CoRE Working Group
Internet-Draft
Updates: 7252, 8323 (if approved)
Intended status: Standards Track
Expires: September 2, 2019

K. Hartke
Ericsson

March 1, 2019

# Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP) draft-ietf-core-stateless-00

## Abstract

This document provides considerations for alleviating CoAP clients and intermediaries of maintaining per-request state. Additionally, it introduces a new, optional CoAP protocol extension for extended token lengths.

This document updates RFCs 7252 and 8323.

## 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 September 2, 2019.

Table of Contents



## 1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a RESTful application-layer protocol for constrained environments [RFC7228]. In CoAP, clients (or intermediaries in the client role) make requests to servers (or intermediaries in the server role), which serve the requests by returning responses.

While a request is ongoing, a client typically maintains some state that it requires for processing the response when it arrives. Identification of this state is done by means of a _token_ in CoAP, an opaque sequence of bytes chosen by the client and included in the CoAP request. The server returns the token verbatim in any resulting CoAP response (Figure 1).

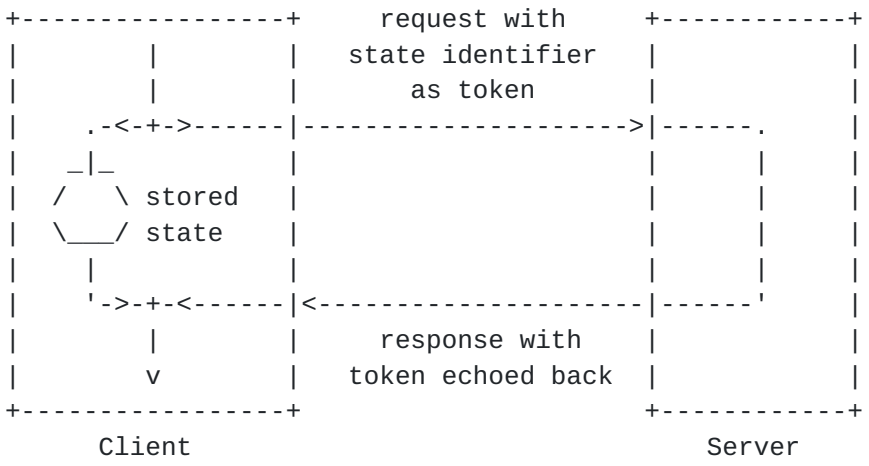



Figure 1: Token as an Identifier for Request State

In some scenarios, it can be beneficial to reduce the amount of state stored at the client at the cost of increased message sizes. Clients can implement this by serializing (parts of) their state into the token itself and recovering the state from the token in the response (Figure 2).

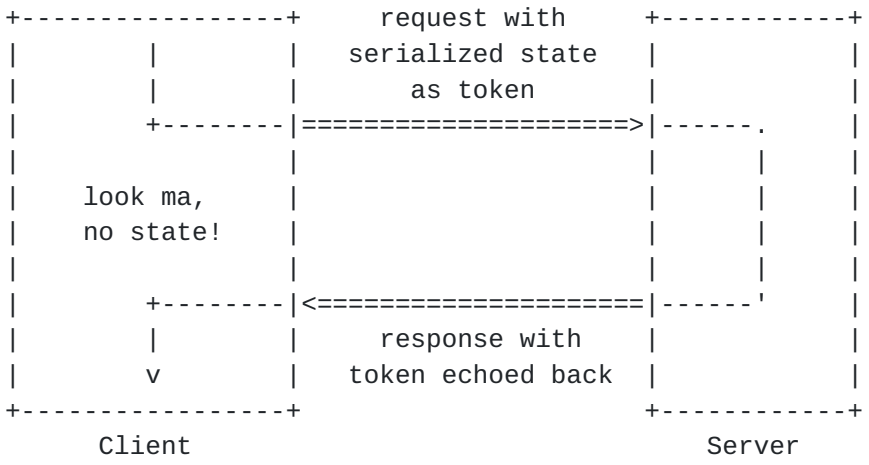



Figure 2: Token as Serialization of Request State

Section 3 of this document provides considerations for making clients "stateless" in this way, i.e., avoiding per-request state. (They'll still need to maintain per-server state and other kinds of state, so they're not entirely stateless.)

Serializing state into tokens is complicated by the fact that both CoAP over UDP [RFC7252] and CoAP over reliable transports [RFC8323] limit the maximum token length to 8 bytes. To overcome this limitation, Section 2 of this document first introduces a CoAP protocol extension for extended token lengths.

While the mechanism (extended token lengths) and the use case (stateless clients) presented in this document are closely related, both can be used independently of the other: Some implementations may fit their state in 8 bytes; some implementations may have other use cases for extended token lengths.

### 1.1. Terminology

Stateless
: In this document, "stateless" refers to an implementation strategy for a client (or intermediary in the client role) that doesn't keep state for the individual requests it sends to a server (or intermediary in the server role). The client still needs to keep state for each server it communicates with (such as state for generating tokens and congestion control), so it's not free of any state.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2. Extended Tokens

### 2.1. Extended Token Length (TKL) Field

This document updates the message formats defined for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with the following new definition of the TKL field, increasing the maximum token length to 65804 bytes.

Token Length (TKL): 4-bit unsigned integer. A value between 0 and 12 inclusive indicates the length of the variable-length Token field in bytes. Three values are reserved for special constructs:

13: An 8-bit unsigned integer precedes the Token field and indicates the length of the Token field minus 13.

14: A 16-bit unsigned integer in network byte order precedes the Token field and indicates the length of the Token field minus 269.

15: Reserved. This value MUST NOT be sent and MUST be processed as a message format error.

All other fields retain their definition.

The updated message formats are illustrated in Appendix A.

## 2.2. Discovering Support

Extended token lengths require support from the server or, if there are one or more intermediaries between the client and the server, the intermediary in the server role that the client is interacting with.

Support can be discovered by a client (or intermediary in the client role) in one of two ways: In case Capabilities and Settings Messages (CSMs) are available, such as in CoAP over TCP, support can be discovered using the Extended-Token-Lengths Capability Option defined in Section 2.2.1. Otherwise, such as in CoAP over UDP, support can only be discovered by trial and error, as described in Section 2.2.2.

### 2.2.1. Extended-Token-Lengths Capability Option

A sender can use the elective Extended-Token-Lengths Capability Option to indicate its support for the new TKL field definition specified in Section 2.1.

| # | C | R | Applies to | Name | Format | Length | Base Value |
|---|---|---|---|---|---|---|---|
| TBD | | | CSM | Extended-Token-Lengths | empty | 0 | (none) |

C=Critical, R=Repeatable

Table 1: The Extended-Token-Lengths Capability Option

### 2.2.2. Trial and Error

A request with a TKL field value outside the range from 0 to 8 will be considered a message format error (Section 3 of RFC 7252) and be rejected by a recipient that does not support the updated TKL field definition. A client thus can determine support by sending a request with an extended token length and checking whether it's rejected by the recipient or not.

In CoAP over UDP, a recipient rejects a malformed confirmable message by sending a Reset message (Section 4.2 of RFC 7252). In case of a non-confirmable message, sending a Reset message is permitted but not required (Section 4.3 of RFC 7252). It is therefore RECOMMENDED that clients use a confirmable message.

As per RFC 7252, Reset messages are empty and don't contain a token; they only return the Message ID (Figure 3). They also don't contain any indication of what caused a message format error. It is therefore RECOMMENDED that clients use a request that contains no potential message format error other than the extended token length.

In CoAP over TCP, TLS, and WebSockets, a recipient rejects a malformed message by sending an Abort message and shutting down the connection (Section 5.6 of RFC 8323).

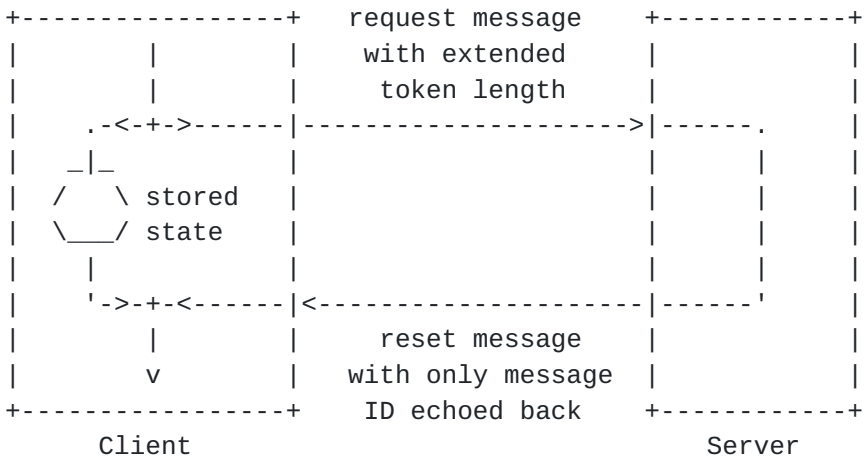



Figure 3: A Confirmable Request With an Extended Token is Rejected With a Reset Message if the Next Hop Does Not Support It

## 2.3. Intermediaries

Tokens are a hop-by-hop feature: When an intermediary receives a request, the only requirement is that it echoes the token back in any resulting response. There is no requirement or expectation that an intermediary passes a client's token on to a server or that an intermediary uses extended token lengths itself when receiving a request with an extended token length.

# 3. Stateless Clients

A client can be alleviated of keeping request state by serializing the state into a sequence of bytes and sending the result as the token of the request. The server will return the token to the client in the response, so that the client can recover the state and process the response as if it had kept the state locally.

The format of the serialized state is an implementation detail of the client and opaque to any server implementation. Using tokens to serialize state has significant and non-obvious security and privacy implications that need to be mitigated; see Section 4.

### 3.1. Intermediaries

Tokens are a hop-by-hop feature: If a client makes a request to an intermediary, that intermediary needs to store the client's token (along with the client's transport address) while it makes its own request to the next hop towards the origin server and waits for the response.

An intermediary might want to be "stateless" as well, i.e., be alleviated of storing the client's token and transport address for ongoing requests. This can be implemented by serializing this information along the request state into the token to the next hop. When the next hop returns the response, the intermediary can recover the information from the token and use it to satisfy the client's request.

The downside of this approach is that an intermediary, without keeping request state, is unable to aggregate requests, which reduces efficiency. In particular, when multiple clients observe [RFC7641] the same resource, aggregating requests is REQUIRED for efficiency (Section 3.1 of RFC 7641). This implies that an intermediary MUST NOT include an Observe Option in requests it sends without keeping request state.

When using blockwise transfers [RFC7959], a server might not be able to distinguish blocks originating from different clients once they have been forwarded by an intermediary. To ensure that this does not lead to inconsistent resource state, a stateless intermediary MUST include the Request-Tag Option [I-D.ietf-core-echo-request-tag] in blockwise transfers with a value that uniquely identifies the next hop towards the client in the intermediary's namespace.

### 3.2. Extended Tokens

A client (or intermediary in the role of a client) that depends on support for extended token lengths (Section 2) from the next hop to avoid keeping request state MUST perform a discovery of support (Section 2.2) before it can be stateless. This discovery MUST be performed in a stateful way, i.e., keeping state for the request (Figure 4): If the client was stateless from the start and the next hop doesn't support extended tokens, then any error message couldn't be processed since the state would neither be present at the client nor returned in the Reset message (Figure 5).

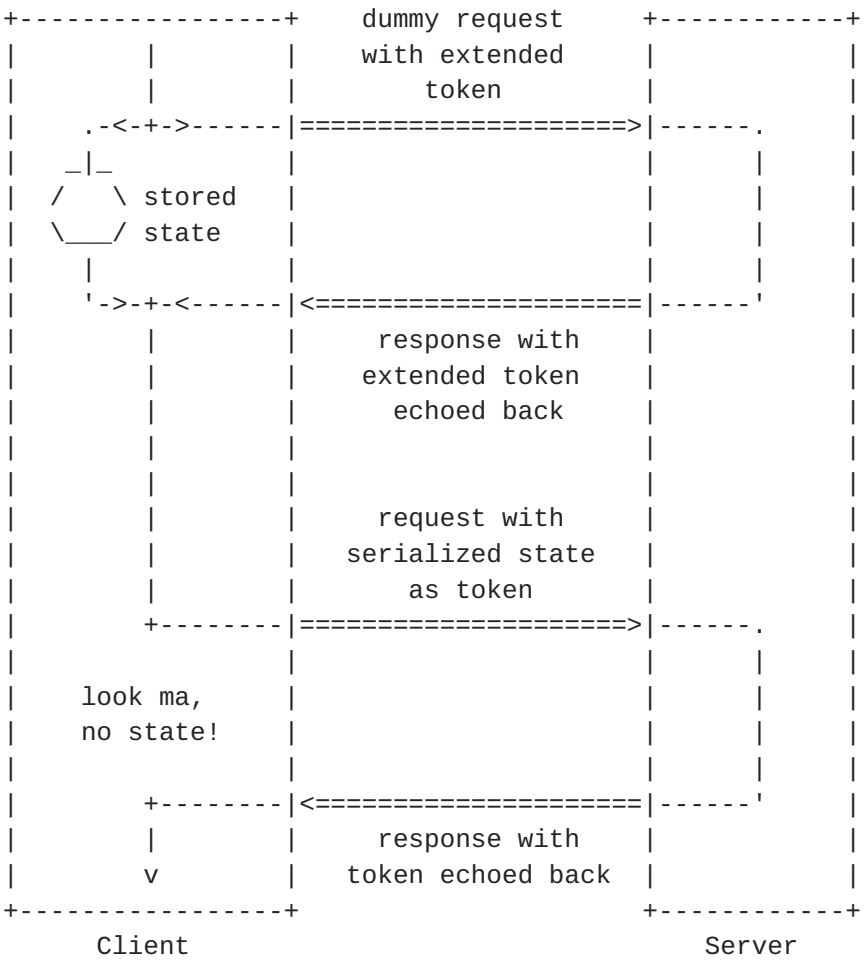



Figure 4: Depending on Extended Tokens for Being Stateless First Requires a Successful Stateful Discovery of Support

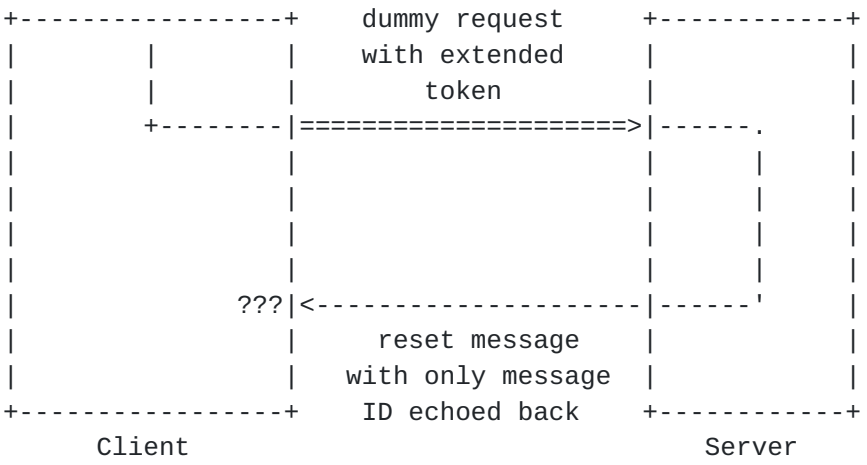



Figure 5: Stateless Discovery of Support Does Not Work

### 3.3. Message Transmission

As a further step in the case of CoAP over UDP [RFC7252], a client (or intermediary in the client role) might want to also avoid keeping message transmission state.

Generally, a client can use confirmable or non-confirmable messages for requests. When using confirmable messages, it needs to keep message exchange state for performing retransmissions and handling Acknowledgement and Reset messages. When using non-confirmable messages, it can keep no message exchange state. However, in either case the client needs to keep congestion control state. That is, it needs to maintain state for each node it communicates with and, e.g., enforce NSTART.

As per RFC 7252, a client must be prepared to receive a response as a piggybacked response, a separate response or non-confirmable response (Section 5.2 of RFC 7252), regardless of the message type used for the request. A stateless client needs to handle these response types as follows:

- If a piggybacked response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST silently ignore the message.

- If a separate response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST reject the message as specified in Section 4.2 of RFC 7252.

- If a non-confirmable response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST reject the message as specified in Section 4.3 of RFC 7252.

## 4. Security Considerations

### 4.1. Extended Tokens

Tokens significantly larger than the 8 bytes specified in RFC 7252 have implications for nodes in particular with constrained memory size that need to be mitigated.

A node in the server role supporting extended token lengths may be vulnerable to a denial-of-service when an attacker (either on-path or a malicious client) sends large tokens to fill up the memory of the node. Implementations MUST be prepared for this and mitigate it.

### 4.2. Stateless Clients

Transporting the state needed by a client to process a response as serialized state information in the token has several significant and non-obvious security and privacy implications that need to be mitigated.

Serialized state information is an attractive target for both unwanted nodes (attackers between the node in client role and the next hop) and wanted nodes (the next hop itself) on the path. Therefore, a node in the client role MUST integrity protect the state information, unless processing a response does not modify state or cause other significant side effects.

Even when the serialized state is integrity protected, an attacker may still replay a response, making the client believe it sent the same request twice. Therefore, the node in client role MUST implement replay protection (e.g., by using sequence numbers and a replay window), unless processing a response does not modify state or cause other significant side effects. Integrity protection is REQUIRED for replay protection.

If processing a response without keeping request state is sensitive to the time elapsed to sending the request, then the serialized state MUST include freshness information (e.g., a timestamp).

Information in the serialized state may be privacy sensitive. A node in client role MUST encrypt the serialized state if it contains privacy sensitive information that an attacker would not get otherwise. For example, an intermediary that serializes the client's token and transport address into its token leaks that information to the next hop, which may be undesirable. In wireless mesh networks, where all traffic is visible to a passive attacker, encryption may not be needed as the attacker can get the same information from analyzing the traffic flows.

A node in client role using OSCORE [I-D.ietf-core-object-security] always MUST encrypt the serialized state.

#### 4.2.1. Recommended Algorithms

The use of encryption, integrity protection, and replay protection of serialized state is recommended in general, unless a careful analysis of any potential attacks to security and privacy is performed. AES_CCM with a 64 bit tag is recommended, combined with a sequence number and a replay window. Where encryption is not needed, HMAC-SHA-256, combined with a sequence number and a replay window, may be used.

## 5. IANA Considerations

### 5.1. CoAP Signaling Option Number

The following entries are added to the "CoAP Signaling Option Numbers" registry within the "CoRE Parameters" registry.

| Applies to | Number | Name | Reference |
|---|---|---|---|
| 7.01 | TBD | Extended-Token-Lengths | [[this document]] |

## 6. References

### 6.1. Normative References

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-03 (work in progress), October 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>.

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

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

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

### 6.2. Informative References

[I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-09 (work in progress), November 2018.

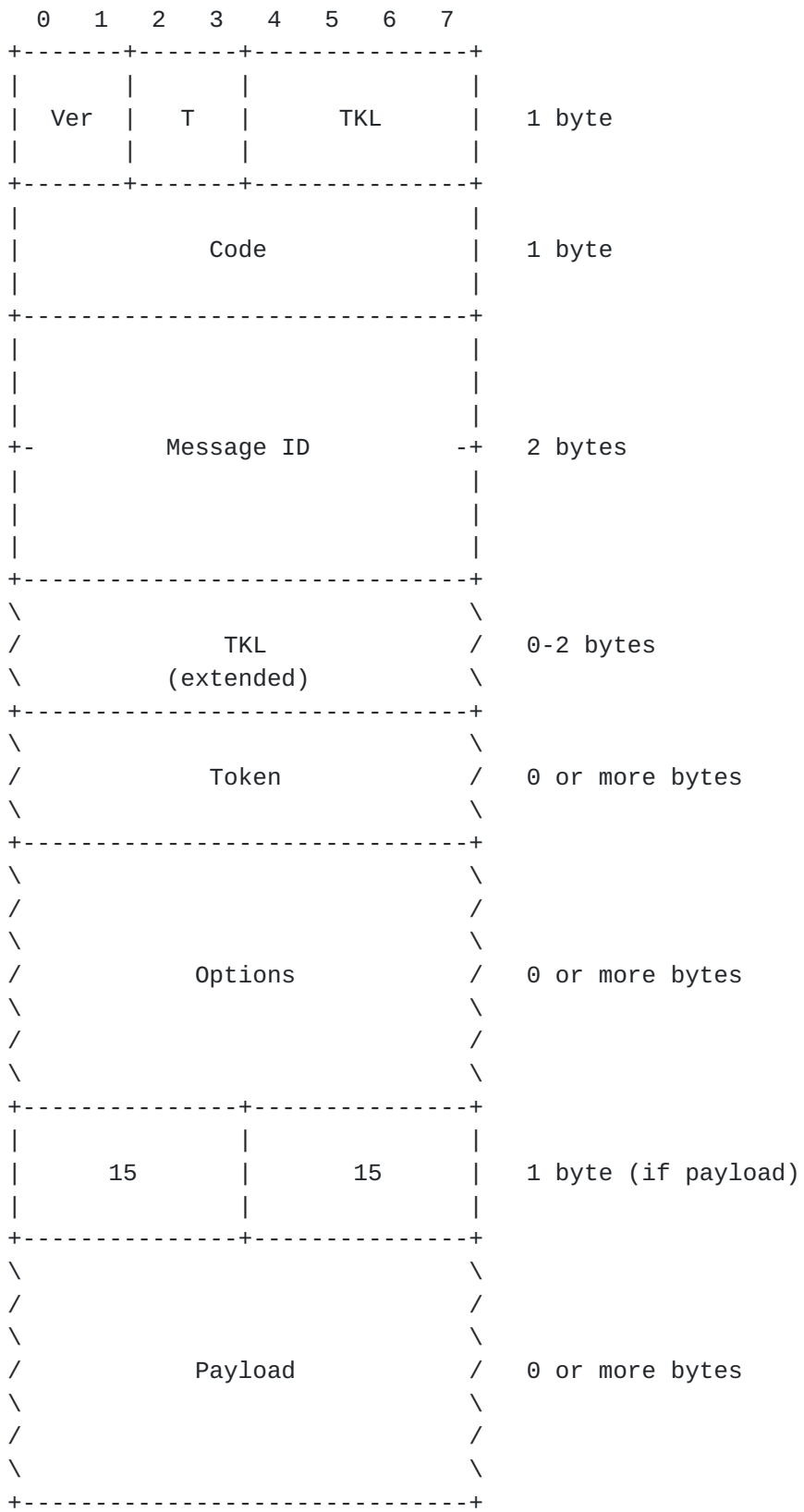
[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-15 (work in progress), August 2018.

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

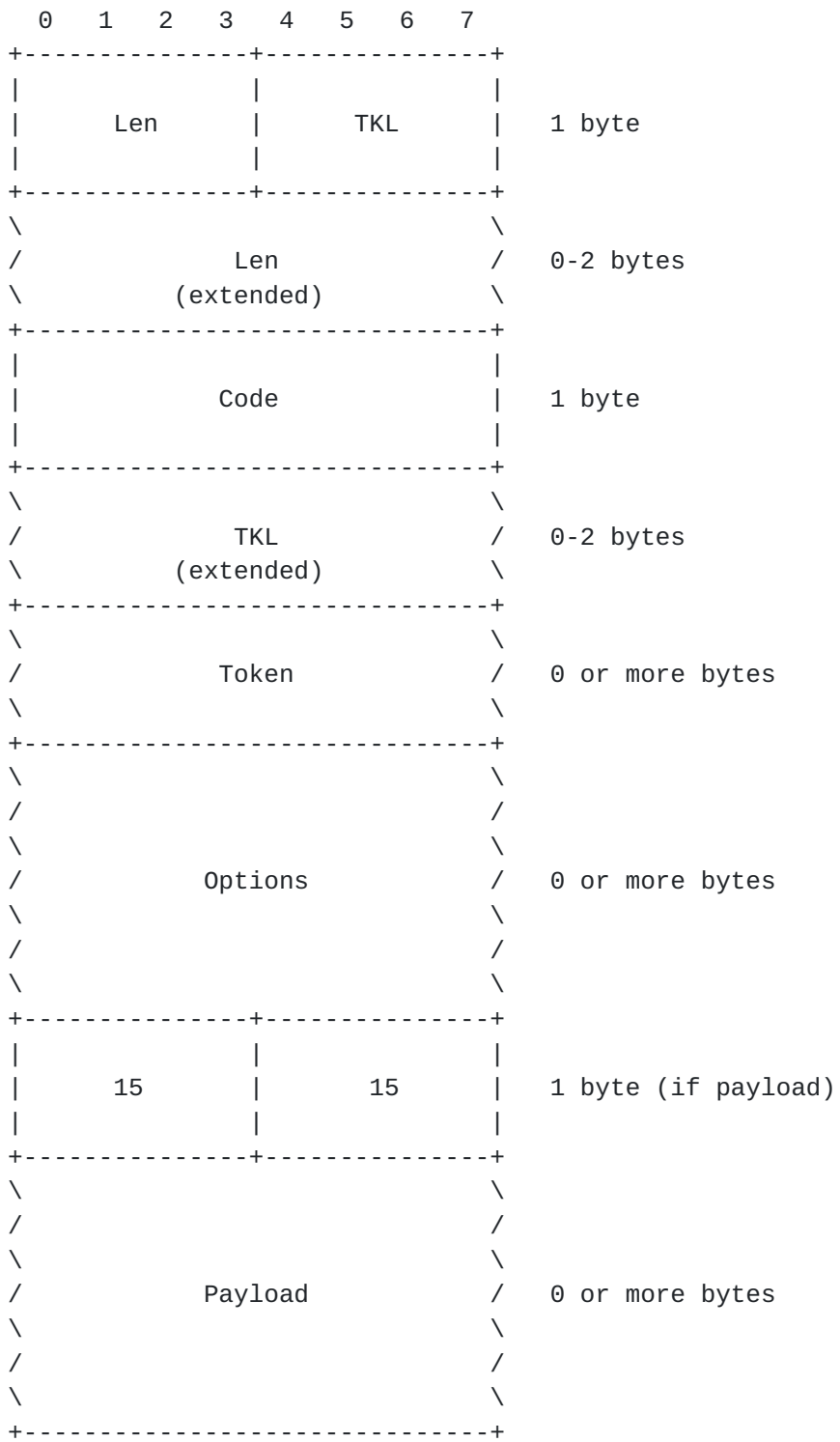
## Appendix A. Updated Message Formats

This appendix illustrates the CoAP message formats updated with the new definition of the TKL field (Section 2).

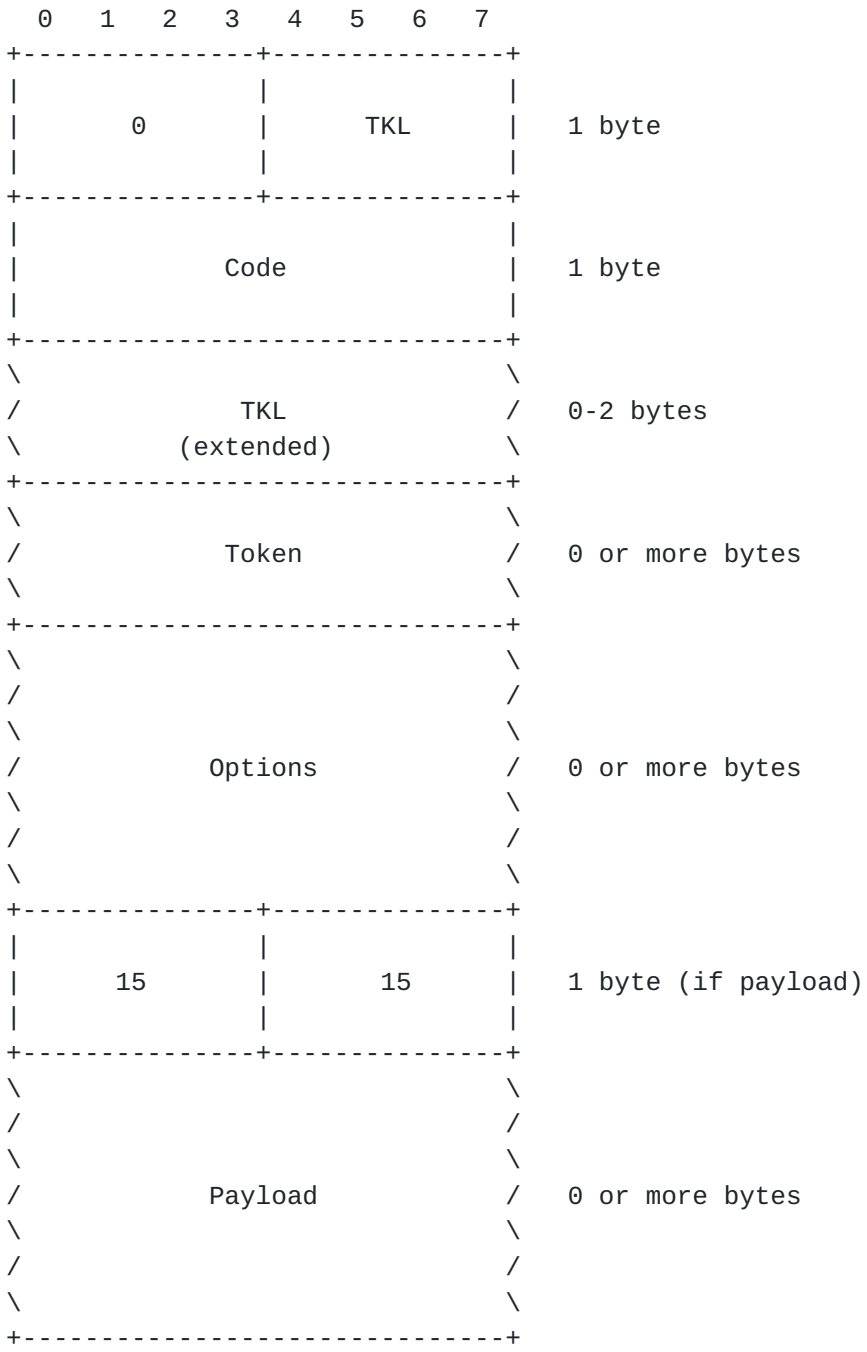
### A.1. CoAP over UDP



### A.2. CoAP over TCP



### A.3. CoAP over WebSockets



## Acknowledgements

This document is based on the requirements of and work on the Minimal Security Framework for 6TiSCH [I-D.ietf-6tisch-minimal-security] by Malisa Vucinic, Jonathan Simon, Kris Pister, and Michael Richardson.

Thanks to Carsten Bormann, Ari Keranen, John Mattsson, Jim Schaad, Goeran Selander, and Malisa Vucinic for helpful comments and discussions that have shaped the document.

## Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm  SE-16483
Sweden

Email: klaus.hartke@ericsson.com