Stream: Internet Engineering Task Force (IETF)
RFC: 9177
Category: Standards Track
Published: March 2022
ISSN: 2070-1721
Authors: M. Boucadair J. Shallow
Orange

RFC 9177
Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission

Abstract

This document specifies alternative Constrained Application Protocol (CoAP) block-wise transfer options: Q-Block1 and Q-Block2.

These options are similar to, but distinct from, the CoAP Block1 and Block2 options defined in RFC 7959. The Q-Block1 and Q-Block2 options are not intended to replace the Block1 and Block2 options but rather have the goal of supporting Non-confirmable (NON) messages for large amounts of data with fewer packet interchanges. Also, the Q-Block1 and Q-Block2 options support faster recovery should any of the blocks get lost in transmission.

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/rfc9177.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252], although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of CoAP over UDP includes support for reliable delivery, simple congestion control, and flow control. CoAP supports two message types (Section 1.2 of [RFC7252]): Confirmable (CON) and Non-confirmable (NON). Unlike NON messages, every CON message will elicit an acknowledgment or a reset.

The CoAP specification recommends that a CoAP message should fit within a single IP packet (i.e., avoid IP fragmentation). To handle data records that cannot fit in a single IP packet, [RFC7959] introduced the concept of block-wise transfers and the companion CoAP Block1 and Block2 options. However, this concept is designed to work exclusively with Confirmable messages (Section 1 of [RFC7959]). Note that the block-wise transfer was further updated by [RFC8323] for use over TCP, TLS, and WebSockets.

The CoAP Block1 and Block2 options work well in environments where there are no, or minimal, packet losses. These options operate synchronously, i.e., each individual block has to be requested. A CoAP endpoint can only ask for (or send) the next block when the transfer of the previous block has completed. The packet transmission rate, and hence the block transmission rate, is controlled by Round-Trip Times (RTTs).

There is a requirement for blocks of data larger than a single IP datagram to be transmitted under network conditions where there may be asymmetrical transient packet loss (e.g., acknowledgment responses may get dropped). An example is when a network is subject to a Distributed Denial of Service (DDoS) attack and there is a need for DDoS mitigation agents relying upon CoAP to communicate with each other (e.g., [RFC9132] [DOTS-TELEMETRY]). As a reminder, [RFC7959] recommends the use of CON responses to handle potential packet loss. However, such a recommendation does not work with a "flooded pipe" DDoS situation (e.g., [RFC9132]).

This document introduces the CoAP Q-Block1 and Q-Block2 options, which allow block-wise transfers to work with a series of Non-confirmable messages instead of lock-stepping using Confirmable messages (Section 3). In other words, this document provides a missing piece of [RFC7959], namely the support of block-wise transfers using Non-confirmable messages where an entire body of data can be transmitted without the requirement that intermediate acknowledgments be received from the peer (but recovery is available should it be needed).

Similar to [RFC7959], this specification does not remove any of the constraints posed by the base CoAP specification [RFC7252] it is strictly layered on top of.

2. Terminology

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.

Readers should be familiar with the terms and concepts defined in [RFC7252], [RFC7959], and [RFC8132]. Particularly, the document uses the following key concepts:

Token: used to match responses to requests independently from the underlying messages (Section 5.3.1 of [RFC7252]).

ETag: used as a resource-local identifier for differentiating between representations of the same resource that vary over time (Section 5.10.6 of [RFC7252]).

The terms "payload" and "body" are defined in [RFC7959]. The term "payload" is thus used for the content of a single CoAP message (i.e., a single block being transferred), while the term "body" is used for the entire resource representation that is being transferred in a block-wise fashion.

Request-Tag refers to an option that allows a CoAP server to match message fragments belonging to the same request [RFC9175].

MAX_PAYLOADS is the maximum number of payloads that can be transmitted at any one time.

MAX_PAYLOADS_SET is the set of blocks identified by block numbers that, when divided by MAX_PAYLOADS, have the same numeric result. For example, if MAX_PAYLOADS is set to 10, a MAX_PAYLOADS_SET could be blocks #0 to #9, #10 to #19, etc. Depending on the overall data size, there could be fewer than MAX_PAYLOADS blocks in the final MAX_PAYLOADS_SET.

3. Alternative CoAP Block-Wise Transfer Options

This document introduces the CoAP Q-Block1 and Q-Block2 options. These options are designed to work in particular with NON requests and responses.

Using NON messages, faster transmissions can occur, as all the blocks can be transmitted serially (akin to fragmented IP packets) without having to wait for a response or next request from the remote CoAP peer. Recovery of missing blocks is faster in that multiple missing blocks can be requested in a single CoAP message. Even if there is asymmetrical packet loss, a body can still be sent and received by the peer whether the body comprises a single payload or multiple payloads, assuming no recovery is required.

A CoAP endpoint can acknowledge all or a subset of the blocks. Concretely, the receiving CoAP endpoint either informs the sending CoAP endpoint of successful reception or reports on all blocks in the body that have not yet been received. The sending CoAP endpoint will then retransmit only the blocks that have been lost in transmission.

Note that similar transmission rate benefits can be applied to Confirmable messages if the value of NSTART is increased from 1 (Section 4.7 of [RFC7252]). However, the use of Confirmable messages will not work effectively if there is asymmetrical packet loss. Some examples with Confirmable messages are provided in Appendix A.

There is little, if any, benefit of using these options with CoAP running over a reliable connection [RFC8323]. In this case, there is no differentiation between CON and NON, as they are not used. Some examples using a reliable transport are provided in Appendix B.

The Q-Block1 and Q-Block2 options are similar in operation to the CoAP Block1 and Block2 options, respectively. They are not a replacement for them but have the following benefits:

- They can operate in environments where packet loss is highly asymmetrical.
- They enable faster transmissions of sets of blocks of data with fewer packet interchanges.
- They support faster recovery should any of the blocks get lost in transmission.
- They support sending an entire body using NON messages without requiring that an intermediate response be received from the peer.

The disadvantages of using the CoAP Block1 and Block2 options are as follows:

- There is a loss of lock-stepping, so payloads are not always received in the correct order (blocks in ascending order).
- Additional congestion control measures need to be put in place for NON messages (Section 7.2).
- To reduce the transmission times for CON transmissions of large bodies, NSTART needs to be increased from 1, but this affects congestion control and incurs a requirement to retune other parameters (Section 4.7 of [RFC7252]). Such tuning is out of scope of this document.
- Mixing of NON and CON during an exchange of requests/responses using Q-Block options is not supported.
- The Q-Block options do not support stateless operation/random access.
- Proxying of Q-Block options is limited to caching full representations.
- There is no multicast support.

The Q-Block1 and Q-Block2 options can be used instead of the Block1 and Block2 options when the different transmission properties are required. If the new options are not supported by a peer, then transmissions can fall back to using the Block1 and Block2 options (Section 4.1).

The deviations from the Block1 and Block2 options are specified in Section 4. Pointers to the appropriate sections in [RFC7959] are provided.

The specification refers to the base CoAP methods defined in Section 5.8 of [RFC7252] and the new CoAP methods, FETCH, PATCH, and iPATCH, which are introduced in [RFC8132].

The No-Response option [RFC7967] was considered but was abandoned, as it does not apply to Q-Block2 responses. A unified solution is defined in the document.

3.1. CoAP Response Code (4.08) Usage

This document adds a media type for the 4.08 (Request Entity Incomplete) response defining an additional message format for reporting on payloads using the Q-Block1 option that are not received by the server.

See Section 5 for more details.

3.2. Applicability Scope

The block-wise transfer specified in [RFC7959] covers the general case using Confirmable messages but falls short in situations where packet loss is highly asymmetrical or there is no need for an acknowledgment. In other words, there is a need for Non-confirmable support.

The mechanism specified in this document provides roughly similar features to the Block1/Block2 options. It provides additional properties that are tailored towards the intended use case of Non-confirmable transmission. Concretely, this mechanism primarily targets applications, such as DDoS Open Threat Signaling (DOTS), that cannot use CON requests/responses because of potential packet loss and that support application-specific mechanisms to assess whether the remote peer is not overloaded and thus is able to process the messages sent by a CoAP endpoint (e.g., DOTS heartbeats in Section 4.7 of [RFC9132]). Other use cases are when an application sends data but has no need for an acknowledgment of receipt and any data transmission loss is not critical.

The mechanism includes guards to prevent a CoAP agent from overloading the network by adopting an aggressive sending rate. These guards **MUST** be followed in addition to the existing CoAP congestion control, as specified in Section 4.7 of [RFC7252]. See Section 7 for more details.

Any usage outside the primary use case of Non-confirmable messages with block transfers should be carefully weighed against the potential loss of interoperability with generic CoAP applications (see the disadvantages listed in Section 3). It is hoped that the experience gained with this mechanism can feed future extensions of the block-wise mechanism that will both be generally applicable and serve this particular use case.

It is not recommended that these options are used in the "NoSec" security mode (Section 9 of [RFC7252]), as the source endpoint needs to be trusted. Using Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] does provide a security context and hence a trust of the source endpoint that prepared the inner OSCORE content. However, even with OSCORE, using the NoSec mode with these options may still be inadequate, for reasons discussed in Section 11.

4. The Q-Block1 and Q-Block2 Options

4.1. Properties of the Q-Block1 and Q-Block2 Options

The properties of the Q-Block1 and Q-Block2 options are shown in Table 1. The formatting of this table follows the one used in Table 4 of Section 5.10 of [RFC7252]. The C, U, N, and R columns indicate the properties Critical, UnSafe, NoCacheKey, and Repeatable, which are defined in

Section 5.4 of [RFC7252]. Only the Critical and UnSafe columns are marked for the Q-Block1 option. The Critical, UnSafe, and Repeatable columns are marked for the Q-Block2 option. As these options are UnSafe, NoCacheKey has no meaning and so is marked with a dash.

No.	C	U	N	R	Name	Format	Length	Default
19	x	x	-		Q-Block1	uint	0-3	(none)
31	x	x	-	x	Q-Block2	uint	0-3	(none)

Table 1: CoAP Q-Block1 and Q-Block2 Option Properties

The Q-Block1 and Q-Block2 options can be present in both the request and response messages. The Q-Block1 option pertains to the request payload, and the Q-Block2 option pertains to the response payload. When the Content-Format option is present together with the Q-Block1 or Q-Block2 option, the option applies to the body, not to the payload (i.e., it must be the same for all payloads of the same body).

The Q-Block1 option is useful with the payload-bearing (e.g., POST, PUT, FETCH, PATCH, and iPATCH) requests and their responses.

The Q-Block2 option is useful, for example, with GET, POST, PUT, FETCH, PATCH, and iPATCH requests and their payload-bearing responses (response codes 2.01, 2.02, 2.04, and 2.05) (Section 5.5 of [RFC7252]).

A CoAP endpoint (or proxy) **MUST** support either both or neither of the Q-Block1 and Q-Block2 options.

If the Q-Block1 option is present in a request or the Q-Block2 option is returned in a response, this indicates a block-wise transfer and describes how this specific block-wise payload forms part of the entire body being transferred. If it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed.

To indicate support for Q-Block2 responses, the CoAP client **MUST** include the Q-Block2 option in a GET or similar request (e.g., FETCH), the Q-Block2 option in a PUT or similar request (e.g., POST), or the Q-Block1 option in a PUT or similar request so that the server knows that the client supports this Q-Block functionality should it need to send back a body that spans multiple payloads. Otherwise, the server would use the Block2 option (if supported) to send back a message body that is too large to fit into a single IP packet [RFC7959].

How a client decides whether it needs to include a Q-Block1 or Q-Block2 option can be driven by a local configuration parameter, triggered by an application (e.g., DOTS), etc. Such considerations are out of the scope of this document.

Implementation of the Q-Block1 and Q-Block2 options is intended to be optional. However, when a Q-Block1 or Q-Block2 option is present in a CoAP message, it **MUST** be processed (or the message rejected). Therefore, the Q-Block1 and Q-Block2 options are identified as critical options.

With CoAP over UDP, the way a request message is rejected for critical options depends on the message type. A Confirmable message with an unrecognized critical option is rejected with a 4.02 (Bad Option) response (Section 5.4.1 of [RFC7252]). A Non-confirmable message with an unrecognized critical option is either rejected with a Reset message or just silently ignored (Sections 5.4.1 and 4.3 of [RFC7252]). To reliably get a rejection message, it is therefore **REQUIRED** that clients use a Confirmable message for determining support for the Q-Block1 and Q-Block2 options. This Confirmable message can be sent under the base CoAP congestion control setup specified in Section 4.7 of [RFC7252] (that is, NSTART does not need to be increased (Section 7.1)).

The Q-Block1 and Q-Block2 options are unsafe to forward. That is, a CoAP proxy that does not understand the Q-Block1 (or Q-Block2) option must reject the request or response that uses either option (see Section 5.7.1 of [RFC7252]).

The Q-Block2 option is repeatable when requesting retransmission of missing blocks but not otherwise. Except for that case, any request carrying multiple Q-Block1 (or Q-Block2) options **MUST** be handled following the procedure specified in Section 5.4.5 of [RFC7252].

The Q-Block1 and Q-Block2 options, like the Block1 and Block2 options, are of both class E and class U for OSCORE processing (Table 2). The Q-Block1 (or Q-Block2) option **MAY** be an Inner or Outer option (Section 4.1 of [RFC8613]). The Inner and Outer values are therefore independent of each other. The Inner option is encrypted and integrity protected between clients and servers and provides message body identification in case of end-to-end fragmentation of requests. The Outer option is visible to proxies and labels message bodies in case of hop-by-hop fragmentation of requests.

Number	Name	E	U
19	Q-Block1	x	x
31	Q-Block2	x	x

Table 2: OSCORE Protection of the Q-Block1 and Q-Block2 Options

Note that, if the Q-Block1 or Q-Block2 options are included in a packet as Inner options, the Block1 or Block2 options **MUST NOT** be included as Inner options. Similarly, there **MUST NOT** be a mix of Q-Block and Block options for the Outer options. Messages that do not adhere to this behavior **MUST** be rejected with a 4.02 (Bad Option). The Q-Block and Block options can be mixed across Inner and Outer options, as these are handled independently of each other. For clarity, if OSCORE is not being used, there **MUST NOT** be a mix of Q-Block and Block options in the same packet.

4.2. Structure of the Q-Block1 and Q-Block2 Options

The structure of the Q-Block1 and Q-Block2 options follows the structure defined in Section 2.2 of [RFC7959].

There is no default value for the Q-Block1 and Q-Block2 options. The absence of one of these options is equivalent to an option value of 0 with respect to the value of block number (NUM) and more bit (M) that could be given in the option, i.e., it indicates that the current block is the first and only block of the transfer (block number is set to 0; M is unset). However, in contrast to the explicit value 0, which would indicate a size of the block (SZX) of 0, and thus a size value of 16 bytes, there is no specific size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length option; therefore, this is semantically different from the absence of the option.)

4.3. Using the Q-Block1 Option

The Q-Block1 option is used when the client wants to send a large amount of data to the server using the POST, PUT, FETCH, PATCH, or iPATCH methods where the data and headers do not fit into a single packet.

When the Q-Block1 option is used, the client **MUST** include a Request-Tag option [RFC9175]. The Request-Tag value **MUST** be the same for all of the requests for the body of data that is being transferred. The Request-Tag is opaque, but the client **MUST** ensure that it is unique for every different body of transmitted data.

> Implementation Note: It is suggested that the client treats the Request-Tag as an unsigned integer of 8 bytes in length. An implementation may want to consider limiting this to 4 bytes to reduce packet overhead size. The initial Request-Tag value should be randomly generated and then subsequently incremented by the client whenever a new body of data is being transmitted between peers.

Section 4.6 discusses the use of the Size1 option.

For Confirmable transmission, the server continues to acknowledge each packet, but a response is not required (whether separate or piggybacked) until successful receipt of the body by the server. For Non-confirmable transmission, no response is required until either the successful receipt of the body by the server or a timer expires with some of the payloads having not yet arrived. In the latter case, a "retransmit missing payloads" response is needed. For reliable transports (e.g., [RFC8323]), a response is not required until successful receipt of the body by the server.

Each individual message that carries a block of the body is treated as a new request (Section 6).

The client **MUST** send the payloads in order of increasing block number, starting from zero, until the body is complete (subject to any congestion control (Section 7)). In addition, any missing payloads requested by the server must be separately transmitted with increasing block numbers.

The following response codes are used:

2.01 (Created)
: This response code indicates successful receipt of the entire body and that the resource was created. The token to use **MUST** be one of the tokens that were received in a request for this block-wise exchange. However, it is desirable to provide the one used in the last received request, since that will aid any troubleshooting. The client should then release all of the tokens used for this body. Note that the last received payload might not be the one with the highest block number.

2.02 (Deleted)
: This response code indicates successful receipt of the entire body and that the resource was deleted when using POST (Section 5.8.2 of [RFC7252]). The token to use **MUST** be one of the tokens that were received in a request for this block-wise exchange. However, it is desirable to provide the one used in the last received request. The client should then release all of the tokens used for this body.

2.04 (Changed)
: This response code indicates successful receipt of the entire body and that the resource was updated. The token to use **MUST** be one of the tokens that were received in a request for this block-wise exchange. However, it is desirable to provide the one used in the last received request. The client should then release all of the tokens used for this body.

2.05 (Content)
: This response code indicates successful receipt of the entire FETCH request body (Section 2 of [RFC8132]) and that the appropriate representation of the resource is being returned. The token to use **MUST** be one of the tokens that were received in a request for this block-wise exchange. However, it is desirable to provide the one used in the last received request.

  If the FETCH request includes the Observe option, then the server **MUST** use the same token as used for the 2.05 (Content) response for returning any triggered Observe responses so that the client can match them up.

  The client should then release all of the tokens used for this body apart from the one used for tracking an observed resource.

2.31 (Continue)
: This response code can be used to indicate that all of the blocks up to and including the Q-Block1 option block NUM (all having the M bit set) have been successfully received. The token to use **MUST** be one of the tokens that were received in a request for this latest MAX_PAYLOADS_SET block-wise exchange. However, it is desirable to provide the one used in the last received request.

  The client should then release all of the tokens used for this MAX_PAYLOADS_SET.

  A response using this response code **MUST NOT** be generated for every received Q-Block1 option request. It **SHOULD** only be generated when all the payload requests are Non-confirmable and a MAX_PAYLOADS_SET has been received by the server. More details about the motivations for this optimization are discussed in Section 7.2.

This response code **SHOULD NOT** be generated for CON, as this may cause duplicated payloads to unnecessarily be sent.

4.00 (Bad Request)
: This response code **MUST** be returned if the request does not include a Request-Tag option or a Size1 option but does include a Q-Block1 option.

4.02 (Bad Option)
: This response code **MUST** be returned for a Confirmable request if the server does not support the Q-Block options. Note that a Reset message may be sent in case of a Non-confirmable request.

4.08 (Request Entity Incomplete)
: As a reminder, this response code returned without content type "application/missing-blocks+cbor-seq" (Section 12.3) is handled as in Section 2.9.2 of [RFC7959].

  This response code returned with content type "application/missing-blocks+cbor-seq" indicates that some of the payloads are missing and need to be resent. The client then retransmits the individual missing payloads using the same Request-Tag, Size1, and Q-Block1 options to specify the same NUM, SZX, and M bit values as those sent initially in the original (but not received) packets.

  The Request-Tag value to use is determined by taking the token in the 4.08 (Request Entity Incomplete) response, locating the matching client request, and then using its Request-Tag.

  The token to use in the 4.08 (Request Entity Incomplete) response **MUST** be one of the tokens that were received in a request for this block-wise body exchange. However, it is desirable to provide the one used in the last received request. See Section 5 for further information.

  If the server has not received all the blocks of a body, but one or more NON payloads have been received, it **SHOULD** wait for NON_RECEIVE_TIMEOUT (Section 7.2) before sending a 4.08 (Request Entity Incomplete) response.

4.13 (Request Entity Too Large)
: This response code can be returned under conditions similar to those discussed in Section 2.9.3 of [RFC7959].

  This response code can be returned if there is insufficient space to create a response PDU with a block size of 16 bytes (SZX = 0) to send back all the response options as appropriate. In this case, the Size1 option is not included in the response.

Further considerations related to the transmission timings of the 4.08 (Request Entity Incomplete) and 2.31 (Continue) response codes are discussed in Section 7.2.

If a server receives payloads with different Request-Tags for the same resource, it should continue to process all the bodies, as it has no way of determining which is the latest version or which body, if any, the client is terminating the transmission for.

If the client elects to stop the transmission of a complete body, then absent any local policy, the client **MUST** "forget" all tracked tokens associated with the body's Request-Tag so that a Reset message is generated for the invalid token in the 4.08 (Request Entity Incomplete) response. On receipt of the Reset message, the server **SHOULD** delete the partial body.

If the server receives a duplicate block with the same Request-Tag, it **MUST** ignore the payload of the packet but **MUST** still respond as if the block was received for the first time.

A server **SHOULD** maintain a partial body (missing payloads) for NON_PARTIAL_TIMEOUT (Section 7.2).

4.4. Using the Q-Block2 Option

In a request for any block number, an unset M bit indicates the request is just for that block. If the M bit is set, this has different meanings based on the NUM value:

NUM is zero: This is a request for the entire body.

'NUM modulo MAX_PAYLOADS' is zero, while NUM is not zero: This is used to confirm that the current MAX_PAYLOADS_SET (the latest block having block number NUM-1) has been successfully received and that, upon receipt of this request, the server can continue to send the next MAX_PAYLOADS_SET (the first block having block number NUM). This is the 'Continue' Q-Block-2 and conceptually has the same usage (i.e., continue sending the next set of data) as the use of 2.31 (Continue) for Q-Block1.

Any other value of NUM: This is a request for that block and for all of the remaining blocks in the current MAX_PAYLOADS_SET.

If the request includes multiple Q-Block2 options and these options overlap (e.g., combination of M being set (this and later blocks) and unset (this individual block)), resulting in an individual block being requested multiple times, the server **MUST** only send back one instance of that block. This behavior is meant to prevent amplification attacks.

The payloads sent back from the server as a response **MUST** all have the same ETag (Section 5.10.6 of [RFC7252]) for the same body. The server **MUST NOT** use the same ETag value for different representations of a resource.

The ETag is opaque, but the server **MUST** ensure that it is unique for every different body of transmitted data.

> Implementation Note: It is suggested that the server treats the ETag as an unsigned integer of 8 bytes in length. An implementation may want to consider limiting this to 4 bytes to reduce packet overhead size. The initial ETag value should be randomly generated and then subsequently incremented by the server whenever a new body of data is being transmitted between peers.

Section 4.6 discusses the use of the Size2 option.

The client may elect to request any detected missing blocks or just ignore the partial body. This decision is implementation specific.

For NON payloads, the client **SHOULD** wait for NON_RECEIVE_TIMEOUT (Section 7.2) after the last received payload before requesting retransmission of any missing blocks. Retransmission is requested by issuing a GET, POST, PUT, FETCH, PATCH, or iPATCH request that contains one or more Q-Block2 options that define the missing block(s). Generally, the M bit on the Q-Block2 option(s) **SHOULD** be unset, although the M bit **MAY** be set to request this and later blocks from this MAX_PAYLOADS_SET; see Section 10.2.4 for an example of this in operation. Further considerations related to the transmission timing for missing requests are discussed in Section 7.2.

The missing block numbers requested by the client **MUST** have an increasing block number in each additional Q-Block2 option with no duplicates. The server **SHOULD** respond with a 4.00 (Bad Request) to requests not adhering to this behavior. Note that the ordering constraint is meant to force the client to check for duplicates and remove them. This also helps with troubleshooting.

If the client receives a duplicate block with the same ETag, it **MUST** silently ignore the payload.

A client **SHOULD** maintain a partial body (missing payloads) for NON_PARTIAL_TIMEOUT (Section 7.2) or as defined by the Max-Age option (or its default of 60 seconds (Section 5.6.1 of [RFC7252])), whichever is less. On release of the partial body, the client should then release all of the tokens used for this body apart from the token that is used to track a resource that is being observed.

The ETag option should not be used in the request for missing blocks, as the server could respond with a 2.03 (Valid) response with no payload. It can be used in the request if the client wants to check the freshness of the locally cached body response.

The server **SHOULD** maintain a cached copy of the body when using the Q-Block2 option to facilitate retransmission of any missing payloads.

If the server detects partway through a body transfer that the resource data has changed and the server is not maintaining a cached copy of the old data, then the transmission is terminated. Any subsequent missing block requests **MUST** be responded to using the latest ETag and Size2 option values with the updated data.

If the server responds during a body update with a different ETag option value (as the resource representation has changed), then the client should treat the partial body with the old ETag as no longer being fresh. The client may then request all of the new data by specifying Q-Block2 with block number '0' and the M bit set.

If the server transmits a new body of data (e.g., a triggered Observe notification) with a new ETag to the same client as an additional response, the client should remove any partially received body held for a previous ETag for that resource, as it is unlikely the missing blocks can be retrieved.

If there is insufficient space to create a response PDU with a block size of 16 bytes (SZX = 0) to send back all the response options as appropriate, a 4.13 (Request Entity Too Large) is returned without the Size1 option.

For Confirmable traffic, the server typically acknowledges the initial request using an Acknowledgment (ACK) with a piggybacked payload and then sends the subsequent payloads of the MAX_PAYLOADS_SET as CON responses. These CON responses are individually ACKed by the client. The server will detect failure to send a packet and **SHOULD** terminate the body transfer, but the client can issue, after a MAX_TRANSMIT_SPAN delay, a separate GET, POST, PUT, FETCH, PATCH, or iPATCH for any missing blocks as needed.

4.5. Using the Observe Option

For a request that uses Q-Block1, the Observe value [RFC7641] **MUST** be the same for all the payloads of the same body. This includes any missing payloads that are retransmitted.

For a response that uses Q-Block2, the Observe value **MUST** be the same for all the payloads of the same body. This is different from Block2 usage where the Observe value is only present in the first block (Section 3.4 of [RFC7959]). This includes payloads transmitted following receipt of the 'Continue' Q-Block2 option (Section 4.4) by the server. If a missing payload is requested by a client, then both the request and response **MUST NOT** include the Observe option.

4.6. Using the Size1 and Size2 Options

Section 4 of [RFC7959] defines two CoAP options: Size1 for indicating the size of the representation transferred in requests and Size2 for indicating the size of the representation transferred in responses.

For the Q-Block1 and Q-Block2 options, the Size1 or Size2 option values **MUST** exactly represent the size of the data on the body so that any missing data can easily be determined.

The Size1 option **MUST** be used with the Q-Block1 option when used in a request and **MUST** be present in all payloads of the request, preserving the same value. The Size2 option **MUST** be used with the Q-Block2 option when used in a response and **MUST** be present in all payloads of the response, preserving the same value.

4.7. Using the Q-Block1 and Q-Block2 Options Together

The behavior is similar to the one defined in Section 3.3 of [RFC7959] with Q-Block1 substituted for Block1 and Q-Block2 substituted for Block2.

4.8. Using the Q-Block2 Option with Multicast

Servers **MUST** ignore multicast requests that contain the Q-Block2 option. As a reminder, the Block2 option can be used as stated in Section 2.8 of [RFC7959].

5. The Use of the 4.08 (Request Entity Incomplete) Response Code

The 4.08 (Request Entity Incomplete) response code has a new content type "application/missing-blocks+cbor-seq" used to indicate that the server has not received all of the blocks of the request body that it needs to proceed. Such messages must not be treated by the client as a fatal error.

Likely causes are the client has not sent all blocks, some blocks were dropped during transmission, or the client sent them a long enough time ago that the server has already discarded them.

The new data payload of the 4.08 (Request Entity Incomplete) response with content type "application/missing-blocks+cbor-seq" is encoded as a Concise Binary Object Representation (CBOR) Sequence [RFC8742]. It comprises one or more missing block numbers encoded as CBOR unsigned integers [RFC8949]. The missing block numbers **MUST** be unique in each 4.08 (Request Entity Incomplete) response when created by the server; the client **MUST** ignore any duplicates in the same 4.08 (Request Entity Incomplete) response.

The Content-Format option (Section 5.10.3 of [RFC7252]) **MUST** be used in the 4.08 (Request Entity Incomplete) response. It **MUST** be set to "application/missing-blocks+cbor-seq" (Section 12.3).

The Concise Data Definition Language (CDDL) [RFC8610] (and see Section 4.1 of [RFC8742]) for the data describing these missing blocks is as follows:

```
; This defines an array, the elements of which are to be used
; in a CBOR Sequence:
payload = [+ missing-block-number]
; A unique block number not received:
missing-block-number = uint
```

Figure 1: Structure of the Missing Blocks Payload

This CDDL syntax **MUST** be followed.

It is desirable that the token to use for the response is the token that was used in the last block number received so far with the same Request-Tag value. Note that the use of any received token with the same Request-Tag would be acceptable, but providing the one used in the last received payload will aid any troubleshooting. The client will use the token to determine what was the previously sent request to obtain the Request-Tag value that was used.

If the size of the 4.08 (Request Entity Incomplete) response packet is larger than that defined by Section 4.6 of [RFC7252], then the number of reported missing blocks **MUST** be limited so that the response can fit into a single packet. If this is the case, then the server can send subsequent 4.08 (Request Entity Incomplete) responses containing those additional missing blocks on receipt of a new request providing a missing payload with the same Request-Tag.

The missing blocks **MUST** be reported in ascending order without any duplicates. The client **SHOULD** silently drop 4.08 (Request Entity Incomplete) responses not adhering to this behavior.

> Implementation Note: Consider limiting the number of missing payloads to MAX_PAYLOADS to minimize the need for congestion control. The CBOR Sequence does not include any array wrapper.

A 4.08 (Request Entity Incomplete) response with content type "application/missing-blocks+cbor-seq" **SHOULD NOT** be used when using Confirmable requests or a reliable connection [RFC8323], as the client will be able to determine that there is a transmission failure of a particular payload and hence that the server is missing that payload.

6. The Use of Tokens

Each new request generally uses a new Token (and sometimes must; see Section 4 of [RFC9175]). Additional responses to a request all use the token of the request they respond to.

> Implementation Note: By using 8-byte tokens, it is possible to easily minimize the number of tokens that have to be tracked by clients, by keeping the bottom 32 bits the same for the same body and the upper 32 bits containing the current body's request number (incrementing every request, including every retransmit). This alleviates the client's need to keep all the per-request state, e.g., per Section 3 of [RFC8974]. However, if using NoSec, Section 5.2 of [RFC8974] needs to be considered for security implications.

7. Congestion Control for Unreliable Transports

The transmission of all the blocks of a single body over an unreliable transport **MUST** either all be Confirmable or all be Non-confirmable. This is meant to simplify the congestion control procedure.

As a reminder, there is no need for CoAP-specific congestion control for reliable transports [RFC8323].

7.1. Confirmable (CON)

Congestion control for CON requests and responses is specified in Section 4.7 of [RFC7252]. In order to benefit from faster transmission rates, NSTART will need to be increased from 1. However, the other CON congestion control parameters will need to be tuned to cover this change. This tuning is not specified in this document, given that the applicability scope of the current specification assumes that all requests and responses using Q-Block1 and Q-Block2 will be Non-confirmable (Section 3.2) apart from the initial Q-Block functionality negotiation.

Following the failure to transmit a packet due to packet loss after MAX_TRANSMIT_SPAN time (Section 4.8.2 of [RFC7252]), it is implementation specific as to whether there should be any further requests for missing data.

7.2. Non-confirmable (NON)

This document introduces the new parameters MAX_PAYLOADS, NON_TIMEOUT, NON_TIMEOUT_RANDOM, NON_RECEIVE_TIMEOUT, NON_MAX_RETRANSMIT, NON_PROBING_WAIT, and NON_PARTIAL_TIMEOUT primarily for use with NON (Table 3).

> Note: Randomness may naturally be provided based on the traffic profile, how PROBING_RATE is computed (as this is an average), and when the peer responds. Randomness is explicitly added for some of the congestion control parameters to handle situations where everything is in sync when retrying.

MAX_PAYLOADS should be configurable with a default value of 10. Both CoAP endpoints **MUST** have the same value (otherwise, there will be transmission delays in one direction), and the value **MAY** be negotiated between the endpoints to a common value by using a higher-level protocol (out of scope of this document). This is the maximum number of payloads that can be transmitted at any one time.

> Note: The default value of 10 is chosen for reasons similar to those discussed in Section 5 of [RFC6928], especially given the target application discussed in Section 3.2.

NON_TIMEOUT is used to compute the delay between sending MAX_PAYLOADS_SET for the same body. By default, NON_TIMEOUT has the same value as ACK_TIMEOUT (Section 4.8 of [RFC7252]).

NON_TIMEOUT_RANDOM is the initial actual delay between sending the first two MAX_PAYLOADS_SETs of the same body. The same delay is then used between the subsequent MAX_PAYLOADS_SETs. It is a random duration (not an integral number of seconds) between NON_TIMEOUT and (NON_TIMEOUT * ACK_RANDOM_FACTOR). ACK_RANDOM_FACTOR is set to 1.5, as discussed in Section 4.8 of [RFC7252].

NON_RECEIVE_TIMEOUT is the initial time to wait for a missing payload before requesting retransmission for the first time. Every time the missing payload is re-requested, the Time-to-Wait value doubles. The time to wait is calculated as:

```
Time-to-Wait = NON_RECEIVE_TIMEOUT * (2 ** (Re-Request-Count - 1))
```

NON_RECEIVE_TIMEOUT has a default value of twice NON_TIMEOUT. NON_RECEIVE_TIMEOUT **MUST** always be greater than NON_TIMEOUT_RANDOM by at least one second so that the sender of the payloads has the opportunity to start sending the next MAX_PAYLOADS_SET before the receiver times out.

NON_MAX_RETRANSMIT is the maximum number of times a request for the retransmission of missing payloads can occur without a response from the remote peer. After this occurs, the local endpoint **SHOULD** consider the body stale, remove any body, and release the tokens and Request-Tag on the client (or the ETag on the server). By default, NON_MAX_RETRANSMIT has the same value as MAX_RETRANSMIT (Section 4.8 of [RFC7252]).

NON_PROBING_WAIT is used to limit the potential wait needed when using PROBING_RATE. By default, NON_PROBING_WAIT is computed in a way similar to EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]) but with ACK_TIMEOUT, MAX_RETRANSMIT, and PROCESSING_DELAY substituted with NON_TIMEOUT, NON_MAX_RETRANSMIT, and NON_TIMEOUT_RANDOM, respectively:

```
NON_PROBING_WAIT = NON_TIMEOUT * ((2 ** NON_MAX_RETRANSMIT) - 1) *
ACK_RANDOM_FACTOR + (2 * MAX_LATENCY) + NON_TIMEOUT_RANDOM
```

NON_PARTIAL_TIMEOUT is used for expiring partially received bodies. By default, NON_PARTIAL_TIMEOUT is computed in the same way as EXCHANGE_LIFETIME (Section 4.8.2 of [RFC7252]) but with ACK_TIMEOUT and MAX_RETRANSMIT substituted with NON_TIMEOUT and NON_MAX_RETRANSMIT, respectively:

```
NON_PARTIAL_TIMEOUT = NON_TIMEOUT * ((2 ** NON_MAX_RETRANSMIT) - 1) *
ACK_RANDOM_FACTOR + (2 * MAX_LATENCY) + NON_TIMEOUT
```

Parameter Name	Default Value
MAX_PAYLOADS	10
NON_MAX_RETRANSMIT	4
NON_TIMEOUT	2 s
NON_TIMEOUT_RANDOM	between 2-3 s
NON_RECEIVE_TIMEOUT	4 s
NON_PROBING_WAIT	between 247-248 s
NON_PARTIAL_TIMEOUT	247 s

Table 3: Congestion Control Parameters

The PROBING_RATE parameter in CoAP indicates the average data rate that must not be exceeded by a CoAP endpoint in sending to a peer endpoint that does not respond. A single body will be subjected to PROBING_RATE (Section 4.7 of [RFC7252]), not the individual packets. If the wait time between sending bodies that are not being responded to based on PROBING_RATE exceeds NON_PROBING_WAIT, then the wait time is limited to NON_PROBING_WAIT.

> Note: For the particular DOTS application, PROBING_RATE and other transmission parameters are negotiated between peers. Even when not negotiated, the DOTS application uses customized defaults, as discussed in Section 4.5.2 of [RFC9132]. Note that MAX_PAYLOADS, NON_MAX_RETRANSMIT, NON_TIMEOUT, NON_PROBING_WAIT, and NON_PARTIAL_TIMEOUT can be negotiated between DOTS peers, e.g., as per [DOTS-QUICK-BLOCKS]. When explicit values are configured for NON_PROBING_WAIT and NON_PARTIAL_TIMEOUT, these values are used without applying any jitter.

Each NON 4.08 (Request Entity Incomplete) response is subject to PROBING_RATE.

Each NON GET or FETCH request using a Q-Block2 option is subject to PROBING_RATE.

As the sending of many payloads of a single body may itself cause congestion, after transmission of every MAX_PAYLOADS_SET of a single body, a delay of NON_TIMEOUT_RANDOM **MUST** be introduced before sending the next MAX_PAYLOADS_SET, unless a 'Continue' is received from the peer for the current MAX_PAYLOADS_SET, in which case the next MAX_PAYLOADS_SET **MAY** start transmission immediately.

> Note: Assuming 1500-byte packets and the MAX_PAYLOADS_SET having 10 payloads, this corresponds to 1500 * 10 * 8 = 120 kbits. With a delay of 2 seconds between MAX_PAYLOADS_SET, this indicates an average speed requirement of 60 kbps for a single body should there be no responses. This transmission rate is further reduced by being subject to PROBING_RATE.

The sending of a set of missing blocks of a body is restricted to those in a MAX_PAYLOADS_SET at a time. In other words, a NON_TIMEOUT_RANDOM delay is still observed between each MAX_PAYLOADS_SET.

For the Q-Block1 option, if the server responds with a 2.31 (Continue) response code for the latest payload sent, then the client can continue to send the next MAX_PAYLOADS_SET without any further delay. If the server responds with a 4.08 (Request Entity Incomplete) response code, then the missing payloads **SHOULD** be retransmitted before going into another NON_TIMEOUT_RANDOM delay prior to sending the next set of payloads.

For the server receiving NON Q-Block1 requests, it **SHOULD** send back a 2.31 (Continue) response code on receipt of all of the MAX_PAYLOADS_SET to prevent the client unnecessarily delaying the transfer of remaining blocks. If not all of the MAX_PAYLOADS_SET were received, the server **SHOULD** delay for NON_RECEIVE_TIMEOUT (exponentially scaled based on the repeat request count for a payload) before sending the 4.08 (Request Entity Incomplete) response code for the missing payload(s). If all of the MAX_PAYLOADS_SET were received and a 2.31 (Continue) response code had been sent, but no more payloads were received for NON_RECEIVE_TIMEOUT (exponentially scaled), the server **SHOULD** send a 4.08 (Request Entity Incomplete) response detailing the missing payloads after the block number that was indicated in the sent 2.31 (Continue) response code. If the repeat response count of the 4.08 (Request Entity Incomplete) exceeds NON_MAX_RETRANSMIT, the server **SHOULD** discard the partial body and stop requesting the missing payloads.

It is likely that the client will start transmitting the next MAX_PAYLOADS_SET before the server times out on waiting for the last block of the previous MAX_PAYLOADS_SET. On receipt of a payload from the next MAX_PAYLOADS_SET, the server **SHOULD** send a 4.08 (Request Entity Incomplete) response code indicating any missing payloads from any previous MAX_PAYLOADS_SET. Upon receipt of the 4.08 (Request Entity Incomplete) response code, the client **SHOULD** send the missing payloads before continuing to send the remainder of the MAX_PAYLOADS_SET and then go into another NON_TIMEOUT_RANDOM delay prior to sending the next MAX_PAYLOADS_SET.

For the client receiving NON Q-Block2 responses, it **SHOULD** send a 'Continue' Q-Block2 request (Section 4.4) for the next MAX_PAYLOADS_SET on receipt of all of the MAX_PAYLOADS_SET to prevent the server unnecessarily delaying the transfer of remaining blocks. Otherwise, the client **SHOULD** delay for NON_RECEIVE_TIMEOUT (exponentially scaled based on the repeat request count for a payload) before sending the request for the missing payload(s). If the repeat request count for a missing payload exceeds NON_MAX_RETRANSMIT, the client **SHOULD** discard the partial body and stop requesting the missing payloads.

The server **SHOULD** recognize the 'Continue' Q-Block2 request per the definition in Section 4.4 and just continue the transmission of the body (including the Observe option, if appropriate for an unsolicited response) rather than treat 'Continue' as a request for the remaining missing blocks.

It is likely that the server will start transmitting the next MAX_PAYLOADS_SET before the client times out on waiting for the last block of the previous MAX_PAYLOADS_SET. Upon receipt of a payload from the new MAX_PAYLOADS_SET, the client **SHOULD** send a request indicating any missing payloads from any previous MAX_PAYLOADS_SET. Upon receipt of such a request, the server **SHOULD** send the missing payloads before continuing to send the remainder of the MAX_PAYLOADS_SET and then go into another NON_TIMEOUT_RANDOM delay prior to sending the next MAX_PAYLOADS_SET.

The client does not need to acknowledge the receipt of the entire body.

> Note: If there is asymmetric traffic loss causing responses to never get received, a delay of NON_TIMEOUT_RANDOM after every transmission of MAX_PAYLOADS_SET will be observed. The endpoint receiving the body is still likely to receive the entire body.

8. Caching Considerations

Caching block-based information is not straightforward in a proxy. For the Q-Block1 and Q-Block2 options, for simplicity, it is expected that the proxy will reassemble the body (using any appropriate recovery options for packet loss) before passing the body onward to the appropriate CoAP endpoint. This does not preclude an implementation doing a more complex per-payload caching, but how to do this is out of the scope of this document. The onward transmission of the body does not require the use of the Q-Block1 or Q-Block2 options, as these options may not be supported in that link. This means that the proxy must fully support the Q-Block1 and Q-Block2 options.

How the body is cached in the CoAP client (for Q-Block1 transmissions) or the CoAP server (for Q-Block2 transmissions) is implementation specific.

As the entire body is being cached in the proxy, the Q-Block1 and Q-Block2 options are removed as part of the block assembly and thus do not reach the cache.

For Q-Block2 responses, the ETag option value is associated with the data (and transmitted onward to the CoAP client) but is not part of the cache key.

For requests with the Q-Block1 option, the Request-Tag option is associated with building the body from successive payloads but is not part of the cache key. For the onward transmission of the body using CoAP, a new Request-Tag **SHOULD** be generated and used. Ideally, this new Request-Tag should replace the Request-Tag used by the client.

It is possible that two or more CoAP clients are concurrently updating the same resource through a common proxy to the same CoAP server using the Q-Block1 (or Block1) option. If this is the case, the first client to complete building the body causes that body to start transmitting to the CoAP server with an appropriate Request-Tag value. When the next client completes building the body, any existing partial body transmission to the CoAP server is terminated, and the transmission of the new body representation starts with a new Request-Tag value. Note that it cannot be assumed that the proxy will always receive a complete body from a client.

A proxy that supports the Q-Block2 option **MUST** be prepared to receive a GET or similar request indicating one or more missing blocks. From its cache, the proxy will serve the missing blocks that are available in its cache in the same way a server would send all the appropriate Q-Block2 responses. If a body matching the cache key is not available in the cache, the proxy **MUST** request the entire body from the CoAP server using the information in the cache key.

How long a CoAP endpoint (or proxy) keeps the body in its cache is implementation specific (e.g., it may be based on Max-Age).

9. HTTP Mapping Considerations

As a reminder, the basic normative requirements on HTTP/CoAP mappings are defined in Section 10 of [RFC7252]. The implementation guidelines for HTTP/CoAP mappings are elaborated in [RFC8075].

The rules defined in Section 5 of [RFC7959] are to be followed.

10. Examples with Non-confirmable Messages

This section provides some sample flows to illustrate the use of the Q-Block1 and Q-Block2 options with NON. Examples with CON are provided in Appendix A.

The examples in the following subsections assume MAX_PAYLOADS is set to 10 and NON_MAX_RETRANSMIT is set to 4.

The list below contains the conventions that are used in the figures in the following subsections.

T: Token value

O: Observe option value

M: Message ID

RT: Request-Tag

ET: ETag

QB1: Q-Block1 option values NUM/More/Size

QB2: Q-Block2 option values NUM/More/Size

Size: Actual block size encoded in SZX

\: Trimming long lines

[[]]: Comments

-->X: Message loss (request)

X<--: Message loss (response)

...: Passage of time

Payload N: Corresponds to the CoAP message that conveys a block number (N-1) of a given block-wise exchange.

10.1. Q-Block1 Option

10.1.1. A Simple Example

Figure 2 depicts an example of a NON PUT request conveying the Q-Block1 option. All the blocks are received by the server.

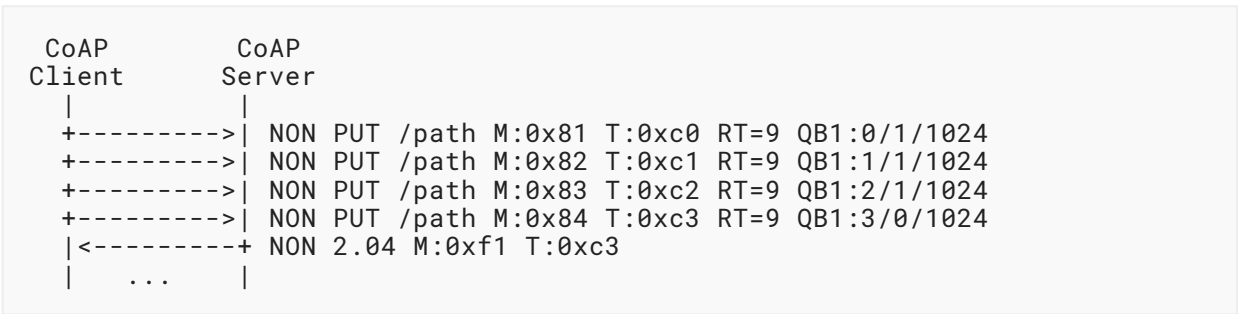

Figure 2: Example of a NON Request with the Q-Block1 option (without Loss)

10.1.2. Handling MAX_PAYLOADS Limits

Figure 3 depicts an example of a NON PUT request conveying the Q-Block1 option. The number of payloads exceeds MAX_PAYLOADS. All the blocks are received by the server.

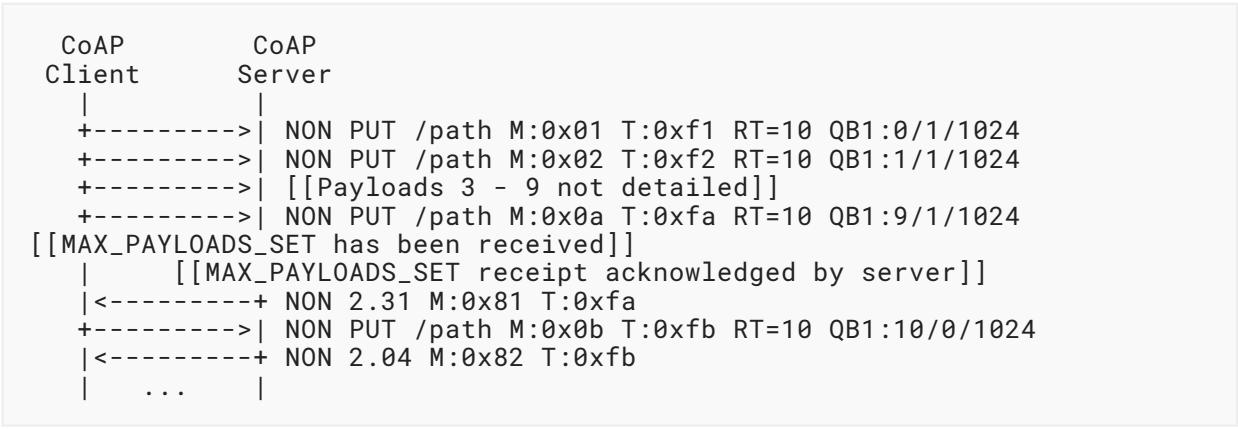

Figure 3: Example of a MAX_PAYLOADS NON Request with the Q-Block1 Option (without Loss)

10.1.3. Handling MAX_PAYLOADS with Recovery

Consider now a scenario where a new body of data is to be sent by the client, but some blocks are dropped in transmission, as illustrated in Figure 4.

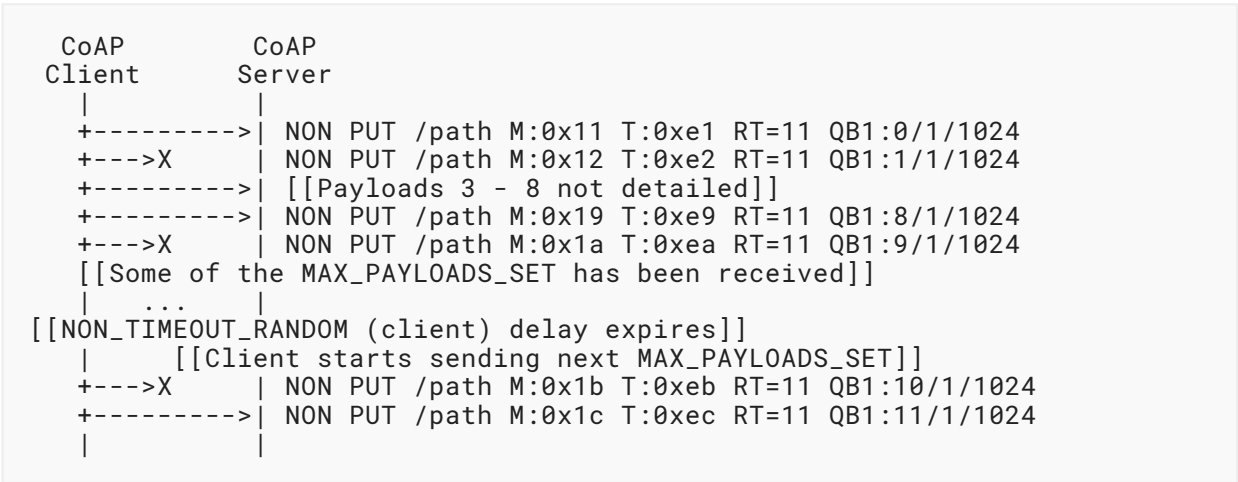

Figure 4: Example of a MAX_PAYLOADS NON Request with the Q-Block1 Option (with Loss)

On seeing a payload from the next MAX_PAYLOADS_SET, the server realizes that some blocks are missing from the previous MAX_PAYLOADS_SET and asks for the missing blocks in one go (Figure 5). It does so by indicating which blocks from the previous MAX_PAYLOADS_SET have not been received in the data portion of the response (Section 5). The token used in the response should be the token that was used in the last received payload. The client can then derive the Request-Tag by matching the token with the sent request.

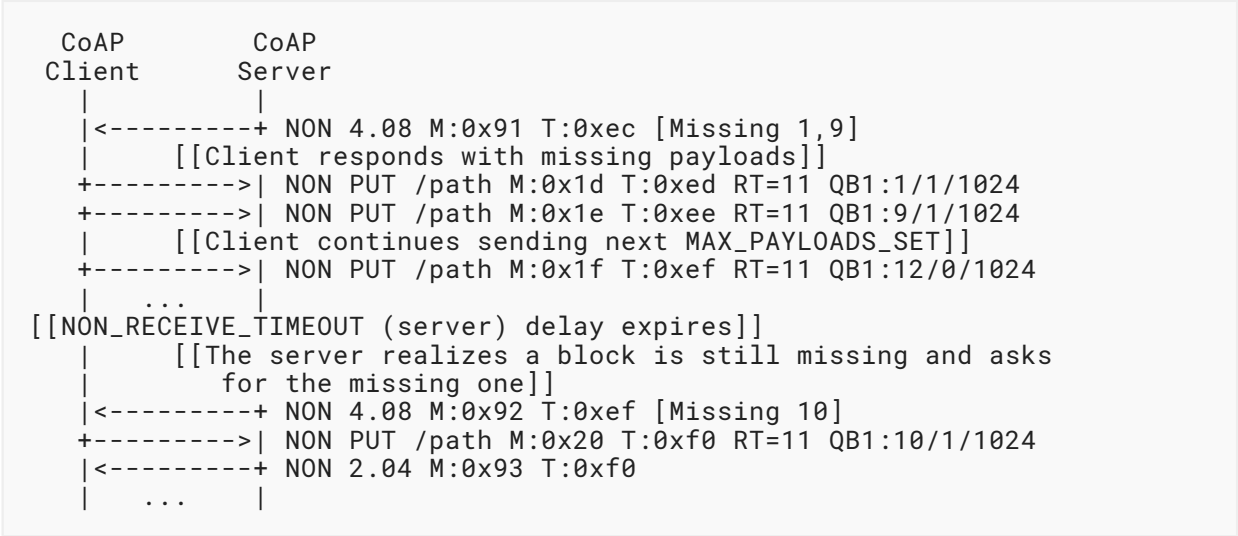

Figure 5: Example of a NON Request with the Q-Block1 Option (Block Recovery)

10.1.4. Handling Recovery if Failure Occurs

Figure 6 depicts an example of a NON PUT request conveying the Q-Block1 option where recovery takes place but eventually fails.

```
  CoAP        CoAP
 Client      Server
   |          |
   +--------->| NON PUT /path M:0x91 T:0xd0 RT=12 QB1:0/1/1024
   +--->X     | NON PUT /path M:0x92 T:0xd1 RT=12 QB1:1/1/1024
   +--------->| NON PUT /path M:0x93 T:0xd2 RT=12 QB1:2/0/1024
   |   ...    |
[[NON_RECEIVE_TIMEOUT (server) delay expires]]
   |     [[The server realizes a block is missing and asks
   |        for the missing one.  Retry #1]]
   |<---------+ NON 4.08 M:0x01 T:0xd2 [Missing 1]
   |   ...    |
[[2 * NON_RECEIVE_TIMEOUT (server) delay expires]]
   |     [[The server realizes a block is still missing and asks
   |        for the missing one.  Retry #2]]
   |<---------+ NON 4.08 M:0x02 T:0xd2 [Missing 1]
   |   ...    |
[[4 * NON_RECEIVE_TIMEOUT (server) delay expires]]
   |     [[The server realizes a block is still missing and asks
   |        for the missing one.  Retry #3]]
   |<---------+ NON 4.08 M:0x03 T:0xd2 [Missing 1]
   |   ...    |
[[8 * NON_RECEIVE_TIMEOUT (server) delay expires]]
   |     [[The server realizes a block is still missing and asks
   |        for the missing one.  Retry #4]]
   |<---------+ NON 4.08 M:0x04 T:0xd2 [Missing 1]
   |   ...    |
[[16 * NON_RECEIVE_TIMEOUT (server) delay expires]]
   |     [[NON_MAX_RETRANSMIT exceeded.  Server stops requesting
   |       the missing blocks and releases partial body]]
   |   ...    |
```

Figure 6: Example of a NON Request with the Q-Block1 Option (with Eventual Failure)

10.2. Q-Block2 Option

These examples include the Observe option to demonstrate how that option is used. Note that the Observe option is not required for Q-Block2.

10.2.1. A Simple Example

Figure 7 illustrates an example of the Q-Block2 option. The client sends a NON GET carrying the Observe and Q-Block2 options. The Q-Block2 option indicates a block size hint (1024 bytes). The server replies to this request using four (4) blocks that are transmitted to the client without any loss. Each of these blocks carries a Q-Block2 option. The same process is repeated when an Observe is triggered, but no loss is experienced by any of the notification blocks.

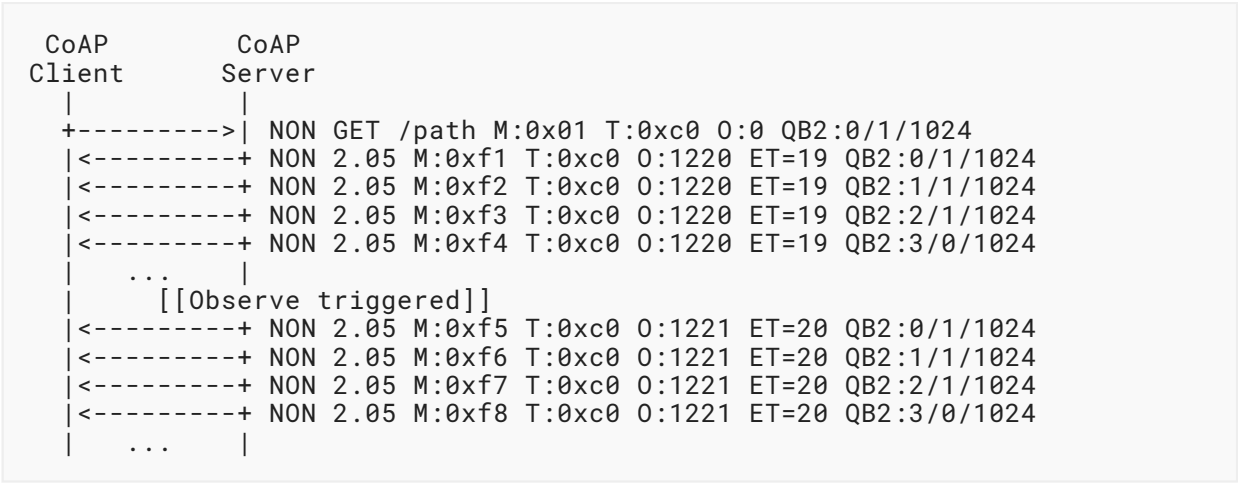

Figure 7: Example of NON Notifications with the Q-Block2 Option (without Loss)

10.2.2. Handling MAX_PAYLOADS Limits

Figure 8 illustrates the same scenario as Figure 7, but this time with eleven (11) payloads, which exceeds MAX_PAYLOADS. There is no loss experienced.

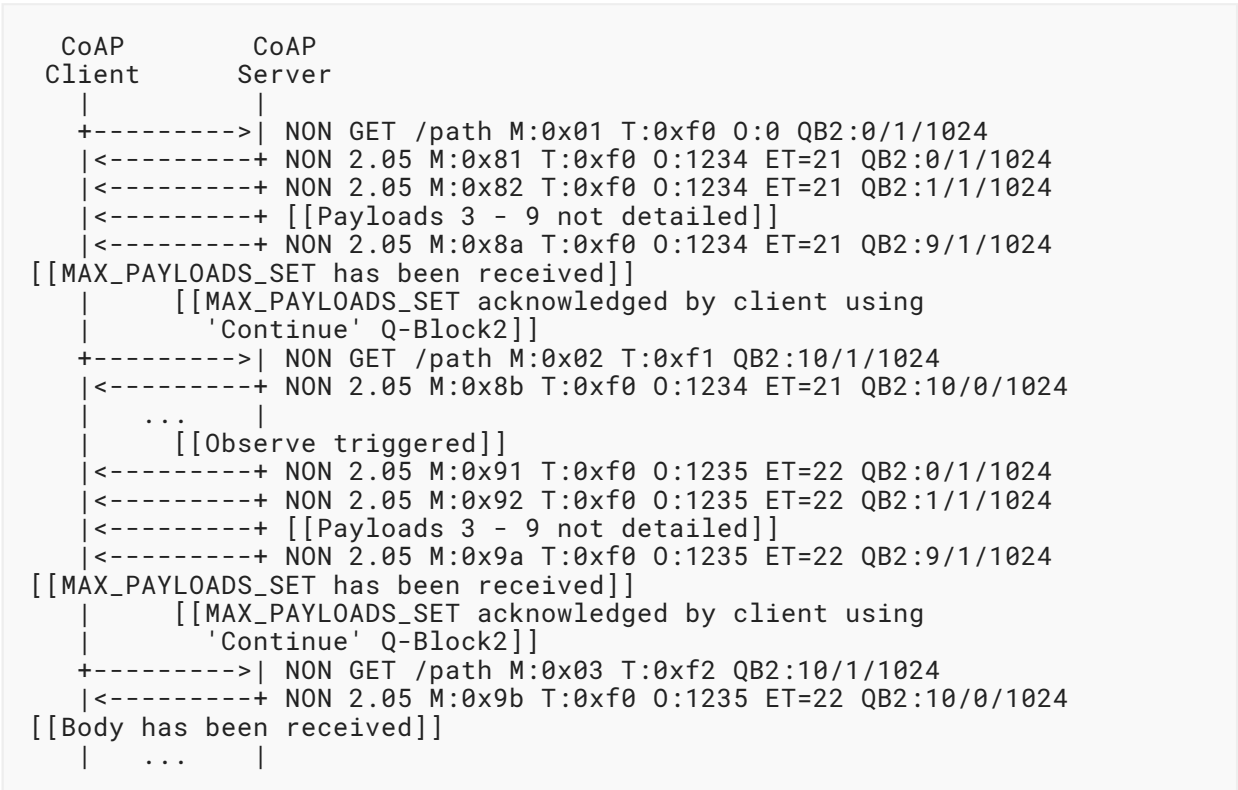

Figure 8: Example of NON Notifications with the Q-Block2 Option (without Loss)

10.2.3. Handling MAX_PAYLOADS with Recovery

Figure 9 shows an example of an Observe that is triggered but for which some notification blocks are lost. The client detects the missing blocks and requests their retransmission. It does so by indicating the blocks that are missing as one or more Q-Block2 options.

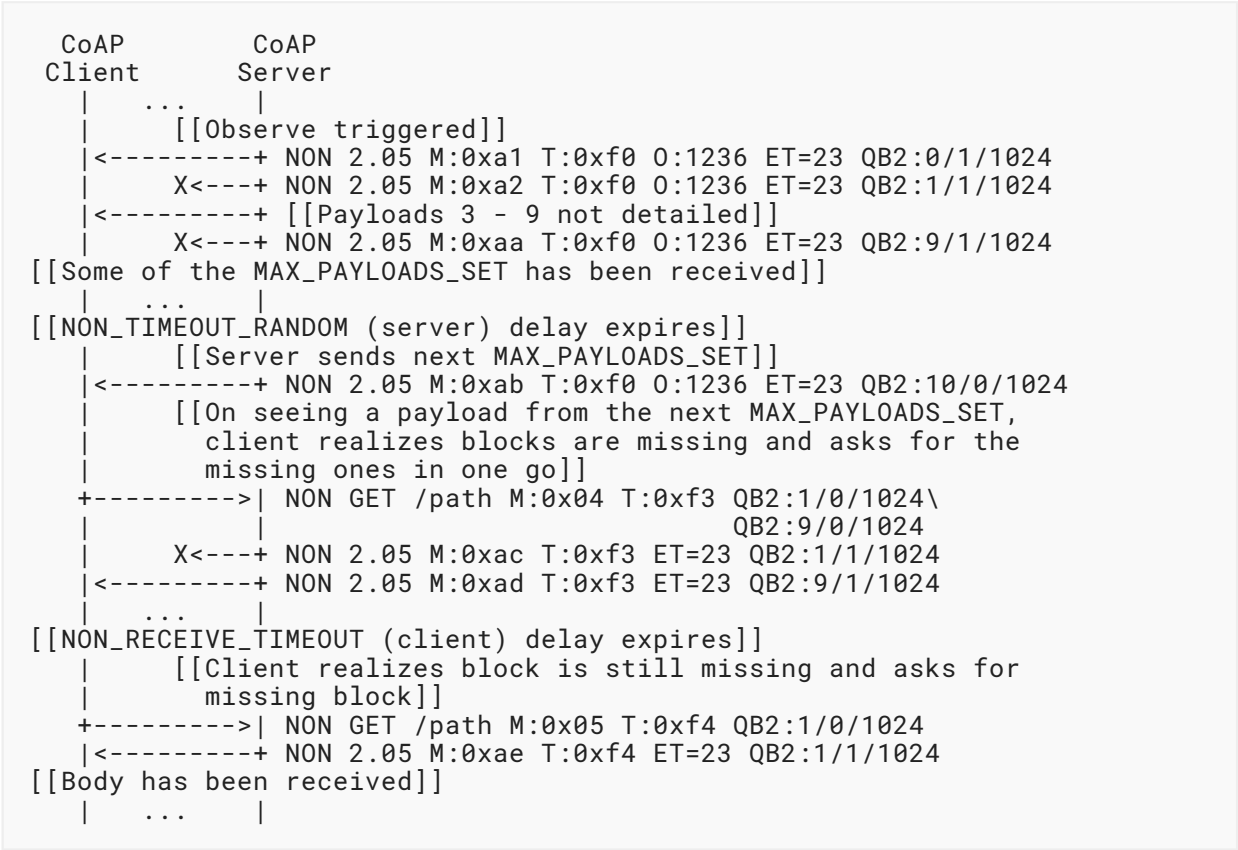

Figure 9: Example of NON Notifications with the Q-Block2 Option (Block Recovery)

10.2.4. Handling Recovery by Setting the M Bit

Figure 10 shows an example where an Observe is triggered but only the first two notification blocks reach the client. In order to retrieve the missing blocks, the client sends a request with a single Q-Block2 option with the M bit set.

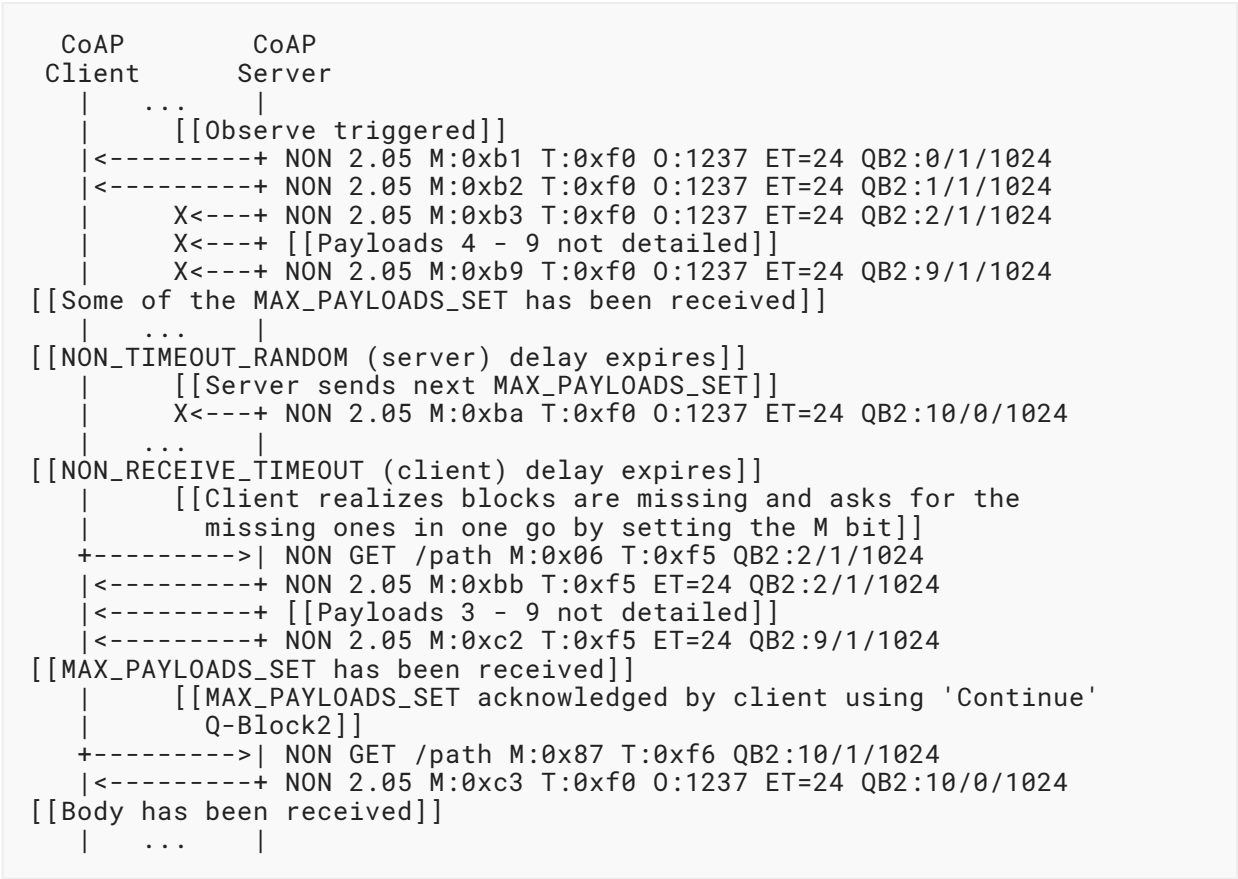

Figure 10: Example of NON Notifications with the Q-Block2 Option (Block Recovery with the M Bit Set)

10.3. Q-Block1 and Q-Block2 Options

10.3.1. A Simple Example

Figure 11 illustrates an example of a FETCH using both the Q-Block1 and Q-Block2 options along with an Observe option. No loss is experienced.

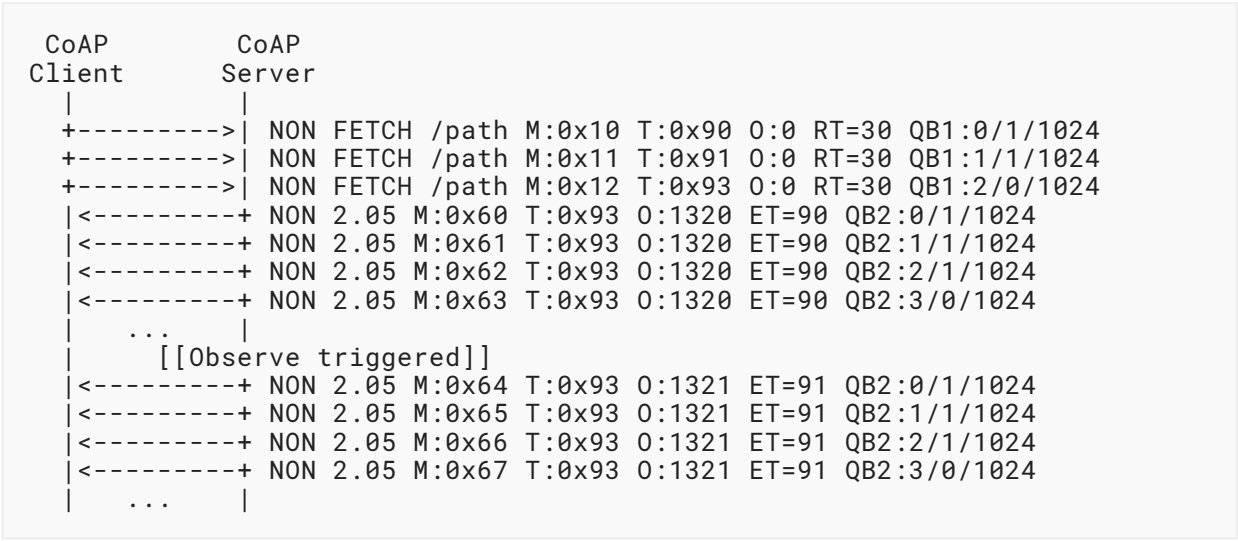

Figure 11: Example of a NON FETCH with the Q-Block1 and Q-Block2 Options (without Loss)

10.3.2. Handling MAX_PAYLOADS Limits

Figure 12 illustrates the same scenario as Figure 11, but this time with eleven (11) payloads in both directions, which exceeds MAX_PAYLOADS. There is no loss experienced.

```
  CoAP        CoAP
 Client      Server
   |          |
   +--------->| NON FETCH /path M:0x30 T:0xa0 O:0 RT=10 QB1:0/1/1024
   +--------->| NON FETCH /path M:0x31 T:0xa1 O:0 RT=10 QB1:1/1/1024
   +--------->| [[Payloads 3 - 9 not detailed]]
   +--------->| NON FETCH /path M:0x39 T:0xa9 O:0 RT=10 QB1:9/1/1024
[[MAX_PAYLOADS_SET has been received]]
   |     [[MAX_PAYLOADS_SET acknowledged by server]]
   |<---------+ NON 2.31 M:0x80 T:0xa9
   +--------->| NON FETCH /path M:0x3a T:0xaa O:0 RT=10 QB1:10/0/1024
   |<---------+ NON 2.05 M:0x81 T:0xaa O:1334 ET=21 QB2:0/1/1024
   |<---------+ NON 2.05 M:0x82 T:0xaa O:1334 ET=21 QB2:1/1/1024
   |<---------+ [[Payloads 3 - 9 not detailed]]
   |<---------+ NON 2.05 M:0x8a T:0xaa O:1334 ET=21 QB2:9/1/1024
[[MAX_PAYLOADS_SET has been received]]
   |     [[MAX_PAYLOADS_SET acknowledged by client using
   |       'Continue' Q-Block2]]
   +--------->| NON FETCH /path M:0x3b T:0xab QB2:10/1/1024
   |<---------+ NON 2.05 M:0x8b T:0xaa O:1334 ET=21 QB2:10/0/1024
   |   ...    |
   |     [[Observe triggered]]
   |<---------+ NON 2.05 M:0x8c T:0xaa O:1335 ET=22 QB2:0/1/1024
   |<---------+ NON 2.05 M:0x8d T:0xaa O:1335 ET=22 QB2:1/1/1024
   |<---------+ [[Payloads 3 - 9 not detailed]]
   |<---------+ NON 2.05 M:0x95 T:0xaa O:1335 ET=22 QB2:9/1/1024
[[MAX_PAYLOADS_SET has been received]]
   |     [[MAX_PAYLOADS_SET acknowledged by client using
   |       'Continue' Q-Block2]]
   +--------->| NON FETCH /path M:0x3c T:0xac QB2:10/1/1024
   |<---------+ NON 2.05 M:0x96 T:0xaa O:1335 ET=22 QB2:10/0/1024
[[Body has been received]]
   |   ...    |
```

Figure 12: Example of a NON FETCH with the Q-Block1 and Q-Block2 Options (without Loss)

Note that, as 'Continue' was used, the server continues to use the same token (0xaa), since the 'Continue' is not being used as a request for a new set of packets but rather is being used to instruct the server to continue its transmission (Section 7.2).

10.3.3. Handling Recovery

Consider now a scenario where some blocks are lost in transmission, as illustrated in Figure 13.

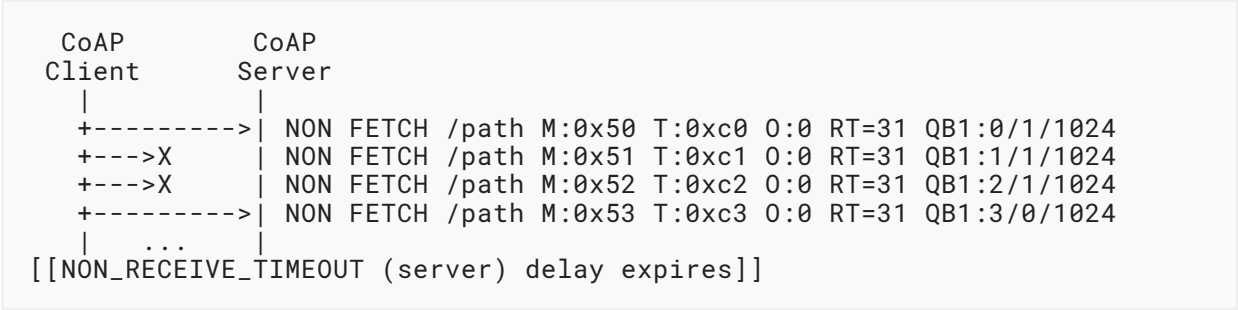

Figure 13: Example of a NON FETCH with the Q-Block1 and Q-Block2 Options (with Loss)

The server realizes that some blocks are missing and asks for the missing blocks in one go (Figure 14). It does so by indicating which blocks have not been received in the data portion of the response. The token used in the response is the token that was used in the last received payload. The client can then derive the Request-Tag by matching the token with the sent request.

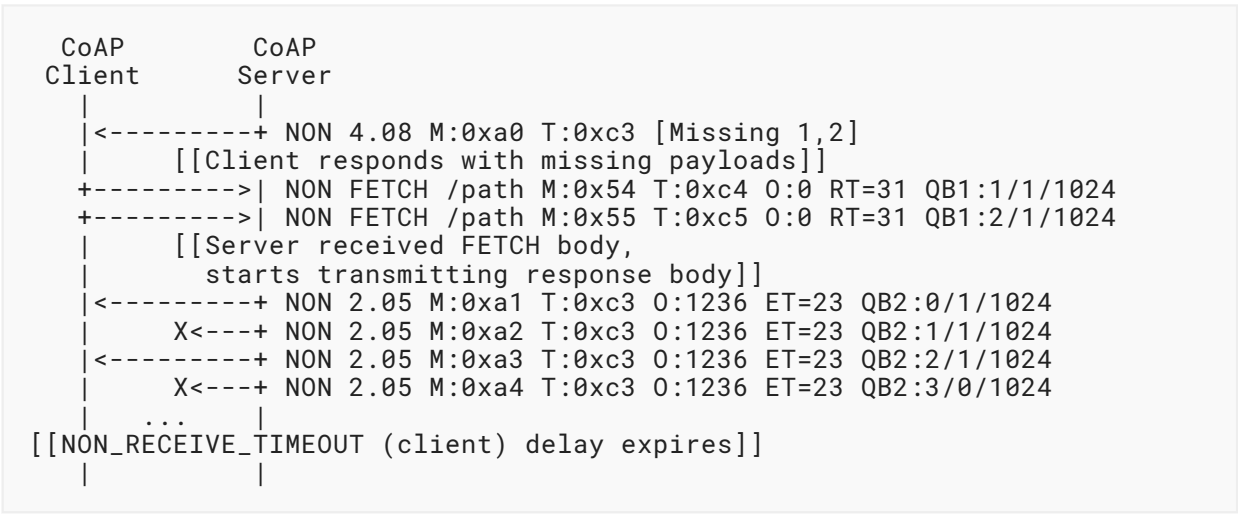

Figure 14: Example of a NON Request with the Q-Block1 Option (Server Recovery)

The client realizes that not all the payloads of the response have been returned. The client then asks for the missing blocks in one go (Figure 15). Note that, following Section 2.7 of [RFC7959], the FETCH request does not include the Q-Block1 or any payload.

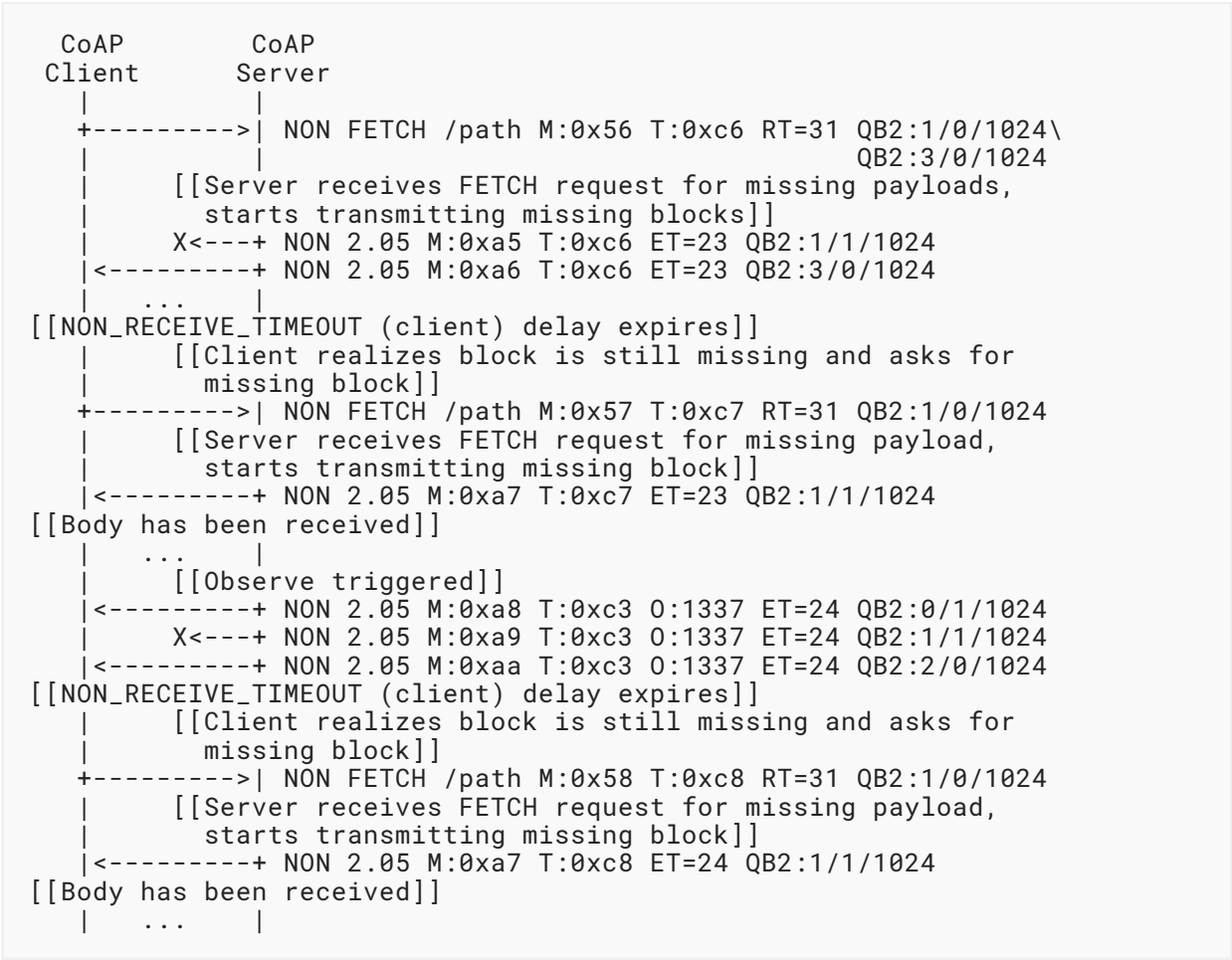

Figure 15: Example of a NON Request with the Q-Block1 Option (Client Recovery)

11. Security Considerations

Security considerations discussed in Section 7 of [RFC7959] should be taken into account.

Security considerations discussed in Sections 11.3 and 11.4 of [RFC7252] should also be taken into account.

OSCORE provides end-to-end protection of all information that is not required for proxy operations and requires that a security context is set up (Section 3.1 of [RFC8613]). It can be trusted that the source endpoint is legitimate even if the NoSec mode is used. However, an intermediary node can modify the unprotected Outer Q-Block1 and/or Q-Block2 options to cause a Q-Block transfer to fail or keep requesting all the blocks by setting the M bit and thus causing attack amplification. As discussed in Section 12.1 of [RFC8613], applications need to consider that certain message fields and message types are not protected end to end and may be spoofed or manipulated. Therefore, it is **NOT RECOMMENDED** to use the NoSec mode if either the Q-Block1 or Q-Block2 option is used.

If OSCORE is not used, it is also **NOT RECOMMENDED** to use the NoSec mode if either the Q-Block1 or Q-Block2 option is used.

If NoSec is being used, Appendix D.5 of [RFC8613] discusses the security analysis and considerations for unprotected message fields even if OSCORE is not being used.

Security considerations related to the use of Request-Tag are discussed in Section 5 of [RFC9175].

12. IANA Considerations

12.1. CoAP Option Numbers Registry

IANA has added the following entries to the "CoAP Option Numbers" subregistry [IANA-Options] defined in [RFC7252] within the "Constrained RESTful Environments (CoRE) Parameters" registry:

Number	Name	Reference
19	Q-Block1	RFC 9177
31	Q-Block2	RFC 9177

Table 4: Additions to CoAP Option Numbers Registry

12.2. Media Type Registration

IANA has registered the "application/missing-blocks+cbor-seq" media type in the "Media Types" registry [IANA-MediaTypes]. This registration follows the procedures specified in [RFC6838].

Type name: application

Subtype name: missing-blocks+cbor-seq

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: Must be encoded as a CBOR Sequence [RFC8742], as defined in Section 5 of RFC 9177.

Security considerations: See Section 11 of RFC 9177.

Interoperability considerations: N/A

Published specification: RFC 9177

Applications that use this media type: Data serialization and deserialization. In particular, the type is used by applications relying upon block-wise transfers, allowing a server to specify non-received blocks and request their retransmission, as defined in Section 4 of RFC 9177.

Fragment identifier considerations: N/A

Additional information: N/A

Person & email address to contact for further information: IETF, iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: none

Author: See Authors' Addresses section of RFC 9177.

Change controller: IESG

Provisional registration? No

12.3. CoAP Content-Formats Registry

IANA has registered the following CoAP Content-Format for the "application/missing-blocks+cbor-seq" media type in the "CoAP Content-Formats" registry [IANA-Format] defined in [RFC7252] within the "Constrained RESTful Environments (CoRE) Parameters" registry:

Media Type	Encoding	ID	Reference
application/missing-blocks+cbor-seq	-	272	RFC 9177

Table 5: Addition to CoAP Content-Format Registry

13. References

13.1. 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>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[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>.

[RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <https://www.rfc-editor.org/info/rfc8075>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[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>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <https://www.rfc-editor.org/info/rfc9175>.

13.2. Informative References

[DOTS-QUICK-BLOCKS] Boucadair, M. and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Configuration Attributes for Robust Block Transmission", Work in Progress, Internet-Draft, draft-bosh-dots-quick-blocks-03, 29 June 2021, <https://datatracker.ietf.org/doc/html/draft-bosh-dots-quick-blocks-03>.

[DOTS-TELEMETRY] Boucadair, M., Ed., Reddy.K, T., Ed., Doron, E., Chen, M., and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Telemetry", Work in Progress, Internet-Draft, draft-ietf-dots-telemetry-19, 4 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-dots-telemetry-19>.

[IANA-Format] IANA, "CoAP Content-Formats", <https://www.iana.org/assignments/core-parameters/>.

[IANA-MediaTypes] IANA, "Media Types", <https://www.iana.org/assignments/media-types/>.

[IANA-Options] IANA, "CoAP Option Numbers", <https://www.iana.org/assignments/core-parameters/>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <https://www.rfc-editor.org/info/rfc7967>.

[RFC8974] Hartke, K. and M. Richardson, "Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)", RFC 8974, DOI 10.17487/RFC8974, January 2021, <https://www.rfc-editor.org/info/rfc8974>.

[RFC9132] Boucadair, M., Ed., Shallow, J., and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 9132, DOI 10.17487/RFC9132, September 2021, <https://www.rfc-editor.org/info/rfc9132>.

Appendix A. Examples with Confirmable Messages

The following examples assume NSTART has been increased to 3.

The conventions provided in Section 10 are used in the following subsections.

A.1. Q-Block1 Option

Let's now consider the use of the Q-Block1 option with a CON request, as shown in Figure 16. All the blocks are acknowledged (as noted with "ACK").

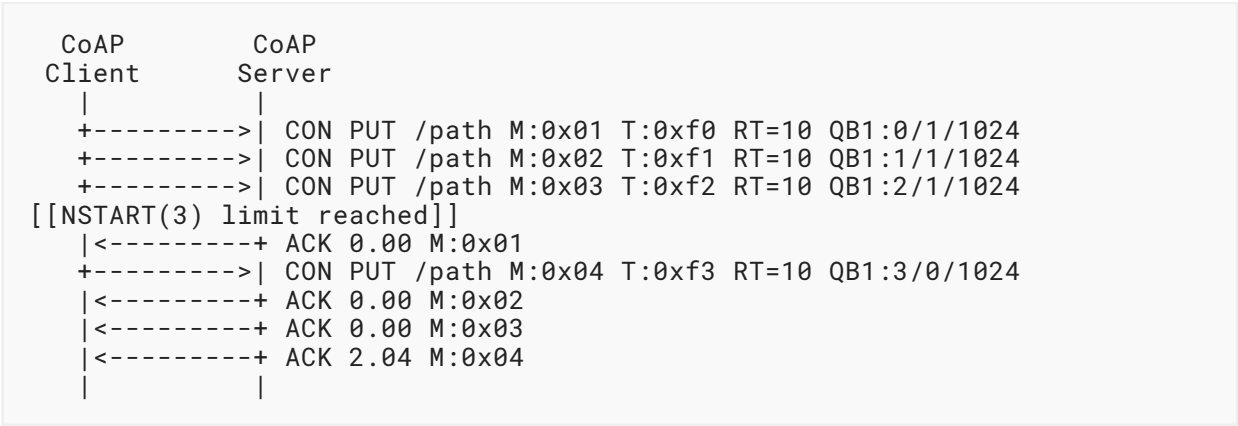

Figure 16: Example of a CON Request with the Q-Block1 Option (without Loss)

Now, suppose that a new body of data is to be sent but with some blocks dropped in transmission, as illustrated in Figure 17. The client will retry sending blocks for which no ACK was received.

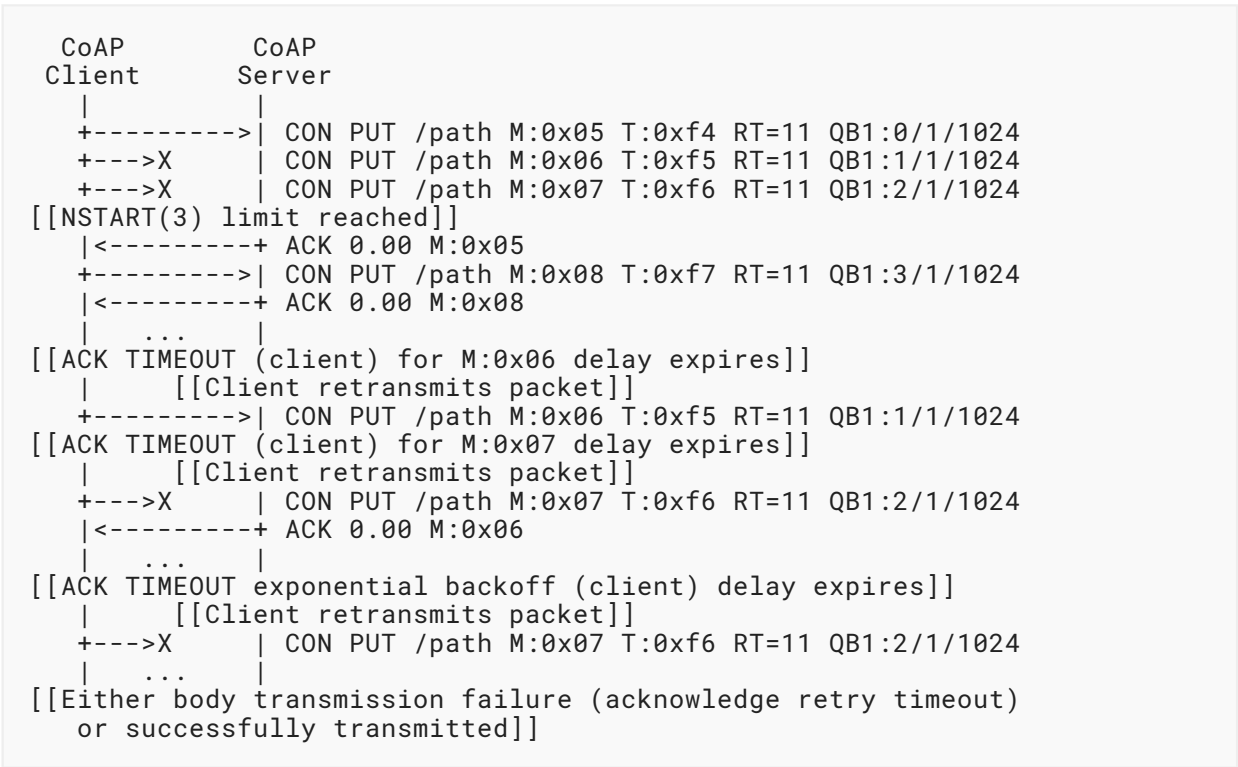

Figure 17: Example of a CON Request with the Q-Block1 Option (Block Recovery)

It is up to the implementation as to whether the application process stops trying to send this particular body of data on reaching MAX_RETRANSMIT for any payload or separately tries to initiate the new transmission of the payloads that have not been acknowledged under these adverse traffic conditions.

If transient network losses are possible, then the use of NON should be considered.

A.2. Q-Block2 Option

An example of the use of the Q-Block2 option with Confirmable messages is shown in Figure 18.

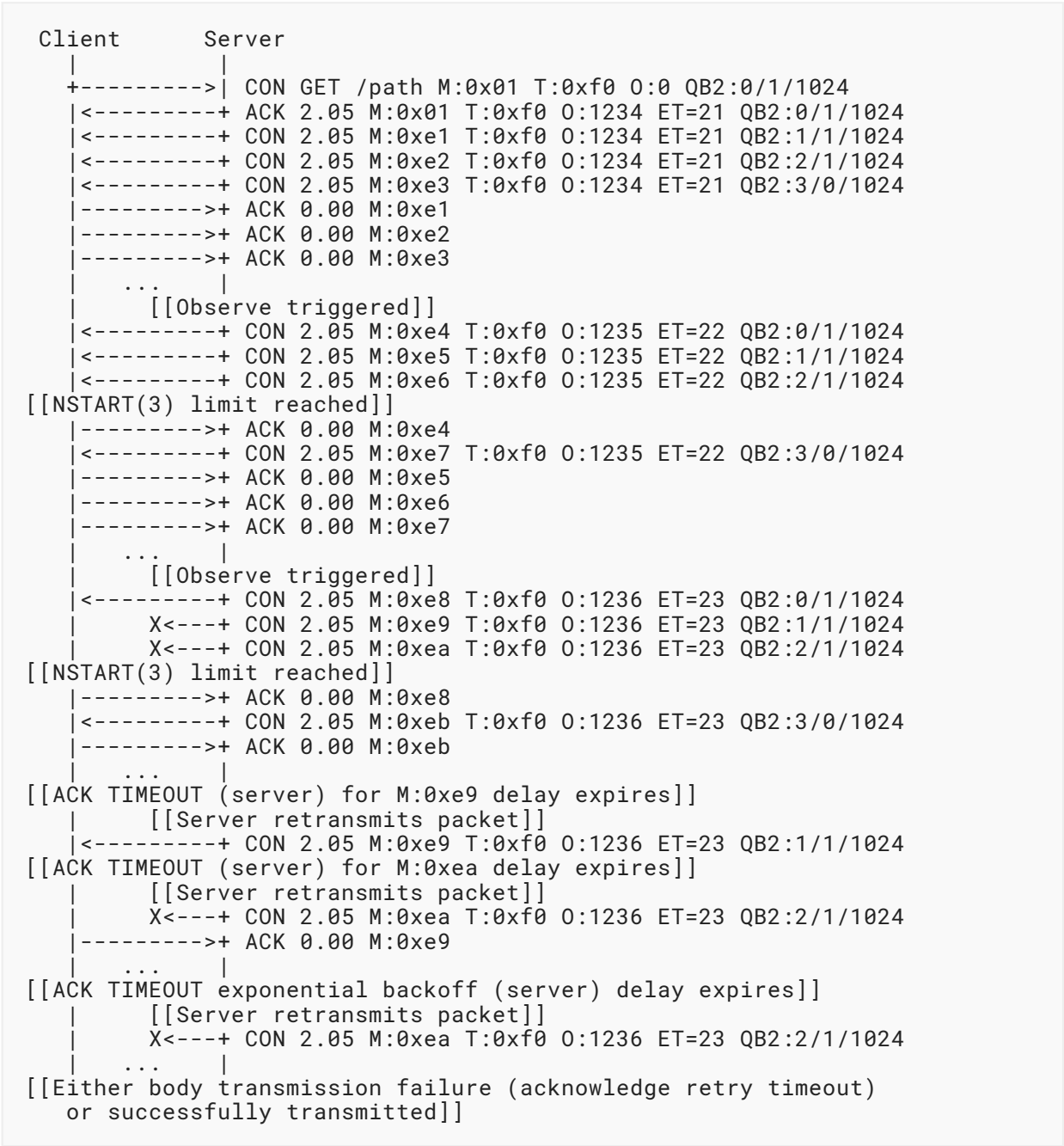

Figure 18: Example of CON Notifications with the Q-Block2 Option

It is up to the implementation as to whether the application process stops trying to send this particular body of data on reaching MAX_RETRANSMIT for any payload or separately tries to initiate the new transmission of the payloads that have not been acknowledged under these adverse traffic conditions.

If transient network losses are possible, then the use of NON should be considered.

Appendix B. Examples with Reliable Transports

The conventions provided in Section 10 are used in the following subsections.

B.1. Q-Block1 Option

Let's now consider the use of the Q-Block1 option with a reliable transport, as shown in Figure 19. There is no acknowledgment of packets at the CoAP layer, just the final result.

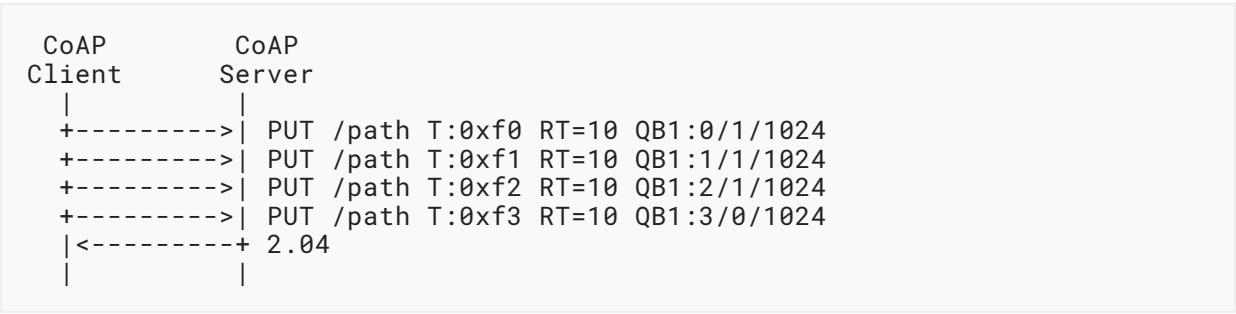

Figure 19: Example of a Reliable Request with the Q-Block1 Option

If transient network losses are possible, then the use of unreliable transport with NON should be considered.

B.2. Q-Block2 Option

An example of the use of the Q-Block2 option with a reliable transport is shown in Figure 20.

```
Client      Server
  |          |
  +--------->| GET /path T:0xf0 O:0 QB2:0/1/1024
  |<---------+ 2.05 T:0xf0 O:1234 ET=21 QB2:0/1/1024
  |<---------+ 2.05 T:0xf0 O:1234 ET=21 QB2:1/1/1024
  |<---------+ 2.05 T:0xf0 O:1234 ET=21 QB2:2/1/1024
  |<---------+ 2.05 T:0xf0 O:1234 ET=21 QB2:3/0/1024
  |   ...    |
  |     [[Observe triggered]]
  |<---------+ 2.05 T:0xf0 O:1235 ET=22 QB2:0/1/1024
  |<---------+ 2.05 T:0xf0 O:1235 ET=22 QB2:1/1/1024
  |<---------+ 2.05 T:0xf0 O:1235 ET=22 QB2:2/1/1024
  |<---------+ 2.05 T:0xf0 O:1235 ET=22 QB2:3/0/1024
  |   ...    |
```

Figure 20: Example of Notifications with the Q-Block2 Option

If transient network losses are possible, then the use of unreliable transport with NON should be considered.

Acknowledgments

Thanks to Achim Kraus, Jim Schaad, and Michael Richardson for their comments.

Special thanks to Christian Amsüss, Carsten Bormann, and Marco Tiloca for their suggestions and several reviews, which improved this specification significantly. Thanks to Francesca Palombini for the AD review. Thanks to Pete Resnick for the Gen-ART review, Colin Perkins for the TSVART review, and Emmanuel Baccelli for the IOT-DIR review. Thanks to Martin Duke, Éric Vyncke, Benjamin Kaduk, Roman Danyliw, John Scudder, and Lars Eggert for the IESG review.

Some text from [RFC7959] is reused for the readers' convenience.

Authors' Addresses

Mohamed Boucadair
Orange
35000 Rennes
France
Email: mohamed.boucadair@orange.com

Jon Shallow
United Kingdom
Email: supjps-ietf@jpshallow.com