ESR Consortium
LWM2M-MQTT-1.0

LWM2M over MQTT
Profile Specification

ESR030

Reference: ESR-SPE-030-LWM2M-MQTT
Version: 1.0
Rev: ADraft4

Contents

Tables

Illustrations

1 PREFACE TO LWM2M-MQTT 1.0 PROFILE, ESR030

This document defines the, *LWM2M-MQTT 1.0* profile.

1.1 Who Should Use this Specification

This specification targets the following audiences:

- Server architects who want to build a server implementation that complies to the LWM2M-MQTT profile specification;
- Device architects who want to build an embedded client implementation that complies to the LWM2M-MQTT profile specification;

1.2 How This Specification is Organized

This specification is organized as follow:

- **Introduction** is a short chapter explaining what is LWM2M-MQTT, why it has been designed and what are its main assets.
- **Specification** is the chapter explaining in details the LWM2M-MQTT concepts and semantic.
- **Appendixes** is a chapter with additional content that helps implementors understanding the LWM2M-MQTT specification.

1.3 Comments

Your comments about LWM2M-MQTT are welcome. Please send them by electronic mail to the following address: `comments @e-s-r.net`, with LWM2M-MQTT in your subject line.

1.4 Glossary

- *ESR*: Embedded Specification Request
- *MQTT*: Message Queuing Telemetry Transport
- *LWM2M*: Lightweight Machine to Machine
- *CoAP*: Contrained Application Protocol

1.5 Document Conventions

In this document, references to pieces of code, standardized tokens, error codes … are written using the default monospace font (e.g. `PUT)`.

1.6 Implementation Notes

The LWM2M-MQTT specification does not include any implementation details. LWM2M-MQTT implementors are free to use whatever techniques they deem appropriate to implement the specification.

2 DOCUMENTS AND DEFINITIONS

2.1 References

[CoAP]: Shelby, Z., Hartke, K., Bormann, C., and B. Frank, “Constrained Application Protocol (CoAP)”, https://tools.ietf.org/html/rfc7252, June 2014.

[LWM2M]: Open Mobile Alliance, “Lightweight Machine to Machine Technical Specification”, Candidate Version 1.0, 07 April 2016

[MQTT]: Oasis, “MQTT Version 3.1.1 Plus Errata 01”, OASIS Standard Incorporating Approved Errata 01, 10 December 2015

[BLOCK]: Bormann, C. and Shelby, Z, "Blockwise transfers in CoAP", Work in Progress, October 2013.

[CORE]: Shelby, Z, “Constrained RESTful Environments (CoRE) Link Format”, https://tools.ietf.org/html/rfc6690, August 2012

2.2 Definitions

2.3 [MQTT] Abstract

MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication 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.

The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections. Its features include:

- *Use of the publish/subscribe message pattern which provides one-to-many message distribution and decoupling of applications.*
- *A messaging transport that is agnostic to the content of the payload.*
- *Three qualities of service for message delivery:*
  - *"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 can occur.*
  - *"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 minimized to reduce network traffic.*
- *A mechanism to notify interested parties when an abnormal disconnection occurs.*

2.4 [LWM2M] Introduction

This enabler defines the application layer communication protocol between a LWM2M Server and a LWM2M Client, which is located in a LWM2M Device. The OMA Lightweight M2M enabler includes device management and service enablement for LWM2M Devices. The target LWM2M Devices for this enabler are mainly resource constrained devices. Therefore, this enabler makes use of a light and compact protocol as well as an efficient resource data model.

A Client-Server architecture is introduced for the LWM2M Enabler, where the LWM2M Device acts as a LWM2M Client and the M2M service, platform or application acts as the LWM2M Server. The LWM2M Enabler has two components, LWM2M Server and LWM2M Client. Four interfaces are designed between these two components as shown below:

- *Bootstrap*
- *Client Registration*
- *Device management and service enablement*
- *Information Reporting*

[...] The LWM2M Enabler uses the Constrained Application Protocol (CoAP) with UDP and SMS bindings. Datagram Transport Layer Security (DTLS) provides security for UDP transport layer.

3 INTRODUCTION

The LWM2M protocol is becoming the standard for remotely managing a device through a set of normalized operations. However, although its semantics can be implemented on any kind of protocols, the LWM2M specification is written in a way that it is tightly coupled to the CoAP protocol, compliant with a lossy network such as UDP or SMS transport.

On an other hand, most of the IoT devices are connected to a central server using a regular TCP connection, for two main reasons:

- This is no more an issue from server side to keep billions of TCP connections up at a time.
- Although the device is viewed as a server that exposes resources upon which requests are initiated from the cloud, the connection is initiated by the device, which removes trying to resolve NAT traversal issues.

There are many initiatives for proposing an implementation of CoAP over TCP, thus having LWM2M implementations already running on a TCP transport without any effort.

The MQTT broker is one of the most commonly used central server, especially because its publish/subscribe paradigm eases sending data to the cloud.

The goal of this specification is to propose an alternative composition of LWM2M semantics with CoAP message format based on MQTT publish/subscribe paradigm on top of a TCP transport. The key points driving choices made in this specification are:

- Keeping a single open link from device to the cloud for data and device management
- Minimizing embedded client footprint for targeting resource-constrained devices, such as those running on microcontrollers or small embedded microprocessors. Especially trying to avoid as much as possible mechanisms specified by CoAP to manage unordered or loss packets (binary data streaming instead of unordered blockwise transfer, de-duplication, sending again after timeouts...), as it is based on a TCP transport that implies no packets loss
- Relying as much as possible on the original specifications

This specification assumes the reader refers to *[LWM2M]*, *[CoAP]*, *[MQTT]* specifications and only describes differences and modifications (subset / updates) compared to the original specifications.

4 SPECIFICATION

This specification defines the use of CoAP messages transferred as payload of general-purpose MQTT topics. MQTT is just viewed as a high-level transport layer for CoAP messages.

A LWM2M Client and a LWM2M server are connected through two general-purpose MQTT topics named *transport topics* that serve for bi-directional communications. Each pair of transport topics connects a single LWM2M Client and a single LWM2M server. As specified by [LWM2M], a LWM2M client may target multiple LWM2M servers through the same MQTT broker connection using distinct transport topics pairs (see 4.1 for naming convention).

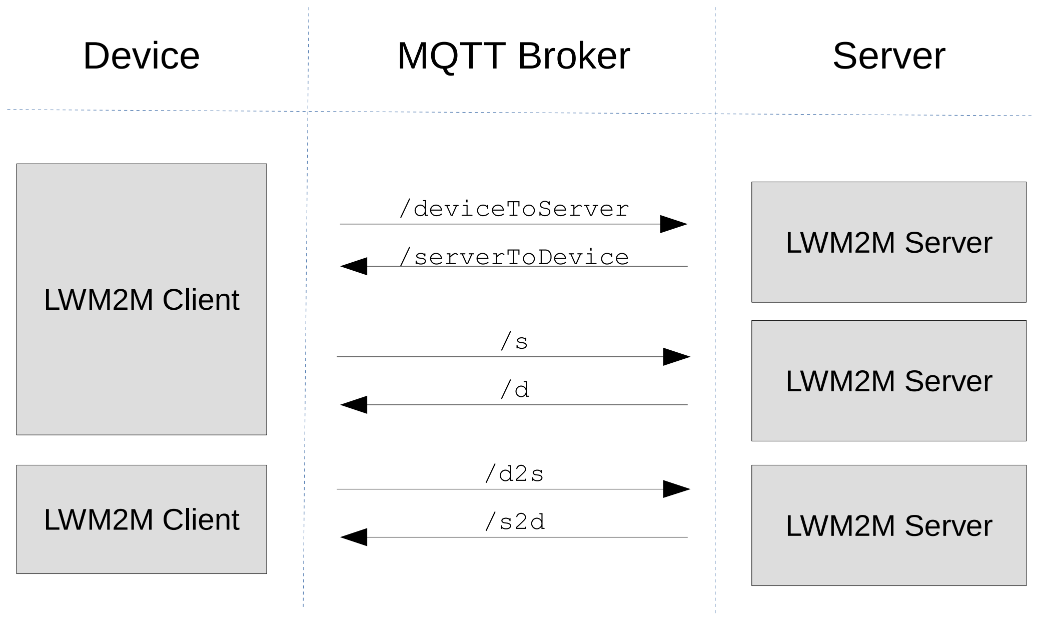

Illustration 4-1: LWM2M over MQTT Overview

4.1 Transport Topic Naming Convention

Transport topics names are structured as follows:

```
[PREFIX]/[DEVICE_ID]/[DIRECTION]
```

Where:

- `[PREFIX]` is an implementation-specific topic prefix (can be empty)
- `[DEVICE_ID]` is a unique device identifier under the chosen `[PREFIX]`
- `[DIRECTION]` is the topic name indicating the message direction. By default, the topic name for communications (i.e. publication) from server to device (resp. device to server) is `serverToDevice` (resp. `deviceToServer`). Topics names may be set in the Security Object during the bootstrap sequence (see 4.6).

Example of valid topics:

```
mymqttbroker/04ac-a451-ff43/deviceToServer
0/deviceToServer
mybroker/myaccount/0001/serverToDevice
```

4.2 Topic Payloads

MQTT packets published on transport topics are CoAP messages.

4.3 MQTT Settings

This specification assumes the LWM2M Client and the LWM2M server are connected to the MQTT broker. When the network connection of one of the two parts is broken the messages exchanged are lost. No session state is maintained across multiple network connections/disconnections.

4.3.1 Specification Version

The MQTT broker MUST implement MQTT version 3.1.1 (*[MQTT]*) or higher.

4.3.2 Quality Of Service

Publication of and subscription to transport topics MUST be done with MQTT quality of service level 1.

4.3.3 Clean Session

LWM2M Client and LWM2M server MUST connect the MQTT broker with the `CleanSession` flag set to 1.

4.3.4 Retain

All publications to transport topics MUST be done with the `RETAIN` flag set to 0.

4.4 Messages Sequences

The following sequence diagram shows the packet sequences transferred on transport topics illustrated with the *[LWM2M] Section 5.3 Client Registration Interface*.

Basically, the initiator (the device in this case) publishes a first CoAP request. Once received, the request is acknowledged by the receiver (the server in this case). Then, when the request has been processed, the server sends the response to the device (in this case the resource has been created from server side means that the device has been successfully registered).

Note that in order to receive device requests, the server first has to subscribe to each `deviceToServer` topic it may be connected to. In this example, this is done by subscribing to any device using a single level wildcard (+).

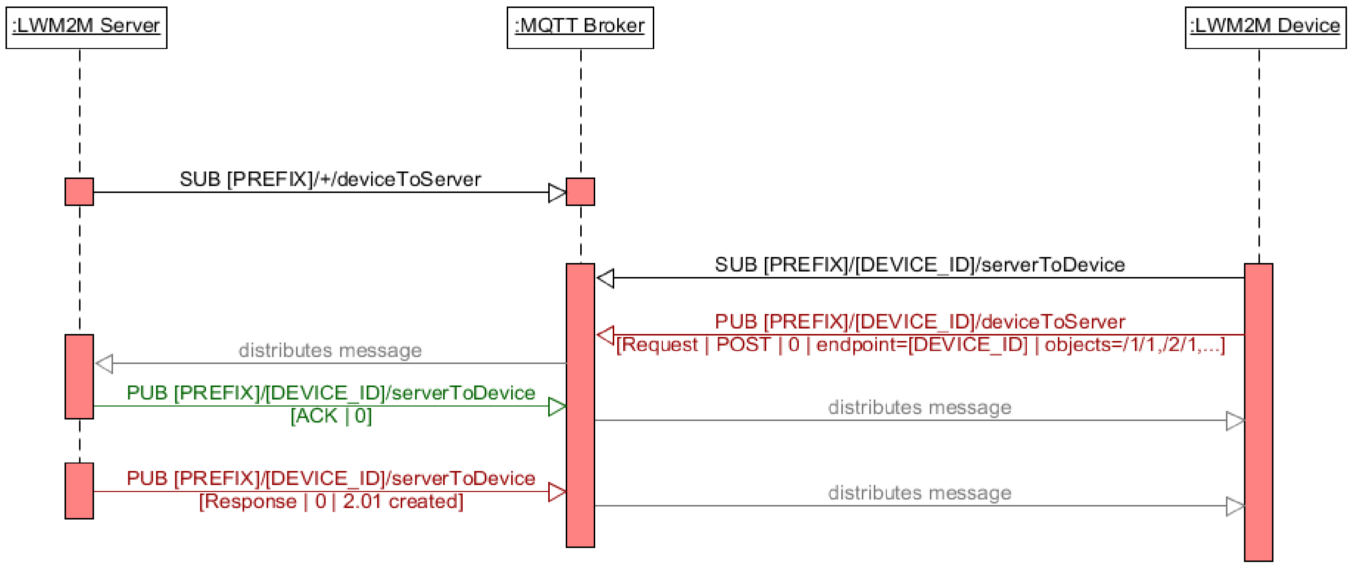
:LWM2M Server
:MQTT Broker
:LWM2M Device
SUB [PREFIX]/+/deviceToServer
SUB [PREFIX]/[DEVICE_ID]/serverToDevice
PUB [PREFIX]/[DEVICE_ID]/deviceToServer
[Request | POST | 0 | endpoint=[DEVICE_ID] | objects=/1/1,/2/1,...]
distributes message
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[ACK | 0]
distributes message
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[Response | 0 | 2.01 created]
distributes message

Illustration 4-2: Message Packets Sequence for LWM2M Client Registration Interface

4.5 Published Messages Rules

4.5.1 Request

A CoAP message request published to a transport topic MUST be marked as confirmable (`CON`), so that an acknowledgment message is expected to be received.

A CoAP message request with a message id `M` can be published to a transport topic if and only if all the following conditions are met:

- The MQTT acknowledgment of the previously published message (`PUBACK`) has been received
- The previously published CoAP message request has been acknowledged or timeout has been reached
- No request with the same message id `M` is pending, i.e. no CoAP response received and the request timeout not yet reached (see 4.5.4)

The previous rules ensure that at most one unacknowledged CoAP request is published to a transport topic at a time.

4.5.2 Response

A CoAP message response is marked as Non-confirmable message (`NON`).

4.5.3 Reliability

The [CoAP] section *4.2. Messages Transmitted Reliably* is overridden by this specification. When a CoAP request timeout is reached, the message is not retransmitted. The pending request is dropped and an error shall be returned to the request initiator.

The [CoAP] section *4.5. Message Deduplication* is overridden by this specification. According to the previous rules, MQTT QoS 1 behavior and MQTT Message Ordering Specification[1], MQTT packets must be dropped and ignored when receiving:

- A CoAP request with a message id equal to the latest received message id
- A CoAP response to an unknown request (i.e. a response to a request with a token that does not match any of the previously sent pending requests)

4.5.4 Transmission Parameters

This specification overrides the [COAP] transmission parameters section, by defining only 2 timeout constants.

A request is considered to be expired if no acknowledgment is received after `ACK_TIMEOUT` delay or if no response is received after `REQUEST_TIMEOUT` delay.

```
+-------------------+----------------+
| name              | default value  |
+-------------------+----------------+
| ACK_TIMEOUT       | 2 seconds      |
| REQUEST_TIMEOUT   | 15 seconds     |
+-------------------+----------------+
```

4.6 Bootstrap Security Object

When a device first connects a LWM2M Bootstrap Server, one ore more LWM2M Security Object is wrote on the device. Table 4-1 extends the Security Object defined in [LWM2M] (section E.1). When a resource ID already exists, it overrides the [LWM2M] definition.

1 As of version 3.1.1, *[MQTT]* clearly specifies the QoS 1 behavior for message ordering and duplicates in the case there is at most one unacknowledged message at a time on a topic. See section 4.6 Message Ordering: *If both Client and Server make sure that no more than one message is "in-flight" at any one time (by not sending a message until its predecessor has been acknowledged), then no QoS 1 message will be received after any later one - for example a subscriber might receive them in the order 1,2,3,3,4 but not 1,2,3,2,3,4. Setting an in-flight window of 1 also means that order will be preserved even if the publisher sends a sequence of messages with different QoS levels on the same topic*

ID	Name	Instances	Mandatory	Type	Range or Enumeration	Description
0	LWM2M Server URI bytes	Single	Mandatory	String	0-255	Uniquely identifies the LWM2M Server or LWM2M Bootstrap Server over MQTT, and is in the form of a MQTT URI: `tcp://host:port` or `ssl://host:port`, where host is an IP address or FQDN, and port is the TCP port of the Server.
100	Topic Prefix	Single	Optional	String	0-255	MQTT transport topics prefix (default value means there is no prefix). See section 4.1.
101	Server To Device Topic Name	Single	Optional	String	0-255	MQTT topic simple name for communication from server to device (default value is `serverToDevice` – see section 4.1).
102	Device To Server Topic Name	Single	Optional	String	0-255	MQTT topic simple name for communication from device to server (default value is `serverToDevice` – see section 4.1).

Table 4-1: LWM2M Security Object Extensions

4.7 Binary Data Transfer

A CoAP request `PUT` method with an Opaque type is used for binary data transfer. This specification defines a suitable protocol for large binary transfer in streaming mode.

4.7.1 Block Option

Although its goal is to replace the CoAP Blockwise Transfer layer specification (*[BLOCK]*), binary data transfer relies on `Block1` and `Block2` options (sections *2.1. The Block2 and Block1 Options* and *2.2. Structure of a Block Option* of *[BLOCK]*). The Block option is extended to a variable size of up to 4 bytes allowing to transfer binary chunks greater than 1KB:

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                   NUM                   |M|       SZX         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.7.2 Binary Transfer Protocol

The binary data is divided into packets, giving the receiver of the request a full control of the data received:

- It decides to send a request to get a new packet when it is ready
- It decides the size of the packet to be received

- It decides the order the packets are received, allowing to process the resource in a streaming mode

Illustration 4-3 describes the full packet sequence.

An initial request is sent (`PUT` opaque method) with the following options:

- `Size1`: indicating the total size in bytes of the data to be transferred
- `Location-Path`: a temporary URI identifying the resource to be transferred from the sender side.

This request is then acknowledged by the receiver. Then the receiver requests each data packet using a GET method with the following options:

- `Block2:` indicating the relative number 0 (`NUM`) and the desired size (`SZX`) of the packet.
- `URI-Path`: the temporary URI identifying the resource on the sender side.

Then the response is sent with the following options

- `Block1:` filled with the same block number, size and more blocks following flag (`M`) set to true.

This sequence is repeated until there is no more blocks to be retrieved from the receiver[2]. The last packet response sets the more blocks following flag (`M`) to false.

Finally, the receiver sends the response corresponding to the initial request (`PUT` opaque) to indicate the resource has been successfully modified.

2 The receiver is free to request the same packet multiple times, or to request packets with different sizes

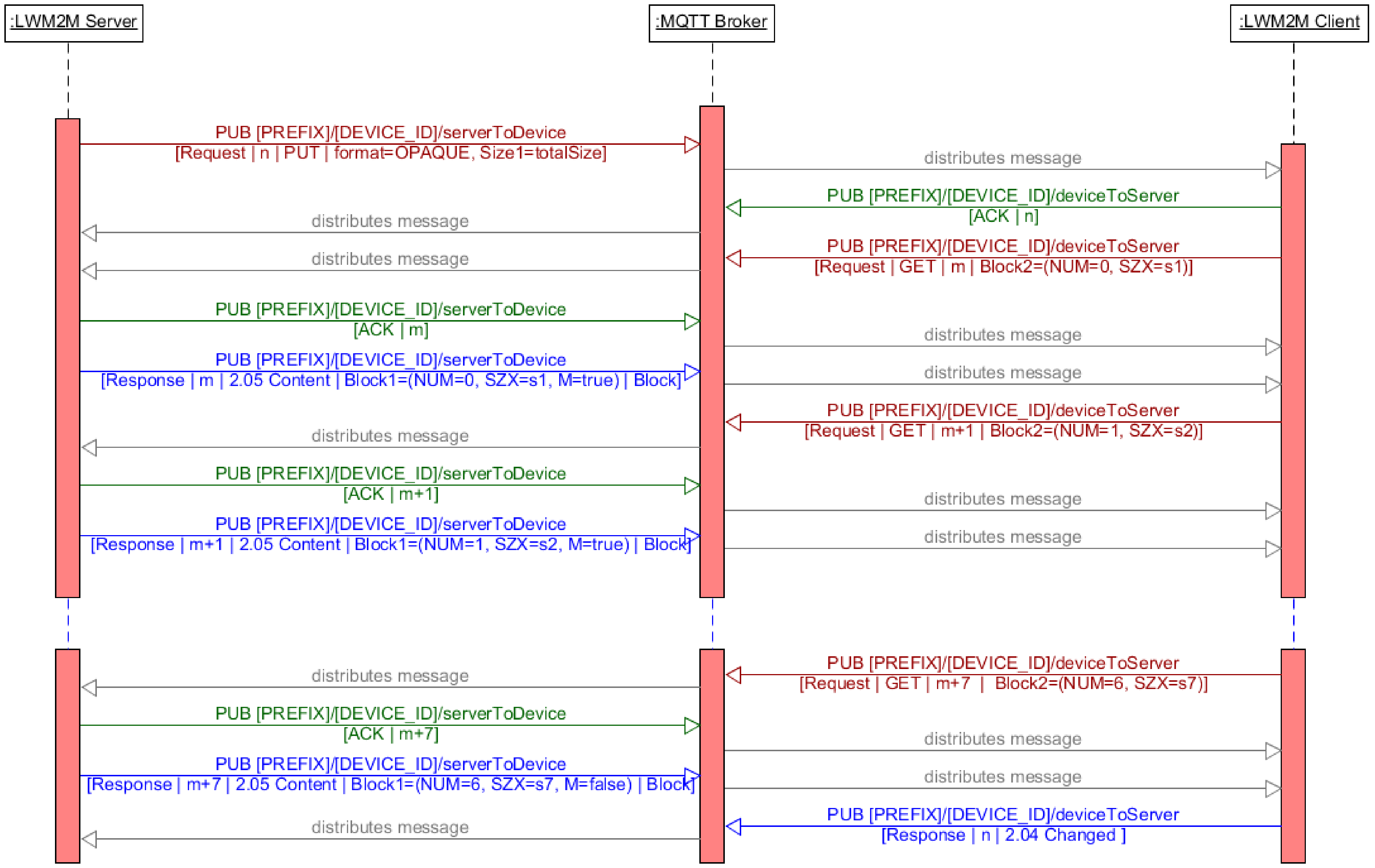
:LWM2M Server
:MQTT Broker
:LWM2M Client
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[Request | n | PUT | format=OPAQUE, Size1=totalSize]
distributes message
PUB [PREFIX]/[DEVICE_ID]/deviceToServer
[ACK | n]
distributes message
PUB [PREFIX]/[DEVICE_ID]/deviceToServer
[Request | GET | m | Block2=(NUM=0, SZX=s1)]
distributes message
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[ACK | m]
distributes message
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[Response | m | 2.05 Content | Block1=(NUM=0, SZX=s1, M=true) | Block]
distributes message
PUB [PREFIX]/[DEVICE_ID]/deviceToServer
[Request | GET | m+1 | Block2=(NUM=1, SZX=s2)]
distributes message
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[ACK | m+1]
distributes message
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[Response | m+1 | 2.05 Content | Block1=(NUM=1, SZX=s2, M=true) | Block]
distributes message
PUB [PREFIX]/[DEVICE_ID]/deviceToServer
[Request | GET | m+7 | Block2=(NUM=6, SZX=s7)]
distributes message
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[ACK | m+7]
distributes message
PUB [PREFIX]/[DEVICE_ID]/serverToDevice
[Response | m+7 | 2.05 Content | Block1=(NUM=6, SZX=s7, M=false) | Block]
distributes message
PUB [PREFIX]/[DEVICE_ID]/deviceToServer
[Response | n | 2.04 Changed]
distributes message

Illustration 4-3: Message Packets Sequence for Binary Data Transfer (PUT Opaque)

4.7.3 Specific Timeout

The `PUT` opaque request timeout specification overrides the default request timeout specified in section 4.5.4. A `PUT` opaque request is considered to be expired when the last `GET` packet request from the receiver has been sent over `REQUEST_TIMEOUT` delay.

When a `PUT` opaque request has expired, the temporary created resource MUST be invalidated by the sender, so that subsequent `GET` requests will be replied with a response code `4.04 Not Found`.

5 APPENDIXES

5.1 CoAP Specification Mapping

Table 5-1 lists the [CoAP] specification sections that are included by this specification without any modifications. When a section is in the list, its potential subsections are also included.

2.1. Messaging Model
2.2. Request/Response Model
3.1. Option Format
3.2. Option Value Formats
4.1. Messages and Endpoints
4.4. Message Correlation
5.1. Requests
5.2. Responses
5.2.2. Separate
5.3. Request/Response Matching
5.4. Options
5.5. Payloads and Representations
5.8. Method Definitions
5.10. Option Definitions
12. IANA Considerations

Table 5-1: CoAP Unmodified Sections

Table 5-2 lists the [CoAP] specification sections that are overridden by this specification.

4.2. Messages Transmitted Reliably	See section 4.5. This specification defines message sequence rules on MQTT topic, so that only the current packet may be received multiple times due to MQTT QoS1 behavior.
4.3. Messages Transmitted without Reliability	
4.5. Message Deduplication	
4.8. Transmission Parameters	See section 4.5.4.
5.2.3. Non-confirmable	See section 4.5. This specification enforces that requests are always marked confirmable and that responses are always marked Non Confirmable.

Table 5-2: CoAP Overridden Sections

Table 5-3 lists the [CoAP] specification sections that are not included by this specification.

2.3. Intermediaries and Caching
4.6. Message Size
4.7. Congestion Control
5.2.1. Piggybacked
8. Multicast CoAP

Table 5-3: CoAP Not Included Section

Table 5-4 lists the [CoAP] specification sections that are out of scope of this specification.

2.4. Resource Discovery
5.6. Caching
5.7. Proxying
6. CoAP URIs
7. Discovery
9. Securing CoAP
10. Cross-Protocol Proxying between CoAP and HTTP
11. Security Considerations

Table 5-4: CoAP Out of Scope Sections