oneM2M
Technical Specification

Document Number	TS-0010-V1.5.1
Document Name:	MQTT Protocol Binding
Date:	2016-February-29
Abstract:	This document defines the binding of the oneM2M protocols to an MQTT transport layer.

Contents

1 Scope

The present document specifies the binding of Mca and Mcc primitives (message flows) onto the MQTT protocol. It specifies

1) How a CSE or AE connects to MQTT.
2) How an Originator (CSE or AE) formulates a Request as an MQTT message, and transmits it to its intended Receiver.
3) How a Receiver listens for incoming Requests.
4) How that Receiver can formulate and transmit a Response.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

The following referenced documents are necessary for the application of the present document.

[1] OASIS MQTT Version 3.1.1 (29 October 2014). OASIS Standard. Edited by Andrew Banks and Rahul Gupta.

NOTE: Available at http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[2] oneM2M TS-0001: "Functional Architecture".

[3] oneM2M TS-0004: "Service Layer Core Protocol Specification".

[4] IETF RFC 793 (September 1981): "Transmission Control Protocol - DARPA Inetернet Program - Protocol Specification", J. Postel.

NOTE: Available at http://www.ietf.org/rfc/rfc793.txt.

[5] IETF RFC 5246 (August 2008): "The Transport Layer Security (TLS) Protocol Version 1.2", T. Dierks.

NOTE: Available at http://tools.ietf.org/html/rfc5246.

[6] IETF RFC 6455 (December 2011): "The WebSocket Protocol", I. Fette.

NOTE: Available at http://tools.ietf.org/html/rfc6455.

[7] oneM2M TS-0003: " Security Solutions".

[8] IETF RFC 3986 (January 2005): " Uniform Resource Identifier (URI): Generic Syntax", T. Berners-Lee.

NOTE: Available at https://tools.ietf.org/html/rfc3986.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1]	oneM2M Drafting Rules.

NOTE: Available at http://ftp.onem2m.org/Others/Rules_Pages/oneM2M-Drafting-Rules-V1_0_1.doc

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

originator [2]: actor that initiates a Request

NOTE: An Originator can either be an Application or a CSE.

receiver [2]: actor that receives the Request

NOTE: A Receiver can be a CSE or an Application.

resource [2]: uniquely addressable entity in oneM2M System such as by the use of a Uniform Resource Identifier (URI)

NOTE: A resource can be accessed and manipulated by using the specified procedures.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in oneM2M TS-0001 [2] and the following apply:

ADN	Application Dedicated Node
ADN-AE	AE which resides in the Application Dedicated Node
AE	Application Entity
ASN	Application Service Node
CSE	Common Service Entity
IN	Infrastructure Node
IN-AE	Application Entity that is registered with the CSE in the Infrastructure Node
IN-CSE	CSE which resides in the Infrastructure Node
MN	Middle Node
MN-CSE	CSE which resides in the Middle Node
TLS	Transport Level Security

4 Conventions

The keywords "Shall", "Shall not", "May", "Need not", "Should", "Should not" in the present document are to be interpreted as described in the oneM2M Drafting Rules [i.1].

5 Introduction

5.1 Use of MQTT

This binding makes use of MQTT to provide reliable two-way communications between two parties (AEs and CSEs). It uses the following features of MQTT:

- Durable Sessions, providing Store and Forward in cases where network connectivity is not available.
- MQTT's "QoS 1" message reliability level. This provides reliability without incurring the overhead implied by QoS 2.
- NAT traversal (neither of the two parties is required to have prior knowledge of the other party's IP address).
- Dynamic topic creation and wild-carded subscription filters.

It does not use the following features:

- One-to-many publish/subscribe.
- Retained Messages.
- Will Messages.
- QoS 0 or QoS 2 message reliability levels.

5.2 Binding overview

5.2.1 Introduction

The MQTT protocol binding specifies how the Mca or Mcc request and response messages are transported across the MQTT protocol. Both communicating parties (AEs and CSEs) typically make use of an MQTT client library, and the communications are mediated via the MQTT server. There is no need for the client libraries or the server to be provided by the same supplier, since the protocol they use to talk to each other is defined by the MQTT specification [1].

Furthermore, the binding does not assume that the MQTT client libraries or server implementations are necessarily aware that they are being used to carry Mca, Mcc or any other oneM2M-defined primitives.

The binding is defined in terms of the MQTT protocol flows that take place between the client libraries and the MQTT server in order to effect the transport of an Mca or Mcc message.

There are two scenarios depending on the location of MQTT server: MQTT server co-located within a node, and MQTT server located independently from nodes.

5.2.2 Scenarios

5.2.2.1 MQTT server co-located within a node

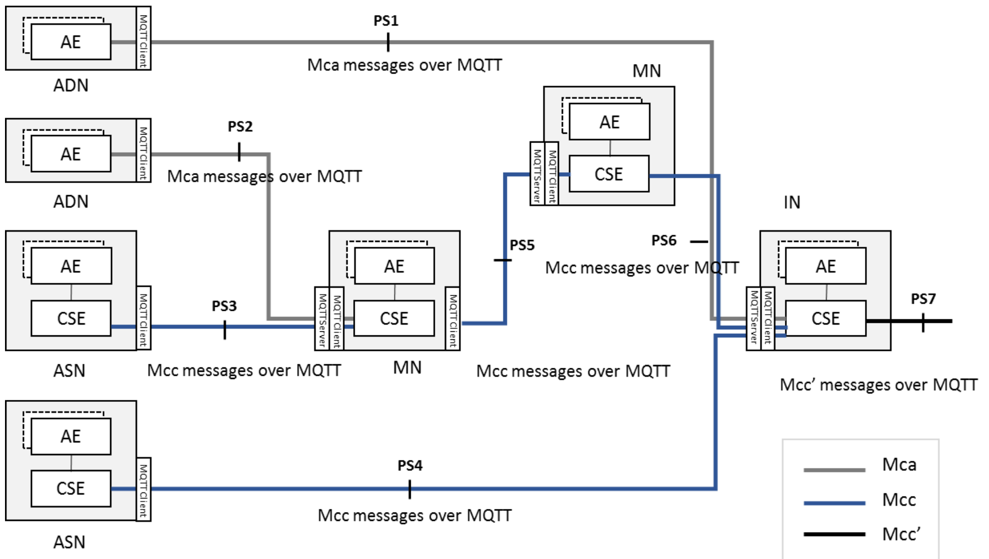

Figure 5.2.2.1-1: MQTT server co-located scenario

Figure 5.2.2.1-1 shows a protocol segment view of the MQTT server co-located scenario. In this scenario, all oneM2M nodes (ADN, ASN, MN, IN) include a MQTT client. MQTT servers are provided within MN and IN.

In this scenario, the protocol segments are illustrated as follows.

Table 5.2.2.1-1

Protocol Segment	oneM2M Message Transported	MQTT Interaction
PS1	Mca (AE of ADN to CSE of IN)	Client in ADN to Server in IN
PS2	Mca (AE of ADN to CSE of MN)	Client in ADN to Server in MN
PS3	Mcc (CSE of ASN to CSE of MN)	Client in ASN to Server in MN
PS4	Mcc (CSE of ASN to CSE of IN)	Client in ASN to Server in IN
PS5	Mcc (CSE of MN to CSE of MN)	Client in MN to Server in MN
PS6	Mcc (CSE of MN to CSE of IN)	Client in MN to Server in IN
PS7	Mcc' (CSE of IN to CSE of IN)	Client in IN to Server in IN

5.2.2.2 MQTT server located independently from nodes

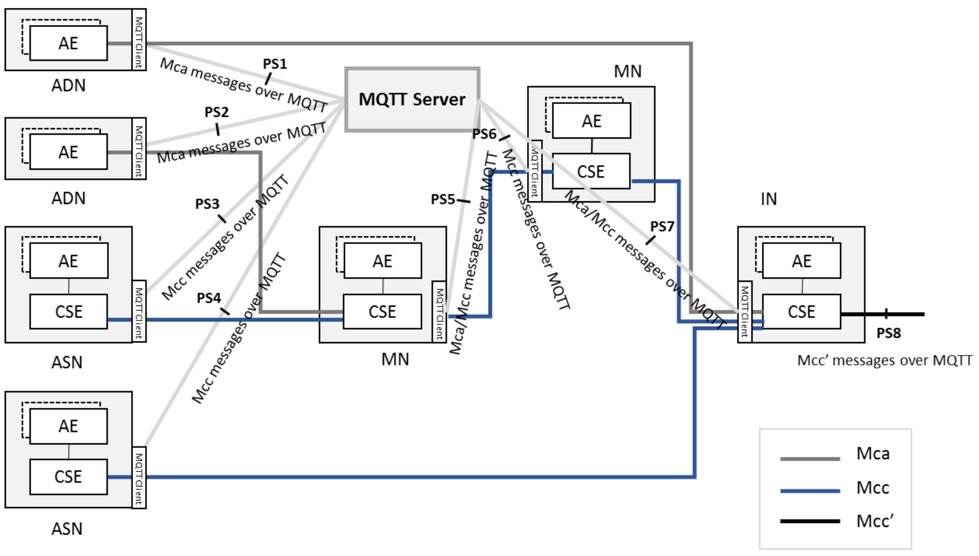

Figure 5.2.2.2-1: MQTT server independently located scenario

Figure 5.2.2.2-1 shows a protocol segment view in which the MQTT server is located independently from the oneM2M nodes. In this scenario, all oneM2M nodes (ADN, ASN, MN, IN) include a MQTT client. MQTT servers exists independently, which means the servers are located outside of the nodes.

In this scenario, the protocol segments are illustrated as follows.

Table 5.2.2.2-1

Protocol Segment	oneM2M Message Transported	MQTT Interaction
PS1	Mca (AE of ADN to CSE of IN)	Client in ADN to Server
PS2	Mca (AE of ADN to CSE of MN)	Client in ADN to Server
PS3	Mcc (CSE of ASN to CSE of MN)	Client in ASN to Server
PS4	Mcc (CSE of ASN to CSE of IN)	Client in ASN to Server
PS5	Mcc (CSE of MN to CSE of MN) Mcc (CSE of MN to CSE of ASN) Mca (CSE of MN to AE of ADN)	Client in MN to Server
PS6	Mcc (CSE of MN to CSE of MN)	Client in MN to Server
PS7	Mcc (CSE of IN to CSE of MN) Mcc (CSE of IN to CSE of ASN) Mca (CSE of IN to AE of ADN)	Client in IN to Server
PS8	Mcc' (CSE of IN to CSE of IN)	Client in IN to Server

The next four clauses show the four configurations in which the MQTT binding can be used in the co-located scenario, followed by similar configurations in the independently-located scenario.

NOTE: Other configurations are possible, but they are currently out of scope.

5.2.3 Configurations

5.2.3.1 AE to IN

This configuration, illustrated in figure 5.2.3.1-1, allows an AE to connect to an IN via MQTT.

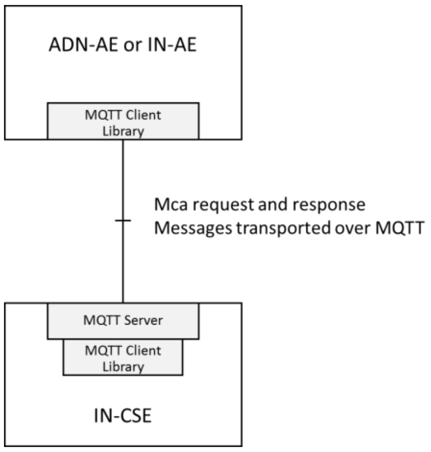

Figure 5.2.3.1-1: Using MQTT between AE and IN-CSE

The MQTT server is co-located with the IN-CSE and allows connection of the ADN-AEs (typically devices) and/or IN-AEs. It can store and forward messages if there is a gap in the connectivity with the devices. Note that the AEs each establish their own separate TCP/IP connection with the MQTT server. Thus the server shall have an accessible IP address, but AEs need not have.

5.2.3.2 AE to MN

This configuration, illustrated in figure 5.2.3.2-1, allows an ADN-AE to connect to an IN via MQTT.

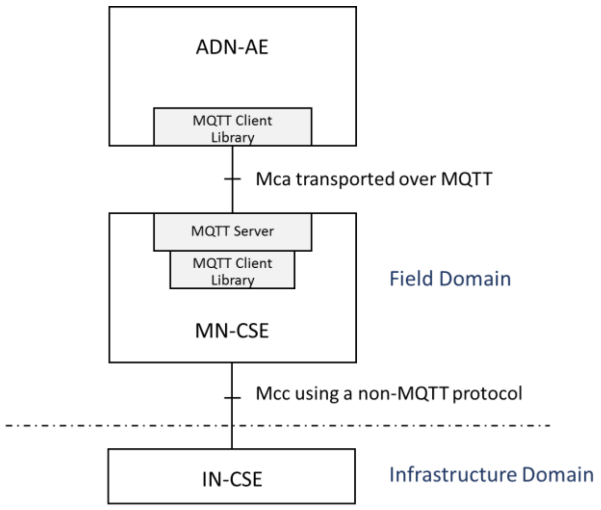

Figure 5.2.3.2-1: Using MQTT between AE and MN-CSE

This configuration is very similar to the AE-IN configuration shown in clause 5.2.3.1, except that the MQTT server is hosted on the MN rather than the IN. Onwards connection to the IN-CSE is via a different transport protocol.

5.2.3.3 MN to IN

This configuration, illustrated in figure 5.2.3.3-1, allows an MN to connect to an IN via MQTT.

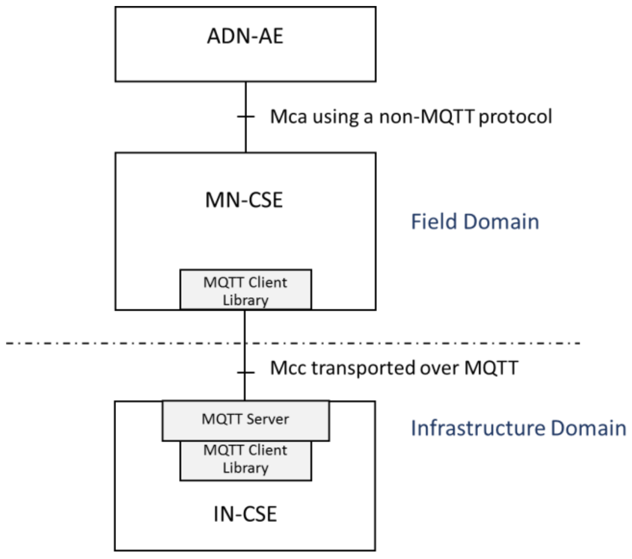

Figure 5.2.3.3-1: Mcc using MQTT between MN and IN

The MQTT server is co-located with the IN-CSE and allows connection of the MNs (typically in-field gateway boxes). It can store and forward messages if there is a gap in the connectivity with the gateways. Note that the MNs each establish their own separate TCP/IP connections with the MQTT server. Thus the server shall have an accessible IP address, but MNs need not have.

5.2.3.4 AE to MN to IN

This configuration, illustrated in figure 5.2.3.4-1, is a combination of the previous two.

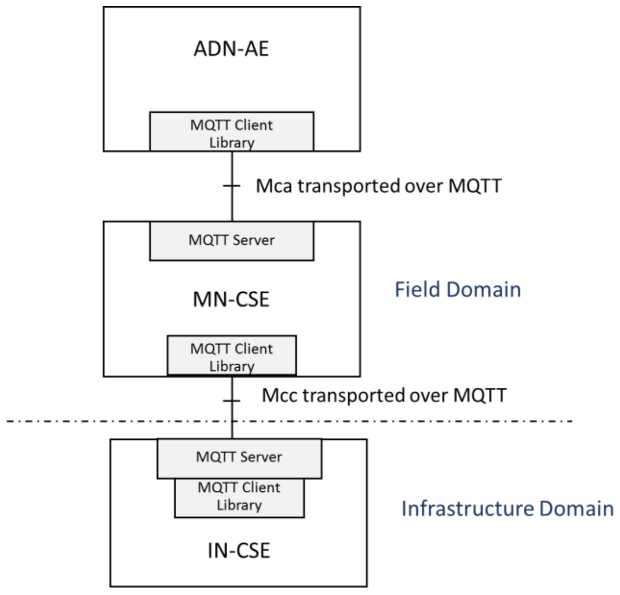

Figure 5.2.3.4-1: Mca and Mcc both using MQTT

In this configuration the two MQTT servers are independent from each other (that is to say they do not have a shared topic space). Any interactions between the ADN-AE and the IN-CSE are mediated by the MN-CSE.

5.2.3.5 AE to IN (Independent scenario)

This configuration, illustrated in figure 5.2.3.5-1, allows an AE to connect to an IN via MQTT.

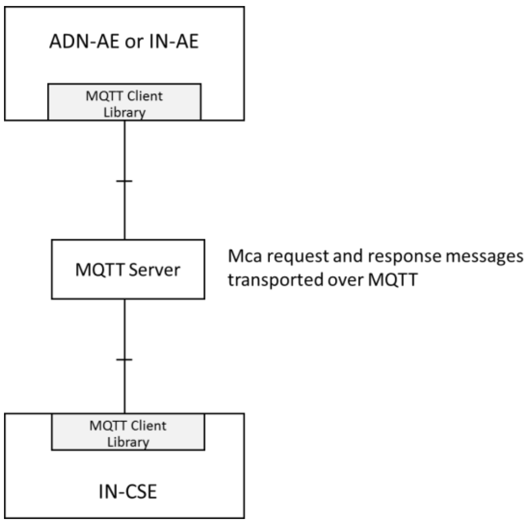

Figure 5.2.3.5-1: Using MQTT between AE and IN-CSE

The MQTT server is an independent entity, located outside of the nodes. In order to deliver Mca messages, MQTT clients within ADN-AE/IN-AE and IN-CSE connect to the MQTT server. After the clients establish TCP/IP connection with the MQTT server, Mca messages between ADN-AE/IN-AE and IN-CSE can be transported via the MQTT server.

5.2.3.6 AE to MN (Independent scenario)

This configuration, illustrated in figure 5.2.3.6-1, allows an ADN-AE to connect to an IN via MQTT.

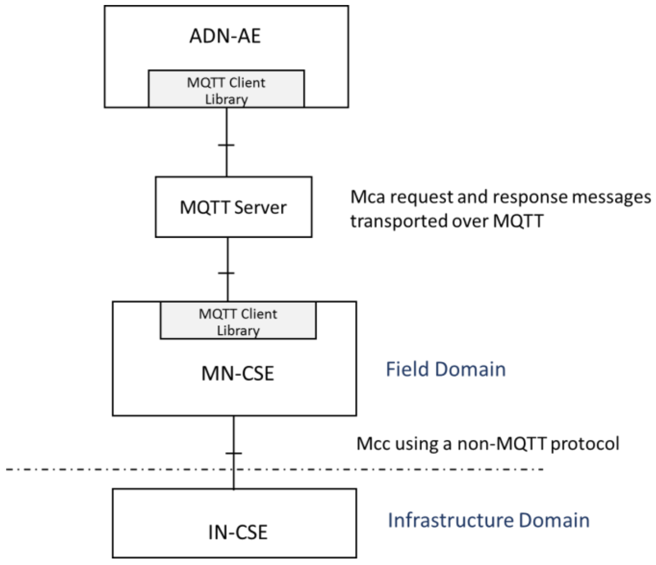

Figure 5.2.3.6-1: Using MQTT between AE and MN-CSE

In this configuration, the MQTT server is an independent entity, located outside of the nodes. MQTT clients within ADN-AE and MN-CSE are connected to the MQTT server, and the MQTT server stores and forwards the Mca messages between ADN-AE and MN-CSE. In addition, this figure shows that the onwards connection to the IN-CSE is via a different transport protocol.

5.2.3.7 MN to IN (Independent scenario)

This configuration, illustrated in figure 5.2.3.7-1, allows an MN to connect to an IN via MQTT.

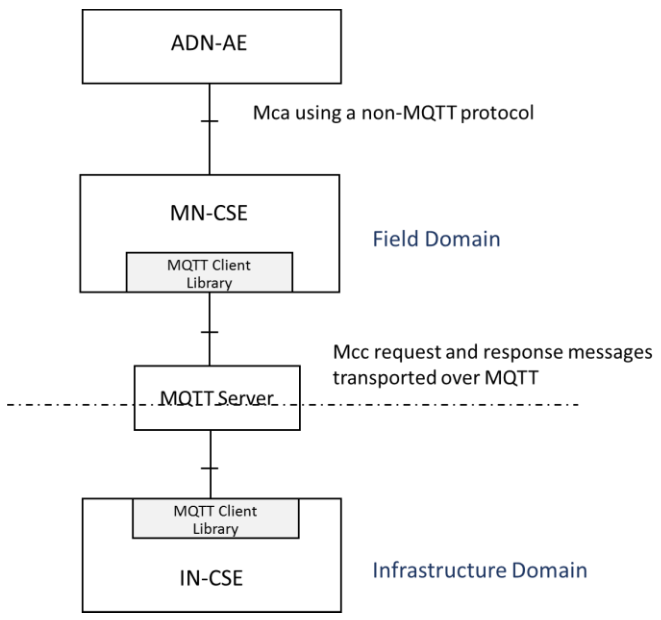

Figure 5.2.3.7-1: Mcc using MQTT between MN and IN

In this configuration, the MQTT server is an independent entity, located outside of nodes. Mcc message delivery between MN-CSE and IN-CSE are performed via the independently located MQTT server. As introduced in the previous clauses, in order to send messages, each MQTT client within MN-CSE and IN-CSE connects to the MQTT server and Mcc messages are transported via MQTT server.

5.2.3.8 AE to MN to IN (Independent scenario)

This configuration, illustrated in figure 5.2.3.8-1, is a combination of the previous two.

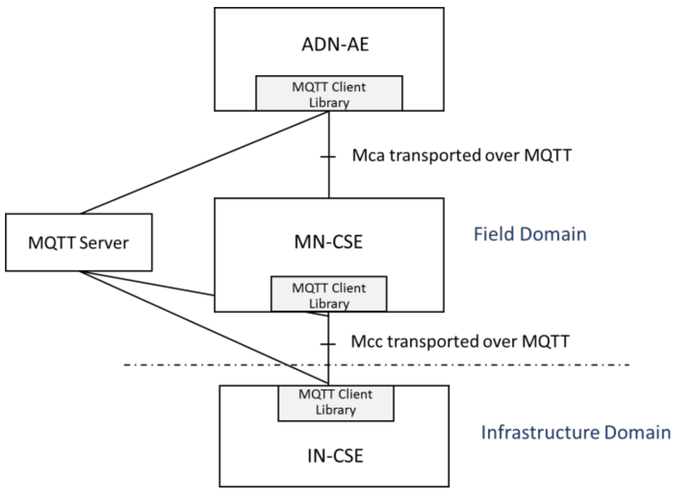

Figure 5.2.3.8-1: Mca and Mcc both using MQTT

In this configuration, the MQTT clients of ADN-AE and MN-CSE and IN-CSE connect to the independently located MQTT server. Any interactions such as Mca or Mcc message delivery among the ADN-AE and the MN-CSE and the IN-CSE are mediated by the MQTT server.

6 Protocol Binding

6.1 Introduction

In this clause the use of MQTT is profiled and the key elements of the binding are defined:

1) How a CSE or AE connects to MQTT.

2) How an Originator (CSE or AE) formulates a Request as an MQTT message, and transmits it to its intended Receiver.

3) How a Receiver listens for incoming Requests, and how it formulates and transmits a Response.

4) How the Mca and Mcc CRUD operations map to MQTT messages.

For more information on MQTT itself see clause A or refer to the MQTT specification [1].

6.2 Use of MQTT

MQTT includes reliability features which allow recovery from loss of network connectivity without requiring explicit involvement of the applications that are using it, however to do this it requires an underlying network protocol that provides ordered, lossless, bi-directional connections. The MQTT specification [1] does not mandate a particular underlying protocol, so this binding specification restricts the choice of underlying protocol: it shall be one of the following:

- Raw TCP/IP [4].
- TCP/IP with Transport Level Security (TLS) [5].
- WebSocket [6] - either with or without the use of TLS.

6.3 Connecting to MQTT

In order to communicate, the two client parties (AE and CSE or CSE and CSE) shall connect to a common MQTT server. The MQTT server shall be hosted in one of the two nodes or shall exist as an independent external entity, following one of the configuration patterns shown in clause 5.2.

Once each party has located the address of the MQTT server, it then connects to it using the standard MQTT CONNECT protocol packet. The following additional considerations apply:

- The CONNECT packet contains a Client Id as described in clause A.3.2. The Client Ids have to be unique at least among all clients that connect to a given MQTT server instance (this is a requirement imposed by the MQTT protocol). This condition will be satisfied if an AE uses its AE-ID and a CSE uses its CSE-ID. See clause 7 of oneM2M TS-0001 [2] for a discussion of these Identifiers. The prefix A:: or C:: shall be added to the ID to show whether it is an AE-ID or a CSE-ID as these ID spaces are not distinct.

  The AE-ID or CSE-ID may not be known during the initial registration process, in which case the client shall use some other appropriate unique ID.

- A client shall set the "Clean Session" flag in the CONNECT packet to false. This means that MQTT Session state related to that client will be retained by the MQTT Server in the event of a disconnection (deliberate or otherwise) of that client.

- A client shall not set the "Will Flag", so Will Messages are not enabled.

- A client may choose to provide a non-zero MQTT KeepAlive value or to provide a KeepAlive of 0 (this disables the MQTT KeepAlive).

- The MQTT server may require that a client provides a User Name and a password (or other credential). For more information see clause

7.

A client might choose to keep the MQTT connection open permanently (restarting it as soon as possible after any unforeseen connection loss), it might choose to connect only when it wants to act as an Originator, or it might choose to connect based on the <schedule> associated with a relevant oneM2M resource.

Once a client has connected to the MQTT server it can then communicate (subject to authorization policies) with any other client connected to its server. There is no need for it to create another connection if it wants to communicate with a different counter-party.

When a client determines that it no longer wishes to participate in an MQTT Session with its MQTT Server it shall perform the following steps:

- Disconnect from that server, if it is currently connected.
- Reconnect with the cleanSession flag set to true.
- Disconnect again.

These steps delete any state that the MQTT server might be holding on behalf of the client.

6.4 Sending and Receiving Messages

6.4.1 Request and Response Messages

An MQTT Control Packet consists of up to three parts: a fixed header, a variable header, a payload. The control packet format is depicted in Figure 6.4.1-1. MQTT does not have a data model to describe or constrain the content of its Application Message payloads (to that extent it is similar to a TCP socket). Mca and Mcc request messages shall be serialized into XML or JSON following the serialization process defined in clause 6.5.

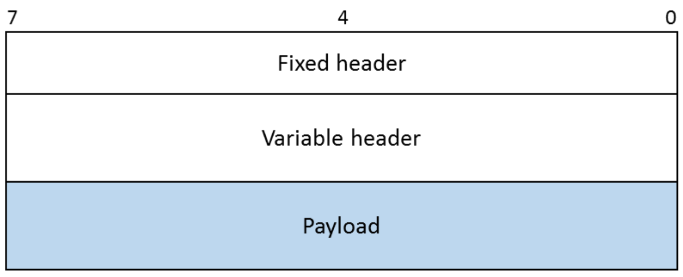

Figure 6.4.1-1: Message format of MQTT Control Packet

The fixed header includes information about the packet type, flags specific to packet type and the packet length. Figure 6.4.1-2 shows the fixed header's structure.

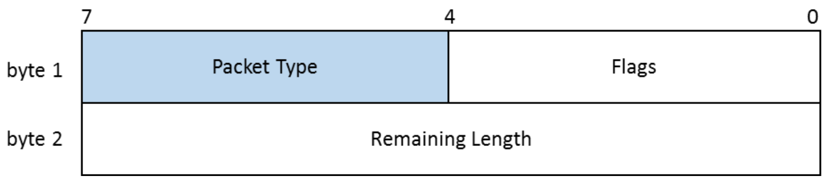

Figure 6.4.1-2: Fixed header structure

The packet type field in figure 6.4.1-2 is used to define the MQTT Control packet type. It is a 4-bit field which possible values are listed in the Table 6.4.1-1. When a oneM2M Request/Response message is bound to MQTT, its packet type shall have value 3, i.e. the Request/Response message is delivered in an MQTT PUBLISH packet.

Table 6.4.1-1: MQTT Control Packet Types

Reserved	0	Reserved for future use
CONNECT	1	Client request to connect to server
CONNACK	2	Connect acknowledgement
PUBLISH	3	Publish message
PUBACK	4	Publish message acknowledgement
PUBREC	5	Publish received (QoS=2)
PUBREL	6	Publish release (QoS=2)
PUBCOMP	7	Publish complete (QoS=2)
SUBSCRIBE	8	Client subscribe request
SUBACK	9	Subscribe acknowledgement
UNSUBSCRIBE	10	Client unsubscribe request
UNSUBACK	11	Unsubscribe acknowledgement
PINGREQ	12	Ping request
PINGRESP	13	Ping response
DISCONNECT	14	Client disconnection request
Reserved	15	Reserved for future use

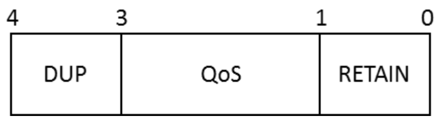

Figure 6.4.1-3: Flags for the PUBLISH packet

Figure 6.4.1-3 shows the flags specific to MQTT PUBLISH packet. The QoS field represents the QoS level of the PUBLISH packet with possible values of 1, 2 or 3. However, since oneM2M messages are idempotent, they should be sent with MQTT QoS 1.

NOTE: MQTT packets are subjected to a theoretical maximum message size of 256MB, but it is good practice not to send packets that are bigger than a 100kB. If a larger amount of data needs to be sent, it should be segmented into multiple PUBLISH packets.

6.4.2 Sending a Request

A request is transmitted by sending it as an MQTT PUBLISH protocol packet to the MQTT Server. It uses a Topic that identifies both the Originator and the Receiver of the request as follows:

- /oneM2M/req/<originator>/<receiver>
  - "oneM2M" is a literal string identifying the topic as being used by oneM2M.
  - <originator> is the SP-relative-AE-ID or SP-relative-CSE-ID of the entity that sends the request on this hop, omitting any leading "/" . In the AE-ID case, any "/" characters embedded in the ID shall be replaced with ":" characters.
  - <receiver> is the SP-relative-AE-ID or SP-relative-CSE-ID of the Receiver (AE, Transit CSE or Hosting CSE) on this hop, omitting any leading "/" . In the AE-ID case, any "/" characters embedded in the ID shall be replaced with ":" characters.
  - "req" is a literal string identifying this as a request.

The payload of the MQTT PUBLISH packet is used to carry the request primitive, as described in clause 6.5.

6.4.3 Listening for and responding to a Request

A Receiver listens for requests arriving via MQTT by subscribing to its Topics using a wildcarded topic filter of the following form

- /oneM2M/req/+/<receiver>
  - “oneM2M” is a literal string identifying the topic as being used by oneM2M

- + is a wildcard which matches any entity
- <receiver> is the SP-relative-AE-ID or SP-relative-CSE-ID of the Receiver (AE, Transit CSE or Hosting CSE), omitting any leading "/" . In the AE-ID case, any "/" characters embedded in the ID shall be replaced with ":" characters.
- “req” is a literal string identifying this as a request.

When it receives a request, the Receiver shall perform the Core Transport operations associated with the request, including any access control policy checks. In particular it shall check the request expiration timestamp (if any) contained in the request, since it is possible that that time might have passed while the message was being stored by MQTT.

It transmits a response by sending an MQTT PUBLISH packet to a response topic. This takes the form:

- /oneM2M/resp/<originator>/<receiver>
    - "oneM2M" is a literal string identifying the topic as being used by oneM2M.
    - <receiver> is the SP-relative-AE-ID or SP-relative-CSE-ID of the Receiver (AE, Transit CSE or Hosting CSE), omitting any leading "/" . In the AE-ID case, any "/" characters embedded in the ID shall be replaced with ":" characters.
    - <originator> is the SP-relative-AE-ID or SP-relative-CSE-ID of the entity that sent the corresponding request, omitting any leading "/" . In the AE-ID case, any "/" characters embedded in the ID shall be replaced with ":" characters.
    - "resp" is a literal string identifying this as a response.

The Originator shall subscribe to this Topic (either explicitly or using a wildcarded filter) in order to see the response.

The payload of the MQTT PUBLISH packet is used to carry the response primitive, as described in clause 6.5.

6.4.4 Initial Registration

In some security scenarios, an Originator might not initially know its AE-ID or CSE-ID. Initial registration exchanges can use the communication pattern described in clauses 6.4.1 and 6.4.2 except that they use Topics containg a credential ID rather than an AE-ID or CSE-ID, as follows:

- /oneM2M/reg_req/<originator>/<receiver>
    - “oneM2M” is a literal string identifying the topic as being used by oneM2M.
    - <originator> is the Credential-ID. Any "/" characters embedded in the ID shall be replaced with ":" characters.
    - <receiver> is the SP-relative-CSE-ID of the Receiver (Transit or Hosting CSE) specified in the corresponding request, omitting any leading "/" .
    - “reg_req” is a literal string identifying it as a registration request.

and

- /oneM2M/reg_resp/<originator>/<receiver>
    - “oneM2M” is a literal string identifying the topic as being used by oneM2M.
    - <originator> is the Credential-ID of the Originator in the corresponding request. Any "/" characters embedded in the ID shall be replaced with ":" characters.
    - <receiver> is the SP-relative-CSE-ID of the Receiver (Transit or Hosting CSE) in the corresponding request, omitting any leading "/" .
    - “reg_resp” is a literal string identifying it as a registration response.

6.4.5 Request/Response Message Flow

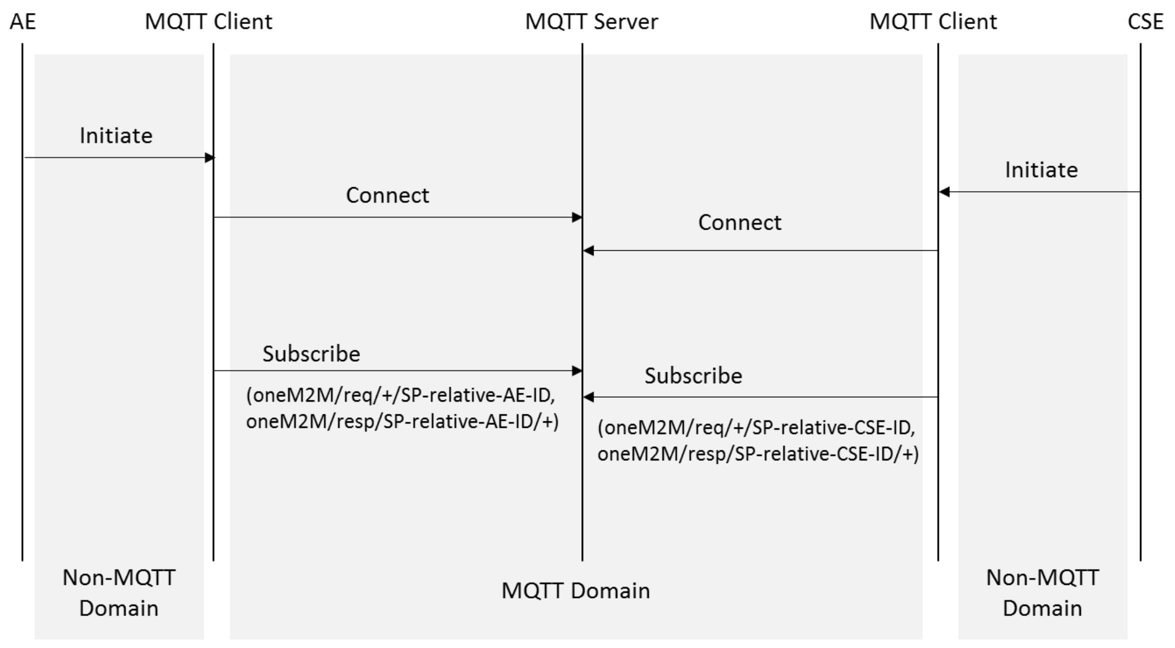
AE
MQTT Client
MQTT Server
MQTT Client
CSE
Initiate
Connect
Initiate
Connect
Subscribe
(oneM2M/req/+/SP-relative-AE-ID, oneM2M/resp/SP-relative-AE-ID/+)
Subscribe
(oneM2M/req/+/SP-relative-CSE-ID, oneM2M/resp/SP-relative-CSE-ID/+)
Non-MQTT Domain
MQTT Domain
Non-MQTT Domain

Figure 6.4.5-1: Initiating Process in MQTT binding

In the MQTT protocol, each client shall subscribe to the MQTT server to receive messages. As shown in figure 6.4.5.1-1, the AE or CSE initiates the MQTT binding process by trying to connect to the MQTT server, as described in clause 6.3.

After each MQTT client successfully connects to the server, it shall subscribe to the MQTT server. The topic names which each MQTT client subscribes to are "/oneM2M/req/+/<receiver>" (to receive requests) and "/oneM2M/resp/<originator>/+" (to receive replies) where <receiver> and <originator> are both set equal to the ID (SP-relative-AE-ID or SP-relative-CSE-ID as appropriate).

Accordingly the plus sign ('+' U+002B) wildcard and the SP-relative-AE-ID or SP-relative-CSE-ID are used in the topic name within the MQTT SUBSCRIBE packet. This enables the MQTT client to receive the PUBLISH packets whose target it is. Therefore, through this process, the AE or CSE receives messages if the request/response messages are published to oneM2M/req/<originator>/<receiver> or oneM2M/resp/<originator>/<receiver>.

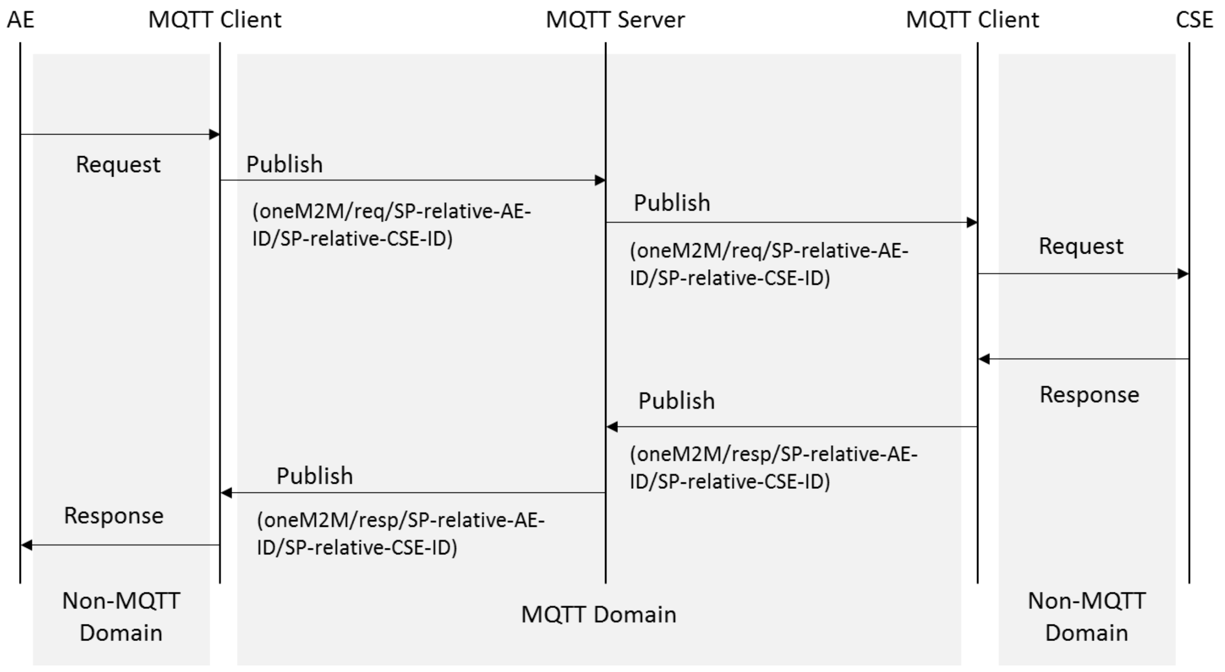
AE
MQTT Client
MQTT Server
MQTT Client
CSE
Request
Publish
(oneM2M/req/SP-relative-AE-ID/SP-relative-CSE-ID)
Publish
(oneM2M/req/SP-relative-AE-ID/SP-relative-CSE-ID)
Request
Response
Publish
(oneM2M/resp/SP-relative-AE-ID/SP-relative-CSE-ID)
Publish
(oneM2M/resp/SP-relative-AE-ID/SP-relative-CSE-ID)
Response
Non-MQTT Domain
MQTT Domain
Non-MQTT Domain

Figure 6.4.5-2: Request/Response message delivery over MQTT

As an example, figure 6.4.5-2 illustrates the request/response message delivery over MQTT protocol between AE and CSE via Mca reference point in oneM2M. The message flow is as follows.

In this scenario, the AE wants to send a Request message to the registered CSE. After that, the AE's MQTT client library sends an MQTT PUBLISH packet to the MQTT server with "oneM2M/req/ SP-relative-AE-ID/SP-relative-CSE-ID" as the topic name. The MQTT PUBLISH packet can include {"op", "fr", "to", "ri", "pc", "ty"} and any other optional message parameters described in clause 8 of [3] in its payload.

After the MQTT server receives the MQTT PUBLISH packet from the MQTT client, the server refers to the topic name and delivers the message to the intended MQTT client. Finally, the MQTT client library delivers the message to the CSE.

Regarding the Response message, the MQTT message is sent using "oneM2M/resp/<originator>/<receiver>" as its topic. In order to send a Response message to the requestor, the CSE can use the "fr" information from the received request message or the topic name of the MQTT PUBLISH packet related with the received request message. After that, the CSE and sends an MQTT PUBLISH packet to the MQTT server with "oneM2M/resp/ SP-relative-AE-ID/SP-relative-CSE-ID" as the topic name in this scenario. The MQTT PUBLISH packet can include {"rsc", "fr", "to", "ri", "pc"} and any other optional message parameters described in clause 8 of [3] in its payload.

Similar to the request message delivery from AE to CSE, the MQTT server delivers the published packet to the intended MQTT client by referring to the topic name. Finally, AE receives the response message from the MQTT client.

6.5 Primitive Mapping

6.5.1 Request primitives

A oneM2M request primitive is made up of a number of control parameters and (optionally) a content part. All the parameters in these parts are serialized into the payload of an MQTT Publish Packet, using the rules given in clause 8 of [3] applied to m2m:requestPrimitive defined in clause 6.4.1of [3].

All the parameters that are present in the primitive shall be serialized, in particular the request shall contain the mandatory parameters such as ***Operation, To, From, Request Identifier*** as specified in clause 8.1.2 of [2] and 7.1.1.1 of [3].

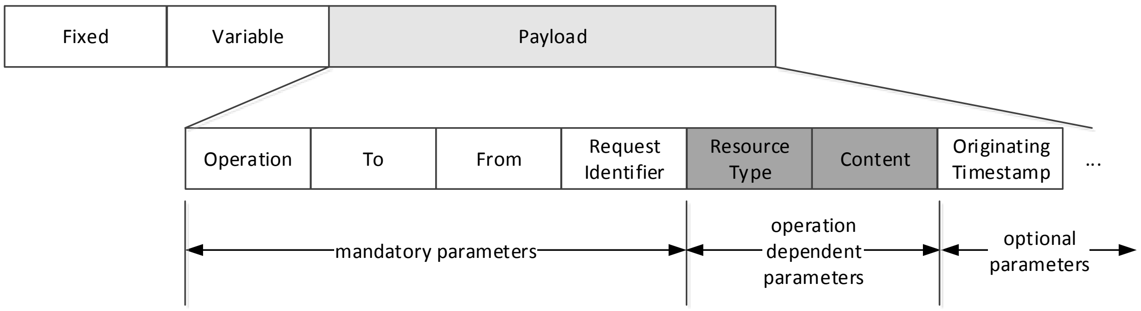

Figure 6.5.1-1: MQTT Request example

An example of an MQTT Request message serialized using JSON is:

```
{"op": 1, "to": "//xxxxx/2345", "fr": "//xxxxx/99", "rqi": "A1234", "ty": 18, "pc":
{"m2m:sch":{"rn":"schedule1", "se": "* 0-5 2,6,10 * * * *"}}, "ot": 20150910T062032}
```

- op: short name of ***Operation*** parameter specified as m2m:operation in [3].
- to: short name of ***To*** parameter specified either xs:anyURI [3] or m2m:nhURI [3]. It is an URI of the target resource.
- fr: short name of ***From*** parameter which is an ID of the Originator e.g. either the AE or CSE
- rqi: short name of ***Request Identifier*** specified as m2m:requestID [3].
- ty: short name of ***Resource Type*** parameter specified as m2m:resourceType [3].
- pc: short name of ***Content*** parameter specified in [3].
- ot: short name of ***Originating Timestamp*** parameter specified as m2m:timestamp [3].

6.5.2 Response primitives

A oneM2M response primitive is serialized using the rules given in clause 8 of [3] applied to m2m:responsePrimitive defined in clause 6.4.2 [3].

In particular, each response primitive shall include the ***Response Status Code*** parameter to indicate success or failure of the operation and the ***Request Identifier*** parameter.

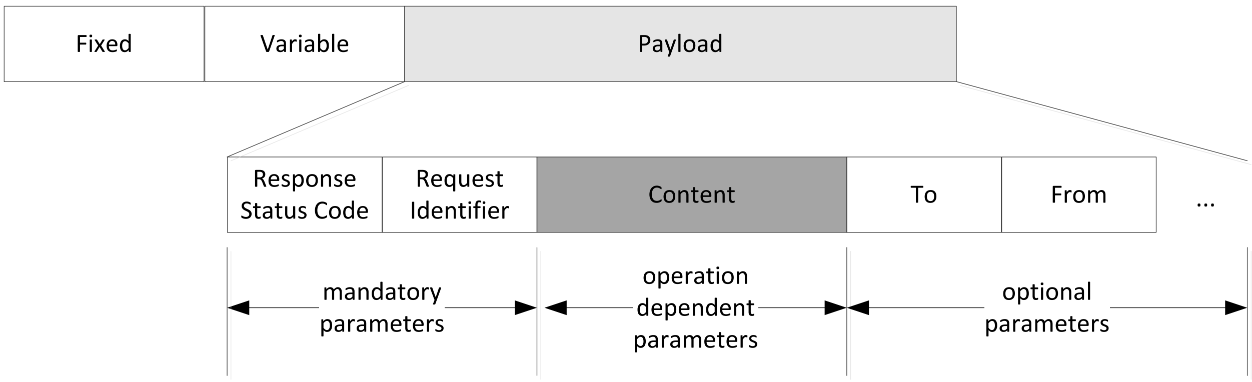

Figure 6.5.2-1: MQTT Response example

An example of an MQTT Response message serialized using JSON is:

```
{"rsc": 2000, "rqi": "A1234", "pc": {"m2m:sch":{"se": "* 0-5 2,6,10 * * * *"}}, "to": "//xxxxx/2345", "fr":
"//xxxxx/99"}
```

- rsc: short name of ***Response Status Code*** parameter specified as m2m:responseStatusCode in [3].

- rqi: short name of ***Request Identifier*** specified as m2m:requestIDin [3].
- pc: short name of ***Content*** parameter specified in [3].
- to: short name of ***To*** parameter specified either xs:anyURI [3] or m2m:nhURI [3]. It is an URI of the target resource.
- fr: short name of ***From*** parameter which is an ID of the Originator e.g. either the AE or CSE.

6.5.3 Serialization Format Negotiation

The request primitive can contain an additional parameter called ***xM2M-Accept*** which indicates the serialization format that the Originator is prepared to accept in a Response.

If the Receiver supports the kind of serialization specified in the request's ***xM2M-Accept*** parameter, the Receiver shall respond using that serialization. If the Receiver does not support the serialization specified in the requests's ***xM2M-Accept*** parameter, "Not Acceptable" shall be sent as the ***Response Status Code*** unless another error code takes precedence for this response.

If the request does not contain the ***xM2M-Accept*** parameter, the Receiver is free to choose to use any oneM2M-supported serialization, since the requestor has not indicated a preference.

Possible values for the ***xM2M-Accept*** parameter are listed below:

- application/xml
- application/json

6.6 Format used in pointOfAccess strings

oneM2M defines an MQTT URI format to be used in the *pointOfAccess* attributes in several entity resource types (e.g., <CSEBase>, <remoteCSE>, <AE>). A *pointOfAccess* attribute contains a list of one or more strings, each of which indicates a way in which that entity can be addressed. An entity can indicate support for MQTT by including strings in either or both of the following forms:

- mqtt://<authority>
- mqtts://<authority>

The <authority> component is defined in clause 3.2 of [8] and gives the host and port of the MQTT Server that is to be used to access the entity in question.

The form with scheme mqtts: shall be used to show that the server requires the use of TLS when this particular point of access is being used.

If the <authority> does not contain a port component, then the IANA-registered MQTT ports shall be assumed. These are 1883 in the case of mqtt: and 8883 in the case of mqtts:

7 Security

7.1 Introduction

The MQTT servers authenticate the clients (both CSEs and AEs) that connect to them and authorize access to Topics used for communicate. The clients do not authenticate each other, instead they use the MQTT server to do this.

Background. The MQTT binding makes use of one or more MQTT Servers to transport messages between AEs and CSEs (or between CSEs) as described in Clause 5. The AE/CSEs both act as MQTT Clients of an MQTT Server that mediates delivery of messages between the two. As described in clause 6, the topic in the MQTT PUBLISH packet either indicates the Originator's identity and Receiver's identity, or includes the Originator's Credential and the Receiver's Identity (in the case that the Originator has not yet been assigned an identifier).

Trust and the MQTT Server. When the oneM2M binding to MQTT is used, some security functions are performed by the MQTT Server, as described further in this clause. In particular:

1) The MQTT Server authenticates the AEs and CSEs as they connect as MQTT Clients. These MQTT Clients themselves never directly authenticate the CSE or AE that is using another MQTT Client – instead they trust the MQTT Server to authenticate the MQTT Clients, and trust the MQTT Server to route the messages between the Originator and Receiver indicated in the topic.

2) The MQTT Server enforces access control policies to ensure that unidentified or unauthorized clients are not able to publish messages to oneM2M topics or subscribe to receive messages from them.

7.2 Authorization

There are two levels of authorization in the oneM2M binding to MQTT.

- *The MQTT Server is responsible for verifying identifiers, for routing messages to the expected CSE or AE, and providing the correct Credential-ID during initial registration*

  The MQTT Server is responsible for verifying that the Client Identifier field in a MQTT CONNECT packet matches the expected AE-ID, CSE-ID or Credential-ID.

  The MQTT Server is responsible for controlling those topics to which an MQTT Client may subscribe and receive published MQTT packets, and those topics to which an MQTT Client may publish MQTT packets.

  Since the topic includes the Receiver's CSE-ID, the Originator can trust that the MQTT packets are routed to and from the expected Receiver. If the topic includes the Originator's CSE-ID or AE-ID, then the Receiver can trust that the MQTT packets are routed to and from the expected Originator.

  If the topic includes the Originator's Credential-ID (which should only occur at initial registration), then the Receiver can use this Credential-ID to determine the CSE-ID or AE-ID or list of allowed AE-ID(s) which are to be used in assigning a CSE-ID or AE-ID to the Originator (as described in TS-0003 [7]). This Credential-ID can be trusted to have been verified by the MQTT Server.

- *The Receiver is responsible for authorizing requests to specific resources, and assigning CSE-ID or AE-ID during initial registration.*

  When the MQTT topic includes the Originator's CSE-ID or AE-ID, then the Receiver is responsible for making access control decisions on requests to perform operations on specific resources hosted on the Receiver. The access control decisions are dictated by the applicable accessControlPolicy resources and the Originator's CSE-ID or AE-ID (and other factors not relevant to the present discussion). This authorization process is as defined in the Architecture Specification [2], the Core Protocol Specification [3] and Security Solutions specification [7].

  When the MQTT topic includes the Originator's Credential-ID (which should only occur at initial registration), then the Receiver is responsible for assigning a CSE-ID or AE-ID to the Originator (which may be dependent on the Originator's Credential-ID).

7.3 Authentication

An MQTT Client and MQTT Server shall apply Transport Layer Security (TLS) using any of the Security Association Establishment Frameworks in TS-0003 [7].

The Security Association Establishment Frameworks provide mutual authentication of the MQTT Client and MQTT Server. The Security Association Establishment Frameworks are described using two main entities Entity A and Entity B: in the case of the oneM2M binding to MQTT, Entity A is a CSE or AE using an MQTT Client, and Entity B is an MQTT Server.

NOTE: In TS-0003 [7], Entity A is described as establishing the CSE-ID of Entity B as a result of Security Association Establishment. The application to MQTT differs because Entity A establishes the identity of the MQTT Server instead. However, the procedures are still applicable.

The Remote Security Provision Frameworks in TS-0003 [7] may be applied to provision a symmetric key shared by a CSE/AE using an MQTT Client and an MQTT Server, with the MQTT Server assuming the role of the Enrolment Target.

Identification of Originator and Receiver. TS-0003 [7] describes a variety of approaches by which successful Security Association Establishment results in Entity B determining the CSE-ID or AE-ID or list of allowed AE-ID(s) for the CSE/AE using the Entity A. These approaches can also be used in the oneM2M binding to MQTT.

It is assumed that the MQTT Server is configured with the information necessary to determine the CSE-ID of the Receiver following successful Security Association Establishment with the Receiver's MQTT Client.

In some scenarios, the MQTT Server can be configured with appropriate information to verify the CSE-ID or AE-ID of the Originator. However, in cases where the Originator has not yet been assigned its CSE-ID or AE-ID, and the MQTT Server has also not been provided with the CSE-ID or AE-ID of the Originator, then the Receiver is responsible for determining the applicable CSE-ID or AE-ID. In these cases, the MQTT Server forms a Credential-ID, identifying the Credential used to authenticate the Originator, and includes this in the topic when forwarding to the initial registration request to the Receiver's MQTT Client. The Receiver extracts the Credential-ID, and the procedures in TS-0001 [3] and TS-0003 [7] determines the CSE-ID or AE-ID of the Originator.

Password Field Authentication. The present document does not specify use of the password field of the MQTT CONNECT control packet. Authentication is performed using the TLS mechanisms described in [7].

7.4 Authorization by the MQTT Server

This procedure describes how an MQTT Server authorizes topics to which an MQTT Client may subscribe. The M2M Service Provider is responsible for configuring the MQTT Server with the relevant information used in these authorization decisions.

1) In the case that the MQTT Server determines that the MQTT Client represents a CSE, and if the CSE has been assigned a CSE's SP-Relative-CSE-ID (which is denoted by <my-SP-Relative-CSE-ID>), then

   a) the MQTT Server authorizes the MQTT Client to subscribe to the topic /oneM2M/req/+/<my-SP-Relative-CSE-ID>, and

   b) for the set of <Registree ID Stem> values corresponding to SP-Relative CSE-ID or AE-ID Stem values of zero or more CSE(s) and/or AE(s) currently registered to this CSE (and known to the MQTT Server), the MQTT Server authorizes the MQTT Client to subscribe to the topics /oneM2M/resp/<Registree ID Stem>/<my-SP-Relative-CSE-ID>, and

   c) If this CSE is registered to a CSE and the SP-Relative-CSE-ID of this CSE is <Registrar-SP-Relative-CSE-ID > then the MQTT Server authorizes the MQTT Client to perform the following:

      i) To subscribe to

   /oneM2M/resp/<my-SP-Relative-CSE-ID>/<Registrar-SP-Relative-CSE-ID>, and

      ii) To publish to

   /oneM2M/resp/<my-SP-Relative-CSE-ID>/<Registrar-SP-Relative-CSE-ID>.

2) In the case that the MQTT Server determines that the MQTT Client represents an AE which has been assigned an S-Type AE-ID Stem equal to < AE-ID-Stem>, then

   a) If the MQTT Server determines that the AE is currently registered, and the AE's Registrar CSE has SP-Relative-CSE-ID equal to <Registrar-SP-Relative-CSE-ID> then the MQTT Server authorizes the MQTT Client to perform the following:

      i) To subscribe to /oneM2M/resp/<AE-ID-Stem>/<Registrar-SP-Relative-CSE-ID>,

      ii) To publish to /oneM2M/req/<AE-ID-Stem>/<Registrar-SP-Relative-CSE-ID>,

      iii) To subscribe to /oneM2M/req/<Registrar-SP-Relative-CSE-ID>/<AE-ID-Stem>, and

      iv) To publish to /oneM2M/resp/<Registrar-SP-Relative-CSE-ID>/<AE-ID-Stem>.

   b) Otherwise, the MQTT Server authorizes the MQTT Client to perform the following:

      i) To subscribe to /oneM2M/reg_resp/<AE-ID-Stem-Credential-ID>/+, and

      ii) To publish to /oneM2M/reg_req/<AE-ID-Stem-Credential-ID>/+

   where <AE-ID-Stem-Credential-ID> is generated from <AE-ID-Stem> as per TS-0003 [7].

3) In the case that the MQTT Server determines that the MQTT Client represents a AE with C-Type AE-ID-Stem equal to <AE-ID-Stem> (which implies that the AE is registered), and the AE's Registrar CSE has SP-Relative-CSE-ID equal to <Registrar-SP-Relative-CSE-ID> then the MQTT Server authorizes the MQTT Client to perform the following:

   a) To subscribe to /oneM2M/resp/<AE-ID-Stem>/<Registrar-SP-Relative-CSE-ID>,

   b) To publish to /oneM2M/req/<AE-ID-Stem>/<Registrar-SP-Relative-CSE-ID>,

   c) To subscribe to /oneM2M/req/<Registrar-SP-Relative-CSE-ID>/<AE-ID-Stem>, and

   d) To publish to /oneM2M/resp/<Registrar-SP-Relative-CSE-ID>/<AE-ID-Stem>.

4) In all other cases, the MQTT Server authorizes the MQTT Client to perform the following:

   a) To subscribe to /oneM2M/reg_resp/<Credential-ID>/+, and

   b) To publish to /oneM2M/reg_req/<Credential-ID>/+

   where <Credential-ID> is obtained from the Security Association Establishment procedure as described in TS-0003 [7].

7.5 General Considerations

Implementors should take note of the Security considerations listed in Chapter 5 of the MQTT Specification [1].

Annex A (informative): Overview of MQTT

A.0 Introduction

This annex provides some background information on MQTT that might by useful to a reader of the normative clauses of this TS. See reference [1] for the definitive source of information about the protocol itself.

A.1 MQTT features

MQTT is a light weight publish/subscribe messaging transport protocol, particularly well suited to event-oriented interactions. It was specifically designed for constrained environments such as those found in Machine to Machine (M2M) and Internet Of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

MQTT includes reliability features which allow recovery from loss of network connectivity without requiring explicit involvement of the applications that are using it, however it does require an underlying network protocol that provides ordered, lossless, bi-directional connections.

The features of MQTT include:

- The use of the publish/subscribe message pattern which provides one-to-many message distribution and decoupling of applications. This is described further in clause 5.1.1.
- Bidirectional communications. An entity can subscribe to receive messages without having a reliable IP address. This could be used to allow unsolicited requests to be sent to a Receiver, or an asynchronous response to be sent to an Originator, where the Originator or Receiver does not have an externally accessible IP address. It thus eliminates the need for long polling and can reduce the need for triggering.
- A messaging transport that is agnostic to the content of the payload. The message payload can be text or binary.
- A Session concept that can survive loss of network connectivity and can persist across multiple consecutive network connections. Messages can be stored and subsequently forwarded when connectivity is restored.
- Three levels of reliability (referred to as "qualities of service") for message delivery within a Session:
    - "At most once", where messages are delivered according to the best efforts of the operating environment. Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
    - "At least once", where messages are assured to arrive but duplicates might occur. This is best suited to messages which have idempotent semantics.
    - "Exactly once", where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.
- A small transport overhead and protocol exchanges designed to minimize network traffic, with consequent additional savings on battery power when compared to HTTP.
- A Retained Message option, allowing new subscribers to get the last message to have been published on a topic prior to their subscription.
- A mechanism to notify interested parties when an abnormal disconnection occurs.

A.2 MQTT implementations

Like HTTP, the MQTT protocol is asymmetric in that it distinguishes between two different roles: client and server.

In MQTT terms, a Client is a program or device that uses MQTT. It always establishes the Network Connection to the Server. A Client can

- Publish application messages that other Clients might be interested in.
- Subscribe to request application messages that it is interested in receiving.
- Unsubscribe to remove a request for application messages.
- Disconnect from the Server.

An MQTT Server is an entity that accepts connections from Clients. Unlike HTTP it generally does not run any application logic, instead an MQTT Server acts as an intermediary between Clients publishing application messages and the Clients which have subscribed to receive them.

The MQTT specification [1] recommends the use of IANA registered ports 1883 (MQTT over raw TCP/IP) and 8883 (MQTT running over TLS).

Although the MQTT protocol is relatively simple to implement, applications normally make use of pre-built implementations:

- The applications themselves link to libraries that provide the MQTT client functionality. Libraries are available for a variety of programming languages and operating environments.
- The MQTT server functionality can be provided by a standalone software process (possibly running on a server that is remote from the clients), a hardware appliance or a cloud-hosted MQTT service.

The Eclipse foundation through their M2M working group, provides open source MQTT client code via its Paho Project, and an open source server implementation via its Mosquitto project. Other open source and commercial implementations are also available.

A.3 MQTT Details

A.3.1 Addressing a message - Topics and Subscriptions

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to an MQTT server and subscribe to topics that they are interested in by sending an MQTT request protocol packet to the server. Clients also connect to the server and publish messages to the server, each message being associated with a topic. Many clients can subscribe to the same topics. The combination of the MQTT protocol and its server provides a simple, common interface for clients to connect to. A publisher can publish a message once and it be received by multiple subscribers.

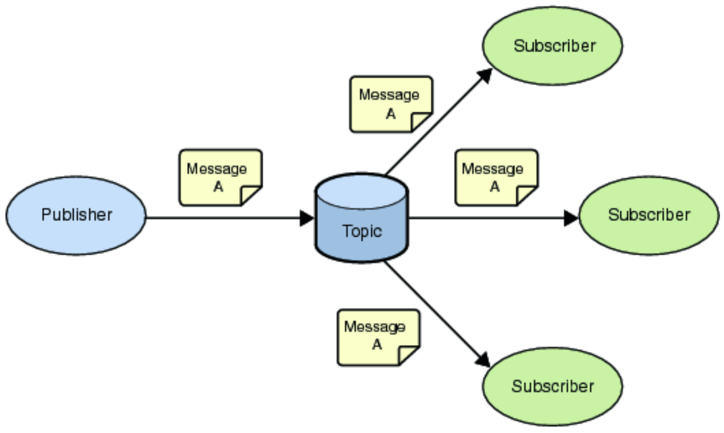

Figure A.3.1-1: MQTT publish-subscribe messaging

A Message in MQTT is associated with a topic when it is published. Topics are structured into topic trees, which are treated as hierarchies, using a forward slash (/) as a separator. This allows arrangement of common themes to be created. Topics and topic trees can be created administratively, although its more common for a server to create a topic on-demand (subject to security policies) when a client first attempts to publish or subscribe to it.

A client registers its interest in topics by providing one or more topics filters. A topic filter can be a simple topic name, or it can contain special "wildcard" characters, which allow clients to subscribe to multiple topics at once, within a single level or within multiple levels in a topic tree.

A.3.2 Reliability

MQTT defines three levels of Quality of Service (QoS). The QoS defines how hard the server & client will try to ensure that a message is received. Messages can be sent at any QoS level, and this affects the way the message is transmitted from the client to the server. When a client requests a subscription, it requests the maximum QoS at which it wants to receive messages on that subscription. This controls the way that messages matching that subscription are transmitted from the server to that client. The QoS used to transmit a message from the server is always less than or equal to the QoS used to transmit it to the server. For example, if a message is published at QoS 2 and a client is subscribed with QoS 0, the message will be delivered to that client with QoS 0. If a second client is also subscribed to the same topic, but with QoS 2, then it will receive the same message but with QoS 2. For a second example, if a client is subscribed with QoS 2 and a message is published on QoS 0, the client will receive it on QoS 0.

QoS 0 messages are the least reliable. They are sent from client to server (or server to client) with no acknowledgement flowing in the opposite direction. A server is free to discard such messages.

QoS 1 is intended for idempotent messages. These messages are transmitted with a short packet ID. When a client (or server) receives such a message it sends an acknowledgement packet back to the message sender. The sender is required to save a copy of that message until it receives the acknowledgement, and if there is a loss of network connectivity before it receives that acknowledgement it is required to resend the message when connectivity is restored.

QoS 2 provides exactly once delivery. It uses a two-step acknowledgement protocol, in which both steps can be repeated an arbitrary number of times (if there is a loss of connectivity) without causing duplication of the original application message. Both client and server are required to save a copy of the message during this process.

In summary, the higher levels of QoS are more reliable, but involve higher latency and have higher bandwidth requirements.

In order to be able to continue with the QoS1 or QoS2 delivery protocols after a network reconnection, the server needs to have a way of distinguishing the individual clients that connect to it. It does this by means of an identifier called a Client Id. A client provides this Id when it first connects and the server records it and uses it as a key to any server-side state (such as the status of incomplete message delivery) associated with that client. When the the client reconnects it presents the same Id, and that allows message delivery to complete. The client Id in effect represents the Id of the MQTT Session that is maintained between the client and the server.

A.3.3 Retained Messages

When a client publishes a message it can request that the message be retained. This means that the server will keep the message even after sending it to all current subscribers. If a new subscription is made that matches the topic of the retained message, then the message will be sent to the client. At most one such message is retained for any single topic. This is useful as a "last known good" mechanism. If a topic is only updated infrequently (such as for "report by exception"), then without a retained message, a newly subscribed client might have to wait a long time to receive an update. With a retained message, the client will receive an instant update.

History

Publication history		
V1.0.1	30-Jan-2015	Release 1 - Publication
V1.5.1	29-Feb-2016	Updated release 1 - Publication