Internet Engineering Task Force (IETF)
Request for Comments: 8974
Updates: 7252, 8323
Category: Standards Track
ISSN: 2070-1721

K. Hartke
Ericsson
M. Richardson
Sandelman

January 2021

Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)

Abstract

This document provides considerations for alleviating Constrained Application Protocol (CoAP) clients and intermediaries of keeping per-request state. To facilitate this, this document additionally introduces a new, optional CoAP protocol extension for extended token lengths.

This document updates RFCs 7252 and 8323 with an extended definition of the "TKL" field in the CoAP message header.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8974.

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 satisfy the requests by returning responses.

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

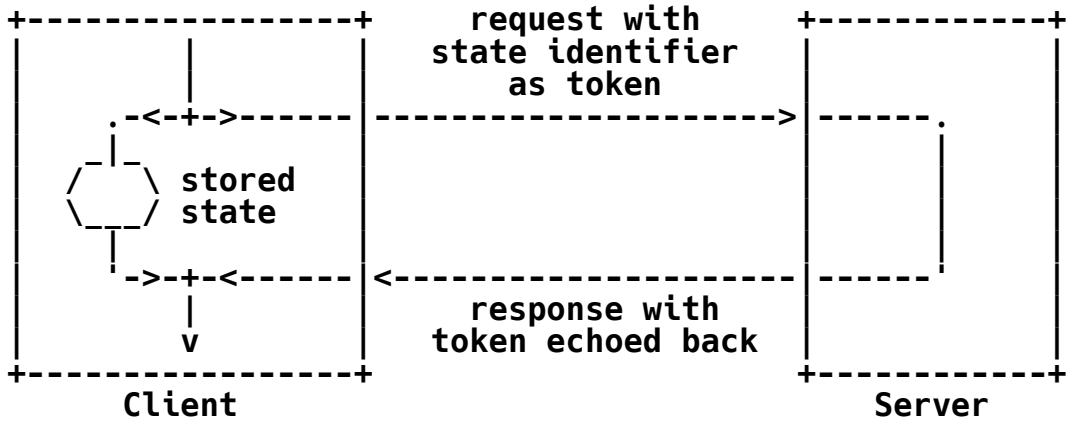


Figure 1: Token as an Identifier for Request State

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

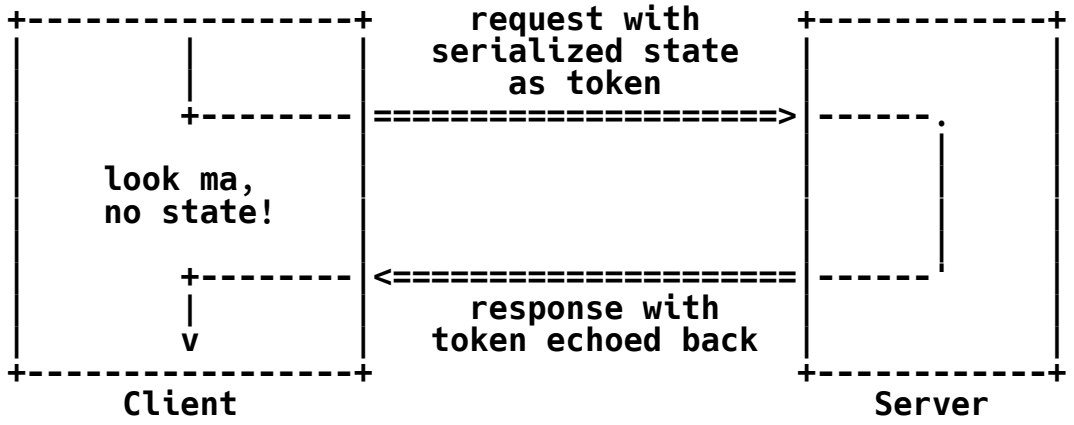


Figure 2: Token as Serialization of Request State

Section 3 of this document provides considerations for clients becoming "stateless" in this way. (The term "stateless" is in quotes here, because it's a bit oversimplified. Such clients still need to maintain per-server state and other kinds of state. So it would be more accurate to just say that the clients are avoiding per-request state.)

Section 4 of this document extends the considerations for clients to intermediaries, which may want to avoid keeping state for not only the requests they send to servers but also the requests they receive from clients.

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

While the use case (avoiding per-request state) and the mechanism (extended token lengths) presented in this document are closely related, each can be used independently of the other. Some implementations may be able to fit their state in just 8 bytes; some implementations may have other use cases for extended token lengths.

1.1. Terminology

In this document, the term "stateless" refers to an implementation strategy for a client (or intermediary in the client role) that does not require it to 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 (e.g., for

token generation, message retransmission, and congestion control).

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

This document updates the message formats defined for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with a new definition of the "TKL" field.

2.1. Extended Token Length (TKL) Field

The definition of the "TKL" field is updated as follows:

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. The other three values are reserved for special constructs:

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

14: A 16-bit unsigned integer in network byte order directly 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 definitions.

The updated message formats are illustrated in Appendix A.

The new definition of the "TKL" field increases the maximum token length that can be represented in a message to 65804 bytes. However, the maximum token length that sender and recipient implementations support may be shorter. For example, a constrained node of Class 1 [RFC7228] might support extended token lengths only up to 32 bytes.

In CoAP over UDP, it is often beneficial to keep CoAP messages small enough to avoid IP fragmentation. The maximum practical token length may therefore also be influenced by the Path MTU (PMTU). See Section 4.6 of [RFC7252] for details.

2.2. Discovering Support

Extended token lengths require support from server implementations. Support can be discovered by a client implementation in one of two ways:

* Where Capabilities and Settings Messages (CSMs) are available,

such as in CoAP over TCP, support can be discovered using the Extended-Token-Length 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-Length Capability Option

A server can use the elective Extended-Token-Length Capability Option to indicate the maximum token length it can accept in requests.

#	C	R	Applies to	Name	Format	Length	Base Value
6			CSM	Extended-Token-Length	uint	0-3	8

Table 1: The Extended-Token-Length Capability Option

C=Critical, R=Repeatable

As per Section 3 of [RFC7252], the base value (and the value used when this option is not implemented) is 8.

The active value of the Extended-Token-Length Option is replaced each time the option is sent with a modified value. Its starting value is its base value.

The option value MUST NOT be less than 8 or greater than 65804. If an option value less than 8 is received, the option MUST be ignored. If an option value greater than 65804 is received, the option value MUST be set to 65804.

Any option value greater than 8 implies support for the new definition of the "TKL" field specified in Section 2.1. Indication of support by a server does not oblige a client to actually make use of token lengths greater than 8.

If a server receives a request with a token of a length greater than what it indicated in its Extended-Token-Length Option, it MUST handle the request as a message-format error.

If a server receives a request with a token of a length less than, or equal to, what it indicated in its Extended-Token-Length Option but is unwilling or unable to handle the token at that time, it MUST NOT handle the request as a message-format error. Instead, it SHOULD return a 5.03 (Service Unavailable) response.

The Extended-Token-Length Capability Option does not apply to responses. The sender of a request is simply expected not to use a token of a length greater than it is willing to accept in a response.

2.2.2. Trial and Error

A server implementation that does not support the updated definition

of the "TKL" field specified in Section 2.1 will consider a request with a "TKL" field value outside the range 0 to 8 to be a message-format error and reject it (Section 3 of [RFC7252]). A client can therefore determine support by sending a request with an extended token length and checking whether or not it is rejected by the server.

In CoAP over UDP, the way a request message is rejected depends on the message type. A Confirmable message with a message-format error is rejected with a Reset message (Section 4.2 of [RFC7252]). A Non-confirmable message with a message-format error is either rejected with a Reset message or just silently ignored (Section 4.3 of [RFC7252]). To reliably get a Reset message, it is therefore REQUIRED that clients use a Confirmable message for determining support.

As per RFC 7252, Reset messages are empty and do not contain a token; they only return the Message ID (Figure 3). They also do not contain any indication of what caused a message-format error. To avoid any ambiguity, it is therefore RECOMMENDED that clients use a request that has no potential message-format error other than the extended token length.

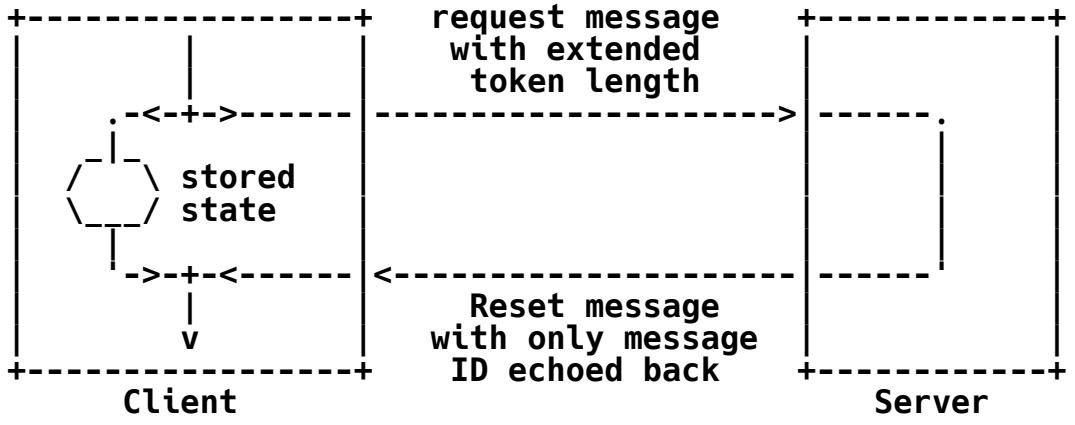


Figure 3: A Confirmable Request with an Extended Token Is Rejected with a Reset Message If the Server Does Not Have Support

An example of a suitable request is a GET request in a Confirmable message that includes only an If-None-Match option and a token of the greatest length that the client intends to use. Any response with the same token echoed back indicates that tokens up to that length are supported by the server.

Since network addresses may change, a client SHOULD NOT assume that extended token lengths are supported by a server for an unlimited duration. Unless additional information is available, the client should assume that addresses (and therefore extended token lengths) are valid for a minimum of 1800 s and a maximum of 86400 s (1 day). A client may use additional forms of input into this determination. For instance, a client may assume a server that is in the same subnet as the client has a similar address lifetime as the client. The client may use DHCP lease times or Router Advertisements to set the limits. For servers that are not local, if the server was looked up

using DNS, then the DNS resource record will have a Time To Live (TTL), and the extended token length should be kept for only that amount of time.

If a server supports extended token lengths but receives a request with a token of a length it is unwilling or unable to handle, it MUST NOT reject the message, as that would imply that extended token lengths are not supported at all. Instead, if the server cannot handle the request at the time, it SHOULD return a 5.03 (Service Unavailable) response; if the server will never be able to handle the request (e.g., because the token is too large), it SHOULD return a 4.00 (Bad Request) response.

> Design Note: The requirement to return an error response when a token cannot be handled might seem somewhat contradictory, as returning the error response requires the server also to return the token it cannot handle. However, processing a request usually involves a number of steps from receiving the message to passing it to application logic. The idea is that a server implementing this extension supports large tokens at least in its first few processing steps, enough to return an error response rather than a Reset message.

> Design Note: To prevent the trial-and-error-based discovery from becoming too complicated, no effort is made to indicate the maximum supported token length. A client implementation would probably already choose the shortest token possible for the task (such as being stateless, as described in Section 3), so it would probably not be able to reduce the length any further anyway should a server indicate a lower limit.

2.3. Intermediaries

Tokens are a hop-by-hop feature: if there are one or more intermediaries between a client and a server, every token is scoped to the exchange between a node in the client role and the node in the server role that it is immediately interacting with.

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 in its request to satisfy a request with an extended token length. Discovery needs to be performed for each hop where extended token lengths are to be used.

3. Stateless Clients

A client can be alleviated of keeping per-request state as follows:

1. The client serializes (parts of) its per-request state into a sequence of bytes and sends those bytes as the token of its request to the server.

2. The server returns the token verbatim in the response to the client, which allows the client to recover the state and process

the response as if it had kept the state locally.

As servers are just expected to return any token verbatim to the client, this implementation strategy for clients does not impact the interoperability of client and server implementations. However, there are a number of significant, nonobvious implications (e.g., related to security and other CoAP protocol features) that client implementations need take into consideration.

The following subsections discuss some of these considerations.

3.1. Serializing Client State

The format of the serialized state is generally an implementation detail of the client and opaque to the server. However, serialized state information is an attractive target for both unwanted nodes (e.g., on-path attackers) and wanted nodes (e.g., any configured forward proxy) on the path. The serialization format therefore needs to include security measures such as the following:

* A client SHOULD protect the integrity of the state information serialized in a token.

* Even when the integrity of the serialized state is protected, an attacker may still replay a response, making the client believe it sent the same request twice. For this reason, the client SHOULD implement replay protection (e.g., by using sequence numbers and a replay window). For replay protection, integrity protection is REQUIRED.

* If processing a response without keeping request state is sensitive to the time elapsed since sending the request, then the client SHOULD include freshness information (e.g., a timestamp) in the serialized state and reject any response where the freshness information is insufficiently fresh.

* Information in the serialized state may be privacy sensitive. A client SHOULD encrypt the serialized state if it contains privacy-sensitive information that an attacker would not get otherwise.

* When a client changes the format of the serialized state, it SHOULD prevent false interoperability with the previous format (e.g., by changing the key used for integrity protection or changing a field in the serialized state).

3.2. Using Extended Tokens

A client that depends on support for extended token lengths (Section 2) from the server to avoid keeping request state needs to 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 server does not support extended tokens, then no error message could be processed, since the state would neither be

present at the client nor returned in the Reset message (Figure 5).

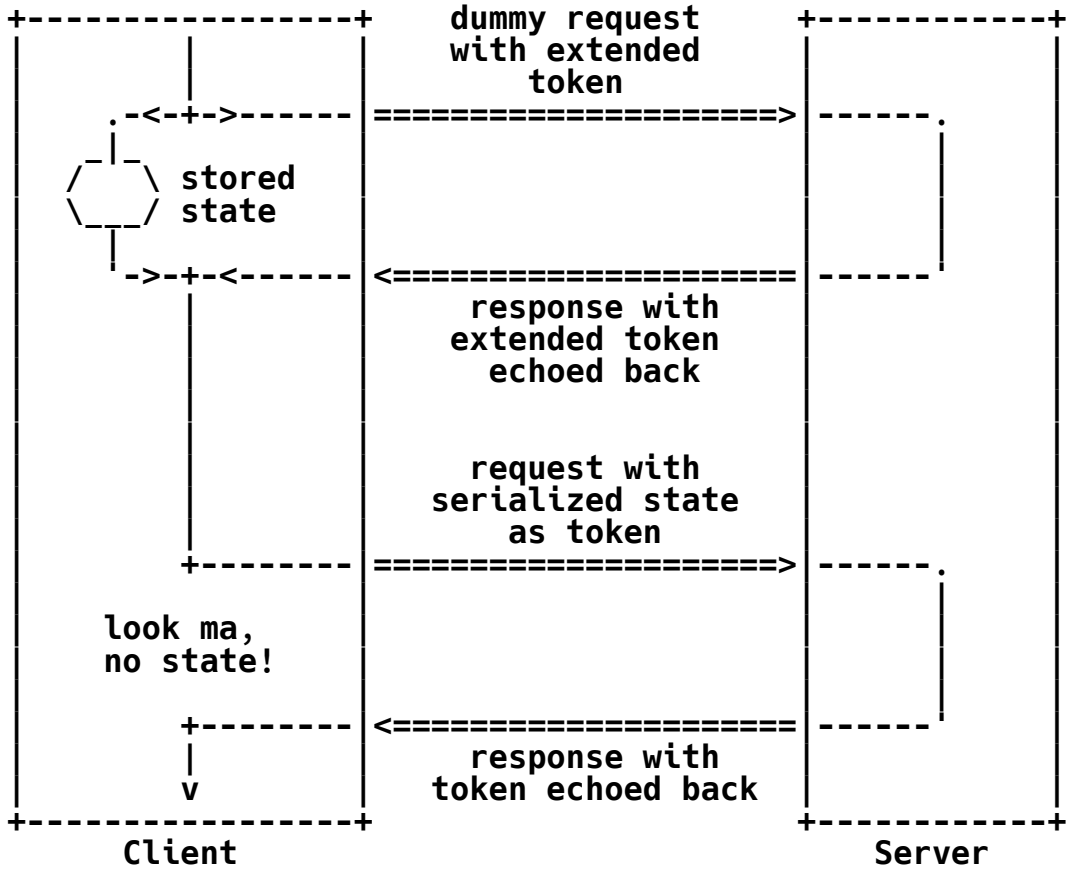


Figure 4: Depending on Extended Tokens for Being Stateless First Requires a Successful Stateful Discovery of Support

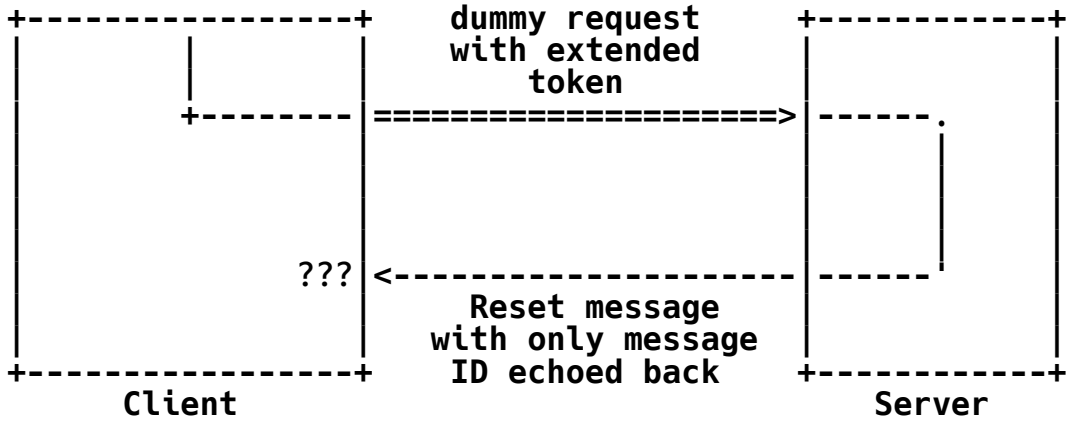


Figure 5: Stateless Discovery of Support Does Not Work

In environments where support can be reliably discovered through some other means, the discovery of support is OPTIONAL. An example for this is the Constrained Join Protocol (CoJP) in a 6TiSCH network [6TISCH-MIN-SEC], where support for extended tokens is required from all relevant parties.

3.3. Transmitting Messages

In CoAP over UDP [RFC7252], a client has the choice between Confirmable and Non-confirmable messages for requests. When using Non-confirmable messages, a client does not have to keep any message-exchange state, which can help in the goal of avoiding state. When using Confirmable messages, a client needs to keep message-exchange state for performing retransmissions and handling Acknowledgement and Reset messages, however. Non-confirmable messages are therefore better suited for avoiding state. In any case, a client still needs to keep congestion-control state, i.e., maintain state for each node it communicates with and enforce limits like NSTART.

As per Section 5.2 of [RFC7252], a client must be prepared to receive a response as a piggybacked response, a separate response, or a Non-confirmable response, regardless of the message type used for the request. A stateless client MUST handle these response types as follows:

* If a piggybacked response passes the checks for token integrity and freshness (Section 3.1), the client processes the message as specified in RFC 7252; otherwise, it processes the acknowledgement portion of the message as specified in RFC 7252 and silently discards the response portion.

* If a separate response passes the checks for token integrity and freshness, the client processes the message as specified in RFC 7252; otherwise, it rejects the message as specified in Section 4.2 of [RFC7252].

* If a Non-confirmable response passes the checks for token integrity and freshness, the client processes the message as specified in RFC 7252; otherwise, it rejects the message as specified in Section 4.3 of [RFC7252].

4. Stateless 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 towards the origin server and waits for the response. When the intermediary receives the response, it looks up the client's token and transport address for the received request and sends an appropriate response to the client.

An intermediary might want to be "stateless" not only in its role as a client but also in its role as a server, i.e., be alleviated of storing the client information for the requests it receives.

Such an intermediary can be implemented by serializing the client information along with the request state into the token towards the origin server. When the intermediary receives the response, it can recover the client information from the token and use it to satisfy the client's request; therefore, the intermediary doesn't need to store the information itself.

The following subsections discuss some considerations for this

approach.

4.1. Observing Resources

One drawback of the approach is that an intermediary, without keeping request state, is unable to aggregate multiple requests for the same target resource, which can significantly reduce efficiency. In particular, when clients observe [RFC7641] the same resource, aggregating requests is REQUIRED (Section 3.1 of [RFC7641]). This requirement cannot be satisfied without keeping request state.

Furthermore, an intermediary that does not keep track of the clients observing a resource is not able to determine whether these clients are still interested in receiving further notifications (Section 3.5 of [RFC7641]) or want to cancel an observation (Section 3.6 of [RFC7641]).

Therefore, an intermediary MUST NOT include an Observe Option in requests it sends without keeping both the request state for the requests it sends and the client information for the requests it receives.

4.2. Block-Wise Transfers

When using block-wise transfers [RFC7959], a server might not be able to distinguish blocks originating from different clients once they have been forwarded by an intermediary. Intermediaries need to ensure that this does not lead to inconsistent resource state by keeping distinct block-wise request operations on the same resource apart, e.g., utilizing the Request-Tag Option [ECHO-REQUEST-TAG].

4.3. Gateway Timeouts

As per Section 5.7.1 of [RFC7252], an intermediary is REQUIRED to return a 5.04 (Gateway Timeout) response if it cannot obtain a response within a timeout. However, if an intermediary does not keep the client information for the requests it receives, it cannot return such a response. Therefore, in this case, the gateway cannot return such a response and as such cannot implement such a timeout.

4.4. Extended Tokens

A client may make use of extended token lengths in a request to an intermediary that wants to be "stateless". This means that such an intermediary may have to serialize potentially very large client information into its token towards the origin server. The tokens can grow even further when it progresses along a chain of intermediaries that all want to be "stateless".

Intermediaries SHOULD limit the size of client information they are serializing into their own tokens. An intermediary can do this, for example, by limiting the extended token lengths it accepts from its clients (see Section 2.2) or by keeping the client information locally when the client information exceeds the limit (i.e., not being "stateless").

5. Security Considerations

5.1. Extended Tokens

Tokens significantly larger than the 8 bytes specified in RFC 7252 have implications -- in particular, for nodes 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 need to be prepared to handle such messages.

5.2. Stateless Clients and Intermediaries

Transporting the state needed by a client to process a response as serialized state information in the token has several significant and nonobvious security and privacy implications that need to be mitigated; see Section 3.1 for recommendations.

In addition to the format requirements outlined there, implementations need to ensure that they are not vulnerable to maliciously crafted, delayed, or replayed tokens.

It is generally expected that the use of encryption, integrity protection, and replay protection for serialized state is appropriate.

In the absence of integrity and replay protection, an on-path attacker or rogue server/intermediary could return a state (either one modified in a reply, or an unsolicited one) that could alter the internal state of the client.

It is for this reason that at least the use of integrity protection on the token is always recommended.

It may be that in some very specific cases, as a result of a careful and detailed analysis of any potential attacks, it is decided that such cryptographic protections do not add value. The authors of this document have not found such a use case as yet, but this is a local decision.

It should further be emphasized that the encrypted state is created by the sending node and decrypted by the same node when receiving a response. The key is not shared with any other system. Therefore, the choice of encryption scheme and the generation of the key for this system is purely a local matter.

When encryption is used, the use of AES-CCM [RFC3610] with a 64-bit tag is recommended, combined with a sequence number and a replay window. This choice is informed by available hardware acceleration of on many constrained systems. If a different algorithm is available accelerated on the sender, with similar or stronger strength, then it SHOULD be preferred. Where privacy of the state is not required, and encryption is not needed, HMAC-SHA-256 [RFC6234], combined with a sequence number and a replay window, may be used.

This size of the replay window depends upon the number of requests that need to be outstanding. This can be determined from the rate at which new ones are made and the expected time period during which responses are expected.

For instance, given a CoAP MAX_TRANSMIT_WAIT of 93 s (Section 4.8.2 of [RFC7252]), any request that is not answered within 93 s will be considered to have failed. At a request rate of one request per 10 s, at most 10 (ceil(9.3)) requests can be outstanding at a time, and any convenient replay window larger than 20 will work. As replay windows are often implemented with a sliding window and a bit, the use of a 32-bit window would be sufficient.

For use cases where requests are being relayed from another node, the request rate may be estimated by the total link capacity allocated for that kind of traffic. An alternate view would consider how many IPv6 Neighbor Cache Entries (NCEs) the system can afford to allocate for this use.

When using an encryption mode that depends on a nonce, such as AES-CCM, repeated use of the same nonce under the same key causes the cipher to fail catastrophically.

If a nonce is ever used for more than one encryption operation with the same key, then the same key stream gets used to encrypt both plaintexts, and the confidentiality guarantees are voided. Devices with low-quality entropy sources -- as is typical with constrained devices, which incidentally happen to be a natural candidate for the stateless mechanism described in this document -- need to carefully pick a nonce-generation mechanism that provides the above uniqueness guarantee.

[RFC8613], Appendix B.1.1 ("Sender Sequence Number") provides a model for how to maintain nonrepeating nonces without causing excessive wear of flash memory.

6. IANA Considerations

6.1. CoAP Signaling Option Number

The following entry has been added to the "CoAP Signaling Option Numbers" registry within the "CoRE Parameters" registry.

Applies to	Number	Name	Reference
7.01	6	Extended-Token-Length	RFC 8974

Table 2: CoAP Signaling Option Number

7. References

7.1. Normative References

[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>.

7.2. Informative References

[6TISCH-MIN-SEC] Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Constrained Join Protocol (CoJP) for 6TiSCH", Work in Progress, Internet-Draft, draft-ietf-6tisch-minimal-security-15, 10 December 2019, <https://tools.ietf.org/html/draft-ietf-6tisch-minimal-security-15>.

[ECHO-REQUEST-TAG] Amsüss, C., Mattsson, J. P., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", Work in Progress, Internet-Draft, draft-ietf-core-echo-request-tag-11, 2 November 2020, <https://tools.ietf.org/html/draft-ietf-core-echo-request-tag-11>.

[RFC3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, DOI 10.17487/RFC3610, September 2003, <https://www.rfc-editor.org/info/rfc3610>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

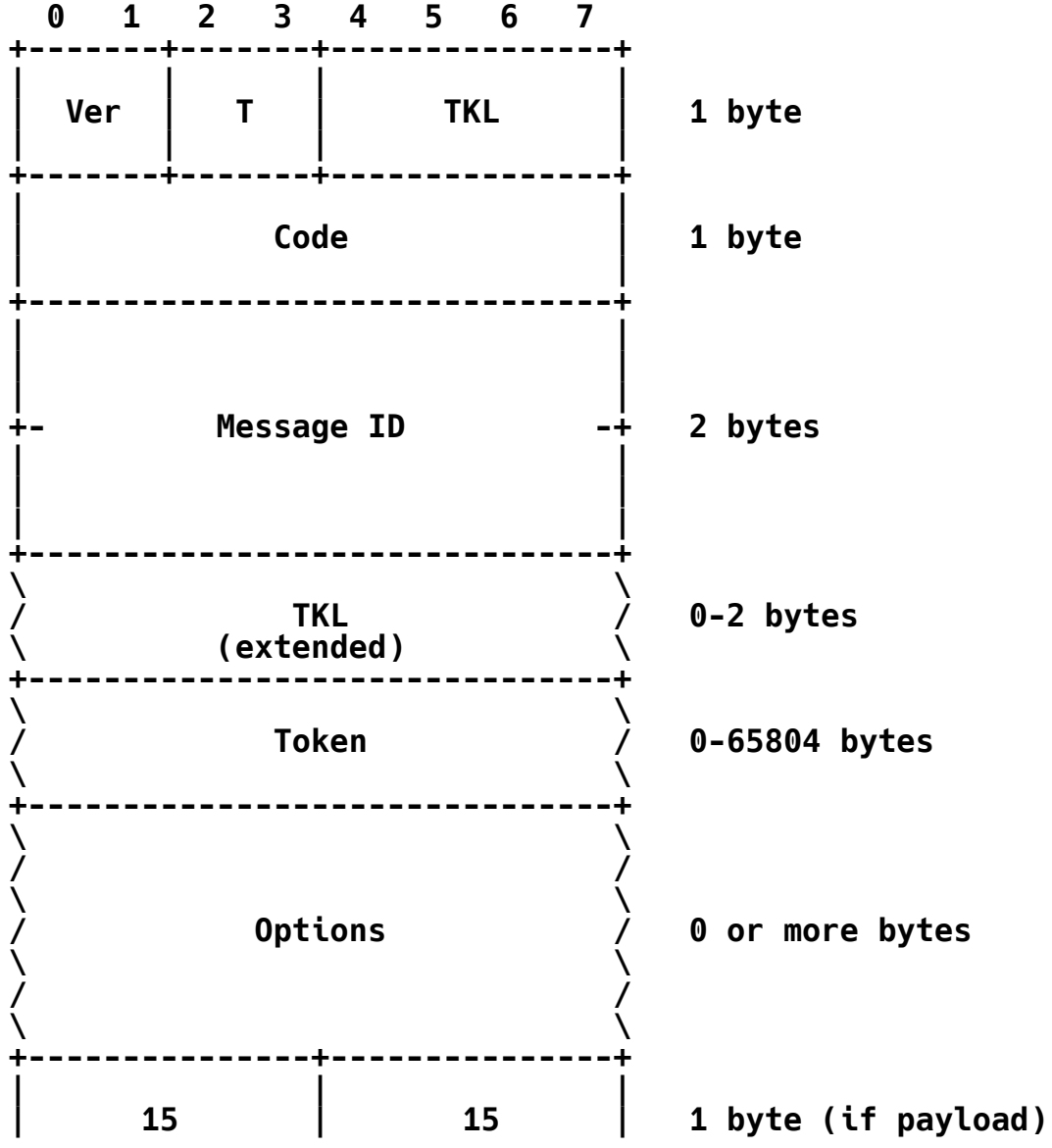
[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

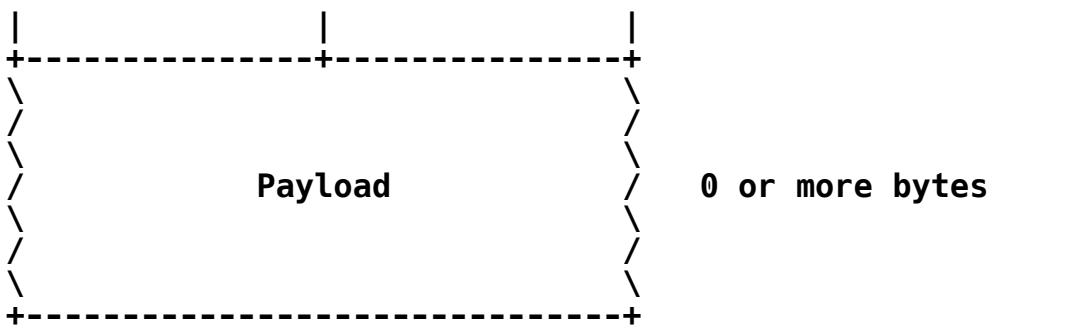
[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

Appendix A. Updated Message Formats

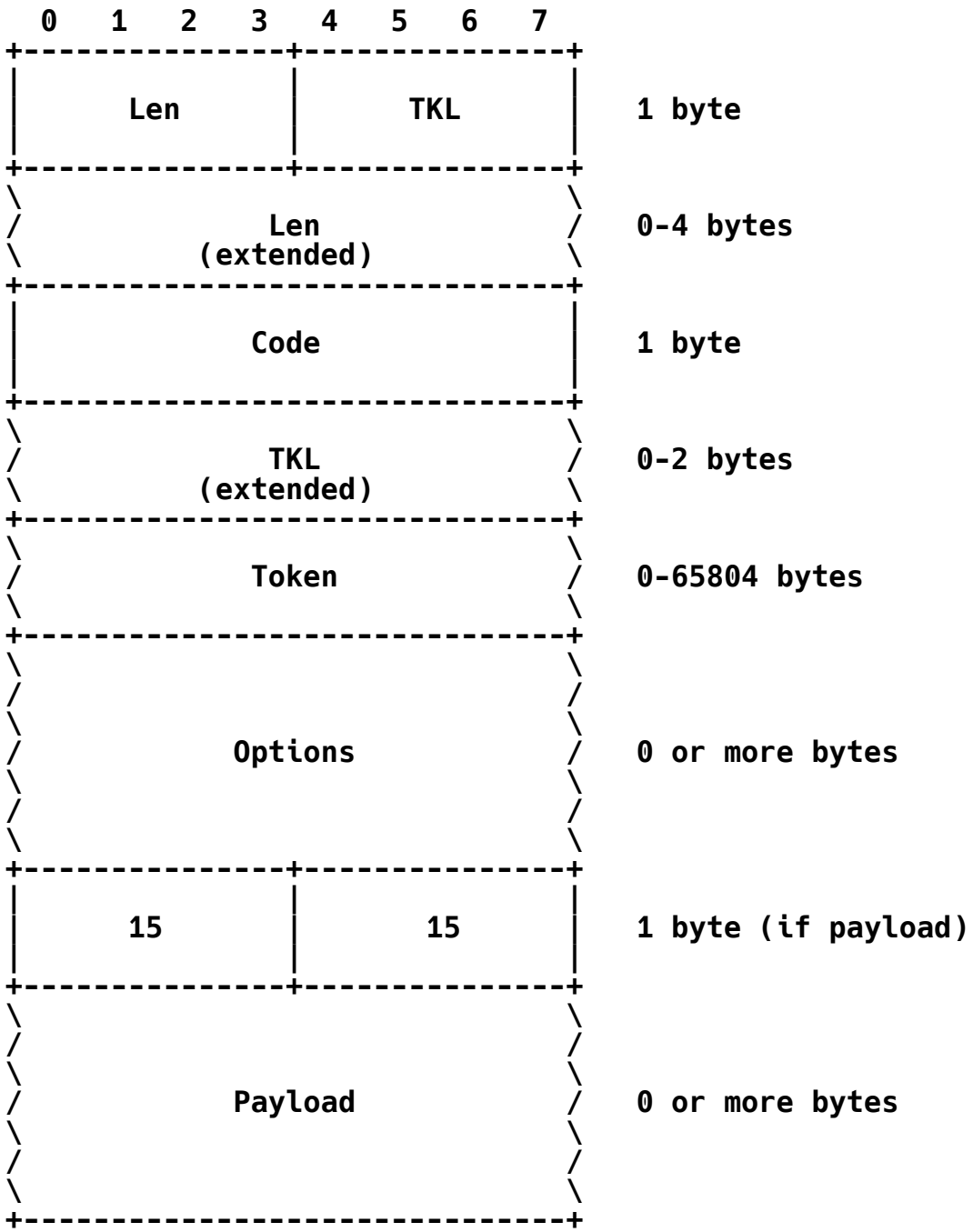
In Section 2, this document updates the CoAP message formats by specifying a new definition of the "TKL" field in the message header. As an alternative presentation of this update, this appendix shows the CoAP message formats for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with the new definition applied.

A.1. CoAP over UDP

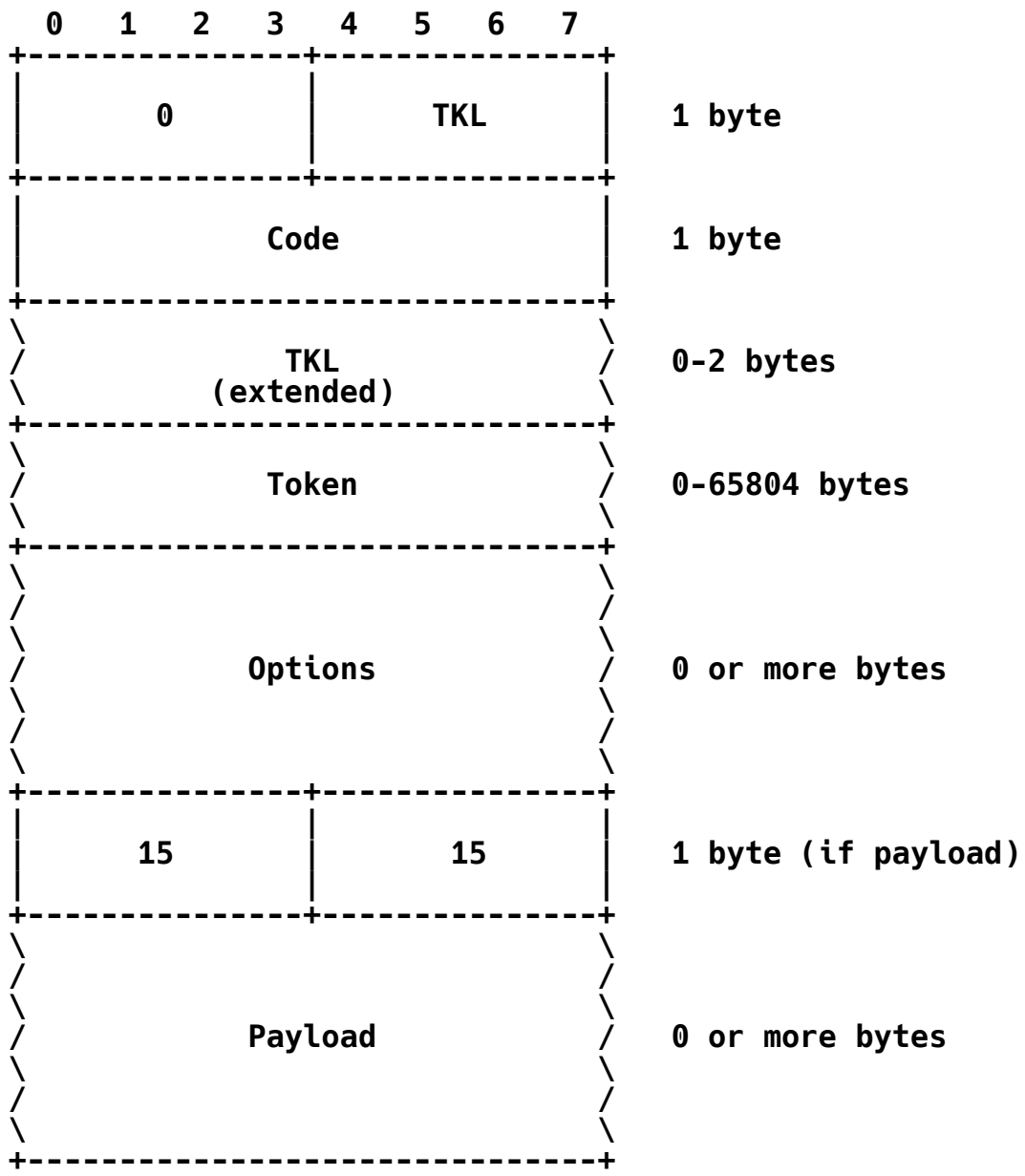




A.2. CoAP over TCP/TLS



A.3. CoAP over WebSockets



Acknowledgements

This document is based on the requirements of, and work on, "Constrained Join Protocol (CoJP) for 6TiSCH" (January 2020) by Mališa Vučinić, Jonathan Simon, Kris Pister, and Michael Richardson.

Thanks to Christian Amsüss, Carsten Bormann, Roman Danyliw, Christer Holmberg, Benjamin Kaduk, Ari Keränen, Erik Kline, Murray Kucherawy, Warren Kumari, Barry Leiba, David Mandelberg, Dan Romascanu, Jim Schaad, Göran Selander, Mališa Vučinić, Éric Vyncke, and Robert Wilton for helpful comments and discussions that have shaped the document.

Special thanks to John Mattsson for his contributions to the security considerations of the document, and to Thomas Fossati for his in-

depth review, copious comments, and suggested text.

Authors' Addresses

Klaus Hartke
Ericsson
Torshamnsgatan 23
SE-16483 Stockholm
Sweden

Email: klaus.hartke@ericsson.com

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/