. The router which manages a network can control IPv6 sleepy nodes and deliver the packets from/to exterior networks. The IPv6 router can keep the IPv6 sleepy node as sleep state and handle the packets from/to the IPv6 sleepy nodes. Also, this document describes the mechanism of sleep mode control using synchronization information.

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 May 11, 2014.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Conventions and Terminology	3
3. Related Work and Background	3
4. Sleep mode control mechanism	3
4.1. Sleep mode request/response	4
4.2. Sleep mode advertisement by Router	5
4.3. Sleep mode control by using synchronization information	7
5. Security Considerations	7
6. IANA Considerations	8
7. References	8
7.1. Normative References	8
7.2. Informative References	8
Authors' Addresses	9

1. Introduction

In order to minimize the use of the battery of devices which are not always connected to a power source, the devices become asleep (sleep mode) for a long time and wake up to communicate (active mode) for a short time. To keep the exact time of sleep mode/active mode of the devices, the network usually utilizes the synchronization information of a Base Station or an Access Point. Although the devices switchover between active mode and sleep mode, the protocol in PHY/MAC layer knows the transit time in advance through the synchronization information.

The goals for power management may be different by the conditions of device and environment. The general strategies for power management of various conditions are depicted as always-on, always-off, and low-power [I-D.arkko-lwig-cellular]. A constrained node, creating constrained node networks, may occasionally go into sleep mode according to strategies of using power for communication [I-D.ietf-lwig-terminology]. In [I-D.ietf-lwig-terminology], a device is divided into four classes according to energy limitation of a device. Here, the constrained nodes classed such as class E1 and E2 in classes of energy limitation may occasionally go into sleep mode. Thus, in constrained node networks, there can exist the end-to-end communications between a sleep mode node and a non-sleep mode node.

Recently, the power management issues arise in not only PHY/MAC layer but also above network layer. In network/transport/application layer, it is assumed that the network devices are always connected

and ready to communicate. But, as various smart object devices become connected to the Internet and power management to save battery of devices is required.

2. Conventions and Terminology

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].

3. Related Work and Background

Power saving in wireless networks is mainly accomplished in PHY/MAC layer. The basic idea of power saving in PHY/MAC layer is to minimize the time of transmission/receipt. In IEEE 802.11 WLAN, the feature of power saving is Power Save Mode (PSM) that is available for nodes existing in an infrastructure based IEEE 802.11 WLAN. PSM is based on a synchronous sleep scheduling policy, in which wireless nodes are able to alternate between an active mode and a sleep mode [PowerMgmt].

Recently, the consideration of power saving is moved to network layer, too. In 6lowpan, the Neighbor Discovery operations must consider the low-power wireless personal area networks such as IEEE 802.15.4. Because the usage of multicast signaling raises severe energy consumption, the Neighbor Discovery optimization for 6lowpan has limited the usage of multicast signaling [I-D.ietf-6lowpan-nd].

In application layer, the CoAP has two functions such as proxy and cache. The proxy function in the CoAP can cache and service requests for sleepy servers. Thus, a client sends a CoAP request to a proxy on behalf of an origin server of sleep mode and then it respond directly to the client through the proxy. Otherwise if the proxy has an invalid representation of the resource in its cache, the proxy has to attempt to get the valid resource from an origin server of sleep mode. The attempt may or may not be successful according to the state of the origin server [I-D.ietf-core-coap].

4. Sleep mode control mechanism

Figure 1 shows the environment where the sleep mode control mechanism is used. In the constrained network, one or more sleepy nodes are connected to the router which manages the constrained network and the router is connected to external network.

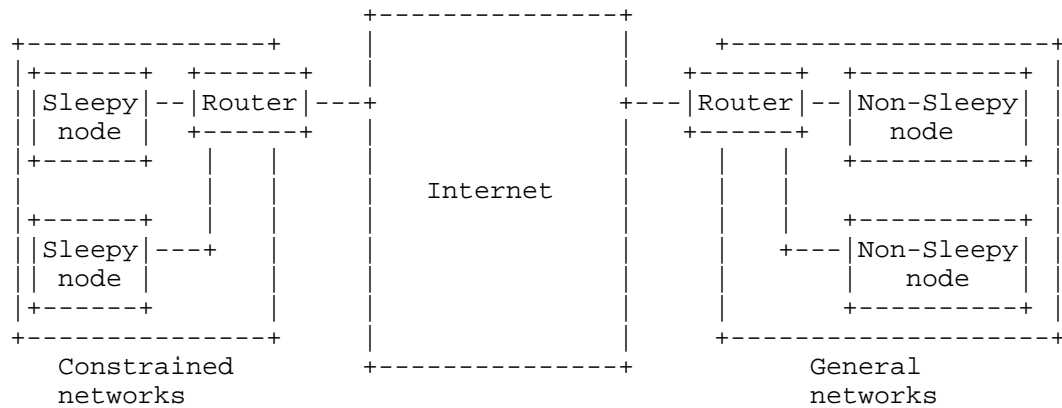
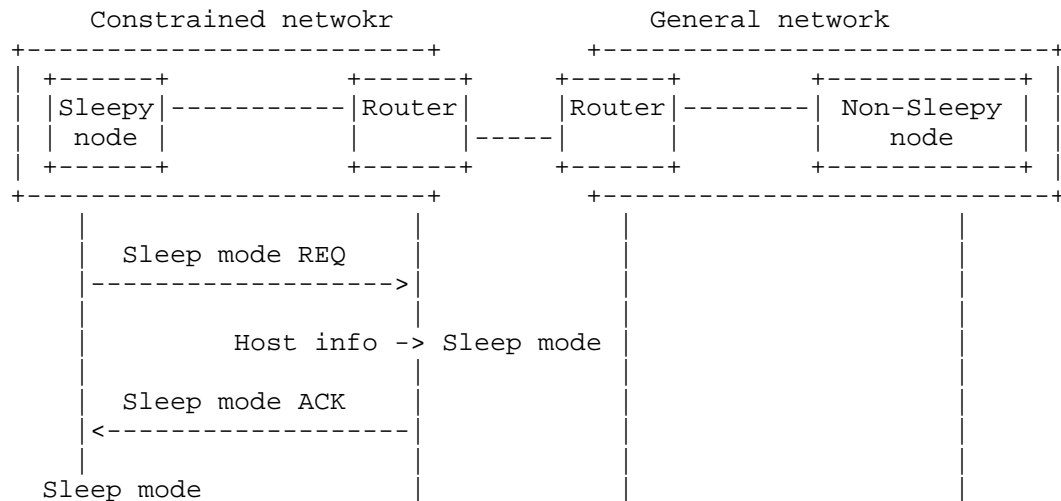


Figure 1: Environment where sleep mode control mechanism is used

4.1. Sleep mode request/response

If a sleepy host wants to enter to sleep mode, it sends a request message to the router. The router which receives the request message records the status of the sleepy node as sleep mode. And then, the router sends a response message to confirm the request. If the sleepy node receives the request message, it enters sleep mode. The router which manages the constrained network sends the status of the sleepy node to the corresponding nodes if the sleepy node is communicating. So, the corresponding nodes immediately know the status of the sleepy node. Figure 2 depicts this procedure.



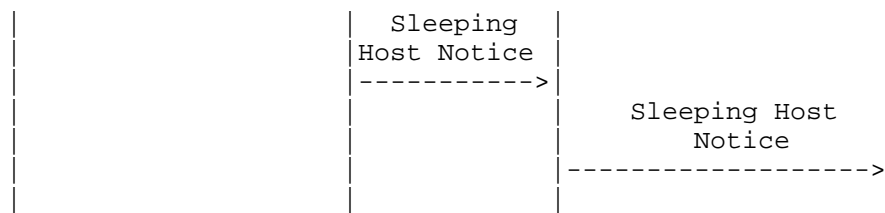


Figure 2: Procedure of sleep mode request/response

4.2. Sleep mode advertisement by Router

If some conditions are satisfied or events are triggered, a router in a constrained network sends a sleep mode advertisement message to sleepy nodes. The sleep mode advertisement message includes the start time and finish time of sleep mode. If sleepy nodes receive the advertisement message, they enter sleep mode. Some cases, the sleepy nodes may respond to the advertisement message. After the router changes the status of the sleepy node to sleep mode and delivers the status information of the sleepy nodes to corresponding nodes if there are corresponding nodes that are communicating to the sleepy nodes. So, the corresponding nodes immediately know the status of the sleepy nodes. Figure 3 depicts this procedure.

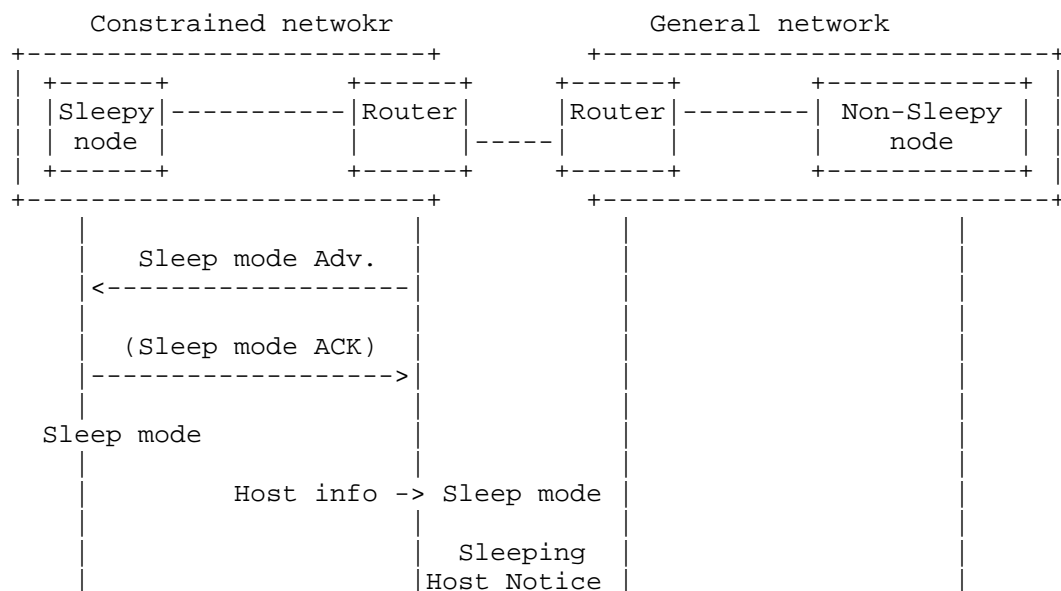




Figure 3: Procedure of sleep mode advertisement

If corresponding nodes sends packets to sleepy nodes, the router which manages a constrained network retrieves sleeping node list. If the target node is not included in the sleeping node list, the router handles the packets as normal process. If the target node is included in the sleeping node list, the router responds to the corresponding nodes and informs that the target node is in sleep mode. Figure 4 depicts this procedure.

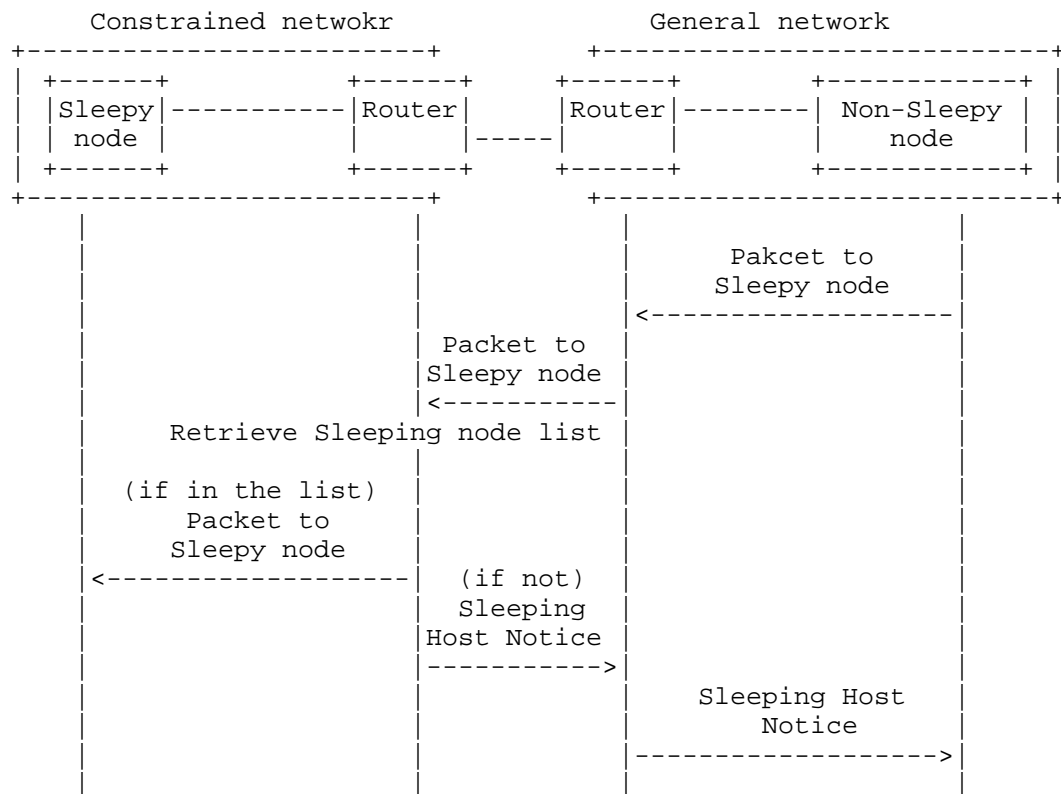


Figure 4: Procedure of the handling of packets from external network

4.3. Sleep mode control by using synchronization information

Generally, Base station or Access Point is used to synchronize the network entities. By using the synchronization information, the network keeps the exact time of sleep mode/active mode of the network entities. This synchronization information can be also used to provide sleep mode control. Figure 5 shows the case where synchronization information is used to provide sleep mode control. In Figure 5, the base station offers the synchronization information and this synchronization information is delivered to a router by using Layer 2 or Layer 3 message or cross-layer mechanism. After the router receives the synchronization information, it knows the sleep mode status of sleepy nodes. If there are corresponding nodes that are communicating to the sleepy nodes, the router delivers the status information of sleepy nodes to the corresponding nodes.

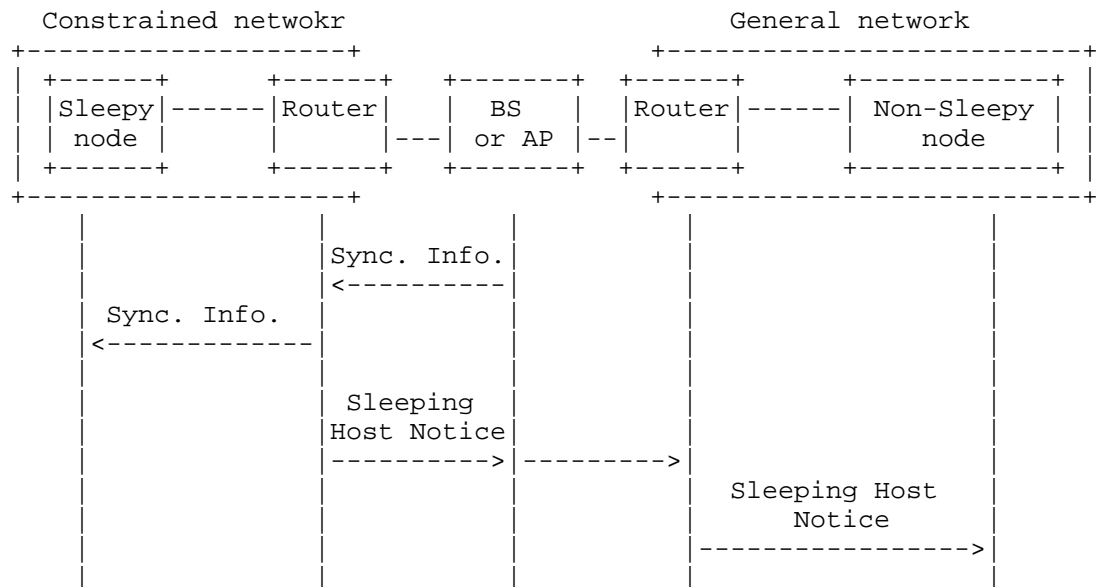


Figure 5: Procedure of sleep mode control by using synchronization information

5. Security Considerations

TBD.

6. IANA Considerations

TBD

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

7.2. Informative References

- [I-D.arkko-lwig-cellular]
Arkko, J., Eriksson, A., and A. Keranen, "Building Power-Efficient CoAP Devices for Cellular Networks", ID draft-arkko-lwig-cellular-00, February 2013.
- [I-D.ietf-6lowpan-nd]
Shelby, Z., Chakrabartiy, S., and E. Nordmark, "Neighbor Discovery Optimization for Low Power and Lossy Networks (6LoWPAN)", ID draft-ietf-6lowpan-nd-21, August 2012.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", ID draft-ietf-core-coap-18, June 2013.
- [I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", ID draft-ietf-lwig-terminology-05, July 2013.
- [PowerMgmt]
Klues, K., "Power Management in Wireless Networks", Report, for Advanced Topics in Networking: Wireless and Mobile Networking by R. Jain, Wash. Univ. in St. Louis, , 2006.

Authors' Addresses

Yong-Geun Hong
ETRI
218 Gajeong-ro Yuseung-Gu
Daejeon 305-700
Korea

Phone: +82 42 860 6557
Email: yghong@etri.re.kr

JooSang Youn
DONG-EUI Univ.
Busan
Korea

Phone: +82 51 890 1993
Email: joosang.youn@gmail.com

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: July 20, 2014

M. Kovatsch
ETH Zurich
O. Bergmann
Universitaet Bremen TZI
E. Dijk
Philips Research
X. He
Hitachi (China) R&D Corp.
C. Bormann, Ed.
Universitaet Bremen TZI
January 16, 2014

CoAP Implementation Guidance
draft-kovatsch-lwig-coap-02

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 (i.e., ~ 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. The guidelines for transmission state management and developer APIs can also help with the implementation of CoAP for less constrained nodes.

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 July 20, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Protocol Handling	3
2.1. Message Processing	4
2.2. Header Information	4
2.2.1. On-the-fly Processing	4
2.2.2. Internal Data Structure	5
2.3. Token Usage	5
2.4. Deduplication	5
2.5. Transmission States	6
2.5.1. Request/Response Layer	6
2.5.2. Message Layer	7
2.6. Out-of-band Information	8
3. Optimizations	8
3.1. Message Buffers	8
3.2. Retransmissions	9
3.3. Observable Resources	10
3.4. Blockwise Transfers	10
3.5. Deduplication with Sequential MIDs	11
4. Alternative Configurations	14
4.1. Transmission Parameters	14
4.2. CoAP over IPv4	14
5. References	14
5.1. Normative References	14
5.2. Informative References	14
Authors' Addresses	15

1. Introduction

The Constrained Application Protocol [I-D.ietf-core-coap] has been designed specifically for machine-to-machine communication in networks with very constrained nodes. Typical application scenarios

therefore include building automation 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 [RFC2616].

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 (such as battery consumption, dynamic memory needs, and static code size) on the other. The present document gives some guidance on possible strategies to solve this tradeoff for very constrained nodes (Class 1 in [I-D.ietf-lwig-terminology]). The main focus is on servers as this is deemed the predominant case where CoAP applications are faced with tight resource constraints.

Additional considerations for the implementation of CoAP on tiny sensors are given in [I-D.arkko-core-sleepy-sensors].

2. Protocol Handling

For constrained nodes of Class 1 or even Class 2, the most limiting factors for (wireless) network communication usually are memory size and battery lifetime. Most applications therefore try to minimize internal buffer space for both transmit and receive operations, and to maximize sleeping cycles.

In the programming styles supported by very simple operating systems, 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 observe notifications). Note that implementations on constrained platforms often not even support the full MTU. Larger messages must then use block-wise 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 plus an estimate of the maximum header size with a worst case option setting.

2.1. Message Processing

T.B.D.: Basic steps and correct error handling

2.2. Header Information

There are two alternatives to handle the header: 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 the compact encoding saves memory and fully reuses the buffer for incoming messages. The basic message header information should always be copied into an internal data structure, as Message ID and/or Token are required for request/response matching and generating the response. 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 can be added to a sparse list (e.g., a one-dimensional array) for fast retrieval.

Once the option list has been processed at least up to the highest option number that is supported by the application, any known

critical option and all elective options can be masked out 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 the remaining options also must be processed to add further critical options included in the original request.)

2.2.2. Internal Data Structure

Using an internal data structure for all parsed options has advantages when processing the values, as they are already in a variable of corresponding type and integers in host byte order. The incoming payload and byte strings of the header can be accessed directly in its IP buffer using pointers. 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. Thus, this alternative also facilitates the usage of the IP buffer for message assembly - all important values are copied from the shared incoming/outgoing buffer.

Setting options for outgoing messages is also easier with an internal data structure. Application developers can set options independent from the option number order required for the delta encoding. The CoAP encoding is then applied in a serialization step before sending. On-the-fly processing might require extensive memmove operations to insert new header options or needs to restrict developers to set options in order.

2.3. Token Usage

T.B.D.: Discuss separation of Token space for correct behavior

2.4. Deduplication

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. Hence, a receiver may have good reasons to decide not to do the duplicate rejection. 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.

Deduplication is heavy for Class 1 devices, as the number of peer addresses can be vast. Servers should be kept stateless, i.e., the REST API should be designed idempotent whenever possible. When this is not the case, the resource handler could perform an optimized deduplication by exploiting knowledge about the application. Another, server-wide strategy is to only keep track of non-idempotent requests.

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.

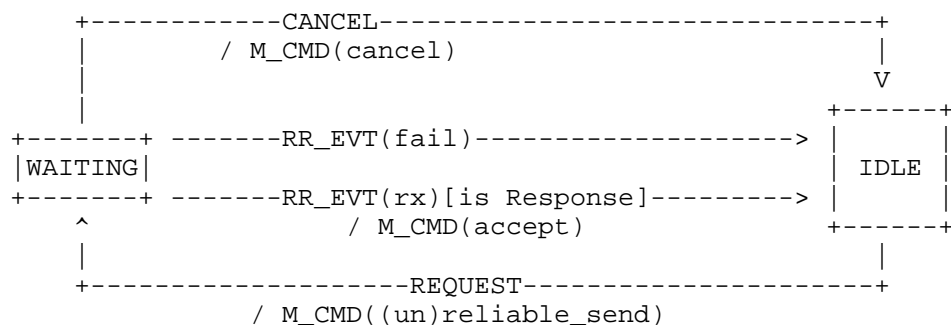


Figure 1: CoAP Client Request/Response Layer FSM

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.

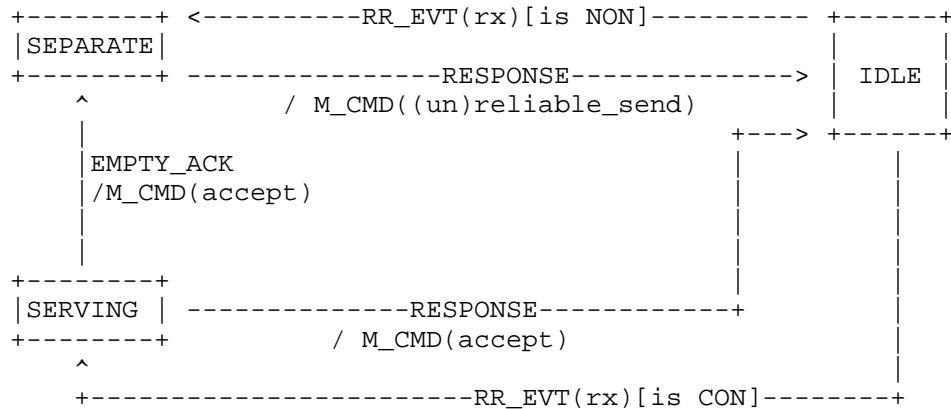


Figure 2: CoAP Server Request/Response Layer FSM

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`.

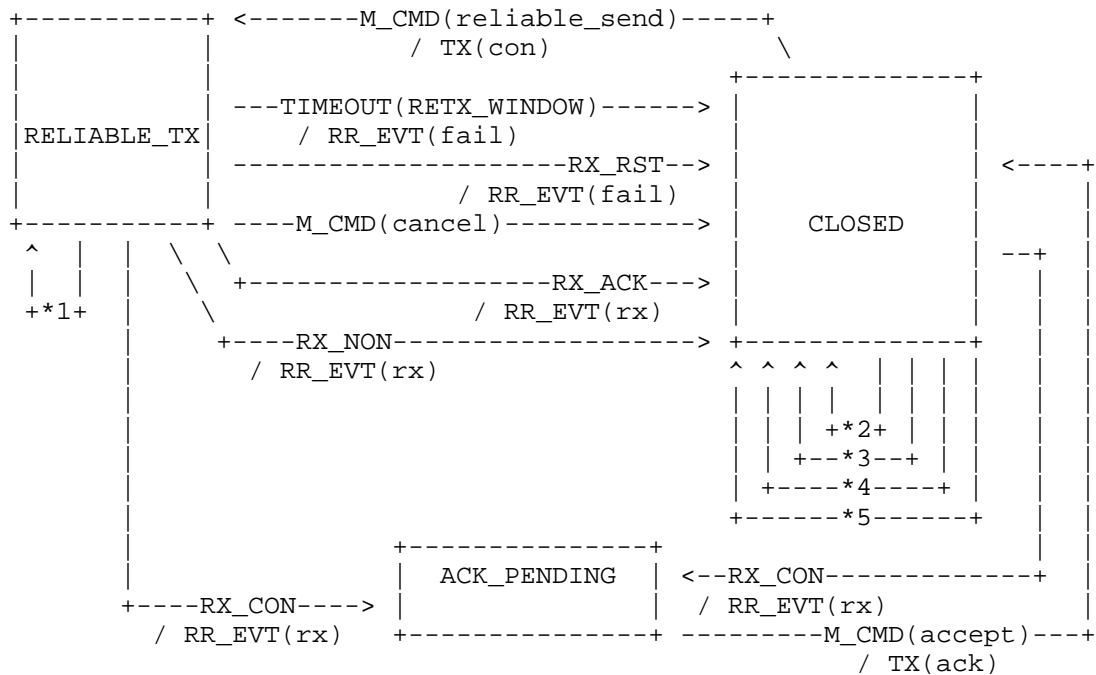


Figure 3: CoAP Message Layer FSM

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

T.B.D.: Discuss where out-of-band information can help

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

Block-wise 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 block-wise 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 block-wise transfer, 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. When transferring the CoRE Link Format from /.well-known/core for instance, 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 end-point by storing, for each remote CoAP end-point ("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_` ([I-D.ietf-core-coap] section "Reliable Messages") and the time packets can stay alive in the network.

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

- o many clients are making periodic, non-idempotent requests to a single CoAP server;
- o 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 using a Message ID variable [I-D.ietf-core-coap] 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 end-point 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 end-points, 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 end-points it communicates with, which partly breaks sequentiality from the receiving CoAP end-point'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 end-point 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 end-point 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 end-point 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 directly communication with the Internet, for instance because it is shielded by a proxy or a closed deployment, alternative transmission parameters can be used.

T.B.D.: Guidance for parameter tweaking

4.2. CoAP over IPv4

CoAP was designed for the properties of IPv6. In some cases, CoAP nodes will communicate over IPv4, though.

T.B.D.: Considerations for IPv4

5. References

5.1. Normative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

5.2. Informative References

- [I-D.arkko-core-sleepy-sensors]
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.

[I-D.ietf-core-block]

Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",
draft-ietf-core-block-14 (work in progress), October 2013.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-
core-observe-11 (work in progress), October 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keranen, "Terminology for
Constrained Node Networks", draft-ietf-lwig-terminology-06
(work in progress), December 2013.

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

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

Xuan He
Hitachi (China) R&D Corp.
301, Tower C North, Raycom, 2 Kexuyuan Nanlu
Beijing, 100190
P.R.China

Email: xhe@hitachi.cn

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

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

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: September 1, 2014

M. Kovatsch
ETH Zurich
O. Bergmann
Universitaet Bremen TZI
E. Dijk
Philips Research
X. He
Hitachi (China) R&D Corp.
C. Bormann, Ed.
Universitaet Bremen TZI
February 28, 2014

CoAP Implementation Guidance
draft-kovatsch-lwig-coap-03

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 (i.e., ~ 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 [I-D.ietf-core-coap], 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 September 1, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. 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. References	19
5.1. Normative References	19
5.2. Informative References	20
Authors' Addresses	20

1. Introduction

The Constrained Application Protocol [I-D.ietf-core-coap] 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 [RFC2616].

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, 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 in [I-D.ietf-lwig-terminology]). For this, it provides guidance on correct implementation of the CoAP specification [I-D.ietf-core-coap], 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). 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 inversed 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 MUST be silently ignored. All other message format errors, such as an incomplete datagram length or the usage of reserved values, MUST be rejected with a Reset (RST) message. 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 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 (i.e., about 16 GiB of RAM only 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 messages (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 code 7.31 is acknowledged; see [I-D.ietf-core-observe] section 3.6). However, the client might never receive the ACK due to a temporary network outages 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 never use the empty Token, but carefully chose the value. 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. Thus, dedicated ranges within the 8-byte Token space should be used when in NoSec mode. This also solves the 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.

Special care has to be taken 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 MUST 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 cancelation 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.

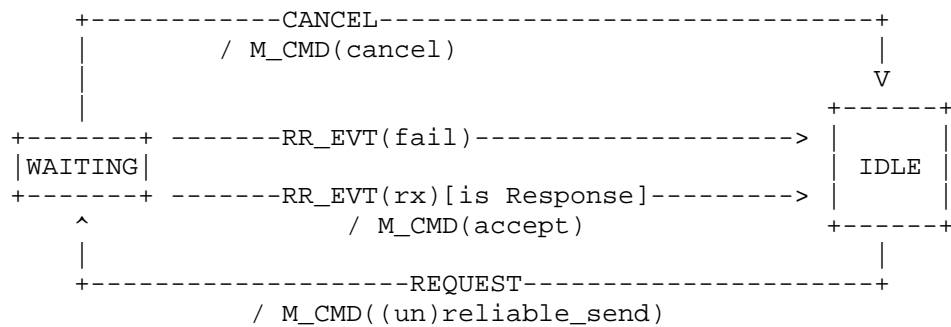


Figure 1: CoAP Client Request/Response Layer FSM

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.

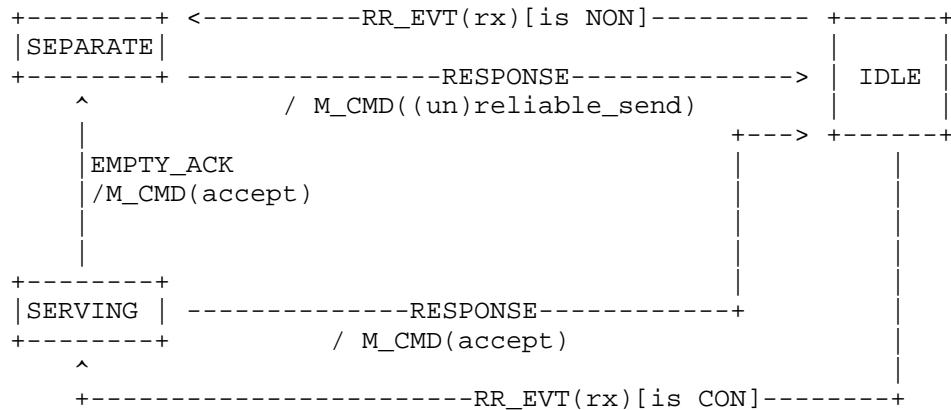


Figure 2: CoAP Server Request/Response Layer FSM

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.

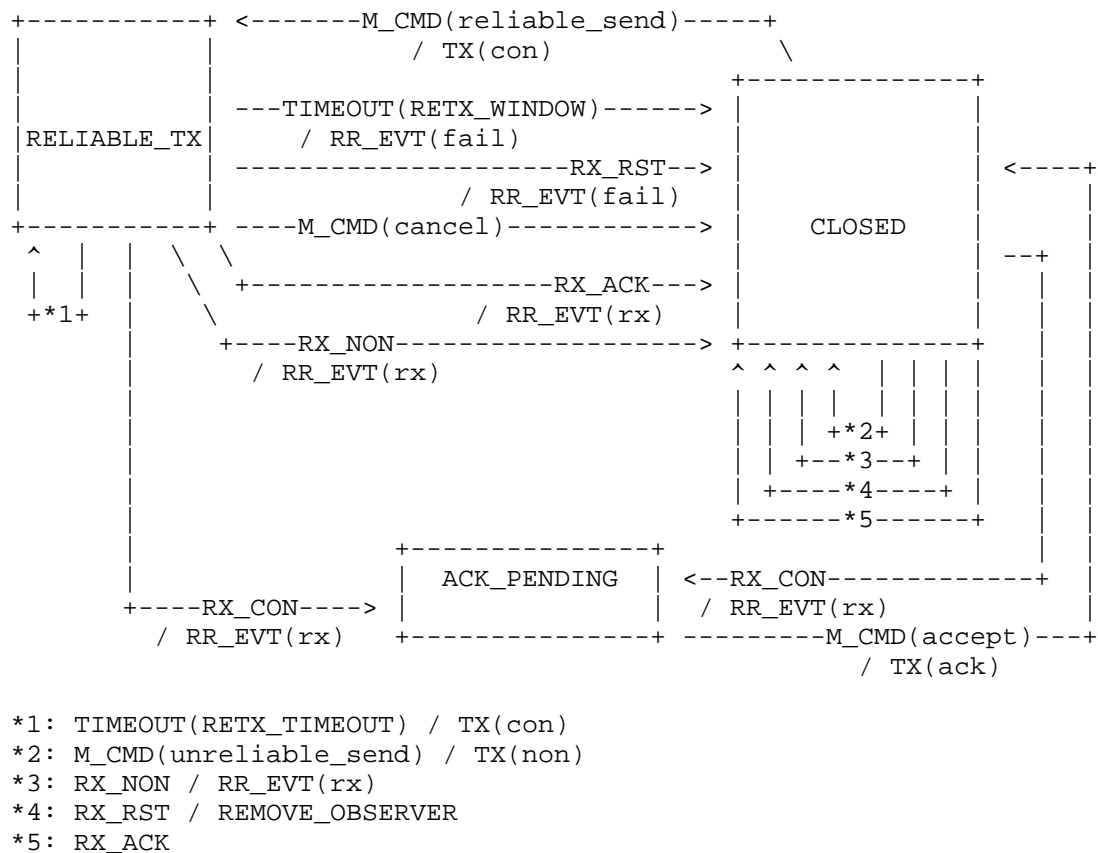


Figure 3: CoAP Message Layer FSM

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

They 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. They should cause the cancelation of ongoing CON transmissions and clearing 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 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. If an advanced mechanism is required to better utilize the network, [I-D.bormann-core-cocoa] should be implemented. Time Exceeded messages inform about possible routing loops or a too small initial Hop Limit value. This is out of scope for CoAP implementations, though.

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 have a minimalistic 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 blocks 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, 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. When transferring the CoRE Link Format from /.well-known/core for instance, 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_` ([I-D.ietf-core-coap] 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:

- o many clients are making periodic, non-idempotent requests to a single CoAP server;
- o 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 using a Message ID variable [I-D.ietf-core-coap] 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` ($0 \dots K-1$) 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 [I-D.ietf-core-coap] 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 [I-D.ietf-core-coap] 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 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 [I-D.ietf-core-coap] already provide guidance and implementation notes to handle the smaller minimal MTUs of IPv4.

5. References

5.1. Normative References

- [I-D.bormann-core-cocoa]
Bormann, C., "CoAP Simple Congestion Control/Advanced", draft-bormann-core-cocoa-01 (work in progress), February 2014.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-14 (work in progress), October 2013.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-12 (work in progress), February 2014.

- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.
- [RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, May 2012.

5.2. Informative References

- [Contiki] Dunkels, A., Groenvald, 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.arkko-core-sleepy-sensors] Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.
- [I-D.ietf-lwig-terminology] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-07 (work in progress), February 2014.
- [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

Esko Dijk
Philips Research

Email: esko.dijk@philips.com

Xuan He
Hitachi (China) R&D Corp.
301, Tower C North, Raycom, 2 Kexuyuan Nanlu
Beijing, 100190
P.R.China

Email: xhe@hitachi.cn

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

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

Light-Weight Implementation Guidance (lwig)
Internet-Draft
Intended status: Informational
Expires: April 24, 2019

D. Migault
Ericsson
T. Guggemos
LMU Munich
October 21, 2018

Minimal ESP
draft-mglt-lwig-minimal-esp-07

Abstract

This document describes a minimal implementation of the IP Encapsulation Security Payload (ESP) defined in RFC 4303. Its purpose is to enable implementation of ESP with a minimal set of options to remain compatible with ESP as described in RFC 4303. A minimal version of ESP is not intended to become a replacement of the RFC 4303 ESP, but instead to enable a limited implementation to interoperate with implementations of RFC 4303 ESP.

This document describes what is required from RFC 4303 ESP as well as various ways to optimize compliance with RFC 4303 ESP.

This document does not update or modify RFC 4303, but provides a compact description of how to implement the minimal version of the protocol. If this document and RFC 4303 conflicts then RFC 4303 is the authoritative description.

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, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Requirements notation

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].

2. Introduction

ESP [RFC4303] is part of the IPsec suite protocol [RFC4301]. IPsec is used to provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity) and limited traffic flow confidentiality.

Figure 1 describes an ESP Packet. Currently ESP is implemented in the kernel of major multi purpose Operating Systems (OS). The ESP and IPsec suite is usually implemented in a complete way to fit multiple purpose usage of these OS. However, completeness of the IPsec suite as well as multi purpose scope of these OS is often performed at the expense of resources, or a lack of performance. As a result, constraint devices are likely to have their own implementation of ESP optimized and adapted to their specificities. With the adoption of IPsec by IoT devices with minimal IKEv2 [RFC7815] and ESP Header Compression (EHC) with [I-D.mglt-ipsecme-diet-esp] or [I-D.mglt-ipsecme-ikev2-diet-esp-extension], it becomes crucial that ESP implementation designed for constraint devices remain interoperable with the standard ESP implementation to avoid a fragmented usage of ESP. This document describes the the minimal properties and ESP implementation needs to meet.

For each field of the ESP packet represented in Figure 1 this document provides recommendations and guidance for minimal implementations. The primary purpose of Minimal ESP is to remain

interoperable with other nodes implementing RFC 4303 ESP, while limiting the standard complexity of the implementation.

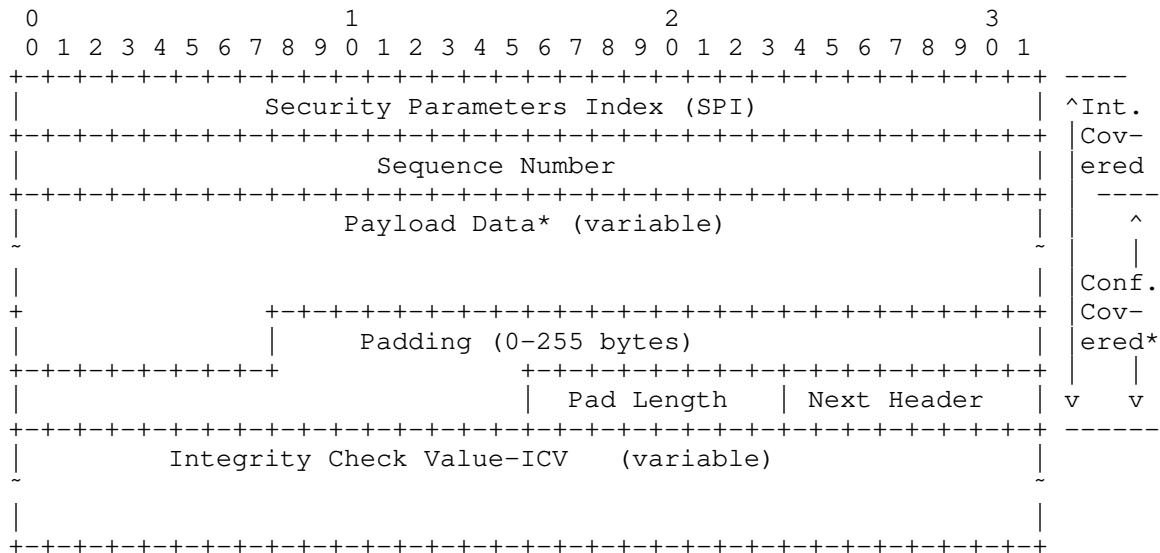


Figure 1: ESP Packet Description

3. Security Parameter Index (SPI) (32 bit)

According to the [RFC4303], the SPI is a mandatory 32 bits field and is not allowed to be removed.

The SPI has a local significance to index the Security Association (SA). From [RFC4301] section 4.1, nodes supporting only unicast communications can index their SA only using the SPI. On the other hand, nodes supporting multicast communications must also use the IP addresses and thus SA lookup needs to be performed using the longest match.

For nodes supporting only unicast communications, it is RECOMMENDED to index SA with the SPI only. Some other local constraints on the node may require a combination of the SPI as well as other parameters to index the SA.

It is RECOMMENDED to randomly generate the SPI indexing each inbound session. A random generation provides a stateless way to generate the SPIs, while keeping the probability of collision between SPIs relatively low. In case of collision, the SPI is simply re-generated.

However, for some constraint nodes, generating a random SPI may consume too much resource, in which case SPI can be generated using predictable functions or even a fix value. In fact, the SPI does not need to be random. Generating non random SPI MAY lead to privacy and security concerns. As a result, this alternative should be considered for devices that would be strongly impacted by the generation of a random SPI and after understanding the privacy and security impact of generating non random SPI.

When a constraint node uses fix value for SPIs, it imposes some limitations on the number of inbound SA. This limitation can be alleviated by how the SA lookup is performed. When fix SPI are used, it is RECOMMENDED the constraint node has as many SPI values as ESP session per host IP address, and that SA lookup includes the IP addresses.

Note that SPI value is used only for inbound traffic, as such the SPI negotiated with IKEv2 [RFC7296] or [RFC7815] by a peer, is the value used by the remote peer when it sends traffic. As SPI are only used for inbound traffic by the peer, this allows each peer to manage the set of SPIs used for its inbound traffic.

The use of fix SPI MUST NOT be considered as a way to avoid strong random generators. Such generator will be required in order to provide strong cryptographic protection and follow the randomness requirements for security described in [RFC4086]. Instead, the use of a fix SPI should only be considered as a way to overcome the resource limitations of the node, when this is feasible.

The use of a limited number of fix SPI or non random SPIs come with security or privacy drawbacks. Typically, a passive attacker may derive information such as the number of constraint devices connecting the remote peer, and in conjunction with data rate, the attacker may eventually determine the application the constraint device is associated to. If the SPI is fixed by a manufacturer or by some software application, the SPI may leak in an obvious way the type of sensor, the application involved or the model of the constraint device. When identification of the application or the hardware is associated to privacy, the SPI MUST be randomly generated. However, one needs to realize that in this case this is likely to be sufficient and a thorough privacy analysis is required. More specifically, traffic pattern MAY leak sufficient information in itself. In other words, privacy leakage is a complex and the use of random SPI is unlikely to be sufficient.

As the general recommendation is to randomly generate the SPI, constraint devices that will use a limited number of fix SPI are expected to be very constraint devices with very limited

capabilities, where the use of randomly generated SPI may prevent them to implement IPsec. In this case the ability to provision non random SPI enables these devices to secure their communications. These devices, due to there limitations, are expected to provide limited information and how the use of non random SPI impacts privacy requires further analysis. Typically temperature sensors, wind sensors, used outdoor do not leak privacy sensitive information. When used indoor, the privacy information is stored in the encrypted data and as such does not leak privacy.

As far as security is concerned, revealing the type of application or model of the constraint device could be used to identify the vulnerabilities the constraint device is subject to. This is especially sensitive for constraint devices where patches or software updates will be challenging to operate. As a result, these devices may remain vulnerable for relatively long period. In addition, predictable SPI enable an attacker to forge packets with a valid SPI. Such packet will not be rejected due to an SPI mismatch, but instead after the signature check which requires more resource and thus make DoS more efficient, especially for devices powered by batteries.

Values 0-255 SHOULD NOT be used. Values 1-255 are reserved and 0 is only allowed to be used internal and it MUST NOT be send on the wire.

[RFC4303] mentions :

"The SPI is an arbitrary 32-bit value that is used by a receiver to identify the SA to which an incoming packet is bound. The SPI field is mandatory. [...]"

"For a unicast SA, the SPI can be used by itself to specify an SA, or it may be used in conjunction with the IPsec protocol type (in this case ESP). Because the SPI value is generated by the receiver for a unicast SA, whether the value is sufficient to identify an SA by itself or whether it must be used in conjunction with the IPsec protocol value is a local matter. This mechanism for mapping inbound traffic to unicast SAs MUST be supported by all ESP implementations."

4. Sequence Number(SN) (32 bit)

According to [RFC4303], the Sequence Number (SN) is a mandatory 32 bits field in the packet.

The SN is set by the sender so the receiver can implement anti-replay protection. The SN is derived from any strictly increasing function that guarantees: if packet B is sent after packet A, then SN of packet B is strictly greater then the SN of packet A.

Some constraint devices may establish communication with specific devices, like a specific gateway, or nodes similar to them. As a result, the sender may know whereas the receiver implements anti-replay protection or not. Even though the sender may know the receiver does not implement anti replay protection, the sender **MUST** implement a always increasing function to generate the SN.

Usually, SN is generated by incrementing a counter for each packet sent. A constraint device may avoid maintaining this context and use another source that is known to always increase. Typically, constraint nodes using 802.15.4 Time Slotted Channel Hopping (TSCH), whose communication is heavily dependent on time, can take advantage of their clock to generate the SN. This would guarantee a strictly increasing function, and avoid storing any additional values or context related to the SN. When the use of a clock is considered, one should take care that packets associated to a given SA are not sent with the same time value.

For inbound traffic, it is **RECOMMENDED** to provide a anti-replay protection, and the size of the window depends on the ability of the network to deliver packet out of order. As a result, in environment where out of order packets is not possible the window size can be set to one. However, while **RECOMMENDED**, there is no requirements to implement an anti replay protection mechanism implemented by IPsec. A node **MAY** drop anti-replay protection provided by IPsec, and instead implement its own internal mechanism.

[RFC4303] mentions :

"This unsigned 32-bit field contains a counter value that increases by one for each packet sent, i.e., a per-SA packet sequence number. For a unicast SA or a single-sender multicast SA, the sender **MUST** increment this field for every transmitted packet. Sharing an SA among multiple senders is permitted, though generally not recommended. [...] The field is mandatory and **MUST** always be present even if the receiver does not elect to enable the anti-replay service for a specific SA."

5. Padding

The purpose of padding is to respect the 32 bit alignment of ESP. ESP **MUST** have at least one padding byte Pad Length that indicates the padding length. ESP padding bytes are generated by a succession of unsigned bytes starting with 1, 2, 3 with the last byte set to Pad Length, where Pad Length designates the length of the padding bytes.

Checking the padding structure is not mandatory, so the constraint device may not proceed to such checks, however, in order to

interoperate with existing ESP implementations, it MUST build the padding bytes as recommended by ESP.

In some situation the padding bytes may take a fix value. This would typically be the case when the Data Payload is of fix size.

[RFC4303] mentions :

"If Padding bytes are needed but the encryption algorithm does not specify the padding contents, then the following default processing MUST be used. The Padding bytes are initialized with a series of (unsigned, 1-byte) integer values. The first padding byte appended to the plaintext is numbered 1, with subsequent padding bytes making up a monotonically increasing sequence: 1, 2, 3, When this padding scheme is employed, the receiver SHOULD inspect the Padding field. (This scheme was selected because of its relative simplicity, ease of implementation in hardware, and because it offers limited protection against certain forms of "cut and paste" attacks in the absence of other integrity measures, if the receiver checks the padding values upon decryption.)"

ESP [RFC4303] also provides Traffic Flow Confidentiality (TFC) as a way to perform padding to hide traffic characteristics, which differs from respecting a 32 bit alignment. TFC is not mandatory and MUST be negotiated with the SA management protocol. TFC has not yet being widely adopted for standard ESP traffic. One possible reason is that it requires to shape the traffic according to one traffic pattern that needs to be maintained. This is likely to require extra processing as well as providing a "well recognized" traffic shape which could end up being counterproductive. As such TFC is not expected to be supported by a minimal ESP implementation.

As a result, TFC cannot not be enabled with minimal, and communication protection that were relying on TFC will be more sensitive to traffic shaping. This could expose the application as well as the devices used to a passive monitoring attacker. Such information could be used by the attacker in case a vulnerability is disclosed on the specific device. In addition, some application use - such as health applications - may also reveal important privacy oriented informations.

Some constraint nodes that have limited battery life time may also prefer avoiding sending extra padding bytes. However the same nodes may also be very specific to an application and device. As a result, they are also likely to be the main target for traffic shaping. In most cases, the payload carried by these nodes is quite small, and the standard padding mechanism may also be used as an alternative to TFC, with a sufficient trade off between the require energy to send

additional payload and the exposure to traffic shaping attacks. In addition, the information leaked by the traffic shaping may also be addressed by the application level. For example, it is preferred to have a sensor sending some information at regular time interval, rather when an specific event is happening. Typically a sensor monitoring the temperature, or a door is expected to send regularly the information - i.e. the temperature of the room or whether the door is closed or open) instead of only sending the information when the temperature has raised or when the door is being opened.

6. Next Header (8 bit)

According to [RFC4303], the Next Header is a mandatory 8 bits field in the packet. Next header is intended to specify the data contained in the payload as well as dummy packet. In addition, the Next Header may also carry an indication on how to process the packet [I-D.nikander-esp-beet-mode].

The ability to generate and receive dummy packet is required by [RFC4303]. For interoperability, it is RECOMMENDED a minimal ESP implementation discards dummy packets. Note that such recommendation only applies for nodes receiving packets, and that nodes designed to only send data may not implement this capability.

As the generation of dummy packets is subject to local management and based on a per-SA basis, a minimal ESP implementation may not generate such dummy packet. More especially, in constraint environment sending dummy packets may have too much impact on the device life time, and so may be avoided. On the other hand, constraint nodes may be dedicated to specific applications, in which case, traffic pattern may expose the application or the type of node. For these nodes, not sending dummy packet may have some privacy implication that needs to be measured. However, for the same reasons exposed in Section 5 traffic shaping at the IPsec layer may also introduce some traffic pattern, and on constraint devices the application is probably the most appropriated layer to limit the risk of leaking information by traffic shaping.

In some cases, devices are dedicated to a single application or a single transport protocol, in which case, the Next Header has a fix value.

Specific processing indications have not been standardized yet [I-D.nikander-esp-beet-mode] and is expected to result from an agreement between the peers. As a result, it is not expected to be part of a minimal implementation of ESP.

[RFC4303] mentions :

"The Next Header is a mandatory, 8-bit field that identifies the type of data contained in the Payload Data field, e.g., an IPv4 or IPv6 packet, or a next layer header and data. [...] the protocol value 59 (which means "no next header") MUST be used to designate a "dummy" packet. A transmitter MUST be capable of generating dummy packets marked with this value in the next protocol field, and a receiver MUST be prepared to discard such packets, without indicating an error."

7. ICV

The ICV depends on the crypto-suite used. Currently recommended [RFC8221] only recommend crypto-suites with an ICV which makes the ICV a mandatory field.

As detailed in Section 8 we recommend to use authentication, the ICV field is expected to be present that is to say with a size different from zero. This makes it a mandatory field which size is defined by the security recommendations only.

[RFC4303] mentions :

"The Integrity Check Value is a variable-length field computed over the ESP header, Payload, and ESP trailer fields. Implicit ESP trailer fields (integrity padding and high-order ESN bits, if applicable) are included in the ICV computation. The ICV field is optional. It is present only if the integrity service is selected and is provided by either a separate integrity algorithm or a combined mode algorithm that uses an ICV. The length of the field is specified by the integrity algorithm selected and associated with the SA. The integrity algorithm specification MUST specify the length of the ICV and the comparison rules and processing steps for validation."

8. Cryptographic Suites

The cryptographic suites implemented are an important component of ESP. The recommended suites to use are expected to evolve over time and implementer SHOULD follow the recommendations provided by [RFC8221] and updates. Recommendations are provided for standard nodes as well as constraint nodes.

This section lists some of the criteria that may be considered. The list is not expected to be exhaustive and may also evolve overtime. As a result, the list is provided as indicative:

1. Security: Security is the criteria that should be considered first for the selection of cipher suites. The security of cipher

suites is expected to evolve over time, and it is of primary importance to follow up-to-date security guidances and recommendations. The chosen cipher suites MUST NOT be known vulnerable or weak (see [RFC8221] for outdated ciphers). ESP can be used to authenticate only or to encrypt the communication. In the later case, authenticated encryption must always be considered [RFC8221].

2. **Interoperability:** Interoperability considers the cipher suites shared with the other nodes. Note that it is not because a cipher suite is widely deployed that is secured. As a result, security SHOULD NOT be weakened for interoperability. [RFC8221] and successors consider the life cycle of cipher suites sufficiently long to provide interoperability. Constraint devices may have limited interoperability requirements which makes possible to reduce the number of cipher suites to implement.
3. **Power Consumption and Cipher Suite Complexity:** Complexity of the cipher suite or the energy associated to it are especially considered when devices have limited resources or are using some batteries, in which case the battery determines the life of the device. The choice of a cryptographic function may consider re-using specific libraries or to take advantage of hardware acceleration provided by the device. For example if the device benefits from AES hardware modules and uses AES-CTR, it may prefer AUTH_AES-XCBC for its authentication. In addition, some devices may also embed radio modules with hardware acceleration for AES-CCM, in which case, this mode may be preferred.
4. **Power Consumption and Bandwidth Consumption:** Similarly to the cipher suite complexity, reducing the payload sent, may significantly reduce the energy consumption of the device. As a result, cipher suites with low overhead may be considered. To reduce the overall payload size one may for example:
 1. Use of counter-based ciphers without fixed block length (e.g. AES-CTR, or ChaCha20-Poly1305).
 2. Use of ciphers with capability of using implicit IVs [I-D.ietf-ipsecme-implicit-iv].
 3. Use of ciphers recommended for IoT [RFC8221].
 4. Avoid Padding by sending payload data which are aligned to the cipher block length - 2 for the ESP trailer.

9. IANA Considerations

There are no IANA consideration for this document.

10. Security Considerations

Security considerations are those of [RFC4303]. In addition, this document provided security recommendations and guidances over the implementation choices for each fields.

11. Acknowledgment

The authors would like to thank Daniel Palomares, Scott Fluhrer, Tero Kivinen, Valery Smyslov, Yoav Nir, Michael Richardson for their valuable comments.

12. References

12.1. Normative References

- [I-D.ietf-ipsecme-implicit-iv] Migault, D., Guggemos, T., and Y. Nir, "Implicit IV for Counter-based Ciphers in Encapsulating Security Payload (ESP)", draft-ietf-ipsecme-implicit-iv-05 (work in progress), June 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.

- [RFC7815] Kivinen, T., "Minimal Internet Key Exchange Version 2 (IKEv2) Initiator Implementation", RFC 7815, DOI 10.17487/RFC7815, March 2016, <<https://www.rfc-editor.org/info/rfc7815>>.
- [RFC8221] Wouters, P., Migault, D., Mattsson, J., Nir, Y., and T. Kivinen, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 8221, DOI 10.17487/RFC8221, October 2017, <<https://www.rfc-editor.org/info/rfc8221>>.

12.2. Informative References

- [I-D.mglt-ipsecme-diet-esp]
Migault, D., Guggemos, T., Bormann, C., and D. Schinazi, "ESP Header Compression and Diet-ESP", draft-mglt-ipsecme-diet-esp-06 (work in progress), May 2018.
- [I-D.mglt-ipsecme-ikev2-diet-esp-extension]
Migault, D., Guggemos, T., and D. Schinazi, "Internet Key Exchange version 2 (IKEv2) extension for the ESP Header Compression (EHC) Strategy", draft-mglt-ipsecme-ikev2-diet-esp-extension-01 (work in progress), June 2018.
- [I-D.nikander-esp-beet-mode]
Nikander, P. and J. Melen, "A Bound End-to-End Tunnel (BEET) mode for ESP", draft-nikander-esp-beet-mode-09 (work in progress), August 2008.

Appendix A. Document Change Log

[RFC Editor: This section is to be removed before publication]

-00: First version published.

-01: Clarified description

-02: Clarified description

Authors' Addresses

Daniel Migault
Ericsson
8400 boulevard Decarie
Montreal, QC H4P 2N2
Canada

Email: daniel.migault@ericsson.com

Tobias Guggemos
LMU Munich
MNM-Team
Oettingenstr. 67
80538 Munich, Bavaria
Germany

Email: guggemos@mn-m-team.org

LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: August 17, 2014

M. Sethi
V. Lehtovirta
P. Salmela
Ericsson
February 13, 2014

Using Generic Bootstrapping Architecture with Constrained Devices
draft-sethi-gba-constrained-01

Abstract

This document discusses the use of the 3GPP Generic Bootstrapping Architecture (GBA) for authenticating and securing constrained devices. While GBA re-uses the 3GPP credentials, it does not require mobile network access, such as LTE, but requires only IP connectivity. Though building devices that employ GBA is obviously well known, this document specifically focuses on techniques necessary to minimize memory and energy consumption which is essential for constrained device 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 <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 August 17, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	5
1.2. Rational for using GBA	5
2. Reason for low power implementation of GBA	6
3. Implementation Considerations	7
3.1. Full HTTP stack is not needed	7
3.2. Resource-Constrained AES implementations are widely available	7
3.3. Purging unnecessary functionality from memory after bootstrapping	7
3.4. Complete State Machine or Complex Error Handling Are Not Needed	8
4. Implementation Status	9
5. Future standardization work	12
6. Security Considerations	13
7. Acknowledgments	14
8. IANA Considerations	15
9. Informative References	16
Authors' Addresses	18

1. Introduction

Generic Bootstrapping Architecture (GBA) is part of the 3GPP standard [3gppts33220] based on 3GPP Authentication and Key Agreement (3GPP AKA). GBA is an application independent mechanism to provide a client application (running on the User equipment (UE)) and an application server with a shared session secret. This shared session secret can subsequently be used to authenticate and protect the communication between the client application and the application server. GBA authentication is based on the permanent secret shared between the UE's Universal Integrated Circuit Card (UICC), for example SIM card, and the corresponding profile information stored within the cellular network operator's Home Subscriber System (HSS) database. The permanent shared secret is used to generate a time-limited master key on the UE and the network operator. The UE and network operator derive a session key from the master key, and the network operator distributes this to the appropriate application server. This session key can then allow authentication and protection of the communication channel between the UE and application server. Figure 1 provides a high-level overview of GBA.

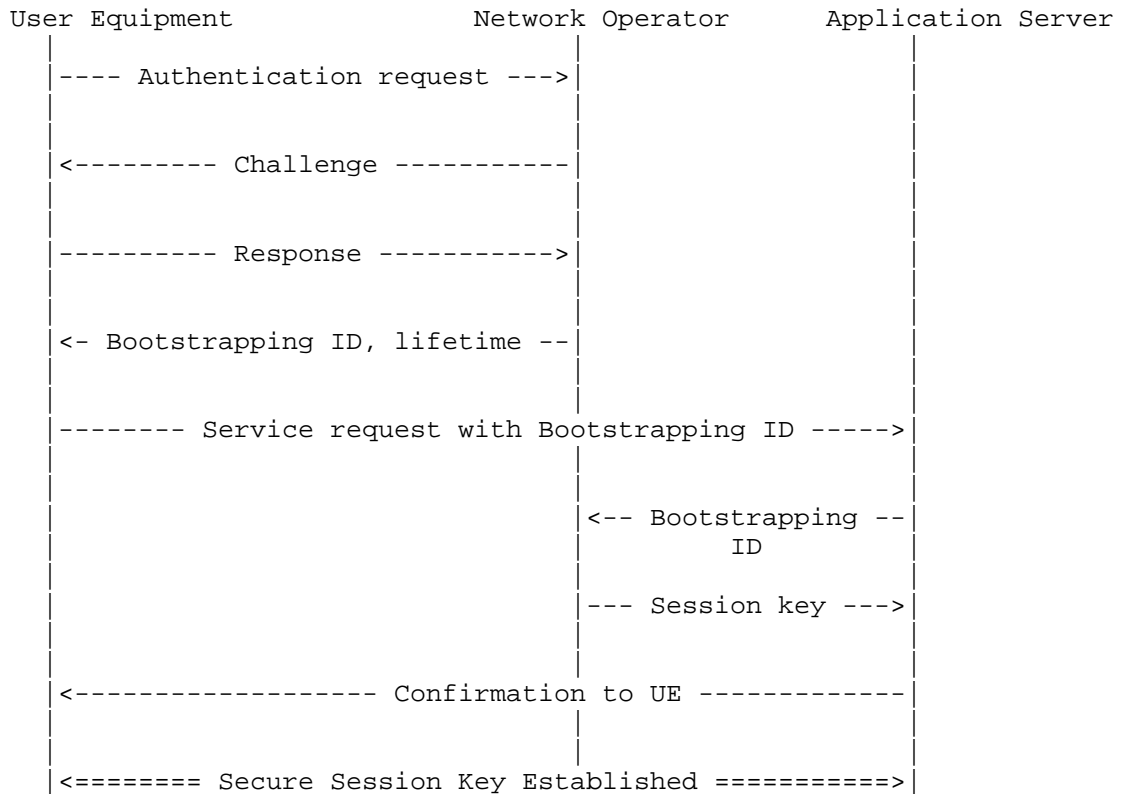


Figure 1: High Level Overview of GBA

As shown in Figure 1, the UE initially identifies itself to the network operator and requests for authentication. The network operator responds by sending a challenge that can be correctly responded to, only with the permanent secret stored on the UICC. The UE responds to the challenge using the permanent secret on the UICC. The network operator then verifies the response and if it is found to be valid, it derives a master key and returns a bootstrapping ID to the UE. The UE then derives a session key from the master key and initiates a secure connection with the application server with which it wants to communicate and provides the bootstrapping ID obtained in the previous step. The application server uses the bootstrapping ID to obtain a session key from the network operator. The session key is derived from the master key by the network operator. The network operator authenticates the application server before generating and revealing the session key. Once the application server is authenticated, it receives the session key. The application authenticates the original message from UE with this session key. If

authenticated correctly, the application server sends a confirmation message to the UE indicating that it can now protect the communication with the session key. While in this scenario, the UE is responsible for initiating secure communication with the application server, the GBA standard also allows the application server to inform the UE to perform GBA authentication for access to the requested resource. In 3GPP terminology, the application server is referred to as Network Application Function (NAF) and the cellular operator interface is referred to as Bootstrapping Server Function (BSF). The bootstrapping ID is known as the Bootstrapping Transaction Identifier (B-TID)

The current GBA standard requires HTTP and TCP along with crypto algorithms like AES and MD5 to be supported on the UE. While the implementation details may be obvious for a UE, using GBA for authentication and channel security on resource-constrained devices may be non-trivial. This draft provides implementation guidelines for using GBA authentication on resource-constrained devices.

1.1. Terminology

This draft uses terminologies for constrained devices defined in [I-D.ietf-lwig-terminology].

1.2. Rational for using GBA

In principle, GBA can be used to protect any application traffic, e.g. with TLS, as GBA provides shared keys for applications to use. Using shared keys avoids the need for complex calculations of asymmetric crypto and the need for verifying certificate chains and checking revocation lists. Instead, GBA relies on the security infrastructure of the network operator. Even though GBA re-uses the credentials stored on the UICC, use of GBA does not require mobile network access, such as LTE, but requires only IP connectivity to the network operator infrastructure. As the credentials of the UICC are used by GBA, there is no need to remember or store passwords to authenticate to applications, and in principle a button-free authentication can be implemented.

2. Reason for low power implementation of GBA

As explained in [I-D.draft-ietf-lwig-cellular-00] there are many situations where focus on a low energy consuming implementation is unnecessary. This would be the case, when for example, the devices are connected to the main, or receive power over wired communications media, such as in Power-over-Ethernet (PoE) devices. Such devices require a limited amount of optimization for energy efficiency. Similarly, devices that are directly connected to the mains do not require extreme optimizations and/or hacks for saving power. Also devices that can gather power from energy harvesting do not necessarily need optimization techniques.

However, wirelessly connected battery powered sensors and actuators are the future and existence of wired infrastructure for communication or power might be impossible or impractical. Such devices require highly efficient implementations of security and network protocols as changing batteries frequently is not possible because of the sheer number of devices and the fact that some of these devices may be physically inaccessible once installed.

Unfortunately, much of our current network and security technology is built with different objectives in mind. Networked devices are assumed to be "always on", and require frequent re-charging. However, most sensor and actuator applications require long battery lifetimes, in the order of years or even a decade, if not longer. While some devices may already reach multi-year lifetimes with continuous improvement in low-power electronics and radio technology, it is rather reasonable to note that battery lifetimes are generally too short currently. Therefore, to use GBA as a mechanism for secure bootstrapping for constrained device applications, a more focused effort is required to ensure an energy-efficient implementation. This memo is a first step for defining considerations and guidelines to achieve an energy-efficient GBA implementation.

3. Implementation Considerations

In this section we describe implementation guidelines that apply to GBA on constrained devices in particular but can be used in general for implementing other similar related network protocols.

3.1. Full HTTP stack is not needed

GBA relies on HTTP digest authentication and requires the client to implement an HTTP stack. Although HTTP client libraries for constrained devices [small-http] are available, they are often unnecessary. This is because the number of HTTP messages required for a GBA-run is small and having an entire library consumes more memory than the templates for these messages. Thus, the required messages can be handcrafted and sent over TCP. Thanks to the plain text nature of HTTP/1.1 this is rather easy to do. An example of a minimalistic HTTP packet is shown here:

```
char httpFirstRequestFormat[82] = "GET /naf/resource HTTP/1.1\r\n"  
                                   "Host: pl23.example.net:8080\r\n"  
                                   "Connection: Keep-Alive\r\n"  
                                   "\r\n";
```

Similarly, the response can easily be parsed from the packets received from the transport layer without necessarily performing all error checks. If an unknown packet or unknown data is received, the application can simply re-start from the first exchange.

3.2. Resource-Constrained AES implementations are widely available

AES is rather easy to implement. There are many open source implementations available specifically for resource-constrained environments [gladman], [avr-crypto-lib]. There are several resources [jpkaps06],[shammid] that show that AES can be implemented with low memory and energy consumption. Also many constrained devices and platforms such as the MicaZ motes are equipped with an AES hardware engine as part of the IEEE 802.15.4 (Zigbee)-compliant RF module.

3.3. Purging unnecessary functionality from memory after bootstrapping

Once a GBA bootstrapping round has finished, only the session key (KsNAF) and B-TID need to be retained in the memory. Optimally, the master key (Ks) and the key derivation function can be retained in the memory if the device will be connecting to multiple NAFs. If a GBA run is only used for authentication, then all GBA related code can be purged from the memory. This means that any libraries or code used for application, transport or cryptography during a GBA run can

be purged and loaded back into the memory when the lifetime of the key expires. This is especially useful with long lived session keys as the device can switch to other application (CoAP) and routing/transport (RIPL/UDP) protocol after secure bootstrapping.

3.4. Complete State Machine or Complex Error Handling Are Not Needed

A typical GBA implementation requires a state-machine to track messages that have been sent and received from the BSF and NAF. However, implementing such a state-machine with appropriate error handling can be rather complex for resource-constrained device. It is therefore advisable for such devices to implement a simple hard fail-over starting from the first message in case of an error, unknown packet or timeout. However, there also needs to be a limit to the number of fail-overs as the device might drain its battery from too many failed GBA authentication tries.

4. Implementation Status

In this section we describe our GBA implementation that was developed for prototyping GBA on constrained devices using the principles described in the previous section. We used an Arduino Mega prototyping board [mega] and an Ethernet Shield for communication. The Mega board has an 8-bit ATmega2560 microprocessor.

For AES, we used Brian Gladman's byte oriented AES implementation [gladman]. Similarly, we used easily available C implementations of SHA256 [sha256] and MD5 [md5]. The default Ethernet and TCP libraries available with the Arduino platform were used for communication.

We tested this GBA implementation against standard 3GPP BSF and NAF interfaces running on our public servers. The sequence of messages for the resource-constrained (sensor/actuator) device in our implementation is as follows:

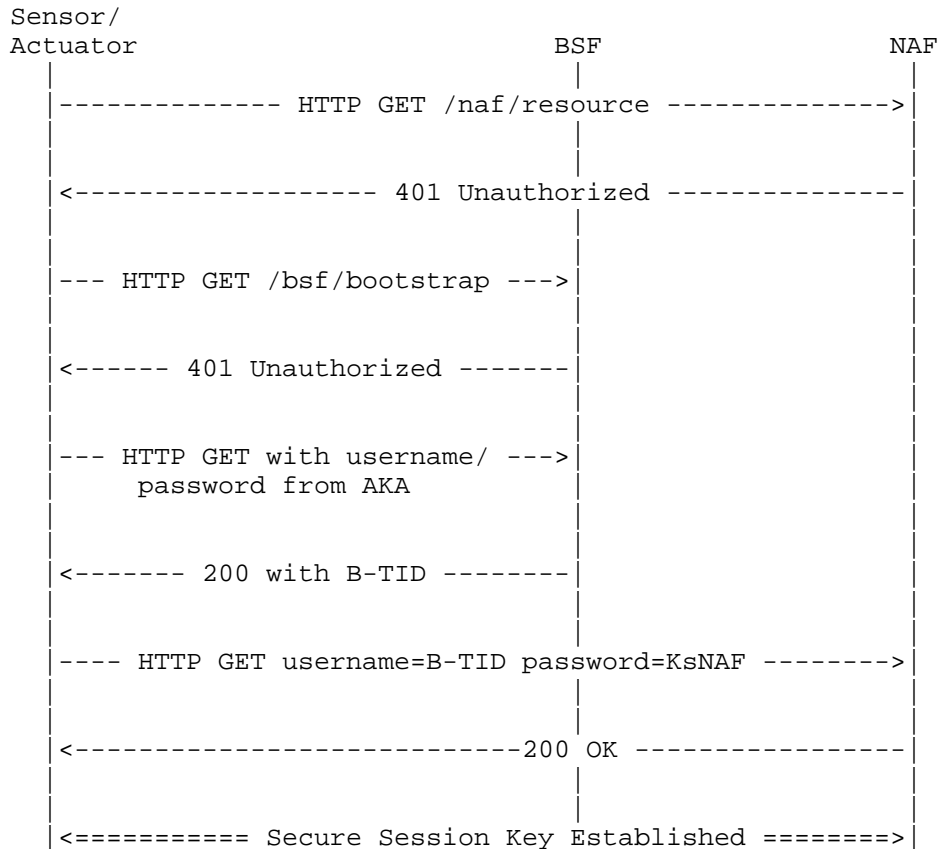


Figure 2: Prototype message sequence

The sensor/actuator begins by contacting the service with which it wishes to communicate (NAF in 3GPP terminology). This means that the sensor sends a HTTP GET for the resource which it wishes to access. The service/NAF in-turn replies with a HTTP 401 Unauthorized with WWW-Authenticate indicating that it requires digest authentication for access to that resource. The HTTP response header also contains the realm prefixed with "3GPP-Bootstrapping@" to indicate that a GBA run can be performed to obtain the appropriate keys for digest authentication. The sensor then contacts the BSF hosted by the operator by sending a HTTP GET. This again results in a 401 unauthorized with the HTTP header containing "algorithm=AKAv1-MD5" indicating that AKA authentication needs to be run. The sensor then creates and sends another HTTP GET containing the username and password from AKA to the BSF. The BSF now replies with an 200 OK message whose body contains the B-TID and the lifetime for which the

B-TID and the derived master key (Ks) would be valid. Finally, the sensor contacts the service/NAF with a HTTP GET containing the B-TID as the username and KsNAF as the password. KsNAF is obtained from the master key Ks and the fully qualified domain name (FQDN) of the NAF. If the service/NAF replies with a 200 OK, the sensor is assured of successful authentication. The sensor and NAF can then use the session key for TLS/DTLS in PSK mode. This key can also be used for message level encryption and/or integrity protection by using, for example, the EAX [mihir04] mode.

Some approximate results from our prototype are listed in the table below:

RAM consumption	<5 kB
ROM consumption	44 kB
Time for 1 GBA run	1.5 s
Energy ($W = U * I * t$)	150mJ
Additional HTTP messages sent/received	8

Prototype Details

Table 1

U=Operating Voltage (5V), I=Current drawn (0.02A for Atmega2560),
t=Execution time (1.5 s)

The energy estimate and time for 1 GBA run would vary significantly depending on the network access (which in our case was Ethernet).

5. Future standardization work

While we have shown how the current GBA standard can be used as a method for secure bootstrapping of constrained devices, it would also be useful in the future for 3GPP and IETF to define GBA over protocols other than HTTP/TCP.

6. Security Considerations

GBA is used for authenticating a device based on the 3GPP subscription credentials stored in the device. Normally the credentials are stored on a UICC, but also embedded UICCs (eUICC) are possible. eUICCs are targeted for scenarios where the 3GPP subscription should be changeable remotely, without having to physically exchange the card in the device. Both UICCs and eUICCs provide hardware based protection for the subscription credentials used for authentication in 3GPP networks. However, in addition it is also essential to protect the access to the (e)UICC from applications to protect against Man-in-the-Middle (MitM) type of attacks [3gpptr33905].

If a UICC or eUICC is not available, GBA digest is an option [3gppts33220]. GBA digest uses SIP digest credentials, basically a username/password pair and secret deducted from TLS, to do GBA. The security requirement defined by 3GPP for the SIP digest credentials are that they need to be securely stored within the terminal.

7. Acknowledgments

The authors would like to thank Bengt Sahlin, Jari Arkko and Ari Keranen for the interesting discussions on this topic.

8. IANA Considerations

This document has no IANA actions.

9. Informative References

- [3gpptr33905]
3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Recommendations for Trusted Open Platforms", September 2012,
<<http://www.3gpp.org/DynaReport/33905.htm>>.
- [3gppts33220]
3GPP, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA) (Release 12)", March 2013,
<<http://www.3gpp.org/DynaReport/33220.htm>>.
- [I-D.draft-ietf-lwig-cellular-00]
Arkko, J., Eriksson, A., and A. Keranen, "Building Power-Efficient CoAP Devices for Cellular Networks", draft-ietf-lwig-cellular-00 (work in progress), December 2013.
- [I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-06 (work in progress), August 2013.
- [avr-crypto-lib]
"AVR Crypto-lib", December 2013,
<<http://avrcryptolib.das-labor.org/trac/wiki/AES>>.
- [gladman] "Byte Oriented AES (Low Resource Version)", December 2013,
<<http://gladman.plushost.co.uk/oldsite/AES/>>.
- [jpkaps06]
Kaps, JP. and B. Sunar, "Energy comparison of AES and SHA-1 for ubiquitous computing", Emerging Directions in Embedded and Ubiquitous Computing. Springer Berlin , 2006.
- [md5] "MD5", December 2013,
<<https://github.com/tzikis/ArduinoMD5/>>.
- [mega] "Arduino Mega", December 2013,
<<http://arduino.cc/en/Main/arduinoBoardMega2560>>.
- [mihir04] Bellare, M., Rogaway, P., and D. Wagner, "The EAX mode of operation", Fast Software Encryption. Springer Berlin Heidelberg , 2004.

- [sha256] "SHA256", December 2013,
<<https://github.com/Cathedrow/Cryptosuite>>.
- [shammid] Didla, S., Ault, A., and S. Bagchi, "Optimizing AES for
embedded devices and wireless sensor networks", 4th
International Conference on Testbeds and research
infrastructures for the development of networks &
communities , 2008.
- [small-http]
"HttpClient for Arduinos", December 2013,
<<https://github.com/amcewen/HttpClient>>.

Authors' Addresses

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: mohit.m.sethi@ericsson.com

Vesa Lehtovirta
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: vesa.lehtovirta@ericsson.com

Patrik Salmela
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: patrik.salmela@ericsson.com

