CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 8, 2011

A. Castellani
University of Padova

March 7, 2011

Constrained Messaging Protocol: an UDP protocol extension useful for CoAP and other protocols. draft-castellani-core-transport-00.txt

Abstract

This document aims at exploring the benefits of designing a more general-purpose transport protocol for CoAP: the proposed protocol is believed to be viable for implementation on constrained devices and proves to be suitable for transporting CoAP request/responses.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 8, 2011.

Copyright Notice

Table of Contents

1. Introduction

Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] design is focused on defining a REST protocol for constrained devices.

During the process of defining the CoAP protocol, a complementary effort has been devoted to the definition of an appropriate "message"-layer, providing basic transport features to facilitate the CoAP protocol design.

2. Motivation

The motivation of this document has been to provide a more clear "message"-layer definition and to add more features to it.

In order to maximize the reusability of this work and to provide a more clear definition of the trasport features, protocol definition has been specified separately from CoAP.

Transport relevant for CoAP is specified in this document as an UDP protocol extension called Constrained Messaging Protocol (CMP).

CMP design is based upon the "message"-layer defined in CoAP, to which the following features have been added:
- multiple message session handling;
- multi-datagram message transport;
- clear session termination definition;
- partly ordered delivery of unconfirmed messages.

These features are especially of interest in the realization of advanced features of CoAP, i.e. [I-D.ietf-core-block] and [I-D.ietf-core-observe].

Thanks to the more general scope of this specification, it provides also a more clear definition of transport features already introduced in CoAP.

Parts of the text present in [I-D.ietf-core-coap] is reusable in CMP, that parts have been explicitly referenced in this specification.

3. Constrained Messaging Protocol (CMP)

Constrained Messaging Protocol (CMP) is an UDP protocol extension intended to reliably or unreliably transport messages from an endpoint to another, and to unreliably transmit messages to a multicast destination.

CMP adds the following features on top of UDP:

- clear session definition supporting multiple messages;
- reliable message transport handling retransmissions and network duplication;
- unreliable message transport partly handling duplication and out-of-order delivery;
- reliable message fragmentation in multiple UDP datagrams.

Differently from TCP segments, using CMP no more than one datagram can be reliably transmitted in a single round-trip time. This important limitation is the main difference between UDP+CMP and TCP and leads to a simpler protocol design, specifically intended to allow its implementation even on very constrained devices.

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

Along the CMP specification some terms have a specific meaning:

Datagram: input received by the UDP layer.

Message: output provided to the higher-layer.

A single message can be formed by multiple datagrams. Some datagrams are called "transparent" to the higher-layer, thus do not generate a message.

See Section 3.4 for more details about which datagrams generate higher-layer messages.

CMP specification is carried in two distinct phases, the whole protocol is defined in terms of the fields required for operation.

Required fields are discussed in Section 3.1.

Fields mapping to CoAP header has lower relevance to the protocol specification, and is discussed in Section 13.

3.1. Protocol fields

CMP requires the following fields for operation:

Datagram type (T): 2-bit unsigned integer. Indicates if this datagram is of type Interrogative (0), Exclamative (1), Partial (2) or Reset (3). See Section 3.4 for the semantics of this datagram types.

More messages (M): 1-bit boolean. Indicates that the endpoint has more messages pending for this session.

Sequence number (Seq): K-bit unsigned integer. Indicates the sequence number of the datagram in the session.

Sequence number size (K) does not change the protocol specification as long as K is higher than 2. If K is equal to 1, the protocol is still valid but has limited capabilities out-of-order handling capabilities.

Performance and validity of the specified protocol depending upon the actual value of K requires deeper investigation currently not present in this version of the document. Some considerations on this topic follow.

When K is small, unreliable message duplication or out-of-order delivery can occur more easily.

Reliable message delivery is always in-order. Reliable message duplication is easier when K is smaller, e.g. late network duplication or intentional duplication by a malicious node. When K is small, reliable message duplication (even if very unlikely) can occur more easily.

3.2. Endpoint turn

CMP introduces the concept of "in turn" endpoint.

An endpoint is said to be "in turn" if it has the ability to reliably send a message.

Endpoints not "in turn" can only send unreliable messages.

In any session, the client always starts "in turn".

The turn is passed to the other endpoint sending a reliable message or sending the last message in the session.

Detailed rules for turn handling are discussed in the following sections.

3.3. Endpoint local variables

CMP endpoints MUST internally keep the following variables related to the Seq field:

iSeq: is the last reliable message sequence number known by an

endpoint.

eSeq: is the last message sequence number known by an endpoint (both reliable and unreliable).

Only endpoints "in turn" can emit sequence numbered messages.

Endpoints not "in turn" can emit only unreliable messages, Seq field of those messages MUST be ignored.

3.4. Datagram types and messages

Each unique non-empty Interrogative (INT) or Exclamative (EXC) datagram always corresponds to a unique message.

Empty INT or EXC datagrams do not generate any message. Partial (PAR) or Reset (RST) datagrams do not generate any message.

Even if non-empty PAR datagrams do not generate any message, their content is always part of a message corresponding to the INT datagram emitted after a set of PAR datagram(s).

Network duplication or retransmissions can generate multiple identical datagrams: rules to avoid message duplication are discussed in Section 7.

The type of a datagram is specified by the T field of the CMP header. The different types of datagrams are summarized below.

3.4.1. Reset (RST)

A RST datagram is used by an endpoint to abruptly close a session.

When a RST is received by an endpoint, it MUST immediately consider the session as closed.

3.4.2. Interrogative (INT)

Some messages require to receive a response or at least an acknowledgment by the other endpoint. These messages MUST be trasmitted using INT datagrams and are called INT messages.

A unique message is generated by each unique INT datagram.

Endpoints MUST NOT emit more than one INT message before receiving at least one INT datagram (even empty) from the other endpoint.

The current endpoint able to send the INT datagram is called "in

turn". When it emits one INT datagram it passes the turn to the other endpoint. At any time there is only one endpoint "in turn". When the session is started, the client is "in turn".

An endpoint emitting a non-empty INT datagram MUST increase iSeq by 1, MUST set eSeq to iSeq and MUST set Seq field accordingly. Empty INT datagrams MUST be emitted with Seq field set to iSeq.

An endpoint can pass the turn to the other endpoint by trasmitting an INT datagram (even empty).

INT datagrams can be retrasmitted. Thus multiple identical datagrams corresponding to the same message could be emitted. Retransmission rules are discussed in Section 6.

If an "in turn" endpoint receives either an INT or PAR datagram, it MUST ignore that datagram.

3.4.3. Exclamative (EXC)

Some messages do not require to receive a response nor an acknowledgment from the other endpoint. These messages SHOULD be transmitted using EXC datagrams and are called EXC messages.

An unique message is generated by each unique EXC datagram.

Differently from INT datagrams, EXC datagrams can be emitted with lower restrictions by an endpoint.

An "in turn" endpoint emitting non-empty EXC datagram MUST increase eSeq by 1 and set Seq field to eSeq, empty EXC datagrams MUST be emitted with Seq field set to eSeq. Moreover not "in turn" endpoints MUST always set Seq field to iSeq.

Because of the implicitly unruled nature of this message type, in order to avoid congestion collapse, endpoints emitting EXC messages MUST limit their output rate to a value known to be suitable along the whole network path.

If an "in turn" endpoint emits an EXC datagram with M=0, the turn is passed to the other endpoint.

3.4.4. Partial (PAR)

Some messages do not fit into a single datagram. Large messages can be fragmented into multiple PAR datagrams. These messages are a special class of INT messages and are called fragmented INT messages.

A multi-datagram message is formed by the concatenation of a set of PAR messages, except its final part that MUST be trasmitted using an INT message type.

An endpoint to emit a PAR datagram MUST be "in turn".

An endpoint emitting a non-empty PAR datagram MUST increase iSeq by 1 and set Seq field to iSeq, empty PAR datagrams MUST be emitted with Seq field set to iSeq.

Endpoints MUST NOT emit more than one PAR datagram before receiving an empty PAR message from the other endpoint. An empty PAR message serves as an acknowledgment of the reception of the current part.

Handling of PAR messages is not mandatory, endpoints not supporting this datagram type MUST immediately send a RST datagram to the other endpoint.

3.4.5. Acknowledgment (ACK)

Differently from INT, EXC, PAR and RST datagrams, ACK datagrams are not an explicit datagram type.

An ACK datagram acknowledges a message transported in an INT datagram.

Any datagram of type INT, EXC or PAR emitted after reception of an INT datagram assumes the role of ACK datagram.

To successfully acknowledge a message, the Seq field contained in the ACK:

- MUST be equal to iSeq, if the ACK is an empty datagram;
- MUST be equal to iSeq+1, if the ACK is a non-empty datagram.

3.4.6. Keep-alive (KPA)

Also KPA datagrams are not an explicit datagram type.

A KPA datagram is an empty PAR message with Seq set to iSeq.

A KPA datagram is used to ping the other endpoint, either to check that the remote session is still alive or to refresh any soft state along the network path.

When a KPA datagram is received on an open session, if Seq is equal to iSeq, the endpoint MUST send an identical datagram to the other endpoint.

3.5. Sequence number

Every endpoint keeps track of the Seq field, using two different local variables iSeq and eSeq.

Client endpoint, before sending the first message, SHOULD initialize iSeq and eSeq to the same random value.

"In turn" endpoints MUST NOT update iSeq and eSeq when receiving a datagram. Not "in turn" endpoints MUST NOT update iSeq and eSeq when sending a datagram.

At every received non-empty PAR or INT datagram, if Seq field in the received datagram is equal to iSeq+1, a not "in turn" endpoint MUST set iSeq to that value.

At every received datagram, if Seq field in the received datagram is equal to iSeq+1, a not "in turn" endpoint MUST set eSeq to that value.

There are more rules for updating the eSeq variable, these rules are related with out-of-order reception of datagrams and are described in Section 8.

When a non-empty PAR, EXC or INT datagram is to be sent by an "in turn" endpoint, iSeq MUST be increased by 1, eSeq set to iSeq and Seq field set accordingly.

When a non-empty EXC datagram is to be sent by an "in turn" endpoint, eSeq MUST be increased by 1 and Seq field set accordingly.

4. Session handling

CMP client session MUST be opened when sending a message to a server. The client MUST select a locally unused source UDP port.

CMP server session SHOULD be opened when receiving a message from a client from a remotely unused source UDP port. If a server cannot open a new session, it MUST immediately send a RST datagram to the client.

A CMP session can be closed either by agreed termination, by arbitrary reset, by endpoint drop or by timeout.

A CMP session is called "terminated" or "closed by agreement" and MUST be closed when all the following conditions are verified:

- Last received datagram has field M=0;
- Last sent datagram has field M=0;
- Last datagram (either sent or received) is of type EXC.

A CMP session is called "reset" or "closed by reset" and MUST be closed when an endpoint sends or receives a RST datagram.

A CMP session is called "dropped" or "closed by drop" and MUST be closed when an endpoint receives no acknowledge to an INT (or PAR) datagram even after MAX_RETRANSMIT retransmissions.

A CMP session is called "timed-out", if no message has been exchanged during CMP_TIMEOUT seconds and SHOULD be closed. Before closing the session the endpoint timing out SHOULD send a KPA datagram to the other endpoint. If no ACK is received back, the already "timed-out" session is also in state "dropped" and MUST be closed.

The endpoint sending the last ACK datagram in a session closed by agreement, MUST keep the session CLOSING state for RESPONSE_TIMEOUT^(MAX_RETRANSMIT-1) seconds.

The client endpoint MUST NOT reuse the same UDP source port used for a previous session before CMP_TIMEOUT seconds have passed after that session has been closed.

4.1. Matching active sessions

Assuming that between two hosts (IPs) there can exist at most one session between two specific UDP ports, protocol session matching can be performed without any further addition on the upper layer protocol.

In a client/server protocol the server is usually spawned on a static port. However, even using a static server port, session matching is still feasible as long as concurrently active clients on a host can use different source ports.

E.g. TCP successfully performs session matching using source/ destination IP and port.

Thus, to successfully perform session matching, as described in this section, it is sufficient that the following synthetic condition is verified:

At any time, there MUST NOT exist two or more concurrent sessions having all the listed parameters equal between themselves:

(source IP, source UDP port, destination IP, destination UDP port)

5. Delayed acknowledging

After receiving a new message in an INT datagram, the endpoint SHOULD wait ACK_TIMEOUT seconds before sending the related ACK datagram.

If the upper-layer provides a new message during this period of time, the new message will be sent immediately piggybacking an ACK to the previous INT datagram.

If the upper-layer has already signaled to the CMP layer that it has no more messages pending in this session, an empty ACK datagram MUST be sent immediately:

- Using an EXC datagram, if the acked datagram has M=0;
- Using an INT datagram, if the acked datagram has M=1.

6. Datagram retransmission

Retransmissions are handled as defined in Section 4.1 of [I-D.ietf-core-coap], for convenience a synthetic description of these rules follows.

An endpoint sending an INT (or PAR) datagram MUST set the session retransmit counter to 0 and MUST start the retransmit timeout.

The retransmit timeout duration is always equal to RESPONSE_TIMEOUT^(retransmit counter).

When an ACK datagram for the sent INT (or PAR) datagram has been received, the retransmit timeout MUST be stopped.

When the retransmit timeout is triggered, the retransmit counter MUST be increased by 1 and depending upon the retransmit counter value one the following actions MUST be done:

If the retransmit counter is less than MAX_RETRANSMIT, the involved INT (or PAR) datagram MUST be sent again and the retransmit timeout MUST be restarted.

Otherwise if the retransmit counter is equal to MAX_RETRANSMIT, the message is dropped and the related session MUST be closed. Higher-layer SHOULD be notified that the message has been dropped.

7. Datagram duplication

Thanks to the reliable nature of INT messages, duplication of these messages can be avoided by handling datagram duplication.

Not "in turn" endpoints MUST NOT accept non-empty INT or PAR datagrams with Seq field not equal to iSeq+1 and MUST NOT accept empty INT or PAR datagrams with Seq field equal to iSeq.

"In turn" endpoint MUST NOT accept INT or PAR datagrams.

Due to unreliable nature of EXC messages, duplication of these messages can only be partly avoided.

Endpoints SHOULD NOT deliver to the upper layer two EXC messages with identical UDP checksum within DUP_TIMEOUT seconds. "In turn" endpoints MAY relax this condition and deliver every EXC message received, not "in turn" endpoints emit EXC messages with identical Seq field (easier UDP checksum collision).

8. Out-of-order message delivery

There is either zero or one INT message in transit between two endpoints, for this reason INT messages are never delivered out-of-order.

Out-of-order delivery of EXC messages emitted by "in turn" endpoints can be partly handled.

If a not "in turn" endpoint receives a non-empty EXC datagram with Seq field:

- between eSeq+2 and eSeq+2^(K-1), the datagram SHOULD be queued for delivery for OOO_TIMEOUT seconds, when the OOO_TIMEOUT timer expires the message SHOULD be delivered to the upper-layer;
- equal to eSeq+1, the contained message MUST be delivered immediately to the upper-layer and if a datagram queued with Seq field equal eSeq+1 is present, it is dequeued and processed as if it was just received;
- between eSeq-2^(K-1)+1 and eSeq, it is probably an out-of-order older message and, depending upon applications, MAY be delivered to the upper-layer or dropped.

Out-of-order delivery of EXC messages emitted by not "in turn" endpoints cannot be handled at all. Applications sensible to out-of-order delivery SHOULD avoid using EXC messages when not "in turn".

9. Congestion control

As noted in Section 4.5 of [I-D.ietf-core-coap], basic congestion control is provided by the retransmission exponential back-off synthesized in Section 6 and further discussion on congestion control

can be found in [I-D.eggert-core-congestion-control].

10. Multi-datagram messaging

Some message requires to be fragmented in many datagrams, these messages can be only transmitted as INT messages.

Every fragment of the message MUST be sent in order using PAR datagrams, except the last fragment which MUST be sent in an INT datagram.

After emitting a new fragment the endpoint MUST wait that the previous fragment has been acknowledged with an empty PAR datagram.

If the endpoint does not receive an empty PAR datagram within RESPONSE_TIMEOUT seconds, retransmission rules discussed in Section 6 apply.

An endpoint can close at any time a multi-datagram exchange by emitting a RST datagram (e.g. out-of-memory).

Multi-datagram messages can be emitted only if "in turn".

11. Multicast

Only EXC messages MUST be sent to a multicast address.

Congestion control for multicast communication is not defined in this document. For this reason endpoints MAY NOT support multicast communication thus multicast messages MUST be emitted at a very limited rate to avoid congestion collapse in the constrained network.

Even if the protocol definition could be easily extended to support reliable multicast message transport, the implication of this on network congestion have to be deeply analyzed before defining this behaviour.

12. Protocol constants

This section defines the relevant protocol constants defined in this document.

ACK_TIMEOUT 0.2 seconds

DUP_TIMEOUT 5 seconds

OOO_TIMEOUT 2 seconds

CMP_TIMEOUT 120 seconds

RESPONSE_TIMEOUT and MAX_RETRANSMIT are defined in Section 9 of [I-D.ietf-core-coap] and reported here for convenience of reading:

RESPONSE_TIMEOUT 2 seconds

MAX_RETRANSMIT 4

13. CoAP header format over UDP+CMP and over TCP

The protocol defined in the previous Sections can be easily plugged into the current CoAP design.

Figure 1 shows the final CoAP/UDP+CMP design.

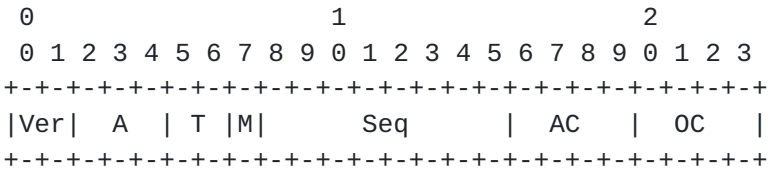

Figure 1: CoAP/UDP+CMP header format

The definition of Ver and OC fields described in Section 3.1 of [I-D.ietf-core-coap] is not changed.

The definition of fields A and AC described in Section 3.2 and 3.2.1 of [I-D.castellani-core-coap-overhead] is not changed.

The definition of fields T, M and Seq described in Section 3 is not changed.

The whole REST protocol semantics and specification described in [I-D.ietf-core-coap] is still valid for CoAP/CMP.

Using a CMP transport protocol extension for UDP, allows also to easily define how CoAP can be trasported over TCP. A proposal for CoAP/TCP header format is shown in Figure 2.

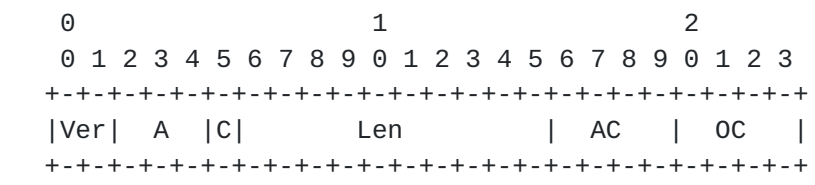

Figure 2: CoAP/TCP header format

The definition of fields already present in Figure 2 is not changed.

The new field Len is defined as follows:

Concatenate messages (C): 1-bit unsigned integer. This field indicates whether the current message should wait for more fragments.

Message lenght (Len): 10-bit unsigned integer. This field indicates the total length of the CoAP message.

If C=0 the message is completed, otherwise if C=1 the message is not completed and more fragments will follow.

The whole message is the result of the concatenation of all the fragments. Thus until a message with C=0 is received, the message MUST NOT be processed.

Ver, A, AC and OC are present in every fragment, to avoid confusion only the value present in the first fragment MUST be processed.

14. Examples

A lot of examples of CMP can be realized, however only a small set is provided in the current version of this document.

More examples will be provided in a planned version of this document.

Even if CMP design is more general, the proposed examples are focused on its application to CoAP.

Examples of retransmissions, duplication and out-of-order handling are not present in the current version of this document, but are intended to be presented in a future version.

For easier reading Seq field has always been initialized to 1.

The current turn before and after the datagram is explicitly indicated using turn=A->B, where A is the turn before the datagram, and B the turn after the datagram. B=X means that no endpoint is in

turn, this happens when the session is closed.

Figure 3 shows a very simple exchange, where a client reliably transmits a message to a server.

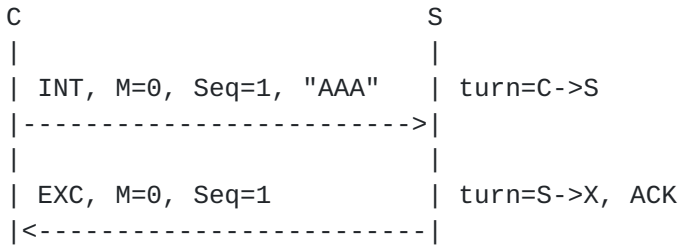

Figure 3: Single INT message, no response

Figure 4 shows a typical CoAP request/immediate-response exchange, the client reliably transmits a request to a server, which sends immediately back an unreliable response.

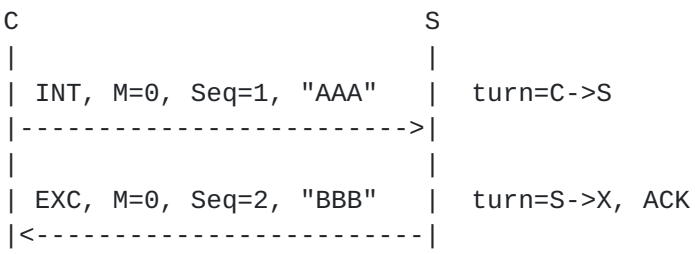

Figure 4: Single INT request, single EXC response

Figure 5 shows a typical CoAP request/deferred-response exchange, the client reliably transmits a request to a server, which later sends reliably back a response.

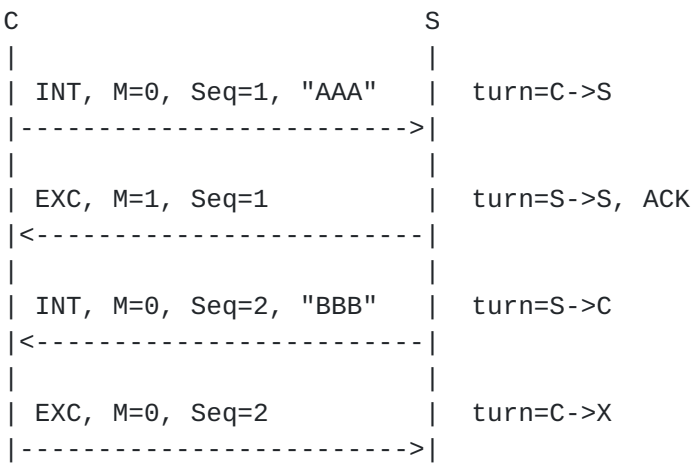

Figure 5: Single INT request, deferred INT response

Figure 6 shows a server sending back a multi-datagram response.

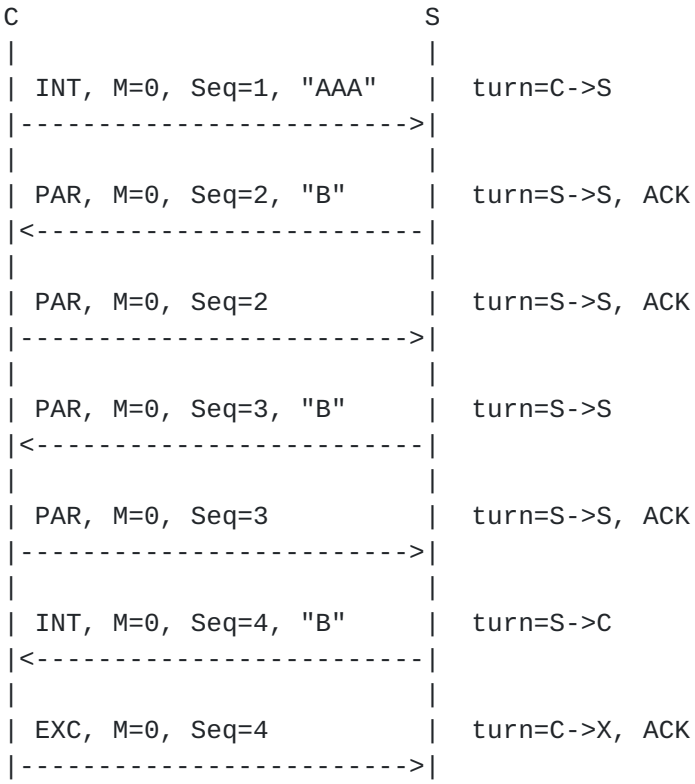

Figure 6: Single INT request, single multi-datagram INT response

Figure 7 shows a client sending a multi-datagram request.

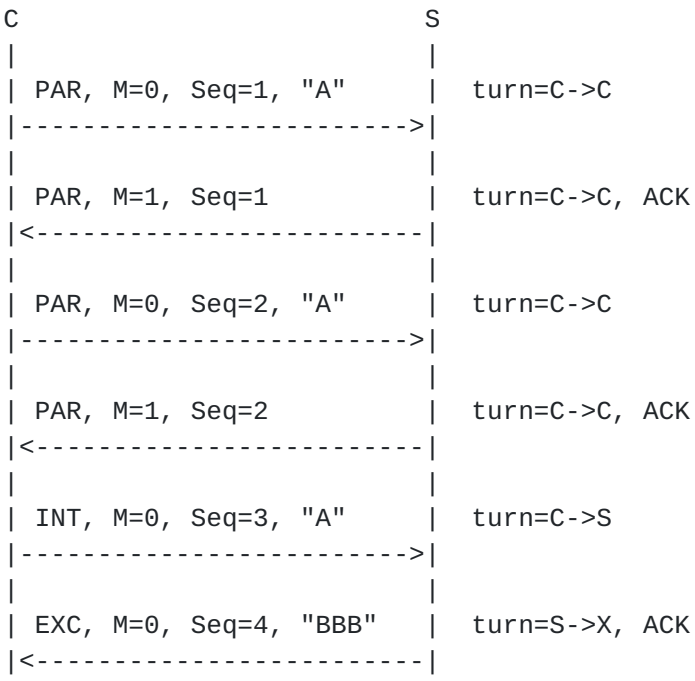

Figure 7: Single multi-datagram INT request, single EXC response

Figure 8 shows a typical CoAP observe session, the client reliably transmits a request to a server, which starts sending infrequent messages back to the client.

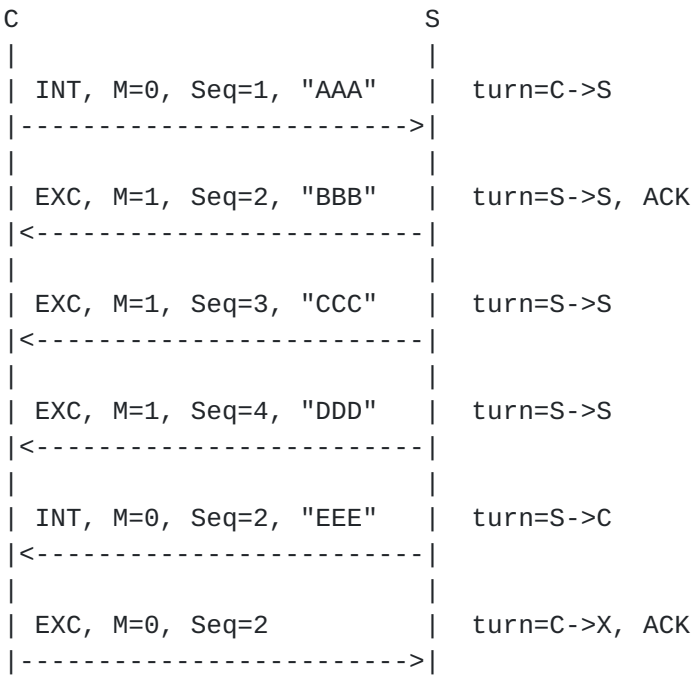

Figure 8: Single INT request, multiple EXC or INT responses

Figure 9 shows a typical CoAP multicast request/response, the client sends an unreliable request to a multicast address, all the servers bound to that address will send back an unreliable response.

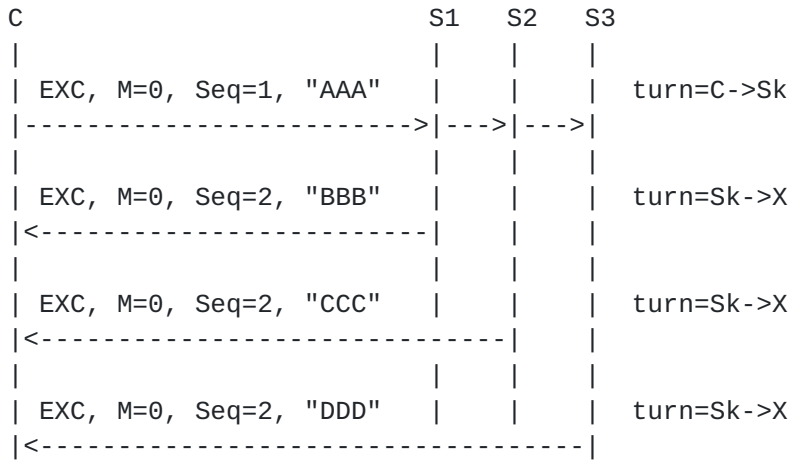

Figure 9: Single multicast EXC request, multiple EXC responses

15. Security considerations

TBD

16. IANA considerations

TBD

17. Acknowledgements

Special thanks to Nicola Bui, Michele Zorzi and Mattia Gheda for the extensive discussions on this protocol, for their support and contributions.

The author of this document relied upon the extensive work done by the members of CoRE working-group.

Especially the work done by the editors and contributors of [I-D.ietf-core-coap], in particular the extensive analysis of issues and techniques strictly related to this document and also for the definitions this document shares with [I-D.ietf-core-coap].

18. References

18.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.castellani-core-coap-overhead]
Castellani, A., "CoAP overhead: protocol analysis and reduction proposals", draft-castellani-core-coap-overhead-00 (work in progress), February 2011.

18.2. Informative References

[I-D.ietf-core-block]
Shelby, Z. and C. Bormann, "Blockwise transfers in CoAP", draft-ietf-core-block-01 (work in progress), January 2011.

[I-D.ietf-core-observe]
Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-01 (work in progress), February 2011.

[I-D.eggert-core-congestion-control]
Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.

Author's Address

Angelo P. Castellani
University of Padova
Via G. Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net