18CSE345T
IoT Architecture And Protocols

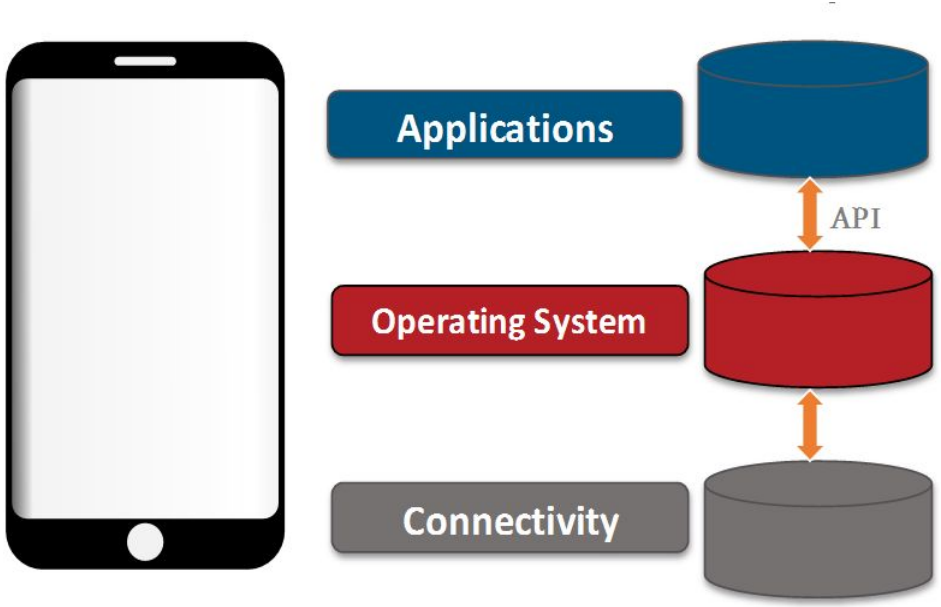
Unit 5

Syllabus

Service Layer Protocols- Introduction, oneM2M, ETSI M2M, OMA, BBF, Understanding Security and Interoperability, Modes of attack: DoS, Getting Access, Guess, Man in Middle, Sniff, Port Scan, Modes of attack: Web Crawl, Search Features and Wild Cards, Breaking Cipher, Tools for achieving Security: VPN, X.509, Authentication, Tools for achieving Security: User names and Passwords, Message Brokers, Tools for achieving Security: Provisioning servers, Centralization versus decentralization, The need for interoperability : Combining Security and Interoperability, Need for Security in IoT Protocols –Introduction, Security in IoT Protocols :MAC 802.15.4, 6LoWPAN, RPL, Security in IoT Protocols :Application Layer

Service Layer Protocols- Introduction

The IoT Service Layer specified in oneM2M can be understood as a **distributed operating system for IoT** providing uniform APIs to IoT applications in a similar way as a mobile OS does for the smart phone eco system.



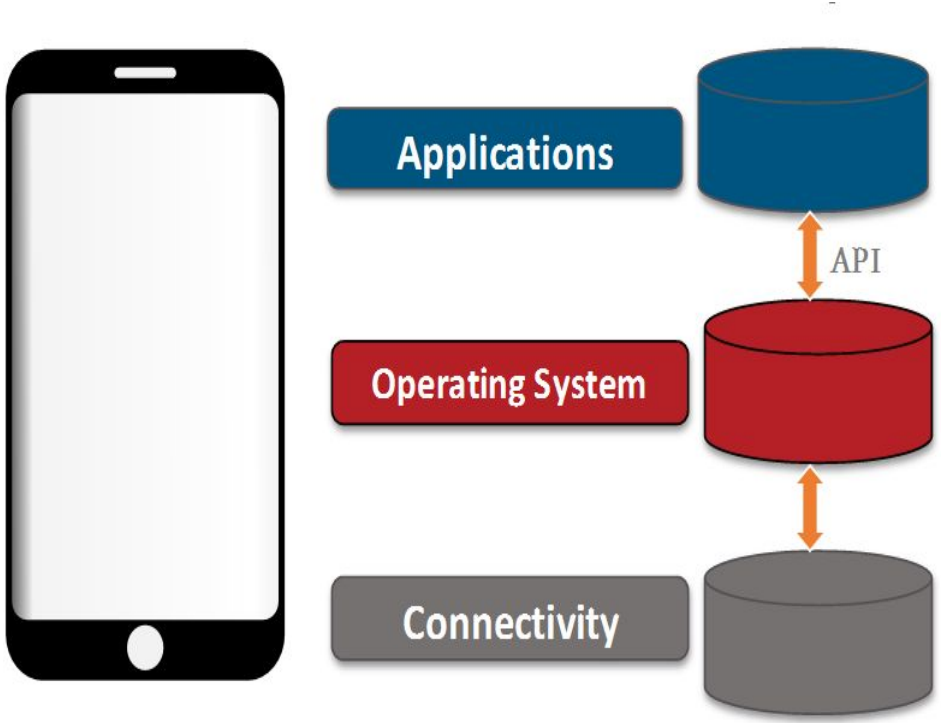
- Applications control connectivity Layer and built-in sensors via API's provided by the Operating System
  => Applications are becoming portable
- Operating System collects data transfer requests from applications. The OS optimizes & controls use the of the network by the device and provides securtiy
- Connectivity Layer provides access to the Internet via the wired and wireless networks

oneM2M

- The architecture standardized by oneM2M defines an **IoT Service Layer**, i.e. a **software** Middleware sitting between processing / communication hardware and IoT applications providing a rich set of functions needed by many IoT applications. It supports among others:
  - secure end-to-end **data/control exchange** between IoT devices and custom applications by providing functions for proper identification
  - authentication, authorization, encryption
  - remote provisioning & activation
  - connectivity setup
  - buffering
  - scheduling
  - synchronization
  - aggregation
  - group communication
  - device management

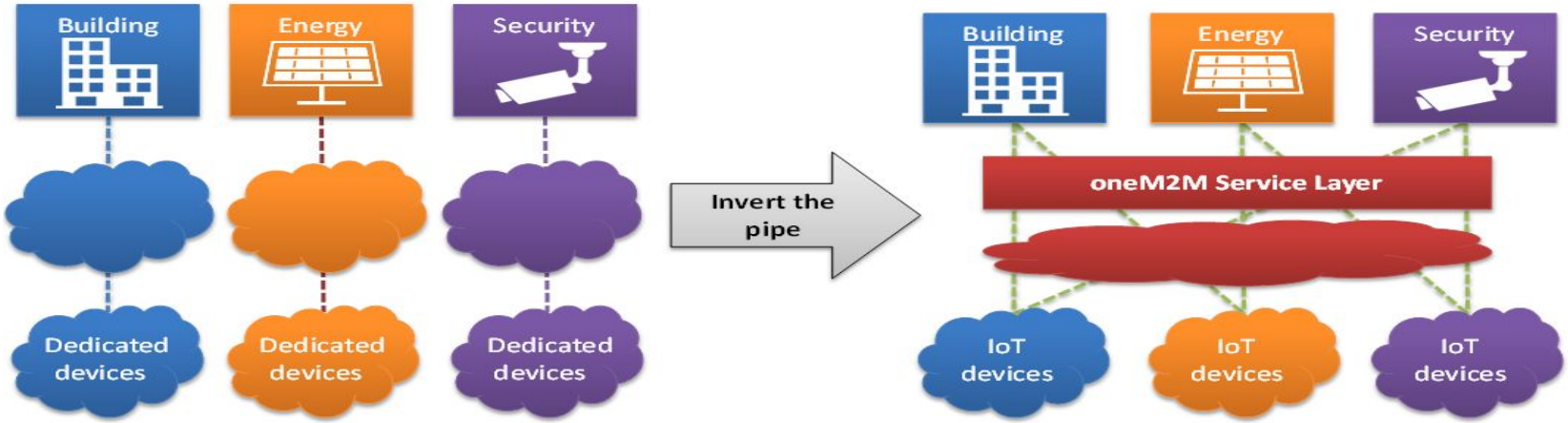
- oneM2M's Service Layer is typically implemented as a software layer and sits between IoT applications and processing or communication hardware and operating system elements that provide **data storage, processing and transport, normally riding on top of IP.**
- However, **non-IP transports** are also supported via interworking proxies. The oneM2M Service Layer provides commonly needed functions for IoT applications across different industry segments.
- **Horizontal architecture**
- oneM2M defines a **horizontal architecture** providing common services functions that enable applications in multiple domains, using a common framework and uniform APIs.
- Using these standardized APIs make it much simpler for M2M/IoT solution providers to cope with complex and heterogeneous connectivity choices by abstracting out the details of using underlying network technologies, underlying transport protocols and data serialization.
- This is all handled by the oneM2M Service Layer without a need for the programmer to become an expert in each of these layers. Therefore, the application developer can focus on the process / business logic of the use case to be implemented and does not need to worry about how exactly the underlying layers work. This is very much like writing a file to a file system without worrying how hard disks and their interfaces actually work.

Therefore, the IoT Service Layer specified in oneM2M can be understood as a **distributed operating system for IoT** providing uniform APIs to IoT applications in a similar way as a mobile OS does for the smart phone eco system.



- Applications control connectivity Layer and built-in sensors via API's provided by the Operating System
  => Applications are becoming portable
- Operating System collects data transfer requests from applications. The OS optimizes & controls use the of the network by the device and provides securtiy
- Connectivity Layer provides access to the Internet via the wired and wireless networks

For example, the following “vertical” domains are isolated silos which makes it difficult to exchange data between each other. Using a “horizontal” architecture allows the providing of a seamless interaction between applications and devices. In the below use case, a security application detects that when nobody is in the building, it triggers the switching off of the light and it stops the air conditioning.



Without oneM2M

- Highly fragmented market with limited vendor-specific applications
- Reinventing the wheel: Same services developed again and again
- Each silo contains its own technologies without interoperability

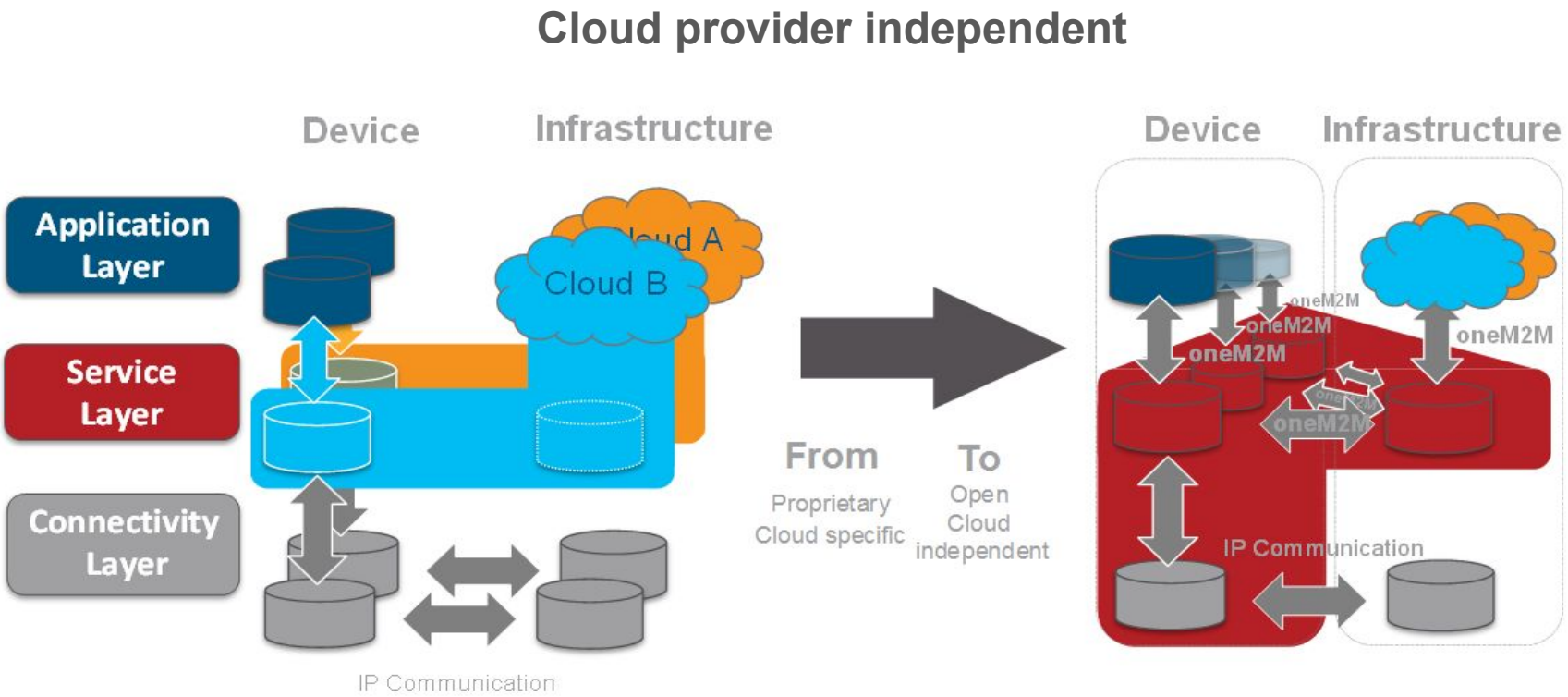
With oneM2M

- End-to-end platform: common service capabilities layer
- Interoperability at the level of data and control exchanges via uniform APIs
- Seamless interaction between heterogeneous applications and devices

IoT Cross-domain interoperability

Cloud provider independent

From Fragmentation to Standards è decoupling device, cloud, and application by open interfaces



Common IoT key problems solved by oneM2M

Application area	• oneM2M provides globally standardized interfaces for the Application developers (device and cloud) • oneM2M enables Application portability
Data Interoperability	• oneM2M provides services towards the Application (Application –Registration & -Discovery, Subscription & Notifications Services, Secure Communication, Device Management etc... • oneM2M enables Device portability (a Device can be connected to any Infrastructure solution)
Connectivity	• oneM2M stores data in case of lack of connectivity • oneM2M can controls the devices usage of connectivity (When, how often communication happens)

oneM2M in a nutshell:

oneM2M's Service Layer corresponds to a **software "framework"**, located between the IoT applications and communication HW/SW that provide storage/connectivity/data transport.

It provides functions that IoT applications across different industry segments commonly need (eg. data transport, security/encryption, scheduling, event notifications, remote software update...)

It supports **cooperative intelligence in distributed applications** i.e. in devices, gateways and back-end/cloud applications

It supports a selection of underlying transport protocols and serialization formats

It is like an **Operating System for the Internet of Things**, sitting on field devices/sensors, gateways and in servers.

And, it is a standard – not controlled by a single private company.

Functional Architecture description

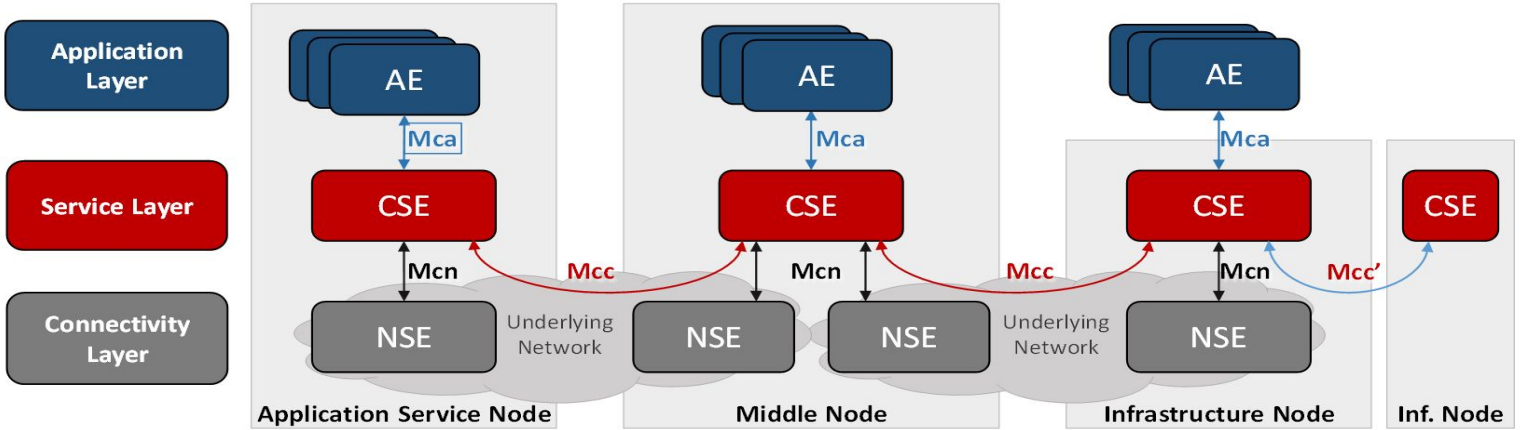
oneM2M Layered Model comprises three layers:

the Application Layer,

the Common Services Layer

the underlying Network Services Layer.

oneM2M Layered Model



oneM2M entities:

The oneM2M functional architecture comprises the following functions:

Application Entity (AE): The Application Entity is an entity in the application layer that implements an M2M application service logic. Each application service logic can be resident in a number of M2M nodes and/or more than once on a single M2M node. Each execution instance of an application service logic is termed an "Application Entity" (AE) and is identified with a unique AE-ID.

Examples of the AEs include an instance of a fleet tracking application, a remote blood sugar measuring application, a power metering application or a pump controlling application.

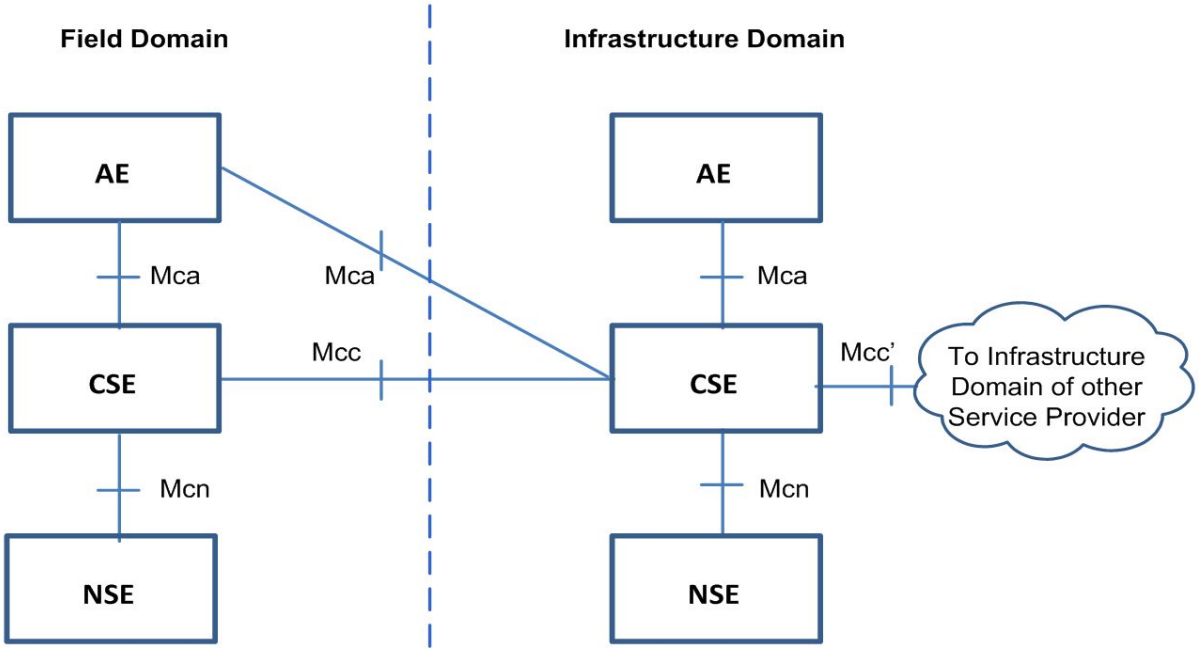
Common Services Entity (CSE): A Common Services Entity represents an instantiation of a set of "common service functions" of the oneM2M Service Layer. A CSE is actually the entity that contains the collection of oneM2M-specified common service functions that AEs are able to use. Such service functions are exposed to other entities through the Mca (exposure to AEs) and Mcc (exposure to other CSEs) reference points. Reference point Mcn is used for accessing services provided by the underlying Network Service Entities such as waking up a sleeping device. Each CSE is identified with a unique CSE-ID.

Examples of service functions offered by the CSE include: data storage & sharing with access control and authorization, event detection and notification, group communication, scheduling of data exchanges, device management, and location services.

Underlying **Network Services Entity (NSE):** A Network Services Entity provides services from the underlying network to the CSEs.

Examples of such services include location services, device triggering, certain sleep modes like PSM in 3GPP based networks or long sleep cycles.

oneM2M Reference Points:



oneM2M Functional Architecture

The oneM2M functional architecture defines the following reference points:

Mca: Reference point for the communication flows between an Application Entity (AE) and a Common Services Entity (CSE). These flows enable the AE to use the services supported by the CSE, and for the CSE to communicate with the AE. The AE and the CSE may or may not be co-located within the same physical entity.

Mcc: Reference point for the communication flows between two Common Services Entities (CSEs). These flows enable a CSE to use the services supported by another CSE.

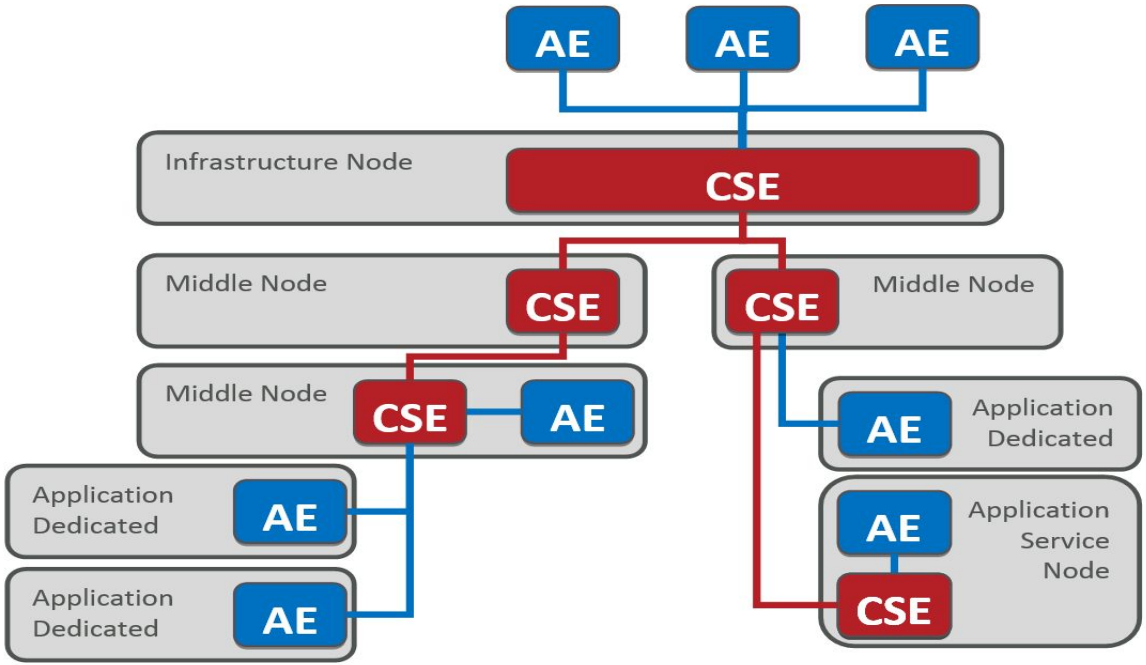
Mcn: Reference point for the communication flows between a Common Services Entity (CSE) and the Network Services Entity (NSE). These flows enable a CSE to use the supported services provided by the NSE. While the oneM2M Service Layer is, usually independent of the underlying network – as long as it supports IP transport – it leverages specific M2M/IoT optimization such as 3GPP's eMTC features (e.g. device triggering, power saving mode, long sleep cycles, etc).

Mcc': Reference point for the communication flows between two Common Services Entities (CSEs) in Infrastructure Nodes (IN) that are oneM2M compliant and that reside in different M2M Service Provider domains.

Additional reference points are defined in oneM2M for specific purposes such as enrolment functions etc. and are not detailed in this overview.

oneM2M Nodes:

oneM2M has defined a set of **Nodes** that are logical entities identifiable in the oneM2M System. oneM2M Nodes typically contain CSEs and/or AEs. For the definition of Node types, oneM2M distinguishes between Nodes in the "Field Domain" – i.e. the domain in which sensors / actors / aggregators / gateways are deployed – and the "Infrastructure Domain" – i.e. the domain in which servers and applications on larger computers reside.



Nodes can be of the following types:

Application Dedicated Node (ADN): a Node that contains at least one AE and does not contain a CSE. It is located in the Field Domain. An ADN would typically be implemented on a resource constraint device that may not have access to rich storage or processing resources and – therefore – may be limited to only host a oneM2M AE and not a CSE. Examples for devices that would be represented by ADNs: simple sensor or actor devices.

Application Service Node (ASN): a Node that contains one CSE and contains at least one Application Entity (AE), located in the Field Domain. An ASN could be implemented on a range of different devices ranging from resource constraint devices up to much richer HW. Examples for devices that would be represented by ASNs: data collection devices, more capable sensors and actors including simple server functions.

Middle Node (MN): a Node that contains one CSE and could contain AEs. MNs are located in the Field Domain. There could be several MNs in the Field Domain of the oneM2M System. Typically an MN would reside in an M2M Gateway. MNs would be used to establish a logical tree structure of oneM2M nodes, e.g. to hierarchically aggregate data of buildings / neighborhoods / cities / counties / states etc.

Infrastructure Node (IN): a Node that contains one CSE and could contain AEs. There is exactly one IN in the Infrastructure Domain per oneM2M Service Provider. An example of physical mapping, an IN could reside in an M2M Service Enablement Infrastructure.

Non-oneM2M Node (NoDN): This Node type is not shown in the figure above. oneM2M specifications also define a Node Type for non-oneM2M Nodes which are Nodes that do not contain oneM2M Entities (neither

ETSI M2M

- ETSI Machine-to-Machine communications is an application diagnostic standard containing an overall end to end M2M functional architecture, identifying the functional entities and the related reference points.

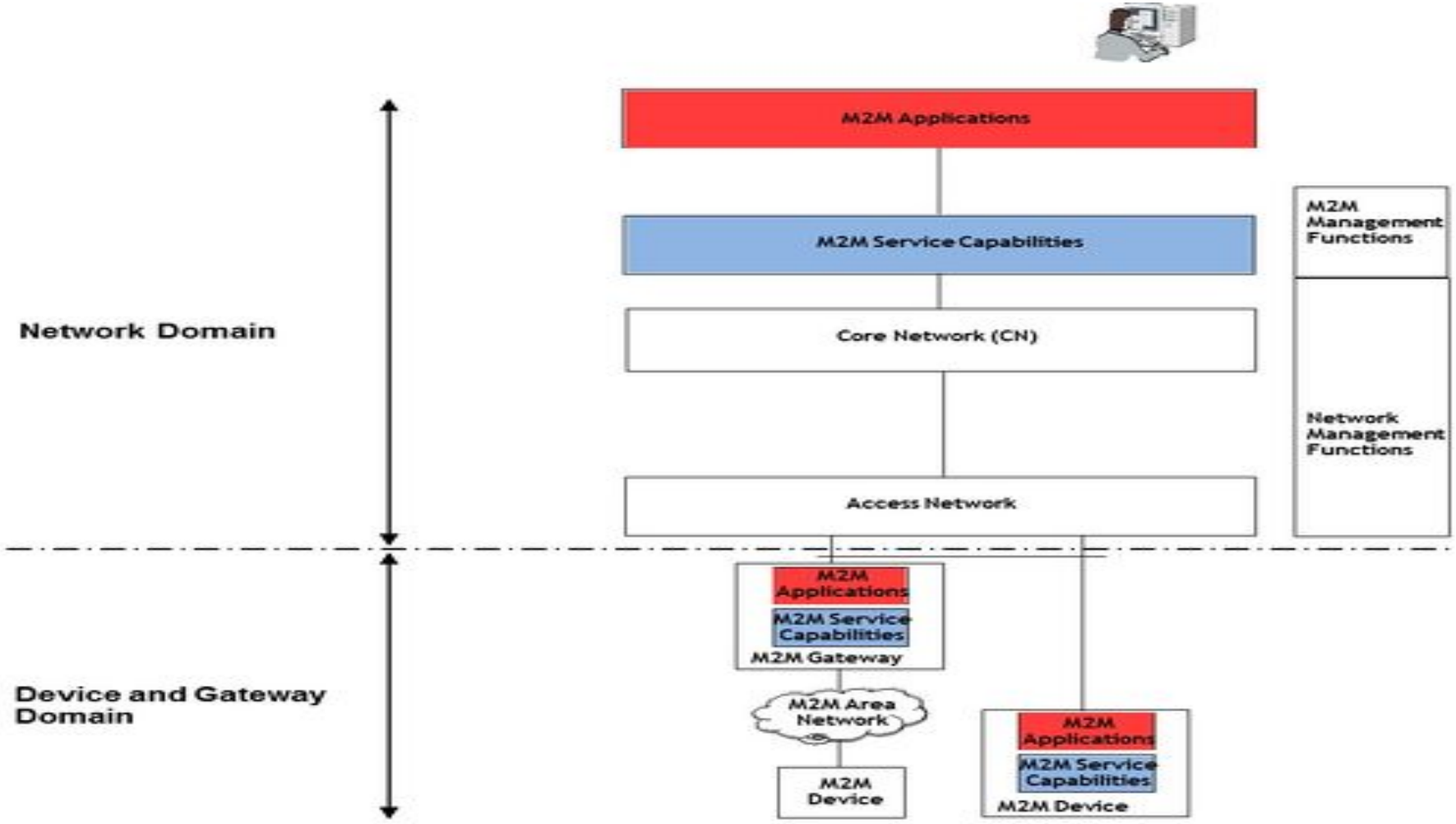
- It describes a resources-based architecture that can be used for the **exchange of data and events** between machines involving communications across networks without requiring human intervention.

- The ETSI M2M High Level Architecture picture (see figure) 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.

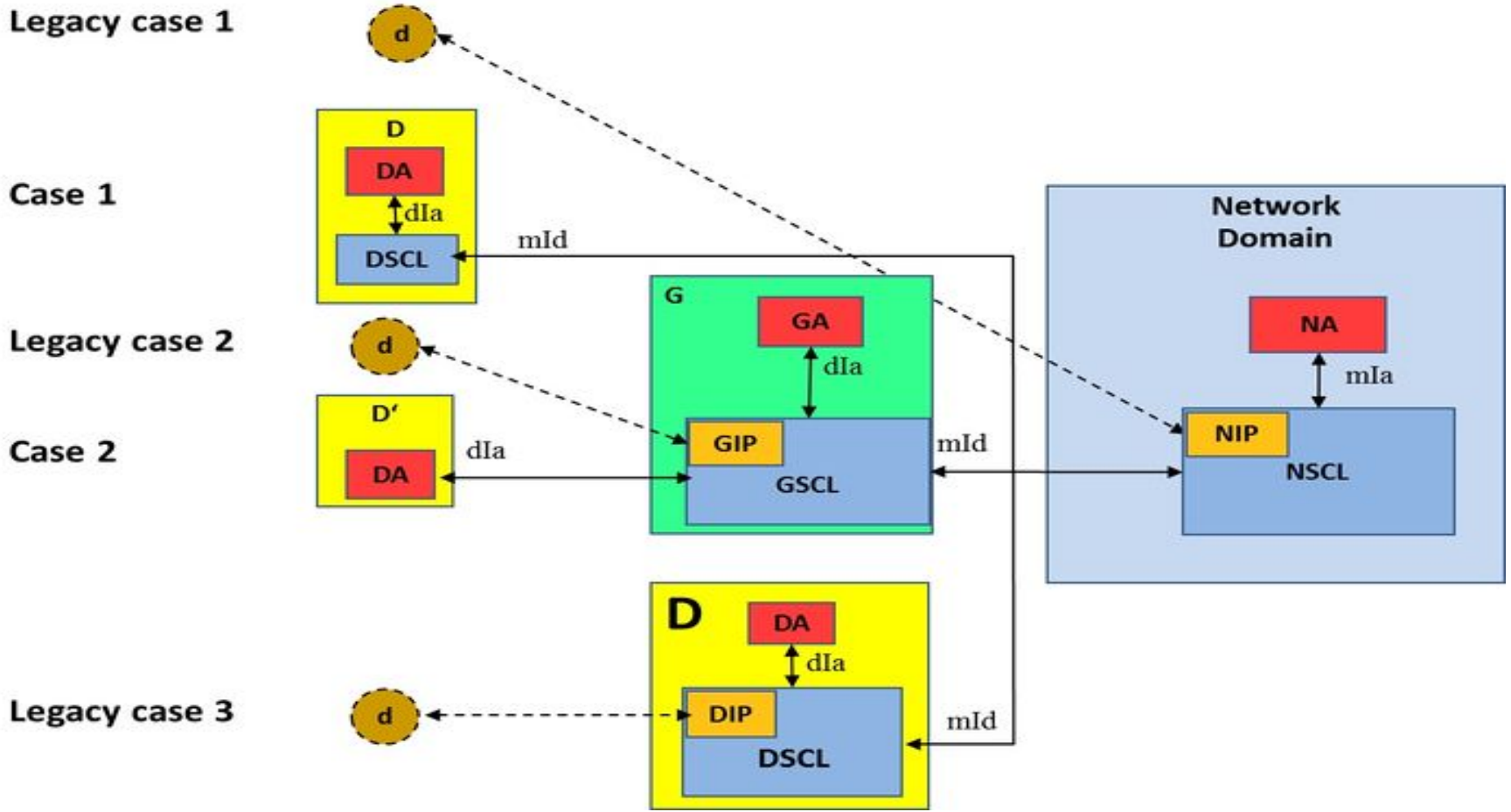
- In essence the goal of this architecture is "to provide the functionalities for the management of interactions between entities (i.e. applications) involving communication across networks without requiring human intervention" as it is described in the ETSI M2M definition of M2M.

ETSI M2M High Level Architecture



- 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 (see figure) 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
- (see the ETSI M2M specifications for a detailed description of these Reference Points)

ETSI M2M Reference Points Mapping



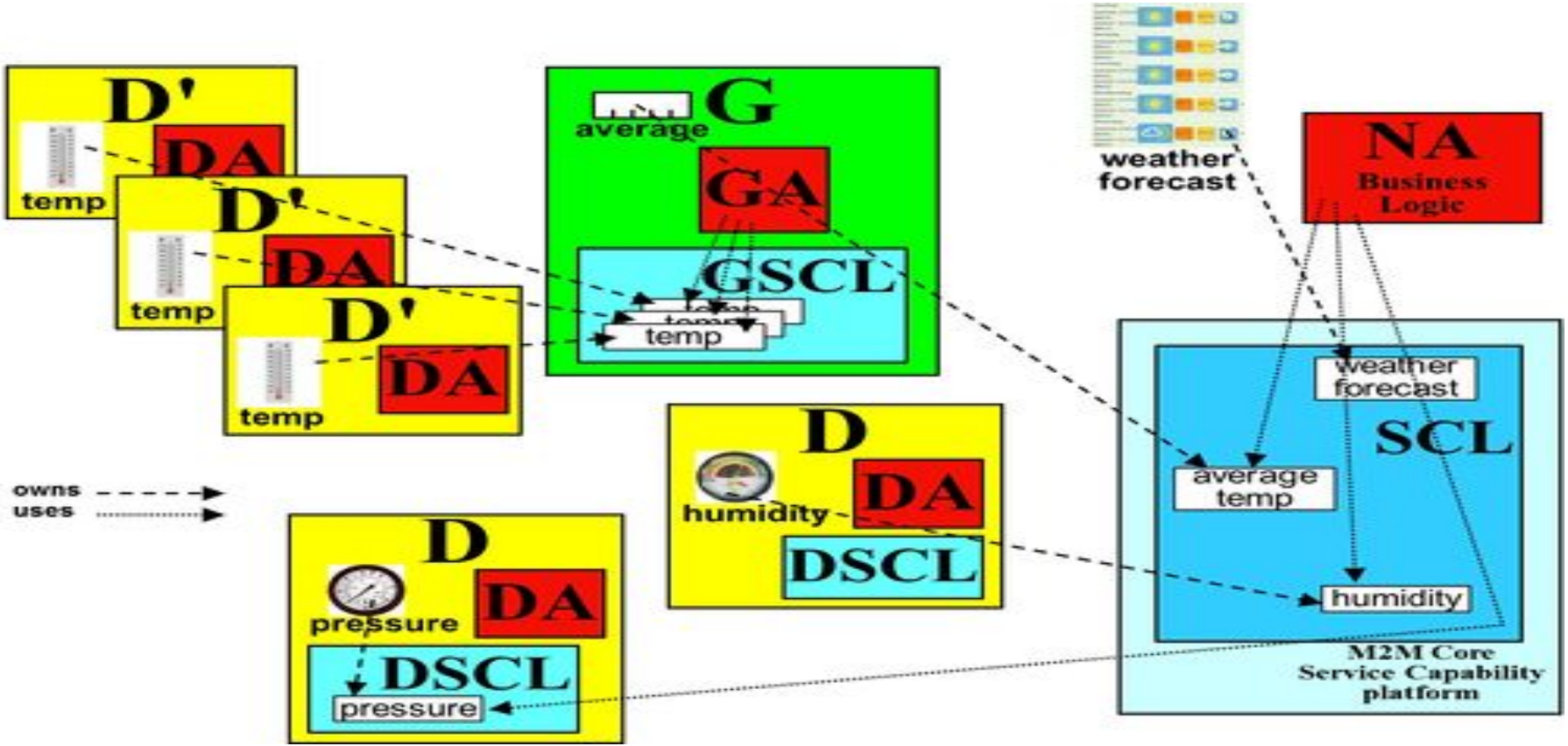
- The main operations that can be performed using these interfaces deal with the concept of M2M Resource. A "resource" can be roughly described as a **shared memory area** that can be used for exchanging data among applications.

- 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 (see figure), 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).

ETSI M2M Resources Usage Example

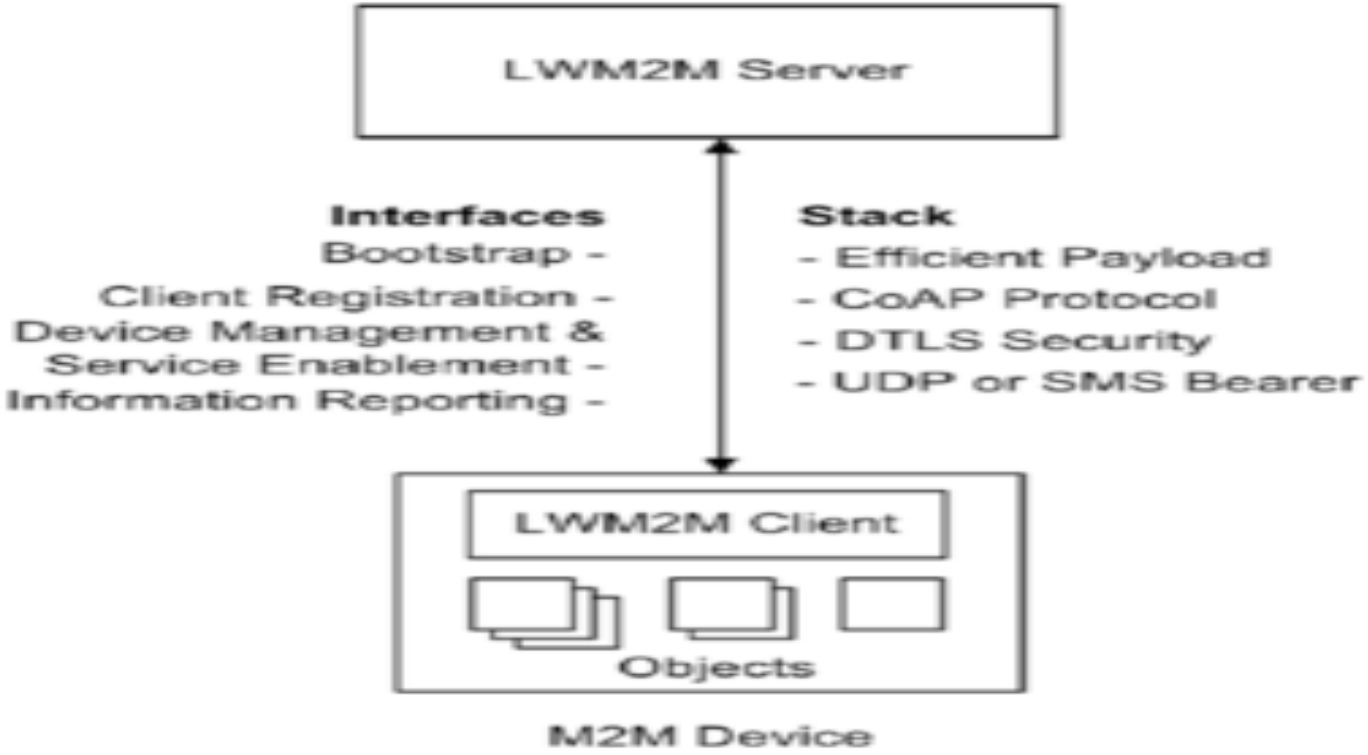


OMA

- **The Open Mobile Association** (OMA) have published a new industry open standard, OMA Lightweight M2M (LWM2M), for 'constrained' M2M & IoT devices. By 'constrained' we mean those devices (like sensor platforms) that are limited by **network bandwidth, computing power and memory, those depending on a limited battery lifetime and those viable only at very low production costs.**
- OMA LWM2M is a good fit for 'Over-The-Air' (OTA) management of IoT wireless sensor platforms because it utilises a simpler protocol and COAP over UDP with a very small overhead (10s of bytes) and the device client is a lot smaller.
- OMA LWM2M defines the application layer communication protocol between a server and a client. The LWM2M Server is typically located in a private or public data centre and can be hosted by the **M2M Service Provider, Network Service Provider or Application Service Provider**. The LWM2M Client resides on the device and is typically integrated as a software library or a built-in function of a module or device.

The LWM2M architecture is illustrated below

LWM2M Architecture



- Four logical interfaces are defined between server and client namely:

  Bootstrap

  Device Discovery and Registration

  Device Management and Service Enablement

  Information Reporting

- The LWM2M protocol stack utilizes the IETF Constrained Application Protocol (CoAP) as the underlying transfer protocol over UDP and SMS bearers. CoAP defines the message header, request/response codes, message options, and retransmission mechanisms. The CoAP protocol was defined by the IETF Constrained RESTful Environment (CoRE) working group.

- CoAP creates an alternative to HTTP for RESTful APIs on resource-constrained devices and supports the basic methods of GET, POST, PUT, DELETE (as with HTTP), which are easily mapped to those of HTTP. Unlike HTTP, CoAP messages are exchanged asynchronously between CoAP end-points over a datagram-oriented transport such as UDP.

- A subset of response codes is supported for LWM2M response message mapping. Built-in resource discovery is supported using the CoRE Link Format standard. CoAP messages are encoded in a simple binary format, allowing this functionality starting with just a 4-byte overhead.

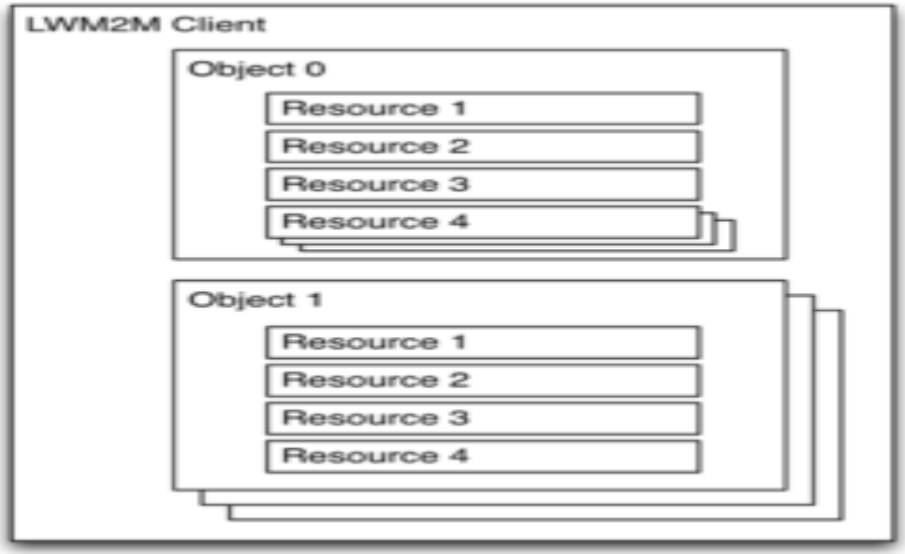
- LWM2M defines **the UDP Binding with CoAP as mandatory whereas the SMS Binding with CoAP is optional.** This means, that LWM2M client-server interaction can happen both via SMS and UDP

- LWM2M includes security to secure communications between client and server using Datagram Transport Layer Security (DTLS). DTLS is used to provide a secure channel between the LWM2M Server and the LWM2M Client for all the messages interchanged.

- DTLS security modes include both pre-shared key and public key technology to support both **very limited and more capable embedded devices**. The LWM2M standard defines provisioning and bootstrapping functionality that allows a LWM2M Bootstrap Server to manage the keying, access control and configuration of a device to enrol with a LWM2M Server.

- The security identifiers, endpoint identifiers and keys are used uniformly throughout the LWM2M system to provide a complete security lifecycle solution.

The LWM2M defines a **simple resource model where each piece of information made available by the LWM2M Client is a Resource**. Resources are logically organized into Objects. The Figure below illustrates this structure, and the relationship between Resources, Objects, and the LWM2M Client. The LWM2M Client may have any number of Resources, each of which belongs to an Object. Resources and Objects have the capability to have multiple instances of the Resource or Object.



LWM2M Object Resource Model

BBF- **Broadband Forum User Services Platform (USP) Data Models**

- The Broadband Forum defines **several data models for use with the User Services Platform** (USP) (TR-369). These data models contain **objects, parameters, commands, and events** that describe the many different service elements that can be exposed via USP Agents.
- USP data models are divided into two types: Root and Service. The root data model, Device, is used to describe the major functions of **network aware devices**, including interfaces, software/firmware, diagnostics, components common to USP and other services, and the basic Agent information necessary to the operation of USP.
- Service data models describe modular functionality that allow the extension of the root data model on a device (under Device.Services.) to **provide particular services**, such as a voice service, set top box service, network attached storage, smart home objects, etc.

- Each data model is defined by a **Name:Version syntax**. The objects, parameters, commands, and events that a particular Agent supports from its implementation of the data model define what is reported to Controllers via the GetSupportedDM message.

How do I use these?

Use the USP data model files to define and implement what exists in your solution's Supported Data Model. This will help Controllers learn what your solution is capable of. The data models will also describe how your solution's Instantiated Data Model will behave during operation.

View the HTML files on this page for a human-readable look at the data model documentation.

Use the XML files in this repository when generating code and performing data validation.

These data models are based on the Broadband Forum's data models for the CPE WAN Management Protocol, also known as "TR-069", with a robust development history.

The source files used to build the complete USP data model can be found on GitHub.

Understanding Security and Interoperability

Understanding the risks

There are many solutions and products marketed today under the label **IoT** that lack basic security architectures.

It is very easy for a knowledgeable person to take control of devices for malicious purposes. Not only devices at home are at risk, but cars, trains, airports, stores, ships, logistics applications, building automation, utility metering applications, industrial automation applications, health services, and so on, are also at risk because of the lack of security measures in their underlying architecture.

It has gone so far that many western countries have identified the lack of security measures in automation applications as a risk to national security, and rightly so. It is just a matter of time before somebody is literally killed as a result of an attack by a hacker on some vulnerable equipment connected to the Internet.

And what are the economic consequences for a company that rolls out a product for use on the Internet that results into something that is vulnerable to well-known attacks?

Reinventing the wheel, but an inverted one

- Engineers working on **machine-to-machine** (**M2M**) communication paradigms, such as industrial automation, might have considered the problem solved when they discovered that machines could talk to each other over the Internet, that is, when the message-exchanging problem was solved.
- This is simply relabelling their previous **M2M** solutions as **IoT** solutions because the transport now occurs over the IP protocol. But, in the realm of the Internet, this is when the problems start. Transport is just one of the many problems that need to be solved.
- The third reason is that when engineers actually re-use solutions and previous experience, they don't really ft well in many cases. The old communication patterns designed for web applications on the Internet are not applicable for **IoT**.
- So, even if the wheel in many cases is reinvented, it's not the same wheel. In previous paradigms, publishers are a relatively few number of centralized high-value entities that reside on the Internet.
- On the other hand, consumers are many but distributed low-value entities, safely situated behind firewalls and well protected by antivirus software and operating systems that automatically update themselves.

Knowing your neighbor

When you decide to move into a new neighborhood, it might be a good idea to know your neighbors frst. It's the same when you move a **M2M** application to **IoT**. As soon as you connect the cable, you have billions of neighbors around the world, all with access to your device.

What kind of neighbors are they? Even though there are a lot of nice and ignorant neighbors on the Internet, you also have a lot of criminals, con artists, perverts, hackers, trolls, drug dealers, drug addicts, rapists, pedophiles, burglars, politicians, corrupt police, curious government agencies, murderers, demented people, agents from hostile countries, disgruntled ex-employees, adolescents with a strange sense of humor, and so on. Would you like such people to have access to your things or access to the things that belong to your children?

If the answer is no (as it should be), then you must take security into account from the start of any development project you do, aimed at IoT. Remember that the Internet is the foulest cesspit there is on this planet. When you move from the M2M way of thinking to IoT, you move from a nice and security gated community to the roughest neighborhood in the world.

Would you go unprotected or unprepared into such an area? IoT is not the same as M2M communication in a secure and controlled network. For an application to work, it needs to work for some time, not just in the laboratory or just after installation, hoping that nobody fnds out about the system. It is not suffcient to just get machines to talk with each other over the Internet.

Modes of attack

- DoS
- Getting Access
- Guess
- Man in Middle
- Sniff
- Post Scan
- Web Crawl
- Search Features
- Wild Cards
- Breaking Cipher

Denial of Service

- A **Denial of Service** (**DoS**) or **Distributed Denial of Service** (**DDoS**) attack is normally used to make a service on the Internet crash or become unresponsive, and in some cases, behave in a way that it can be exploited. The attack consists in making repetitive requests to a server until its resources gets exhausted.

- In a distributed version, the requests are made by many clients at the same time, which obviously increases the load on the target. It is often used for blackmailing or political purposes

- However, as the attack gets more effective and difficult to defend against when the attack is distributed and the target centralized, the attack gets less effective if the solution itself is distributed.

- To guard against this form of attack, you need to build decentralized solutions where possible. In decentralized solutions, each target's worth is less, making it less interesting to attack.

Guessing the credentials

- One way to get access to a system is to impersonate a client in the system by trying to guess the client's credentials.

- To make this type of attack less effective, make sure each client and each device has a long and unique, perhaps randomly generated, set of credentials. Never use pre set user credentials that are the same for many clients or devices or **factory default credentials** that are easy to reset.

- Furthermore, set a **limit to the number of authentication attempts per time unit permitted by the system**; also, log an event whenever this limit is reached, from where to which credentials were used. This makes it possible for operators to detect systematic attempts to enter the system.

Getting access to stored credentials

- One common way to illicitly enter a system is **when user credentials are found somewhere else and reused**. Often, people reuse credentials in different systems.
- There are various ways to avoid this risk from happening. One is to make sure that credentials are not reused in different devices or across different services and applications.
- Another is to randomize credentials, lessening the desire to reuse memorized credentials. A third way is to never store actual credentials centrally, even encrypted if possible, and instead store hashed values of these credentials.
- This is often possible since authentication methods use hash values of credentials in their computations.
- Furthermore, these hashes should be unique to the current installation. Even though some hashing functions are vulnerable in such a way that a new string can be found that generates the same hash value, the probability that this string is equal to the original credentials is miniscule.
- And if the hash is computed uniquely for each installation, the probability that this string can be reused somewhere else is even more remote.

Man in the middle

- Another way to gain access to a system is to try and impersonate a server component in a system instead of a client. This is often referred to as a **Man in the middle** (**MITM**) attack.
- The reason for the middle part is that the attacker often does not know how to act in the server and simply forwards the messages between the real client and the server. In this process, the attacker gains access to confidential information within the messages, such as client credentials, even if the communication is encrypted.
- The attacker might even try to modify messages for their own purposes. To avoid this type of attack, it's important for all clients (not just a few) to always validate the identity of the server it connects to. If it is a high-value entity, it is often identified using a certificate.
- This certificate can both be used to verify the domain of the server and encrypt the communication. Make sure this validation is performed correctly, and do not accept a connection that is invalid or where the certificate has been revoked, is self-signed, or has expired.
- Another thing to remember is to never use an unsecure authentication method when the client authenticates itself with the server. If a server has been compromised, it might try to fool clients into using a less secure authentication method when they connect. By doing so, they can extract the client credentials and reuse them somewhere else. By using a secure authentication method, the server, even if compromised, will not be able to replay the authentication again or use it somewhere else. The communication is valid only once.

Sniffing network communication

- If communication is not encrypted, everybody with access to the communication stream can read the messages using simple **sniffing applications**, such as Wireshark.

- If the communication is point-to-point, this means the communication can be heard by any application on the sending machine, the receiving machine, or any of the bridges or routers in between.

- If a simple hub is used instead of a switch somewhere, everybody on that network will also be able to **eavesdrop.**

- If the communication is performed using multicast messaging service, as can be done in UPnP and CoAP, anybody within the range of the **Time to live** (**TTL**) parameter (maximum number of router hops) can eavesdrop.

- Remember to always use encryption if sensitive data is communicated. If data is private, encryption should still be used, even if the data might not be sensitive at first glance.

Sniffing network communication

- A burglar can know if you're at home by simply monitoring temperature sensors, water flow meters, electricity meters, or light switches at your home. Small variations in temperature alert to the presence of human beings.

- Change in the consumption of electrical energy shows whether somebody is cooking food or watching television. The flow of water shows whether somebody is drinking water, flushing a toilet, or taking a shower.

- No flow of water or a relatively regular consumption of electrical energy tells the burglar that nobody is at home. Light switches can also be used to detect presence, even though there are applications today that simulate somebody being home by switching the lights on and off.

- Note: If you haven't done so already, make sure to download a sniffer to get a feel of what you can and cannot see by sniffing the network traffic. Wireshark can be downloaded from https://www.wireshark.org/download.html.

Port scanning and web crawling

- Port scanning is a method where you systematically test a **range of ports across a range of IP addresses** to see which ports are open and serviced by applications. This method can be combined with different tests to see the applications that might be behind these ports.
- If HTTP servers are found, standard page names and web-crawling techniques can be used to try to figure out which web resources lie behind each HTTP server.
- CoAP is even simpler since devices often publish well-known resources. Using such simple brute-force methods, it is relatively easy to find (and later exploit) anything available on the Internet that is not secured.
- To avoid any private resources being published unknowingly, make sure to close all the incoming ports in any firewalls you use. Don't use protocols that require incoming connections.
- Instead, use protocols that create the connections from inside the firewall. Any resources published on the Internet should be authenticated so that any automatic attempt to get access to them fails.
- Always remember that information that might seem trivial to an individual might be very interesting if collected en masse. This information might be coveted not only by teenage pranksters but by public relations and marketing agencies, burglars, and government agencies (some would say this is a repetition).

Search features and wildcards

- Don't make the mistake of thinking it's difficult to find the identities of devices published on the Internet. Often, it's the reverse.
- For devices that use **multicast communication**, such as those using UPnP and CoAP, anybody can listen in and see who sends the messages.
- For devices that use **single-cast communication**, such as those using HTTP or CoAP, port-scanning techniques can be used.
- For devices that are **protected by firewalls** and use message brokers to protect against incoming attacks, such as those that use XMPP and MQTT, **search features or wildcards** can be used to find the identities of devices managed by the broker, and in the case of MQTT, even what they communicate.
- You should always assume that the identity of all devices can be found, and that there's an interest in exploiting the device.
- For this reason, it's very important that each device authenticates any requests made to it if possible. Some protocols help you more with this than others, while others make such authentication impossible.

- XMPP only permits messages from accepted friends. The only thing the device needs to worry about is which friend requests to accept. This can be either configured by somebody else with access to the account or by using a provisioning server if the device cannot make such decisions by itself.

- The device does not need to worry about client authentication, as this is done by the brokers themselves, and the XMPP brokers always propagate the authenticated identities of everybody who send them messages.

- MQTT, on the other hand, resides in the other side of the spectrum. Here, devices cannot make any decision about who sees the published data or who makes a request since identities are stripped away by the protocol.

- The only way to control who gets access to the data is by building a proprietary end-to-end encryption layer on top of the MQTT protocol, thereby limiting interoperability.

- In between the two resides protocols such as HTTP and CoAP that support some level of local client authentication but lacks a good distributed identity and authentication mechanism.

- This is vital for **IoT** even though this problem can be partially solved in local intranets.

Breaking ciphers

- Many believe that by using encryption, data is secure. This is not the case, as discussed previously, since the encryption is often only done between connected parties and not between end users of data (the so-called end-to-end encryption).

- At most, such encryption safeguards from eavesdropping to some extent. But even such encryption can be broken, partially or wholly, with some effort.

- Ciphers can be broken using known vulnerabilities in code where attackers exploit program implementations rather than the underlying algorithm of the cipher. This has been the method used in the latest spectacular breaches in code based on the OpenSSL library.

- To protect yourselves from such attacks, you need to be able to update code in devices remotely, which is not always possible.

- Other methods use irregularities in how the cipher works to figure out, partly or wholly, what is being communicated over the encrypted channel. This sometimes requires a considerable amount of effort.

Breaking ciphers

- To safeguard against such attacks, it's important to realize that an attacker does not spend more effort into an attack than what is expected to be gained by the attack.
- By storing massive amounts of sensitive data centrally or controlling massive amounts of devices from one point, you increase the value of the target, increasing the interest of attacking it.
- On the other hand, by decentralizing storage and control logic, the interest in attacking a single target decreases since the value of each entity is comparatively lower.
- Decentralized architecture is an important tool to both mitigate the effects of attacks and decrease the interest in attacking a target.
- However, by increasing the number of participants, the number of actual attacks can increase, but the effort that can be invested behind each attack when there are many targets also decreases, making it easier to defend each one of the attacks using standard techniques.

Tools for achieving security

- There are a number of tools that architects and developers can use to **protect against malicious** use of the system. An exhaustive discussion would fill a smaller library. Here, we will mention just a few techniques and how they not only affect security but also interoperability.

The tools for achieving security are:

- VPN
- X.509
- Authentication
- User names and Passwords
- Message Brokers
- Provisioning servers
- Centralization versus decentralization

Virtual Private Networks

- A method that is often used to protect unsecured solutions on the Internet is to protect them using **Virtual Private Networks** (**VPNs**). Often, traditional **M2M** solutions working well in local intranets need to expand across the Internet.

- One way to achieve this is to create such **VPNs** that allow the devices to believe they are in a local intranet, even though communication is transported across the Internet.

- Even though transport is done over the Internet, it's difficult to see this as a true **IoT** application. It's rather a **M2M** solution using the Internet as the mode of transport.

- Because telephone operators use the Internet to transport long distance calls, it doesn't make it **Voice over IP** (**VoIP**). Using **VPNs** might protect the solution, but it completely eliminates the possibility to interoperate with others on the Internet, something that is seen as the biggest advantage of using the **IoT** technology.

X.509 certificates and encryption

- We've mentioned the use of certificates to validate the identity of high-value entities on the Internet.
- Certificates allow you to validate not only the identity, but also to check whether the certificate has been revoked or any of the issuers of the certificate have had their certificates revoked, which might be the case if a certificate has been compromised.
- Certificates also provide a **Public Key Infrastructure** (**PKI**) architecture that handles encryption. Each certificate has a public and private part.
- **The public part** of the certificate can be freely distributed and is used to encrypt data, whereas only the holder of the **private part** of the certificate can decrypt the data. Using certificates incurs a cost in the production or installation of a device or item.
- They also have a limited life span, so they need to be given either a long lifespan or updated remotely during the life span of the device.
- Certificates also require a scalable infrastructure for validating them. For these reasons, it's difficult to see that certificates will be used by other than high-value entities that are easy to administer in a network. It's difficult to see a cost-effective, yet secure and meaningful, implementation of validating certificates in low-value devices such as lamps, temperature sensors, and so on, even though it's theoretically possible to do so.

Authentication of identities

- Authentication is the process of validating whether the identity provided is actually correct or not. Authenticating a server might be as simple as validating a **domain certificate** provided by the server, making sure it has not been revoked and that it corresponds to the domain name used to connect to the server.
- Authenticating a client might be more involved, as it has to authenticate the credentials provided by the client. Normally, this can be done in many different ways.
- It is vital for developers and architects to understand the available authentication methods and how they work to be able to assess the level of security used by the systems they develop.
- Some protocols, such as HTTP and XMPP, use the standardized **Simple Authentication and Security Layer** (**SASL**) to publish an extensible set of authentication methods that the client can choose from. This is good since it allows for new authentication methods to be added.
- But it also provides a weakness: clients can be tricked into choosing an unsecure authentication mechanism, thus unwittingly revealing their user credentials to an impostor.
- Make sure clients do not use unsecured or obsolete methods, such as PLAIN, BASIC, MD5-CRAM, MD5-DIGEST, and so on, even if they are the only options available.

Authentication of identities

- Instead, use secure methods such as SCRAM-SHA-1 or SCRAM-SHA-1-PLUS, or if client certificates are used, EXTERNAL or no method at all. If you're using an unsecured method anyway, make sure to log it to the event log as a warning, making it possible to detect impostors or at least warn operators that unsecure methods are being used.
- Other protocols do not use secure authentication at all. MQTT, for instance, sends user credentials in clear text (corresponding to PLAIN), making it a requirement to use encryption to hide user credentials from eavesdroppers or client-side certificates or pre-shared keys for authentication.
- Other protocols do not have a standardized way of performing authentication. In CoAP, for instance, such authentication is built on top of the protocol as security options.
- The lack of such options in the standard affects interoperability negatively.

Usernames and passwords

- A common method to provide user credentials during authentication is by providing a simple username and password to the server. This is a very human concept.
- Some solutions use the concept of a **pre-shared key** (**PSK**) instead, as it is more applicable to machines, conceptually at least.
- If you're using usernames and passwords, do not reuse them between devices, just because it is simple. One way to generate secure, **difficult-to-guess usernames and passwords is to randomly create them**.
- In this way, they correspond more to pre-shared keys.
- One problem in using randomly created user credentials is how to administer them.
- Both the server and the client need to be aware of this information. The identity must also be distributed among the entities that are to communicate with the device.
- In the case of XMPP, this problem has been solved, *The XMPP Protocol*. Here, the device creates its own random identity and creates the corresponding account in the XMPP server in a secure manner.
- There is no need for a common factory default setting

Usernames and passwords

- It then reports its identity to a Thing Registry or provisioning server where the owner can claim it and learn the newly created identity.
- This method never compromises the credentials and does not affect the cost of production negatively.
- Furthermore, passwords should never be stored in clear text if it can be avoided.
- This is especially important on servers where many passwords are stored. Instead, hashes of the passwords should be stored.
- Most modern authentication algorithms support the use of password hashes. Storing hashes minimizes the risk of unwanted generation of original passwords for attempted reuse in other systems.

Using message brokers and provisioning servers

- Using message brokers can greatly enhance security in an **IoT** application and lower the complexity of implementation when it comes to authentication, as long as message brokers provide authenticated identity information in messages it forwards.

- In XMPP, all the federated XMPP servers authenticate clients connected to them as well as the federated servers themselves when they intercommunicate to transport messages between domains.

- This relieves clients from the burden of having to authenticate each entity in trying to communicate with it since they all have been securely authenticated. It's sufficient to manage security on an identity level.

- Even this step can be relieved further by the use of provisioning, as described in *The XMPP Protocol*.

- Unfortunately, not all protocols using message brokers provide this added security since they do not provide information about the sender of packets. MQTT is an example of such a protocol.

Centralization versus decentralization

- Comparing centralized and decentralized architectures is like comparing the process of putting all the eggs in the same basket and distributing them in many much smaller baskets.

- The effect of a breach of security is much smaller in the decentralized case; fewer eggs get smashed when you trip over.

- Even though there are more baskets, which might increase the risk of an attack, the expected gain of an attack is much smaller.

- This limits the motivation of performing a costly attack, which in turn makes it simpler to protect it against.

- When designing IoT architecture, try to consider the following points:
  • **Avoid storing data in a central position if possible**. Only store the data centrally that is actually needed to bind things together.
  • **Distribute logic, data, and workload**. Perform work as far out in the network as possible. This makes the solution more scalable, and it utilizes existing resources better.

Centralization versus decentralization

• **Use linked data to spread data** across the Internet, and use standardized grid computation technologies to assemble distributed data (for example, SPARQL) to avoid the need to store and replicate data centrally.

• **Use a federated set of small local brokers** instead of trying to get all the devices on the same broker. Not all brokered protocols support federation, for example, XMPP supports it but MQTT does not.

• **Let devices talk directly to each other** instead of having a centralized proprietary API to store data or interpret communication between the two.

• **Contemplate the use of cheap small and energy-efficient** microcomputers such as the Raspberry Pi in local installations as an alternative to centralized operation and management from a datacenter.

The need for interoperability

- What has made the Internet great is not a series of isolated services, but the ability to coexist, interchange data, and interact with the users. This is important to keep in mind when developing for **IoT**.
- Avoid the mistakes made by many operators who failed during the first Internet bubble. You cannot take responsibility for everything in a service.
- The new Internet economy is based on the interaction and cooperation between services and its users.
  1. **Solves complexity**
  2. **Reduces cost**
  3. **Allows new kinds of services and reuse of devices**

Solves complexity

- The same must be true with the new **IoT**. Those companies that believe they can control the entire value chain, from things to services, middleware, administration, operation, apps, and so on, will fail, as the companies in the first Internet bubble failed.
- Companies that built devices with proprietary protocols, middleware, and mobile phone applications, where you can control your things, will fail. Why? Imagine a future where you have a thousand different things in your apartment from a hundred manufacturers.
- Would you want to download a hundred smart phone apps to control them? Would you like five different applications just to control your lights at home, just because you have light bulbs from five different manufacturers? An alternative would be to have one app to rule them all.
- There might be a hundred different such apps available (or more), but you can choose which one to use based on your taste and user feedback. And you can change if you want to.
- But for this to be possible, things need to be interoperable, meaning they should communicate using a commonly understood language.

Reduces cost

- Interoperability does not only affect simplicity of installation and management, but also the price of solutions. Consider a factory that uses thousands (or hundreds of thousands) of devices to control and automate all processes within.
- Would you like to be able to buy things cheaply or expensively? Companies that promote proprietary solutions, where you're forced to use their system to control your devices, can force their clients to pay a high price for future devices and maintenance, or the large investment made originally might be lost.
- Will such a solution be able to survive against competitors who sell interoperable solutions where you can buy devices from multiple manufacturers? Interoperability provides competition, and competition drives down cost and increases functionality and quality.
- This might be a reason for a company to work against interoperability, as it threatens its current business model. But the alternative might be worse. A competitor, possibly a new one, might provide such a solution, and when that happens, the business model with proprietary solutions is dead anyway.
- The companies that are quickest in adapting a new paradigm are the ones who would most probably survive a paradigm shift, as the shift from **M2M** to **IoT** undoubtedly is.

Allows new kinds of services and reuse of devices

- There are many things you cannot do unless you have an interoperable communication model from the start. Consider a future smart city. Here, new applications and services will be built that will reuse existing devices, which were installed perhaps as part of other systems and services.
- These applications will deliver new value to the inhabitants of the city without the need of installing new duplicate devices for each service being built.
- But such multiple use of devices is only possible if the devices communicate in an open and interoperable way.
- However, care has to be taken at the same time since installing devices in an open environment requires the communication infrastructure to be secure as well.
- To achieve the goal of building smart cities, it is vitally important to use technologies that allow you to have both a secure communication infrastructure and an interoperable one.

Combining security and interoperability

- As we have seen, there are times where security is contradictory to interoperability. If security is meant to be taken as exclusivity, it opposes the idea of interoperability, which is by its very nature inclusive.
- Depending on the choice of communication infrastructure, you might have to use security measures that directly oppose the idea of an interoperable infrastructure, prohibiting third parties from accessing existing devices in a secure fashion.
- It is important during the architecture design phase, before implementation, to thoroughly investigate what communication technologies are available, and what they provide and what they do not provide. You might think that this is a minor issue, thinking that you can easily build what is missing on top of the chosen infrastructure. This is not true.
- All such implementation is by its very nature proprietary, and therefore not interoperable. This might drastically limit your options in the future, which in turn might drastically reduce anyone else's willingness to use your solution.

Combining security and interoperability

- The more a technology includes, in the form of global identity, authentication, authorization, different communication patterns, common language for interchange of sensor data, control operations and access privileges, provisioning, and so on, the more interoperable the solution becomes.
- If the technology at the same time provides a secure infrastructure, you have the possibility to create a solution that is both secure and interoperable without the need to build proprietary or exclusive solutions on top of it.

Need for Security in IoT Protocols

- IoT is perhaps the most complex and undeveloped area of network security.
- **Communication security**: Secure, trusted, and privacy-protected communication capability is required, so that unauthorized access to the content of data can be prohibited, integrity of data can be guaranteed and privacy-related content of data can be protected during data transmission or transfer in IoT.
- **Data management security**: Secure, trusted, and privacy-protected data management capability is required, so that unauthorized access to the content of data can be prohibited, integrity of data can be guaranteed and privacy-related content of data can be protected when storing or processing data in IoT.
- **Service provision security**: Secure, trusted, and privacy-protected service provision capability is required, so that unauthorized access to service and fraudulent service provision can be prohibited and privacy information related to IoT users can be protected.

Need for Security in IoT Protocols

- **Integration of security policies and techniques:** The ability to integrate different security policies and techniques is required, so as to ensure a consistent security control over the variety of devices and user networks in IoT.

- **Mutual authentication and authorization**: Before a device (or an IoT user) can access the IoT, mutual authentication and authorization between the device (or the IoT user) and IoT is required to be performed according to predefined security policies.

- **Security audit**: Security audit is required to be supported in IoT. Any data access or attempt to access IoT applications are required to be fully transparent, traceable and reproducible according to appropriate regulation and laws. In particular, IoT is required to support security audit for data transmission, storage, processing, and application access.

Security in IoT Protocols

- Security is another aspect of IoT applications which is critical and can be found in all almost all layers of the IoT protocols. Threats exist at all layers including datalink, network, session, and application layers.

- In this section, we briefly discuss the security mechanisms built in the IoT protocols that we have discussed in this survey.

- MAC 802.15.4
- 6LoWPAN
- RPL
- Application Layer

Communication Protocol	*Standards*	*Encryption Protocol*	*Energy Consumption*	*Advantages*	*Disadvantages*
6LoWPAN	IEEE 802.15.4	AES	Low	Low processing	Lack of authentication
RPL	IETF RPL	AES	Low	Low processing	Vulnerability to many attacks
NFC	ISO/IEC 14443	RSA, DSA	Low	Simplicity of deployment	Limited Range
Bluetooth	IEEE 802.16	AES, ECDH	Medium/Very Low (BLE)	Low consumption	Privacy/Identity Tracking
Wi-Fi	IEEE 802.11i/e/g	AES	High	Mobility and efficiency	Limited reachability
Zigbee	IEEE 802.15.4	AES	Low	Low-cost, low-energy devices	one-time transmission of the unprotected key
WiMAX	IEEE 802.16	RSA	Medium	Supports authentication	Limited mobility

MAC 802.15.4

- MAC 802.15.4 offers different security modes by utilizing the “Security Enabled Bit” in the Frame Control field in the header.
- Security requirements include
- Confidentiality
- Authentication
- Integrity
- Access control mechanisms and
- secured Time-Synchronized Communications.
- Refer unit 3 IEEE 802.15.4 in depth security

6LoWPAN

- 6LoWPAN by itself does not offer any mechanisms for security. However, relevant documents include discussion of security threats, requirement and approach to consider in IoT network layer.
- For example, RFC 4944 discusses the possibility of duplicate EUI-64 interface addresses which are supposed to be unique [RFC4944].
- RFC 6282 discusses the security issues that are raised due to the problems introduced in RFC 4944 [RFC6282].
- RFC 6568 addresses possible mechanisms to adopt security within constrained wireless sensor devices. In addition, a few recent drafts in [6Lo] discuss mechanisms to achieve security in 6loWPAN.

The main security needs for 6LoWPAN networks are:

- **Data Confidentiality:** makes the data content non understandable to unauthorized devices or users, Data Authentication: verifies the identity of the data source,

- **Data Integrity**: ensures that the received data is correct, Data Freshness: guarantees that the received data is original and has not been replayed,

- **Network Availability**: ensures that the network services are always available for the legitimate devices or users,

6LoWPAN

- **Network Robustness**: makes the network usable even when an attack occurs, Network Resiliency: maintains a given security level over the network even when a node is compromised,
- **Network Resistance**: is the ability to avoid that an attacker takes the control of the network via a compromise node,
- **Energy Efficiency:** prevents battery drain in the network,
- **Assurance**: is the ability to dispatch information over the network to ensure their security,
- **Device Authorization:** checks the legitimacy of a device and enables it to join the network.

RPL

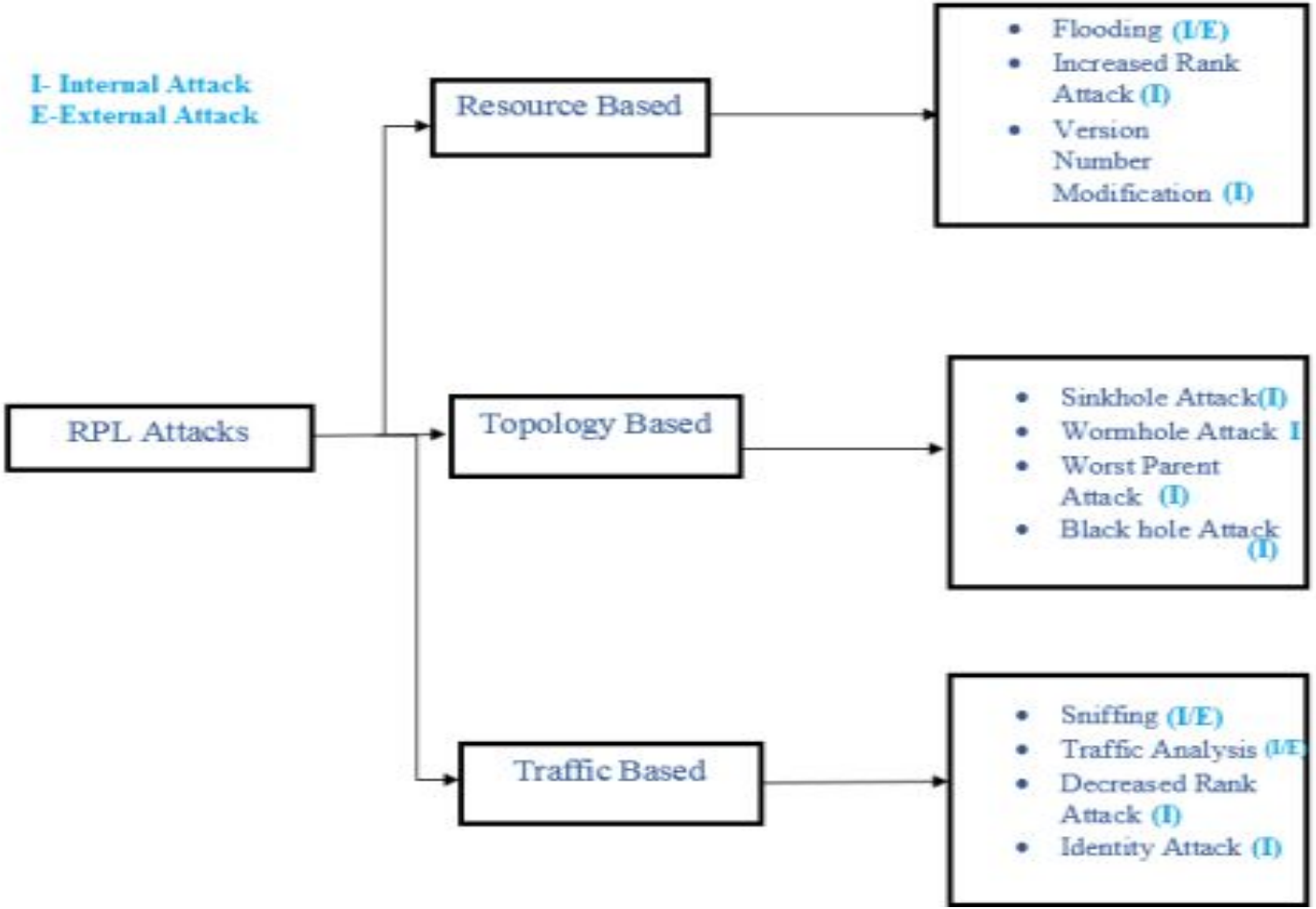
- RPL offers different level of security by utilizing a “Security” field after the 4-byte ICMPv6 message header. Information in this field indicates the level of security and the cryptography algorithm used to encrypt the message.
- RPL offers support for data authenticity, semantic security, protection against replay attacks, confidentiality and key management.
- Levels of security in RPL include Unsecured, Preinstalled, and Authenticated.
- RPL attacks include Selective Forwarding, Sinkhole, Sybil, Hello Flooding, Wormhole, Black hole and Denial of Service attacks.

RPL

- RPL stands for Routing Protocol for Low-Power and Lossy Networks and was defined by IETF. It is a network layer protocol which is mainly designed for the wireless devices with small battery and to provide reliable performance in lossy environments. It uses a distance vector routing protocol for Low Power and Lossy Networks(LLNs) using IPv6. RPL protocol are based on Directed Acyclic Graphs (DAG) and is better than tree because it allows node to select multiple neighbour nodes as its parent. In DAG node ranks is always higher than its parent node.

- RPL has three basic security modes.
- **Unsecured:** It is the default mode in RPL in which control message is send without using any additional security.
- **Preinstalled:** It uses key to have confidentiality, integrity and authentication. So, therefore nodes which have devices with pre-configured symmetric keys only join the RPL instance.
- **Authenticated**: In it, when a node wants to join the network with a pre-installed key than key authority will authenticate and then authorize the devices

RPL


I- Internal Attack
E-External Attack
RPL Attacks
Resource Based
Topology Based
Traffic Based
Flooding (I/E)
Increased Rank Attack (I)
Version Number Modification (I)
Sinkhole Attack(I)
Wormhole Attack I
Worst Parent Attack (I)
Black hole Attack (I)
Sniffing (I/E)
Traffic Analysis (I/E)
Decreased Rank Attack (I)
Identity Attack (I)

- ***Sinkhole Attack:*** In this internal attack, attacker or compromised node advertises beneficial path to attract many nearby nodes to route traffic through it. This attack disrupt the network topology and can become very stronger when combined with another attacks.
- ***Version Number Modification Attack****:* This internal attack is occurred by changing version number(lower to higher) of a DODAG tree. When nodes receive the new higher version number DIO message they start the formation of new DODAG tree. This results unoptimized or inconsistency of network topology, increases control overhead and higher packet loss.
- ***Denial of Service Attack:*** Denial of service or Distributed denial of service attack is attempt to make resources unavailable to its valuable user. In RPL this attack can be done by the IPv6 UDP packet flooding.
- ***Neighbor Attack:*** In this attack, intruder broadcast DIO messages that it received without adding information of himself. The node who receives this messages may conclude that new neighbor node send this DIO message. The victim nodes select the out range neighbors node for routing purpose which affects network quality parameters.
- ***Wormhole Attack****:* This attack can occurred by creating tunnel between the two attackers and transmitting the selective traffic through it which Disrupt the network topology and traffic flow.

- ***Decreased Rank Attacks**:* In a DODAG, rank is used for positions of nodes relative to the sink node. It means lower the rank is close to the root and vice verse. When a malicious node advertises a lower rank value, it means, it wants majority of traffic pass through it. As a result, many legitimate nodes connect to the attacker. An attacker can change its rank value through the falsification of DIO messages. finally, an attacker either drop this messages or selectively forward these messages. It becomes more powerful when combined with other attacks.
- ***Identity Attacks**:* Identity attacks contain clone ID and sybil attacks. In a clone ID attack, an attacker copies the identities of a valid node onto another physical node. In a sybil attack, which is similar to a clone ID attack, an attacker uses several logical entities on the same physical node. These attacks can be used to take control over large parts of a network without deploying physical nodes.
- ***Sniffing Attacks**:* A sniffing attack perform by the listening of the packets transmitted over the network. This attack compromises the confidentiality of communications. It also check the pattern of traffic for major attacks

Application Layer

- Applications can provide additional level of security using TLS or SSL as a transport layer protocol. In addition, end to end authentication and encryption algorithms can be used to handle different levels of security as required.
- For further discussion on security, It should be noted that a number of new security approaches are also being developed that are suitable for resource constrained IoT devices. Some of these protocols are listed in Figure 2.

Layer	Protocols
Application Layer	CoAP, MQTT, XMPP, AMQP, RESTFUL, Websockets
Transport Layer	UDP, DTLS
Internet Layer	RPL, 6LoWPAN
Physical/Link Layer	IEEE 802.15 Series, IEEE 802 11 series

Summary of application layer protocol with security

Protocols	Transport	QoS	Architecture	Security
CoAP	UDP	YES	Request/Response	DTLS
MQTT	TCP	YES	Publish/Subscribe	TLS/SSL
XMPP	TCP	NO	Request/Response Publish/Subscribe	TLS/SSL
RESTFUL	HTTP	NO	Request/Response	HTTPS
AMQP	TCP	YES	Publish/Subscribe	TLS/SSL
Web socket	TCP	NO	Client/Server Publish/Subscribe	TLS/SSL
DDS	TCP/UDP	YES	Publish/Subscribe	TLS/SSL
SMQTT	TCP	YES	Publish/Subscribe	It have own