Security Monitoring of LwM2M Protocol

Technical Report – FIT-TR-2017-16

Ondřej Ryšavý

Technical Report no. FIT-TR-2017-16
Faculty of Information Technology
Brno University of Technology Brno,
Czech Republic

December, 2017

Table of Contents

1. Introduction 3
2. LwM2M Interfaces 5
2.1 Bootstrapping 5
2.2 Client Registration 8
2.3 Device Management 10
2.4 Information Reporting 12
2.5 Queue Mode 13
3. Resources 14
4. Security 16
4.1 LwM2M and DTLS 16
5. Transport Layer Binding 17
6. Experimental Environment 17
7. LwM2M Exported Information 18
6.1 Events 18
8. References 20

1. Introduction

LwM2M is a device management protocol. It has the purpose of supporting many aspects of IoT device management. The protocol defines four interfaces:

- Bootstrapping - configures servers and keying material.
- Device Registration - registers the client and its resource objects to Resource Directory.
- Information Access - server reads or writes resource value on a device.
- Information Reporting - a device asynchronously notifies about new resource value.

Bootstrapping and device registration interfaces are used when a new device (LwM2M client) is turned on and enrolls in the system. Bootstrapping and successive registration aims to prepare a device to be accessible from the server for management.

Information Access and Reporting interfaces are utilized for querying the state of a client, modifying parameters of a client or executing a specific action on a client. In LwM2M environment, the client is necessarily a collection of resources organized in objects that can be read, write or executed. All management and monitoring tasks are thus represented by a series of read/write operations initiated by the server and executed on the client. Moreover, Information Reporting interface enables for asynchronous information delivery. The server can ask a client for providing periodic or event-triggered information on the selected set of resources.

The protocol considers an extensible object and resource model. The resource model employs global registry and public lookup of all objects. The reusable objects provide application semantics.

A client has one or more object instance. Each object is represented as a collection of resources. A resource can be read, write or executed. Objects, resources, and instances have integer identifier. Accessing a resource one needs to provide a unique path to this resources, which is represented as URI string. The new object can be defined by OMA working groups or enterprise organizations. Because all objects are global, the object definition needs to be registered.

The specification also provides security model and binding of LwM2M to transport protocols. Security is based on DTLS [DTLS_REF!] allowing three authentication modes:

- pre-shared,

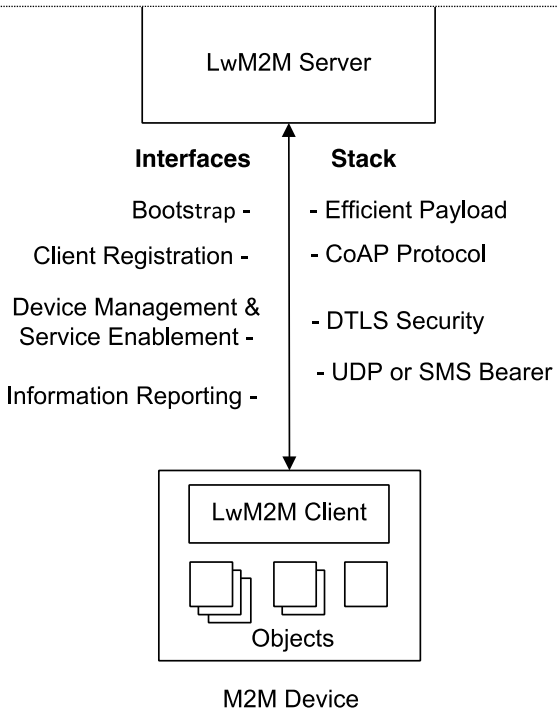

Figure X: Architecture of LwM2M (from [1])

- public key, and
- certificate.

Bootstrapping is utilized for key management. Key material can be preconfigured or read from smart card, etc.

The standard applies to different type of communication networks, namely, cellular, 6LoWPAN, WiFi, ZigBee or other IP based networks. The protocol mainly targets mobile devices connected to the cellular network. Because of constrained resources, the protocol has a lightweight design.

LwM2M does not define its message protocol. Instead, it uses CoAP protocol to encode messages. LwM2M utilizes several types of payload representation [1]:

- Link (application/link-format)[RFC6690]
- Plain (text/plain)
- Opaque (application/octet-stream)
- TLV (application/vnd.oma.lwm2m+tlv)
- JSON (application/vnd.oma.lwm2m+json)

The manner how these data representations are used for payload encoding is further discussed in section 3.

[1] http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats

2. LwM2M Interfaces

There are four interfaces: 1) Bootstrap, 2) Client Registration, 3) Device Management and Service Enablement, and 4) Information Reporting.
Figure {INTERFACES} shows the messages related to each of these interfaces. Bootstrap interface and Client Registration interface are utilized only during client initialization. Device Management and Information Reporting are used during normal operation for retrieving information from clients or updating and executing resources as a part of device management.

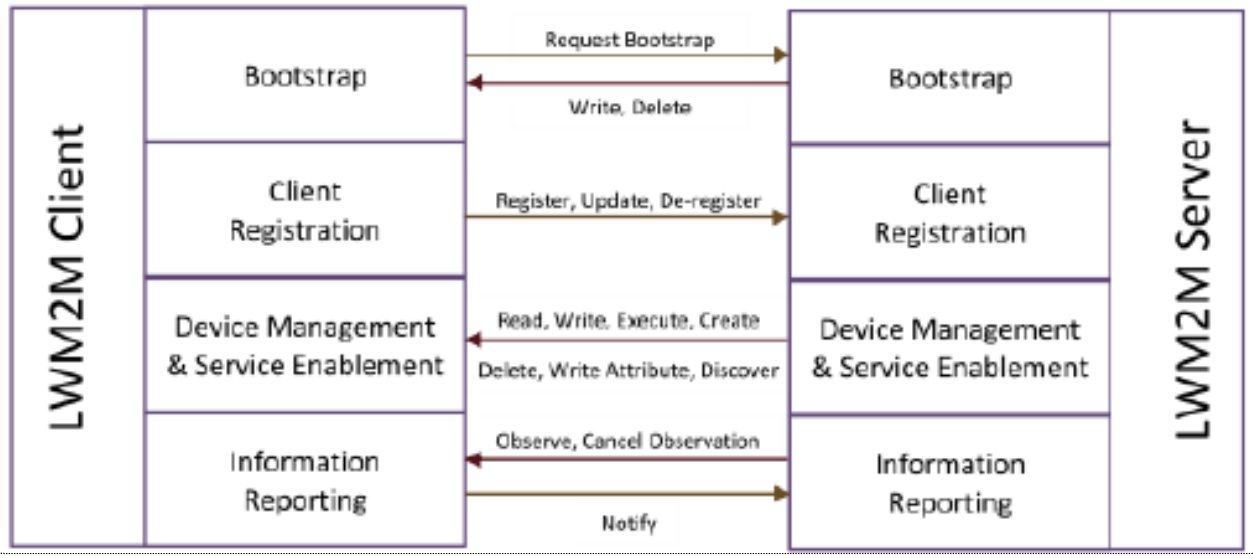

Figure INTERFACES: Interface Flows (from [2])

General operations that interact with Resources, Resource Instances, Objects, and Object instances are as follows:

- Read - is used to read current values.
- Create - is used to create a new instance of resource or object.
- Delete - is used to delete the instance of resource of object.
- Write - is used to update the values.
- Execute - is used to initiate an action.
- Discover - is used to discover attributes and to discover which Resources are implemented in a certain Object.

Interfaces also define specific operations, such as Request-Bootstrap, Cancel Observation, etc.

2.1 Bootstrapping

The Bootstrap interface serves for initializing necessary information in the LwM2M client before the client can perform REGISTER operation with LwM2M server.

A client may ignore requests and response in this phases if not related to its Bootstrap mode, but it must support at least one Bootstrap mode defined. There are predefined Bootstrap modes:

- Factory bootstrap - all necessary information is defined in the local permanent memory and is loaded in the device initialization. The information must be provided before the device is deployed.
- Bootstrap from Smart Card - information is provided from a Smart Card, e.g., SIM, and is loaded in the device initialization. This mode improves the security and provides more flexibility.
- Client Initiated Bootstrap - information is obtained from the LwM2M Bootstrap server. The client initiates the Bootstrap process. The client needs to be preloaded with the LwM2M Bootstrap-Server Account information that also contains security credentials for a DTLS connection established to the bootstrap server. The communication is presented in Figure {IBOOTSTRAP}.
- Server Initiated Bootstrap - information is obtained from the LwM2M Bootstrap server. The server initiates the Bootstrap process. The communication is the same as presented in Figure {IBOOTSTRAP) except that there is not the Bootstrap-Request message. Still, the server needs to find out when the client is ready for bootstrap operation. In this case, this is implementation dependent.

For the last two Bootstrap modes, the LwM2M Bootstrap server is used to provide clients with the necessary information. The main result of the bootstrap process is the receipt on how to contact a Registration server.

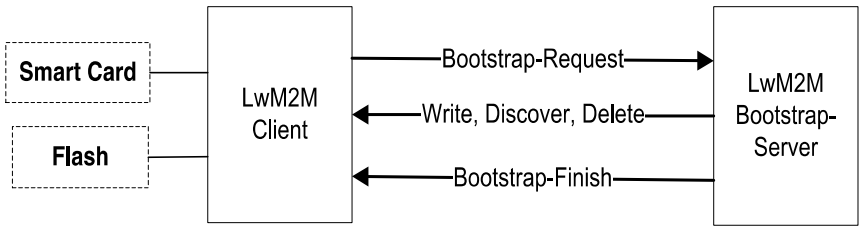

Figure IBOOTSTRAP: Bootstrap Communication (from [1])

Bootstrap communication starts with Bootstrap-Request message and ends with a Bootstrap-Finish message. Between these two messages, the bootstrap server may configure the client with necessary information. Bootstrap Information can be of the following two types:

- **LwM2M Server Bootstrap Information** is used by the LwM2M client to register and connect to the LwM2M server. This information contains LwM2M Server

Account and additional object instance (access control, connectivity object)

- **LwM2M Bootstrap-Server Bootstrap Information** is optional information providing LwM2M Bootstrap-Server Account. It is used by the client to contact LwM2M Bootstrap Server securely.

Bootstrap phase finishes by sending BOOTSTRAP-FINISH command. Error response code must report any inconsistencies during this phase. The bootstrap server may initiate corrective actions when receiving an error response code.

The following table summarizes the operation of Bootstrap interface. Because LwM2M maps to CoAP, the operations are defined regarding CoAP methods and URI:

Operation	Method	URI	Success	Failure
BOOTSTRAP-REQUEST	POST	/bs?ep={ClientName}	2.04	4.00 4.15
BOOTSTRAP-FINISH	POST	/bs	2.04	4.00 4.06
BOOTSTRAP-DISCOVER	GET	/{ObjectID}	2.05	4.00 4.04
BOOTSTRAP-WRITE	PUT	/{ObjectID}/{ObjectInstanceID}/{ResourceID}	2.04	4.00
BOOTSTRAP-DELETE	DELETE	/{ObjectID}/{ObjectInstanceID}	2.02	4.00

An example of client initiated bootstrap conversation is depicted in Figure {CLIENT-BOOTSTRAP}.

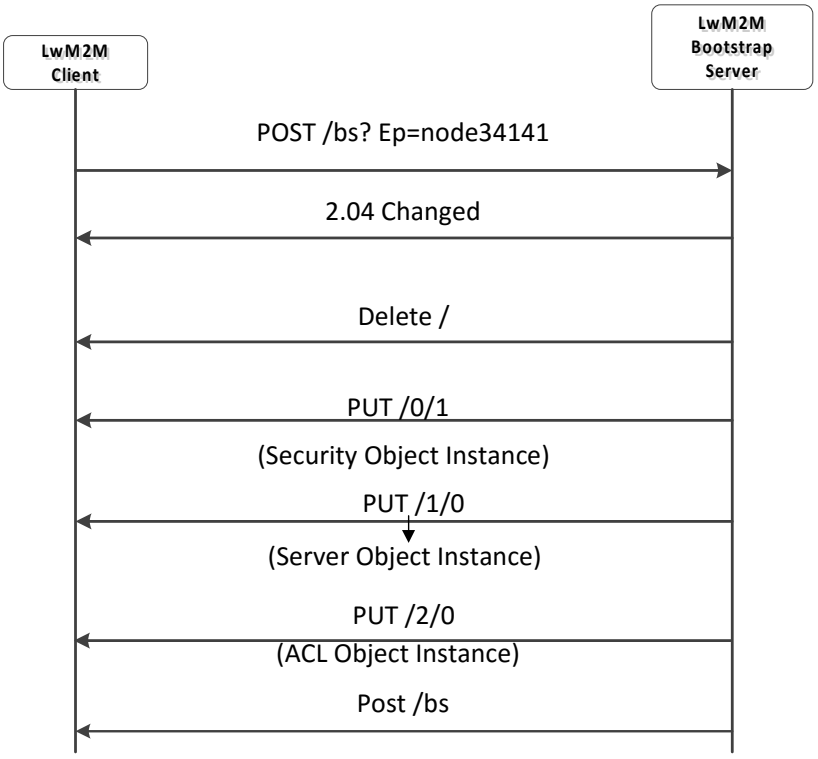

Figure CLIENT-BOOTSTRAP: Example of Client initiated Bootstrap (from [1])

2.2 Client Registration

The client registration is performed by a client to register with one or more servers. In registration, the client provides the properties necessary for the server to recognize the client. Registration information includes the following data:

- End point name of the client
- Registration lifetime and queue mode
- Objects supported by a client

Registered client updates lifetime resource of the corresponding server object instance.

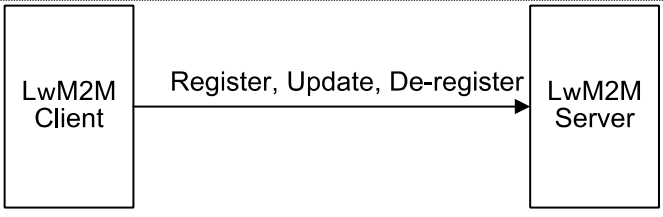

Figure REGISTRATION: Client Registration (from [1])

The following table summarizes operation of Registration interface

Operation	Method	URI	Success	Failure
REGISTER	POST	/rd?ep={ClientName}&lt={Lifetime}&sms={MSISDN} &lwm2m={version}&b={binding}	2.04	4.00 4.03 4.12
UPDATE	POST	/{location}?lt={Lifetime}&sms={MSISDN} &b={binding}	2.04	4.00 4.04
DEREGISTER	DELETE	/{location}	2.02	4.00 4.04

The register operation is performed by the client on a server after the bootstrap procedure has completed. The parameters provided by the client is as follows:

- Endpoint client name identifies the client node for further reference.
- Lifetime indicates the expected duration of the registration.
- Version specifies the version of the protocol.
- Binding mode defines the current transport binding and queue mode. Currently, there are several queue modes for UDP and SMS transport.
- SMS Number is the MSISDN if the client is reachable through SMS binding.
- Objects and Object instances are lists of available objects and instance on the client.

The register message has a payload of type application/link-format. The payload contains a list of objects and instances using link format. For example, if the client supports Access Control, Device, and Firmware Update it provides the following list of objects: </2>,</3>,</5>. Result of registration is location information that represents an object instance created for registered client.

The update operation is used by a client or server to refresh the registration information. Lifetime, binding mode, SMS number and object list can be updated. Usually, the update is used to restore the lifetime parameter. Also, adding or removing objects is announced by the update operation.

The deregister operation is executed when the client no longer requires registration to a server. Server processed this operation by removing all registration information belonging to the client.

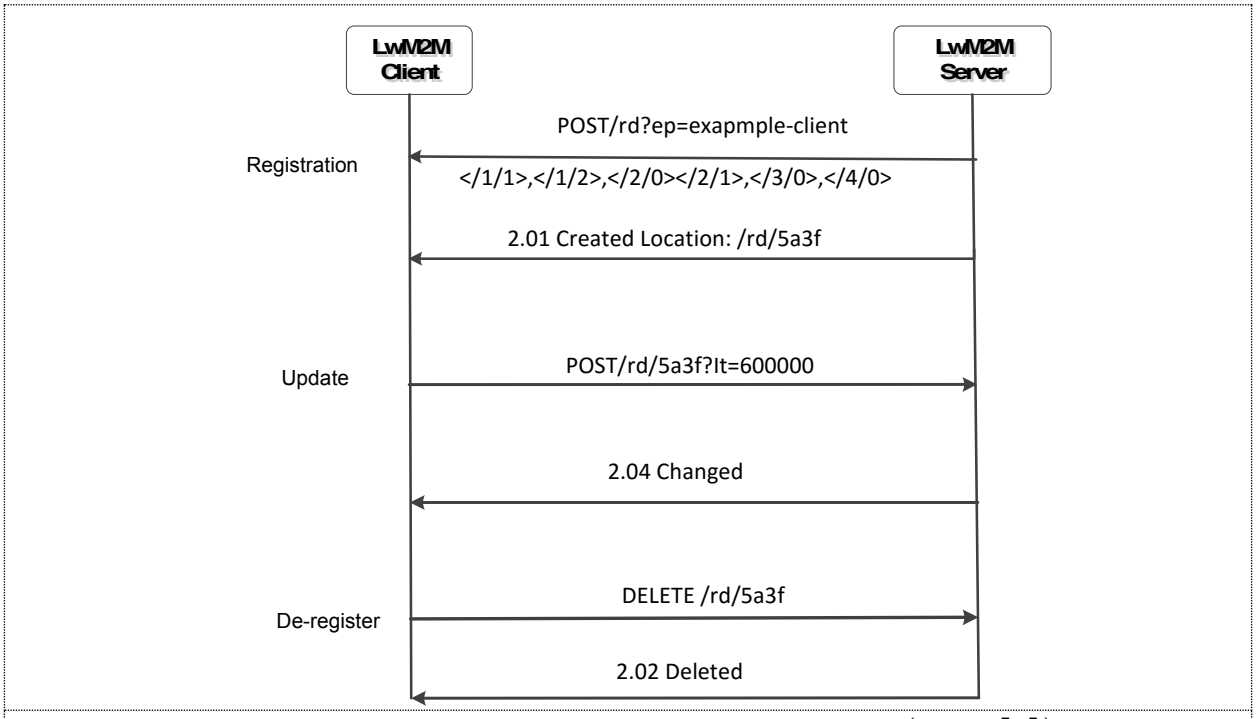

Figure CLIENT-REGISTER: Example of Client registration (from [1])

2.3 Device Management

The Device Management interface is used by a server to access Object Instances and Resources on a registered client. This interface defines the following operations:

Operation	Method	URI	Success	Failure
READ	GET	/{ObjectID}/{ObjectInstanceID}/{ResourceID}	2.05	4.00 4.01 4.04 4.05 4.06
DISCOVER	GET	/{ObjectID}/{ObjectInstanceID}/{ResourceID}	2.05	4.00 4.04
WRITE	PUT	/{ObjectID}/{ObjectInstanceID}/{ResourceID}	2.04	4.00 4.01 4.04 4.05 4.06
WRITE	POST	/{ObjectID}/{ObjectInstanceID}	2.31	4.08 4.13
WRITE-ATTRIBUTES	PUT	/{ObjectID}/{ObjectInstanceID}/{ResourceID} ?pmin={minimumPeriod}&pmax={maximumPeriod} &gt={greaterThan}&lt={lessThan}&st={step}	2.04	4.00 4.01 4.04 4.05

EXECUTE	POST	/{ObjectID}/{ObjectInstanceID}/{ResourceID}	2.04	4.00 4.01 4.04 4.05
CREATE	POST	/{ObjectID}	2.01	4.00 4.01 4.04 4.05 4.06
DELETE	DELETE	/{ObjectID}/{ObjectInstanceID	2.02	4.00 4.01 4.04 4.05

All operations use the identification of objects/resources by the path:

/{ObjectId}/{ObjectInstanceId}/{ResourceId}.

Read operation expect the response including a value in plain text, Opaque, TLV or JSON format according to the specified Content-Format. Preferred Content-Format may be suggested in the request.

Values of resources or object instances are set by WRITE operation that is represented by sending CoAP PUT or POST request with the specified URI. The request contains a value in plain text, Opaque, TLV or JSON format.
Some resources can be executed which is achieved by sending CoAP POST request. The request may contain a list of arguments.

A new object instance is created by CREATE operation which is encoded as CoAP POST request with URI specifying the object ID. The request may contain values in TLV or JSON format to be used in object creation. The ID of an object instance is returned in CoAP response and "2.01 Created" is signalized that this operation completed successfully.

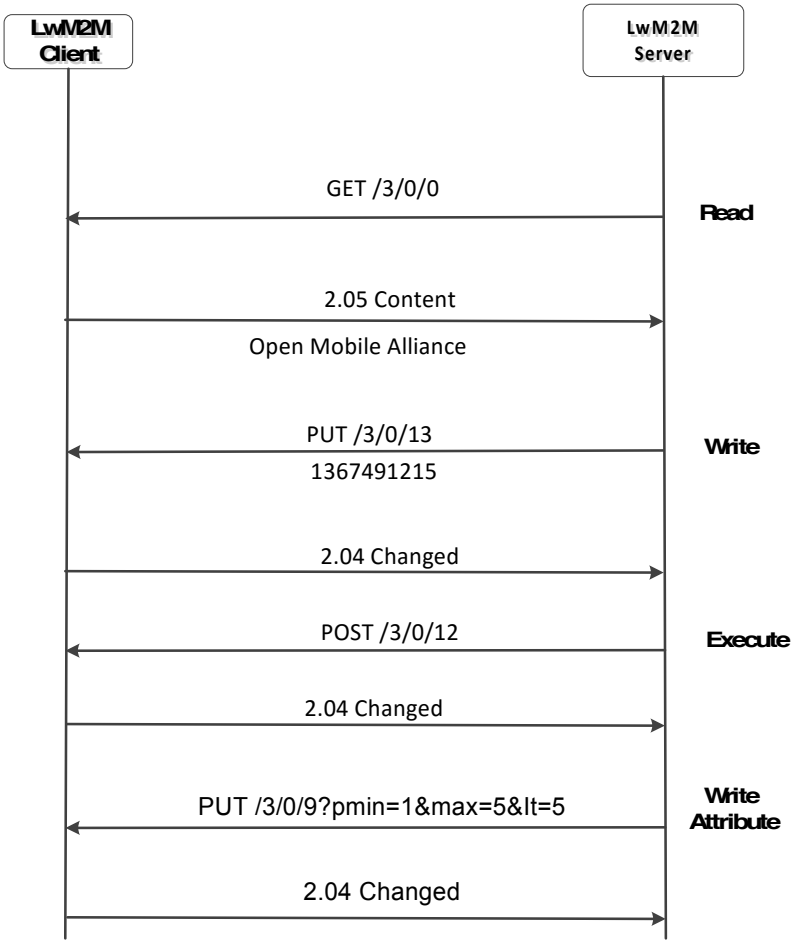

Figure CLIENT-REGISTER: Example of Device Management communication (from [1])

2.4 Information Reporting

Information Reporting interface serves for periodic or event-triggered reporting about resource values from a client to the server. Operations of this interface rely on the CoAP Observe mechanism.

Operation	Method	URI	Success	Failure
OBSERVE	GET OBS=0	/{ObjectID}/{ObjectInstanceID}/{ResourceID}	2.05	4.00 4.01 4.04 4.05
CANCEL OBSERVATION	GET OBS=1	/{ObjectID}/{ObjectInstanceID}/{ResourceID}	2.05	4.00 4.01 4.04 4.05
NOTIFY	Async Response		2.05	

The OBSERVE operation sends a CoAP GET message with Observe set option, which causes that notification will be sent every time the value changes or periodically. The CoAP message token value is used to match notifications to the observe GET message. Observing a resource is canceled either by sending CoAP GET with the Observe reset option sent or responding to the notification with an RST CoAP message.

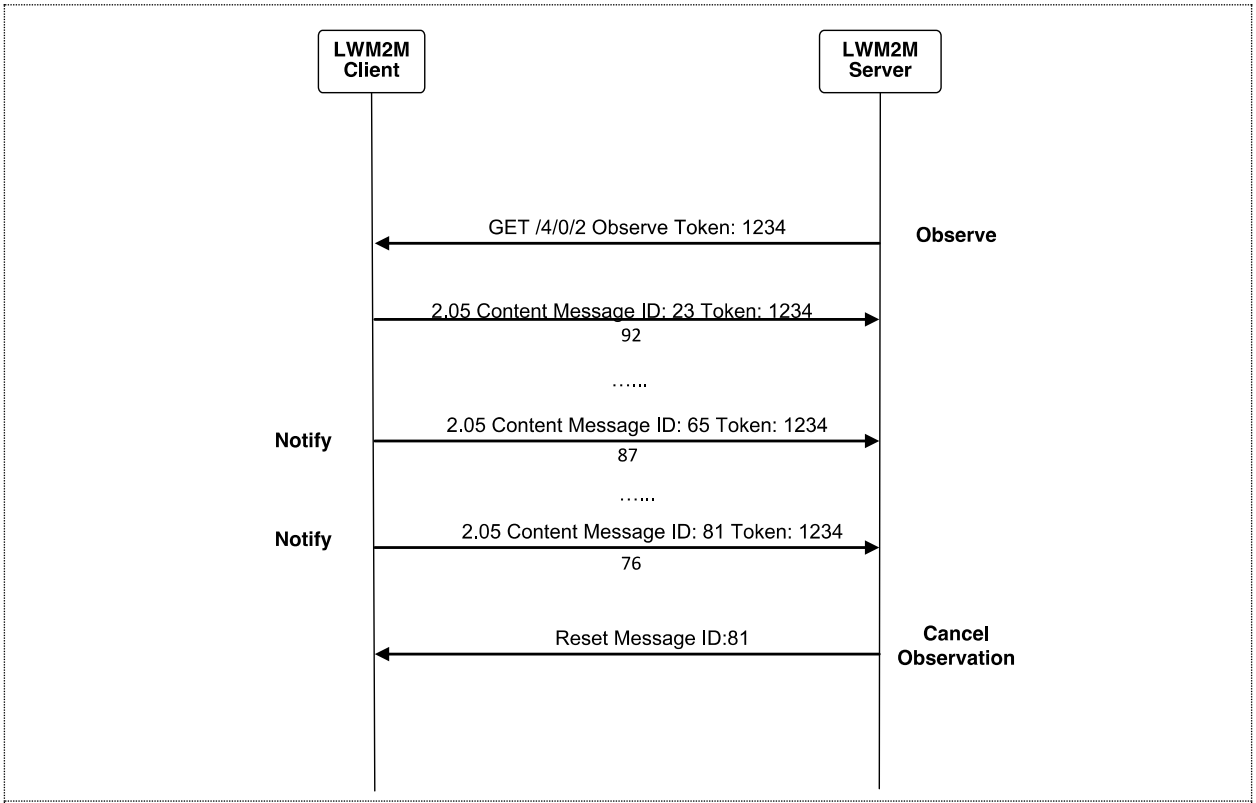

Figure CLIENT-BOOTSTRAP: Example of Information Reporting (from [1])

2.5 Queue Mode

The Queue Mode supports disconnected clients. Clients may request Queue Mode during registration. When this mode is used, the server does not send the request immediately but queues them until the client is online. Clients inform servers about any period they become offline and also signalize their reachability. The client indicates its availability by UPDATE message through Registration interface.

3. Resources

Each client has one or more object instances. An object is a collection of resources. A resource can be read, write or executed. Figure {RESOURCE} presents a data model of a typical client.

An integer number identifies objects and resources. Server access a resource located at a client by using a simple URI pattern:

/ObjectID/ObjectInstance/ResourceID

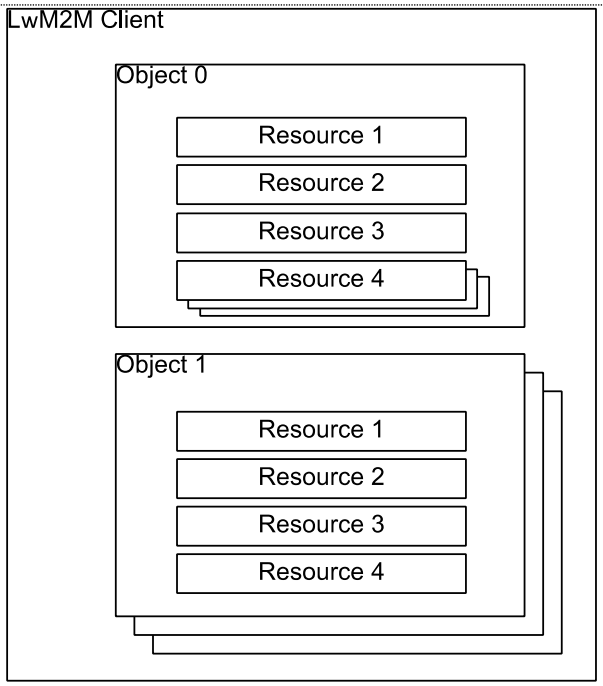

Figure RESOURCE: Resource Model (from [1])

Each object has a specification that defines supported operations and available resources. Objects and resources may have multiple instances. Objects may exist in different versions which endorses the situation when resources are added or removed from the object. The version is composed of two digits separated by a dot representing major and minor version.

The standard defines some predefined objects:

- **LWM2M Server (1)** - This objects provides data about LWM2M server.

- **Access Control (2)** - This object is used to check if the server has enough rights to perform specified operations.
- **Device (3)** - This object provides device related information.
- **Connectivity Monitoring (4)** - This object can be used to monitor network related parameters.
- **Firmware (5)** - This object provides functions and information related to firmware management.
- **Location (6)** - This object provides location information for the devices based on GPS information.

LwM2M uses four data formats for representing possible values of resources:

- **Plain text** it is used for READ and WRITE operations. It provides a value in plain text format.
- **Opaque** it is used for READ and WRITE operations associated with resources whose values have binary format, such as images or application specific binary data.
- **TLV** is used for READ and WRITE operations enable to represents either singular value or an array of values in a compact way. The format is self describing. It consists of four parts, namely, TYPE, IDENTIFIER, LENGTH and VALUE. This format is defined in [OMA-TS-LightweightM2M-V1_0-20170208-A, p.48][2].
- **JSON** is used for READ and WRITE operations to transport single or multiple resource values. It provides a value in JSON text format. The data must contain resource array assigned to JSON variable "e". For example, the following shows three values of a temperature sensor "72/1/2":

```
{
   "bn" : "/72/ ",
   "e" :[
     {"n":"1/2","v":22.4,"t":-5},
     {"n":"1/2","v":22.9,"t":-30},
     {"n":"1/2","v":24.1,"t":-50}
   ],
   "bt" :25462634
}
```

The meaning of JSON variables are explained in [OMA-TS-LightweightM2M-V1_0-20170208-A, p.56].

[2] Kaitai specification of LwM2M TLV is available from https://github.com/rysavy-ondrej/Netfox.NDX/blob/master/Ndx.Packets/Kaitai/lwm2m-tlv.ksy

4. Security

Security of LwM2M environment within UDP binding is based on the use of DTLS protocol for protecting CoAP data exchange. Enforcing a secure client-server communication requires addressing the two aspects:

- Authentication - mutual authentication of a client and a server is required prior to exchange of any information.
- Encryption and integrity protection - any communication between a client and a server should be protected by encryption.

For authentication, LwM2M considers employing credentials that can be of different types. The following types of credentials are considered in the LwM2M environment:

- Certificates - clients and severs have certificates that can be validated as a part of authentication.
- Raw public keys[3] - private/public key pairs exist for servers and clients. Public key is known in the system.
- Pre-shared keys - both client and server share a key used for authentication.

Because LwM2M expects that the underlying DTLS protocol provides security, the use of a particular type of credentials expects to select appropriate cipher suite of DTLS as discussed in the next section.

4.1 LwM2M and DTLS

Typical DTLS implementation provides a collection of cipher suites for providing secure communication. A cipher suite defines at least three algorithms:

- Key exchange algorithm is used for exchange keys during the secure channel initialization. The agreed keys are then used in encryption algorithm.
- Message encryption algorithm is used to protect application data during the entire communication.
- Message authentication code algorithm is used for providing message integrity check.

[3] The use of raw public keys is considered in RFC7250. In this architecture, it is assumed that public key is distributed by other means thus operations involved in certificate processing are not necessary.

From the available cipher suites, the LwM2M standard selects a subset that is mandatory for client and server implementations. The only required cipher suites are[4]:

- Pre-Shared Keys:
  TLS_PSK_WITH_AES_128_CCM_8
  TLS_PSK_WITH_AES_128_CBC_SHA256
- Raw Public Keys and X.509 Certificates:
  TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
  TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

These cipher suites are based on AES block encryption algorithm. The length of the key is 128 bits. The AES encryption algorithm is used either in counter mode (CCM) that provides both authentication and encryption mode or in the encryption mode complemented with cipher block chaining (CBC) message authentication code provided by SHA256.

5. Transport Layer Binding

Both client and server must support UDP binding and should support SMS binding. UDP binding means that the communication is transported in UDP datagrams and the protocol has registered scheme **coap://** at the default port of **5683**. SMS binding stands for transmitting a CoAP message in SMS payload. As SMS is limited to 140 characters concatenation is used for messages larger than that.

6. Experimental Environment

We have developed an experimental environment that consists of several LwM2M software emulated clients and LwM2M server. The environment simulates the smart metering application.[5] This demonstration environment consists of a single server equipped with a simple UI to present information read from multiple clients. Each client provides measurement values for a number of households.

To simulate a real world situation, the NREL dataset https://data.nrel.gov/submissions/69 was used as the source of data. The data

[4] These ciphersuites are suitable for resource constrained devices and are available as a part of TinyDTLS implementation (https://projects.eclipse.org/projects/iot.tinydtls).
[5] https://github.com/rysavy-ondrej/Ironstone.LwM2M/tree/master/SmartMetering

set contains values of power demands for 200 households meassured every 10 minutes.
To get more realistic data we used the data set for reference points and compute every second actual values of demand.

This implementation is the source of LwM2M traffic that can be used for validation and testing the monitoring system.

7. References

1. Open Mobile Alliance. (2017). Lightweight Machine to Machine Technical Specification.
2. Rao, S., Chendanda, D., Deshpande, C., & Lakkundi, V. (2016). Implementing LWM2M in constrained IoT devices. In *2015 IEEE Conference on Wireless Sensors, ICWiSE 2015* (pp. 52-57).