LwM2M Over MQTT

Source code

The source code for this project is available in GitHub at this location.

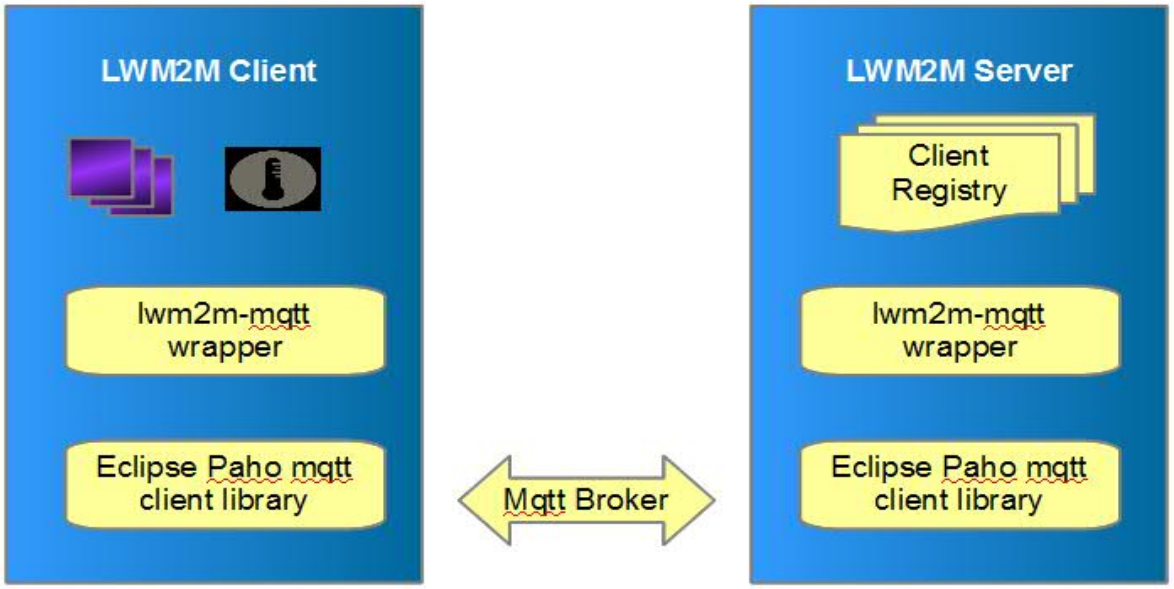
Introduction

OMA Lightweight M2M is a protocol for device and service management. The main purpose of this technology is to address service and management needs for constrained M2M devices, over a number of transports and bearers. The current stack for LWM2M relies on CoAP as the protocol. Along with CoAP, **MQTT is another standard which is widely used in Internet Of Things**. Our solution involves development of an LWM2M server prototype, as well as, a client prototype, which make use of MQTT as the underlying M2M protocol. Thus LWM2M can be used for both CoAP, as well as, MQTT.

In this project we used,

- Eclipse Paho MQTT Client - Open source client implementation for MQTT
- Eclipse Leshan - OMA LwM2M implementation in java
- Open Source Mosquitto Broker - This MQTT broker is hosted at eclipse
- Rasbperry Pi device with built-in CPU temperature - LwM2M client

and developed a MQTT wrapper that bridges the LwM2M and MQTT. The high-level architecture is outlined below,



As shown in the above diagram, both the client and server use the MQTT wrapper that talks to the mqtt broker and transfer the LwM2M operations over the network. The MQTT wrapper for client provides a tree based resource model which the client uses to build the list of Objects that it represents. Also, the resource model defines a basic structure to handle the incoming http verb calls like POST, PUT, GET and DELETE where the LwM2M requests are handled and responded. Along with the resource model, the MQTT wrapper defines a framework for exchanging the LwM2M request/response messages, defines a mechanism for observation & notification of messages for both client and server. On the server side, we removed the Californium implementation of COAP from the Leshan code and added the MQTT support.

The following section talks about the topic names that we have used to make MQTT work as request/response model,

Topic name formation

By nature MQTT is a publish/subscribe protocol and in order to make it work as a request/response model, the topic names must be designed such a way that it uniquely identifies the application that are connected to MQTT broker. We derived the topic names from eclipse kura MQTT namespace guidelines and it includes the following sections,

organization-name

Identifies a group of devices and users within the organization.

device-id

Identifies a single gateway device within the organization. The device_id maps to the Client Identifier (Client ID) as defined in the MQTT specifications or the endpoint name defined in the lwm2m enabler specification.

application-id

Identifies an application running on the lwm2m device uniquely.

LwM2M operations require a request/response message pattern to be established over MQTT. To initiate a solicited conversation, the client/server first sends a request message to a given application running on a specific device and then waits for a response.

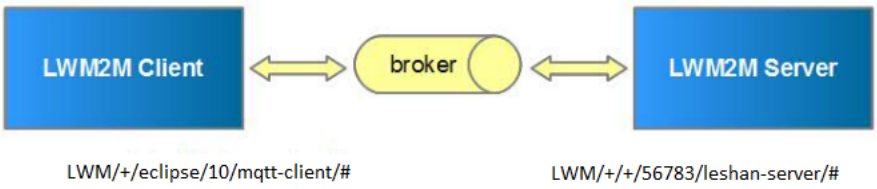
To ensure the delivery of request messages, both client and server must subscribe to the following topic on startup:

LWM/+/<organization-name>/<device-id>/<app-id>/#

where

LWM/S/.... : Topic to send the request
LWM/R/.... : Topic to send the response

following diagram shows an example of the topics used in the server and client,



REST Verbs

In the MQTT binding, we used the traditional & simple http methods to perform all the LwM2M operations like CoAP,

- POST
- GET
- PUT
- DELETE

These method names will be appended to the topic (shown below) when the client/server initiates the request,

LWM/+/<organization-name>/<device-id>/<app-id>/POST/#

In the following section we explain the message format that is used to accomplish the Client registration,

Client Registration

The Client Registration Interface is used by a LWM2M Client to register with one or more LWM2M Servers, maintain each registration and de-register from a LWM2M Server. As per the LwM2M spec, When registering, the LWM2M Client performs the “Register” operation and provides the properties the LWM2M Server requires to contact the LWM2M Client (e.g., End Point Name); maintain the registration and session (e.g., Lifetime, Queue Mode) between the LWM2M Client and LWM2M Server as well as knowledge of the Objects the LWM2M Client supports and existing Object Instances in the LWM2M Client.

Register command

Client connects to the MQTT Broker and initiates the registration request through the following events:

- Generates a conversation identifier known as a message-id (e.g., sequence number starting from 0)
- Similar to COAP, we use /rd resource for registration,

- Sends the registration request to the following topic along with below mentioned parameters,
  - **message-id** (identifier used to match a response with a request)
  - **requester.client.id** (device id of the client)
  - **requester.client.app-id** (application id of the client)
  - Appends the endpoint-name, lifetime and other parameters separated by &
  - List of Objects it supports

LWM/S/account_name/server_id/app_id/POST/rd

For example,

Topic: LWM/S/eclipse/56783/leshan-server/POST/rd
Message: [0 10 mqtt-client ep=10&lt=86400 </1/0>,</3/0>,</3303/0>]

where

0 - represents the message-id where the server must send the response

10 - the endpoint-id of the client

mqtt-client - client application id

ep=10&lt=86400 - Registration parameters as per the LwM2M specification

</1/0>,</3/0>,</3303/0> - List of objects exposed by the client

- Server responds to the following topic with below mentioned properties
  - response.code
  - Location-path (used for update and de-register call)

LWM/R/account_name/requester.client.id/requester.app_id/message-id

for example,

Topic: LWM/R/eclipse/10/mqtt-client/0
Message: [2.01 rd/DQ3lH0uWSq]

where

2.01 - response code

rd/DQ3lH0uWSq - registration id

The client will receive the endpoint-id (device id) and application id of the server via the bootstrap process as part of the LwM2M Server object (/1). The short server id is already exposed by the server object at resource 0, i.e (/1/0/0) and this MQTT binding require the LwM2M to include the following resource as part of the LwM2M Server Object to carry the server application name.

Resource -name	Resource-id	Access type	Multiple Instances	Description
Server Application name	9	R	No	Name of the server application

In the case of server, it will learn the client id and application name via the registration operation where client sends the requestor.client.id and requestor.application.id. The server stores the same in the client registry along with other registration parameters like, lifetime and etc..

Along with registration, the current implementation supports the following LwM2M operations in plain text message format,

- Registration Interface
  - register
  - update
  - de-register
- Device Management & Service Enablement Interface
  - read
  - write
  - write-attributes
  - execute
  - create
  - delete
  - deregister
- Information Reporting Interface
  - observe
  - notify
  - cancel-observation

In this project we support only the plain text message format, and other message formats like, TLV, JSON and opaque will be supported later.

Real Temperature reading

In this project we used a Raspberry Pi device with in-built CPU temperature sensor that reports the

temperature reading to the LwM2M server. In order to report the temperature values to the remote LwM2M server, we have taken the IPSO temperature object definition from OMA which represents following resources,

Object info:

Object	Object ID	Object URN	Multiple Instances
IPSO Temperature	3303	urn:oma:lwm2m:ext:3303	Yes

Resource info:

Resource Name	Resource ID	Access Type	Multiple Instances?	Descriptions
Sensor Value	5700	R	No	Temperature Value in °C
Min Measured Value	5601	R	No	minimum value measured since its ON
Max Measured Value	5602	R	No	maximum value measured since its on
Min Range Value	5603	R	No	minimum value that can be measured by the sensor
Max Range Value	5604	R	No	The maximum value that can be measured by the sensor

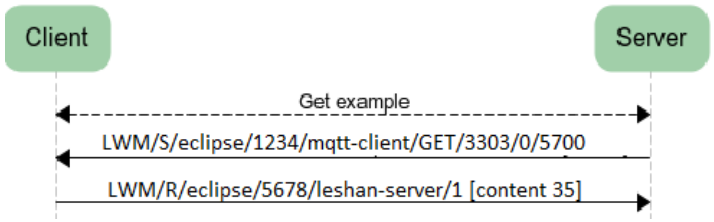
The LwM2M Client (running in Raspberry Pi device) will create the IPSO temperature object instance (3303/0) representing the CPU temperature during the startup and will inform the same to the LwM2M server, at the time of registration via MQTT protocol.

In this client example we represent the following object at client side and expose the same to the LwM2M server via the MQTT protocol,

1. IPSO temperature Object representing the temperature sensor attached to the device
2. LwM2M Device Object representing the client device
3. LwM2M server object representing the LwM2M server parameters

The client/server registers an observer to the MQTT wrapper for every request that it sends to the client. Upon response, the MQTT wrapper notifies the success or failure to the appropriate observer registered earlier for this request. Apart from the simple observer that observes a single message, MQTT wrapper, in Leshan server, defines an MQTT observer that provides the LwM2M information reporting capability at the server.

Following example shows a MQTT message flow for the LwM2M device management interface where the server tries to read the value of the temperature resource.



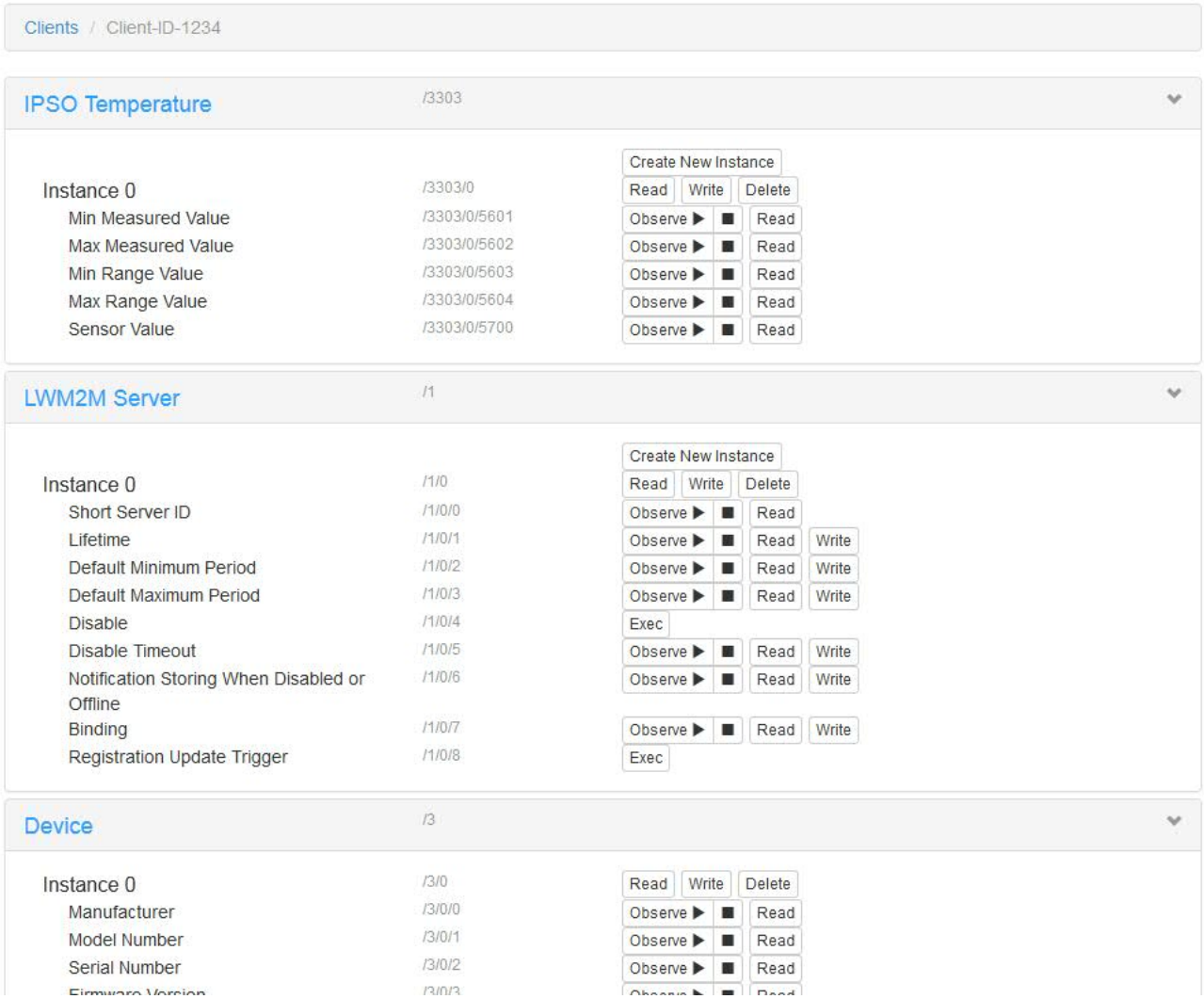
As shown in the above diagram, the read operation is initiated by the LwM2M server with GET operation on a Sensor value Resource (/3303/0/5700) of the IPSO temperature Object. Below is the MQTT message sent by the server to client,

Topic: LWM/S/eclipse/10/mqtt-client/GET/3303/0/5700
Message: 0 56783 leshan-server 1

where it mentions the resource name, operation name as part of the topic and mentions the parameters related to response in the message body. As shown above, Client successfully responds with the temperature value 35.

Leshan Server into the action

The following diagram shows the Leshan web interface that shows the client id and the list of objects it supports. This information is learnt by the server during registration via MQTT. In our example the client exposes the LwM2M Device object, LwM2M Server object and the IPSO Temperature object and hence the server can perform one or more actions on these resources.



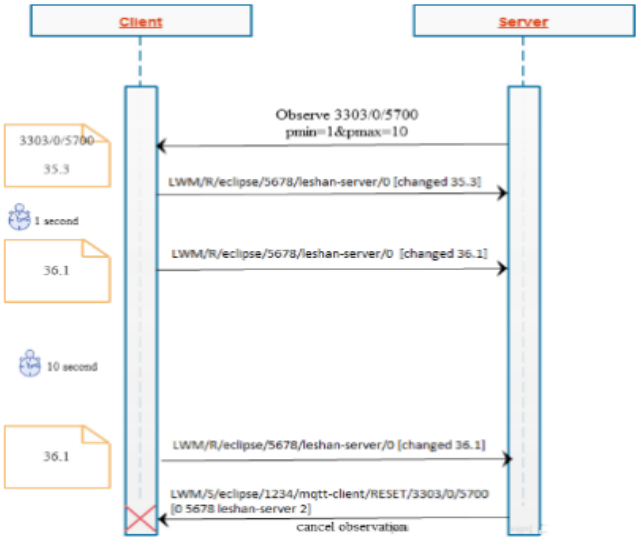
The following diagram shows the information about the Rasbperry Pi device where the client program is running,

The following diagram shows the value of the CPU temperature and also the observe relation established for the Sensor Value Resource.

As shown in below diagram, the Leshan Server initiates the observe relation by sending a GET request on the CPU temperature resource (3303/0/5700). Below is the Topic name used to send the observe relation,

Topic: LWM/S/eclipse/10/mqtt-client/GET/3303/0/5700
Message: 1 56783 leshan-server 2

Similar to read operation, observe as well uses the GET verb but uses a sub operation on the message to differenciate between the observe and read operation.



Once client receives the observe request, creates a observe packet and adds it to the scheduler service that is built to provide the notification. Client responds with a new value to the server within the minimum period set or sends the old value with the maximum period set when there is no change in value. Following is the topic thats used to send the notification messgae by client,

Topic: LWM/R/eclipse/56783/leshan-server/1
Message: 2.04 36.1

The key here is the message-id, so that the server knows its the response for the earlier observe request. Also server does not use the same message-id for further operations until the observe relation is canceled. Server cancels the observation by sending the RESET messages whenever it decides to cancel the observation.

The code is present in the GitHub to build and run the samples follow this link present in the GitHub readme.

Team Members

Amit M Mangalvedkar

Jeffrey I Dare

Sathiskumar Palaniappan

Shalini Kapoor