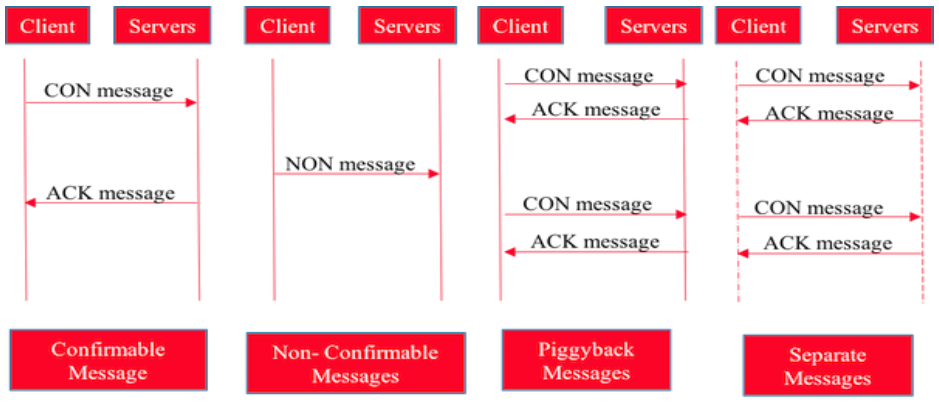
HTTP

- HTTP stands for Hypertext Transfer Protocol. It's the network protocol used to deliver virtually all files and other data (collectively called resources) on the World Wide Web, whether they're HTML files, image files, query results, or anything else. Usually, HTTP takes place through TCP/IP sockets
- A browser is an HTTP client because it sends requests to an HTTP server (Web server), which then sends responses back to the client. The standard (and default) port for **HTTP servers to listen on is 80**, though they can use any port.
- **What are "Resources"?**
- HTTP is used to transmit resources, not just files. A resource is some chunk of information that can be identified by a URL (it's the R in URL). The most common kind of resource is a file, but a resource may also be a dynamically-generated query result, the output of a CGI script, a document that is available in several languages, or something else.
- Like most network protocols, HTTP uses the client-server model: An HTTP client opens a connection and sends a request message to an HTTP server; the server then returns a response message, usually containing the resource that was requested. After delivering the response, the server closes the connection (making HTTP a stateless protocol, i.e. not maintaining any connection information between transactions).

6.2 CoAP

The Constrained Application Protocol (CoAP) is another session layer protocol designed by IETF Constrained RESTful Environment (Core) working group to provide lightweight RESTful (HTTP) interface. **Representational State Transfer (REST) is the standard interface between HTTP client and servers.** However, for lightweight applications such as IoT, REST could result in significant overhead and power consumption. CoAP is designed to enable low-power sensors to use RESTful services while meeting their power constrains. It is built over UDP, instead of TCP commonly used in HTTP and has a light mechanism to provide reliability. **CoAP architecture is divided into two main sublayers: messaging and request/response.** The messaging sublayer is responsible for reliability and duplication of messages while the request/response sublayer is responsible for communication. **CoAP has four messaging modes: confirmable, non- confirmable, piggyback and separate.** Confirmable and non-confirmable modes represent the reliable and unreliable transmissions, respectively while the other modes are used for request/response. **Piggyback is used for client/server direct communication where the server sends its response directly after receiving the message**, i.e., within the acknowledgment message. On the other hand, the separate mode is used when the server response comes in a message separate from the acknowledgment, and may take some time to be sent by the server. **As in HTTP, CoAP utilizes GET, PUT, PUSH, DELETE messages requests to retrieve, create, update, and delete, respectively**

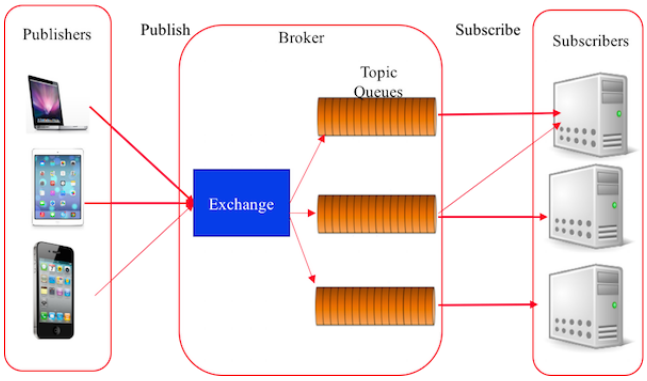


6.3 XMPP

Extensible Messaging and Presence Protocol (XMPP) is a messaging protocol that was **designed originally for chatting and message exchange applications.** It was standardized by IETF more than a decade ago. Hence, it is well known and has proven to be highly efficient over the internet. Recently, it has been reused for IoT applications as well as a protocol for SDN. This reusing of the same standard is due to its use of XML which makes it easily extensible. **XMPP supports both publish/ subscribe and request/ response** architecture and it is up to the application developer to choose which architecture to use. It is designed for near real-time applications and, thus, efficiently supports low-latency small messages. It does not provide any quality of service guarantees and, hence, is not practical for M2M communications. Moreover, XML messages create additional overhead due to lots of headers and tag formats which increase the power consumption that is critical for IoT application. Hence, XMPP is rarely used in IoT but has gained some interest for enhancing its architecture in order to support IoT applications

6.4 AMQP

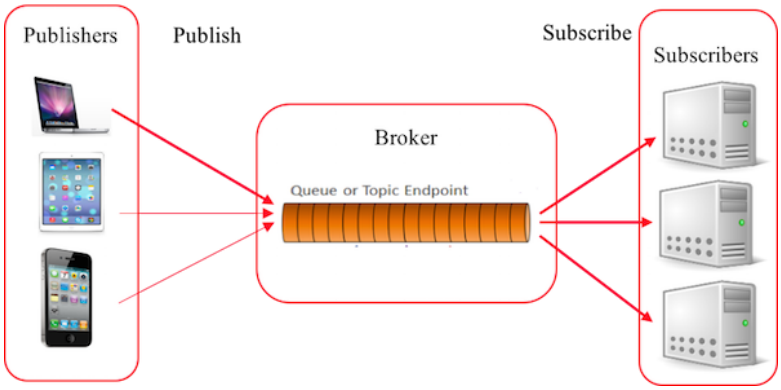
The Advanced Message Queuing Protocol (AMQP) **is another session layer protocol that was designed for financial industry**. It runs over TCP and provides a publish/ subscribe architecture which is similar to that of MQTT. The difference is that the **broker is divided into two main components: exchange and queues.** The exchange is responsible for receiving publisher messages and distributing them to queues based on pre-defined roles and conditions. Queues basically represent the topics and subscribed by subscribers which will get the sensory data whenever they are available in the queue



AMQP Architecture Diagram.

6.5 MQTT

Message Queue Telemetry Transport (MQTT) was introduced by IBM in 1999 and standardized by OASIS in 2013. It is designed to provide embedded connectivity between applications and middleware on one side and networks and communications on the other side. It follows a publish/subscribe architecture, where the system consists of **three main components: publishers, subscribers, and a broker**. From IoT point of view, publishers are basically the lightweight sensors that connect to the broker to send their data and go back to sleep whenever possible. Subscribers are applications that are interested in a certain topic, or sensory data, so they connect to brokers to be informed whenever new data are received. The brokers classify sensory data in topics and send them to subscribers interested in the topics.



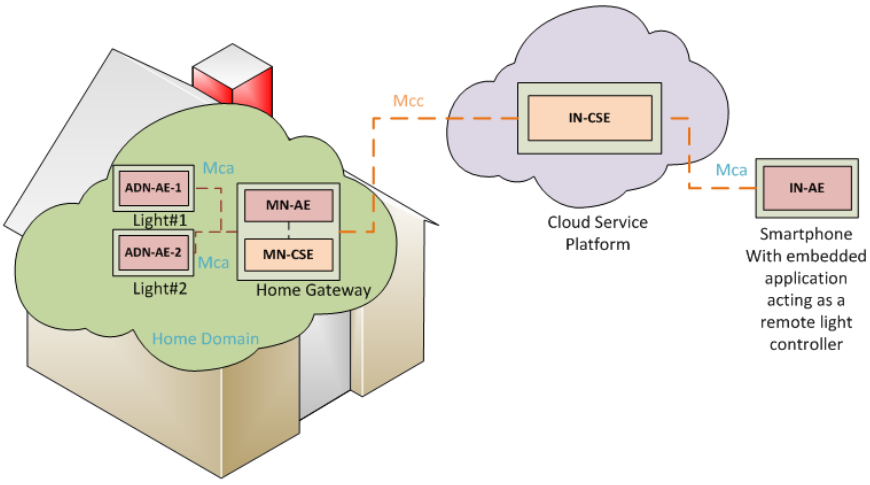
MQTT Architecture diagram

oneM2M

oneM2M is a global organization that creates requirements, architecture, API specifications, security solutions and interoperability for Machine-to-Machine and IoT technologies.
oneM2M specifications provide a framework to support a wide range of applications and services such as smart cities, smart grid, connected car, home automation, public safety, and health.

oneM2M standard employs a simple horizontal, platform architecture that fits within a **three layer model comprising applications, services and networks**. In the first of these layers, Application Entities (AEs) reside within individual device and sensor applications. They provide a standardized interface to manage and interact with applications. **Common Services Entities (CSEs) play a similar role in the services layer which resides between the applications layer and the in the network layer**. The network layer ensures that devices and sensors and applications are able to function in a network-agnostic manner.

In the oneM2M functional architecture two basic types of entities are defined. One is an AE (short for Application Entity) and the other is a CSE (short for Common Services Entity). In this use case, the lights and smartphone each host an AE. Also an IN-CSE (short for Infrastructure Node CSE) is hosted in the cloud by the oneM2M Service Provider and a MN-CSE (short for Middle Node CSE) is hosted on the Home Gateway.



The oneM2M defined Mca reference point is used to interface an AE and CSE. The oneM2M defined Mcc reference point is used to interface CSEs. In this use case, the Mca reference point is used between the Light ADN-AEs and home gateway MN-CSE and between the Smartphone AE and IN-CSE. The Mcc reference point is used between the home gateway MN-CSE and Cloud service platform IN-CSE. In summary, applications used in the current use case are classified as follows:

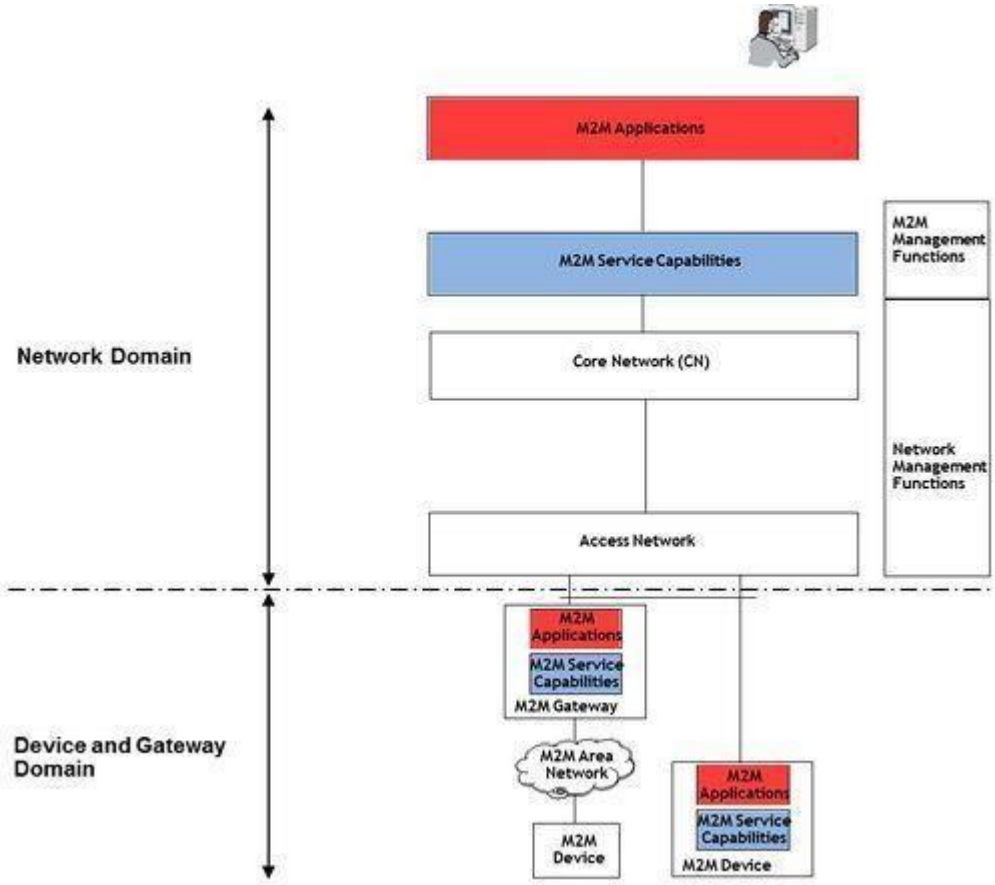
- **ADN-AE1:** an application embedded in Light#1 with capabilities to control Light#1 and interact with the home gateway MN-CSE through Mca reference point.
- **ADN-AE2:** an application embedded in Light#2 with capabilities to control Light#2 and interact with the home gateway MN-CSE through Mca reference point

- **IN-AE:** a smartphone application embedded in the smartphone device with capabilities to interact directly with the cloud service platform IN-CSE through Mcc reference point and thereby remotely control Light#1 and Light#2.
- **MN-AE:** a gateway application embedded into the home gateway that interacts with MN-CSE through Mca reference point.

ETSI M2M

ETSI Machine-to-Machine communications is an application agnostic standard containing an overall end to end M2M functional architecture, identifying the functional entities and the related reference points. **It describes a resources-based architecture that can be used for the exchange of data and events between machines involving communications across networks without requiring human intervention**

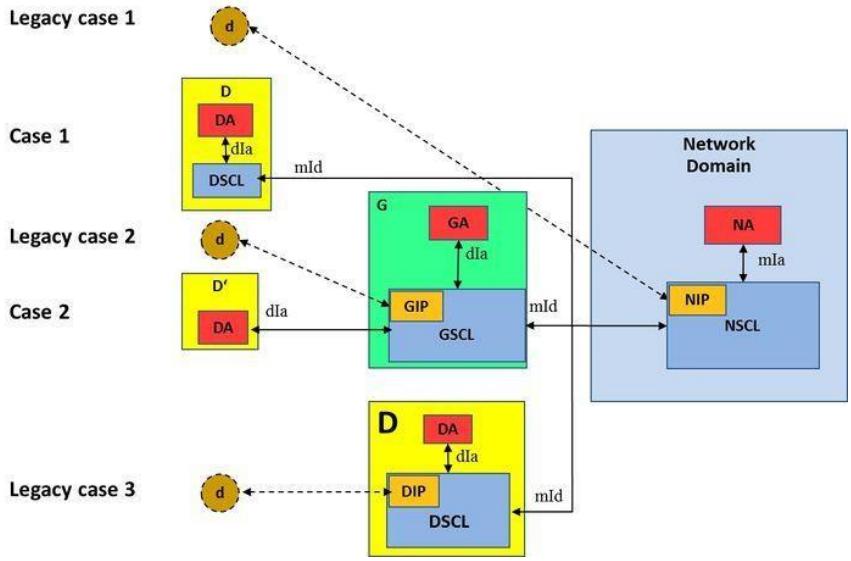
The ETSI M2M High Level Architecture picture shows that this is a distributed system: the M2M Service Capabilities are both at network level (M2M Service Capabilities in the Network Domain) and at local level (M2M Service Capabilities in the M2M Gateway and in the M2M Device). These Service Capabilities are the set of functionalities defined in the specification and are used to put in communication applications among them; both network applications, and gateway and device applications. In essence the goal of this architecture is "to provide the functionalities for the management of interactions between entities (i.e. applications) involving communication across networks without requiring human intervention" as it is described in the ETSI M2M definition of M2M.

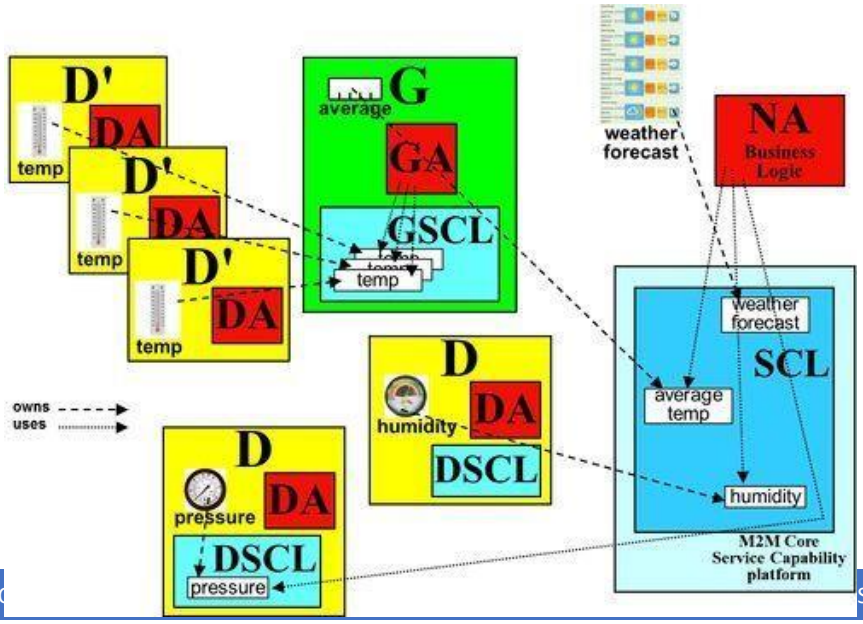


In order to standardize the procedures that can be used for enabling these entities (i.e. applications) to communicate the ETSI M2M specifications have defined a number of Reference Points and the operations that can be used for this communication. These Reference Points are:

- mIa - M2M application interface: it is used by the Network Applications (NA) to communicate with the Network Service Capability Layer (NSCL)
- dIa - Device application interface: it is used by the Device and Gateway Applications (DA and GA) to communicate with the local service capabilities, i.e. Device Service Capability Layer (DSCL) and Gateway Service Capability Layer (GSCL)

- mId - M2M to device interface: it is used for the inter-SCLs communication





The main operations that can be performed using these interfaces deal with the concept of M2M Resource. A "resource" can be roughly described as a shared memory area that can be used for exchanging data among applications. The resources can be created by an application within an SCL (at network, gateway or device level), and can be read, updated, subscribed, notified, announced, discovered, deleted by the same or other applications (provided they have the permissions to perform the actions requested). Resources can be created and used freely across the ETSI M2M architecture. Looking at the following example of a weather forecast system including applications and sensors, a resource can be created by a Device or a Gateway Application at local level, in a GSCL (e.g. "temp" resource) or in a DSCL (e.g. "pressure" resource), or it can be created at network level in the NSCL (e.g. "average temp" or "humidity" resources). Network Applications can discover and access resources created both at network level and at local level, and they can create resources too (e.g. "weather forecast" resource).

OMA

OMA Lightweight M2M is a protocol from the **Open Mobile Alliance** for M2M or IoT device management. Lightweight M2M 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.** It provides a choice for the M2M Service Provider to deploy a M2M system to provide service to the M2M User. It is frequently used with CoAP

OMA Lightweight M2M is designed to:

- Provide Device Management functionality over sensor or cellular networks
- Transfer service data from the network to devices
- Extend to meet the requirements of most any application

BBF- Broadband Forum

[https://www.iot-now.com/2018/04/18/80848-broadband-forum-brings-iot-home-new-usp-standard/]

It brings IoT home with new USP standard. The Broadband Forum published its User Services Platform (USP) – a new standard for implementing, deploying and managing all types of devices in the broadband home. For the first time, USP brings a unified, common approach to securely deploy, manage, and control network-aware consumer electronics, including home and enterprise Wi-Fi, Internet of Things (IoT), and more. This enables a vast number of devices from different suppliers to be integrated into service providers' offerings, enabling them to offer new, revenue-generating services to end-users. "This means deployments with IoT devices, smart Wi-Fi, set-top boxes, and smart gateways can be controlled by anyone in the household, while permission levels for service providers can be created to allow any necessary updates or troubleshooting of the network and devices connected to it. Ultimately, we designed USP to be flexible, scalable and secure."

With USP, service providers, consumer electronics manufacturers and end-users can perform lifecycle management of connected devices and carry out upgrades, for example, for critical security updates. Newly installed or purchased devices and virtual services can also be easily added, while customer support is improved by remote monitoring and troubleshooting of connected devices, services and home network links.

The specification enables secure control of IoT, smart home and smart networking functions and helps map the home network to manage service quality and monitor threats.

[ETSI TS 118 106 V1.1.0 (2016-03)
oneM2M; Management Enablement (BBF)
(oneM2M TS-0006 version 1.1.4 Release 1)]
The BBF TR-069 [4] specification defines three (3) types of devices, known as CPEs, that are capable of being managed from the perspective of the TR-069 agent:

- **CPE that hosts the TR-069 agent**: section A.3.3.1 Inform of BBF TR-069 [4] defines the required fields for a CPE to be identified. These fields include the OUI and Serial Number of the CPE assigned by the CPE manufacturer. Optionally the manufacturer may assign a Product Class to the CPE. The format of the identifier is as follows: OUI-[PC-]SN.
- **Virtual Device:** This type of device is addressed as a CPE. The Virtual Device has its own OUI-[PC-]SN as represented by the CPE Proxier. The CPE Proxier emulates a CWMP agent for each Virtual Device.
- **Embedded Device:** This type of device is addressed as one or more objects within the data model of the CPE that hosts the TR-069 agent.