OneM2M

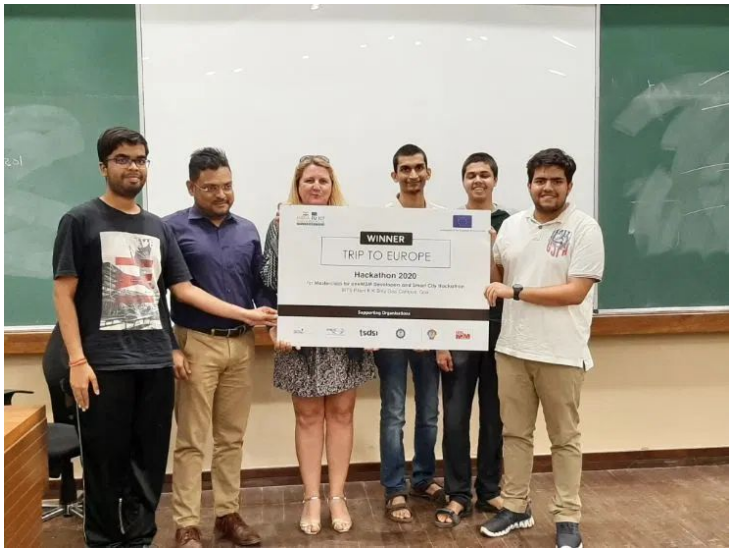
Spring 2020, IIIT-H

India-EU Collaboration

- Promote India and EU ties to produce ICT standards
- OneM2M standards developed for standard IoT framework

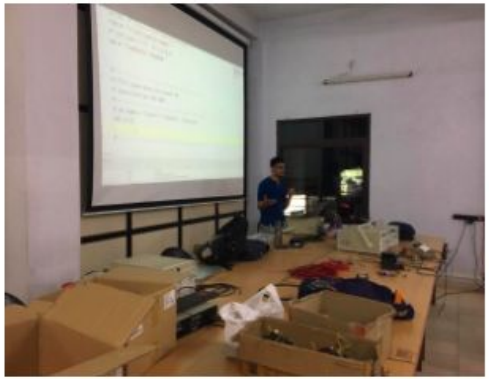
Activities under the collaboration

1. **Hackathons**
   IIIT students won 2 of the 4 hackathons



Activities under the collaboration

2. **Labs sessions**



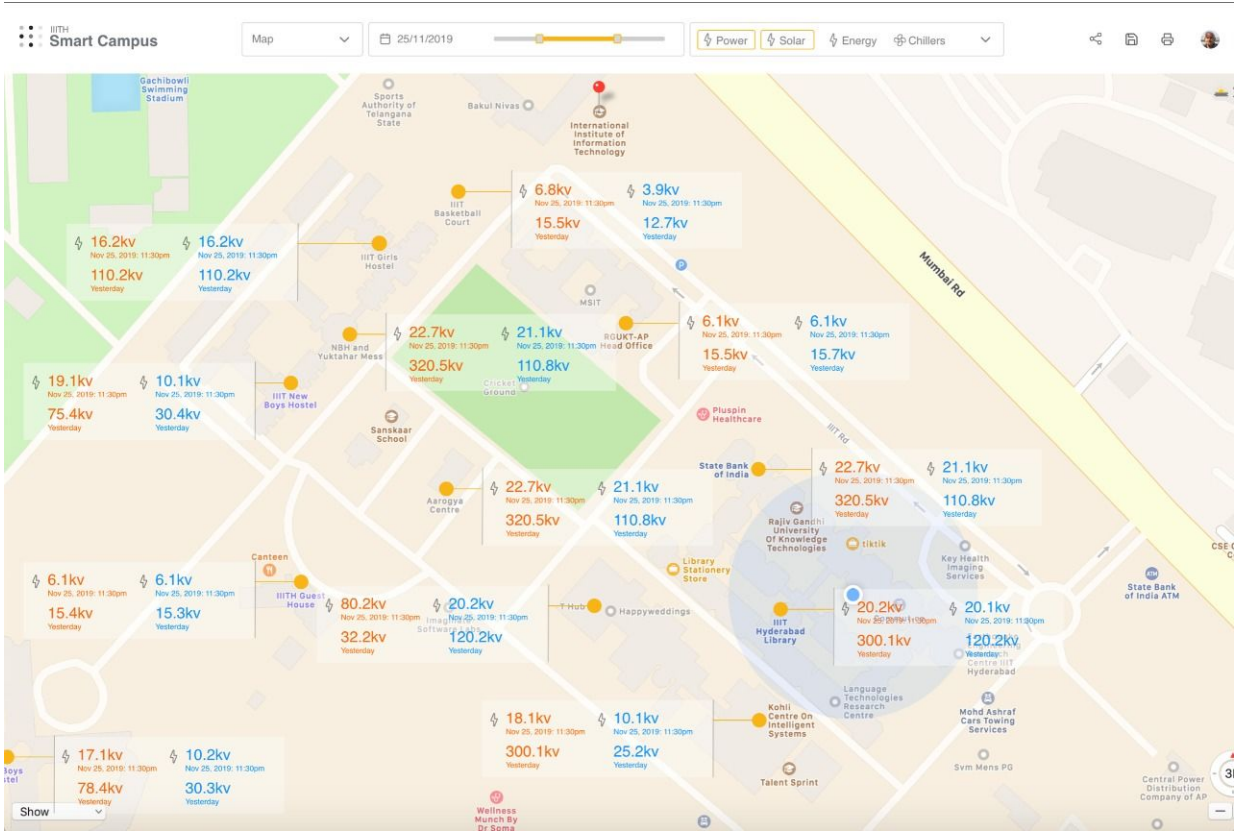
Activities under the collaboration

3. **Smart Campus**

   Deployed multiple IoT devices across the IIIT-H campus

Activities under the collaboration

4. **Smart Campus**



Today’s topics

1. Case study : Existing IoT Solutions
2. The OSI model
3. General look at interoperability
4. Introduction to OneM2M
5. Why OneM2M
6. Ontology (or Nomenclature)
7. Using OneM2M
8. Open Source opportunities

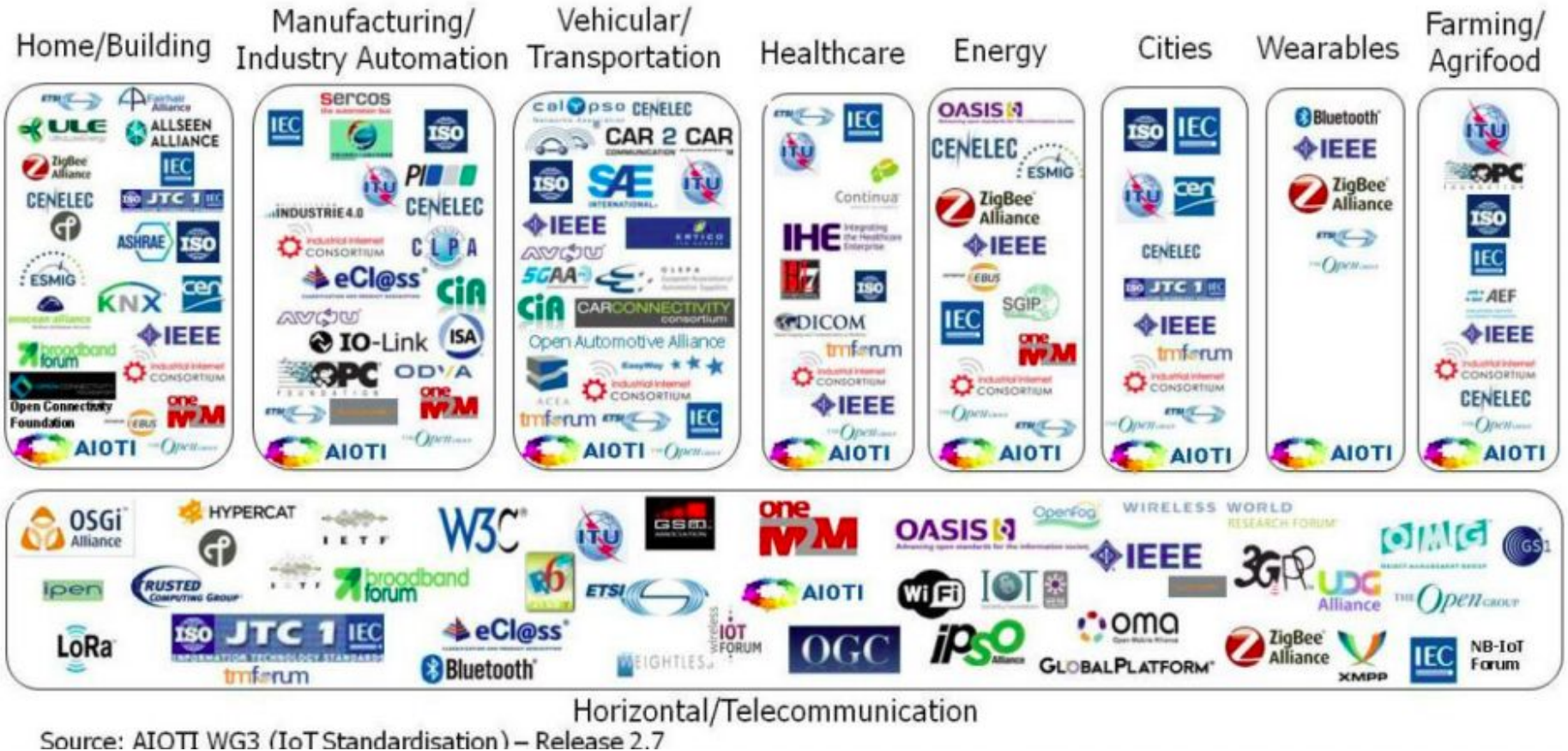
Today’s topics

1. **Case study : Existing IoT Solutions**
2. The OSI model
3. General look at interoperability
4. Introduction to OneM2M
5. Why OneM2M
6. Ontology (or Nomenclature)
7. Using OneM2M
8. Open Source opportunities

A simple case

Source : http://www.onem2m.org/images/app_dev_guide/tr-0057/sillos.png

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


Home/Building
Manufacturing/ Industry Automation
Vehicular/ Transportation
Healthcare
Energy
Cities
Wearables
Farming/ Agrifood
Horizontal/Telecommunication
Source: AIOTI WG3 (IoT Standardisation) – Release 2.7

Source

The challenges

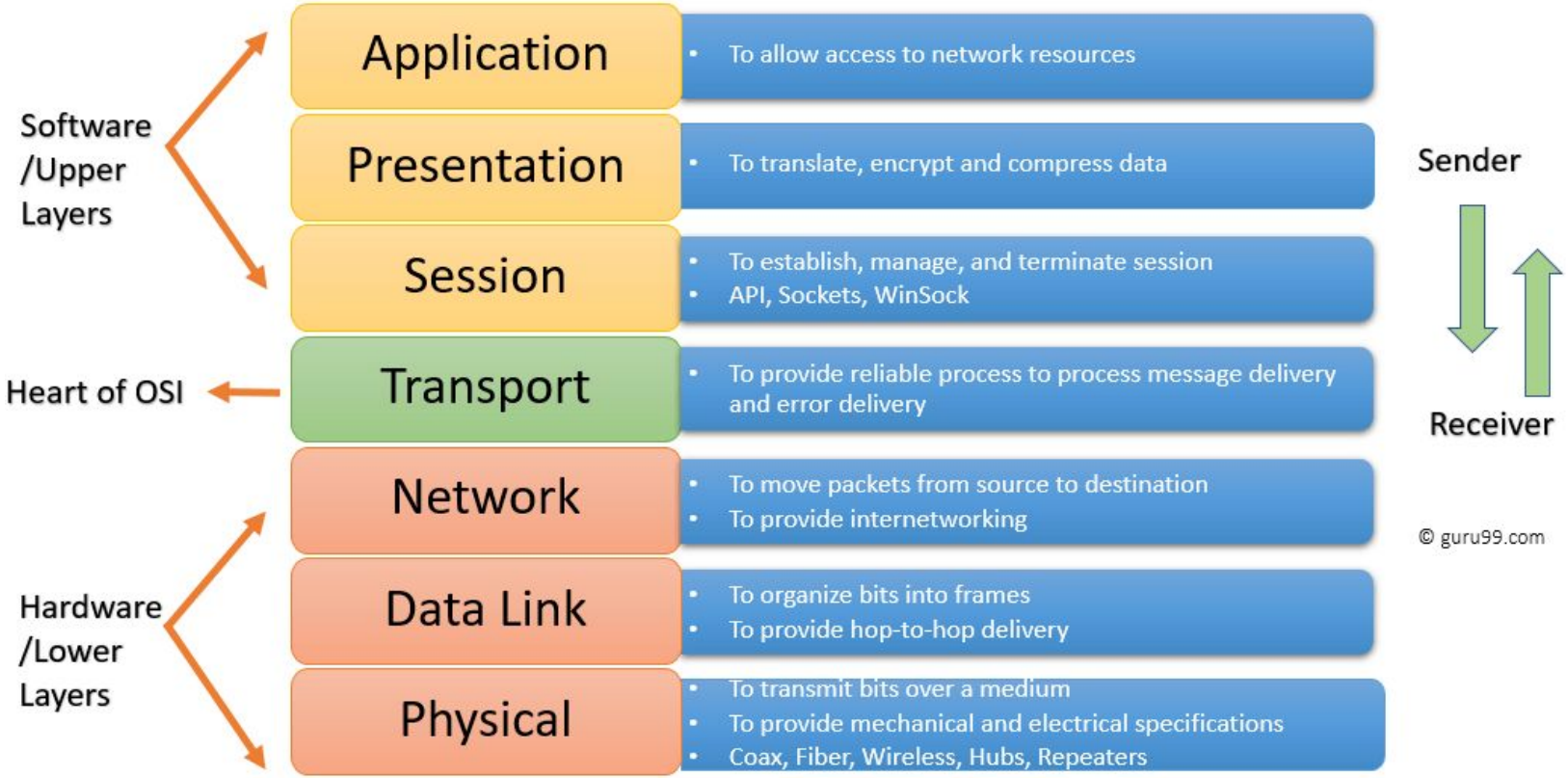
1. Interoperability
2. Scalability
3. Device management

Today’s topics

1. Case study : Existing IoT Solutions
2. **The OSI model**
3. General look at interoperability
4. Introduction to OneM2M
5. Why OneM2M
6. Ontology (or Nomenclature)
7. Using OneM2M
8. Open Source opportunities

OSI model

APSTNDP

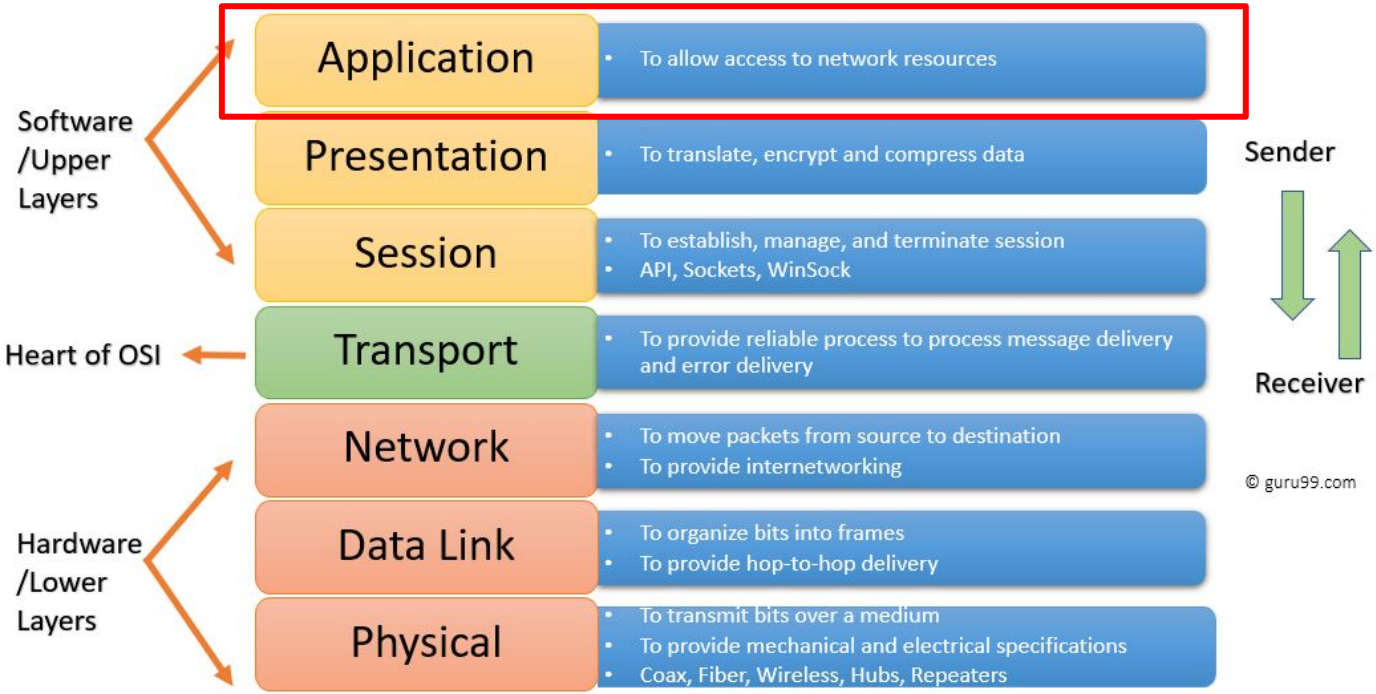


Today’s topics

1. Case study : Existing IoT Solutions
2. The OSI model
3. **General look at interoperability**
4. Introduction to OneM2M
5. Why OneM2M
6. Ontology (or Nomenclature)
7. Using OneM2M
8. Open Source opportunities

OSI model

NOTE:
This is a loose interpretation of OSI model and interoperability for understanding.

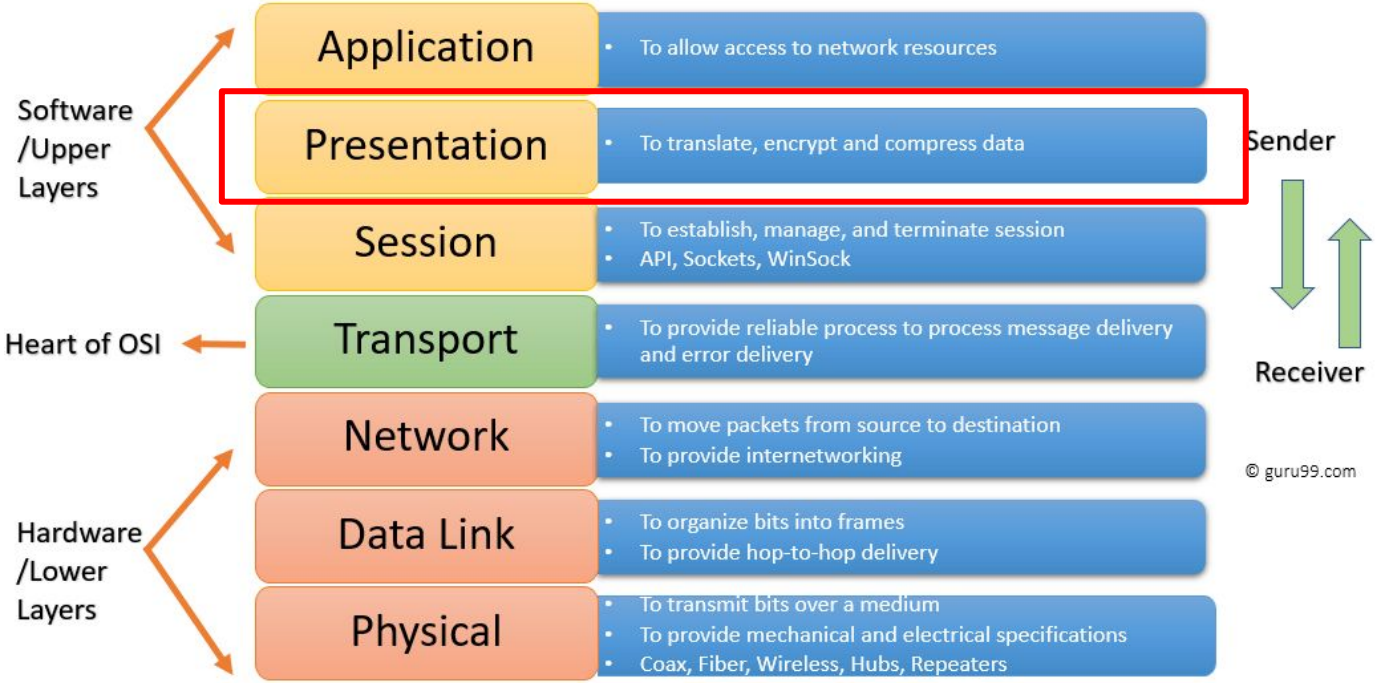


Semantic interoperability

- Standardization of *application layer*
- Formal definition of W3C "*Enabling different agents, services, and applications to exchange information, data and knowledge in a meaningful way, on and off the Web*"
- In short, various applications should be able to understand the meaning or semantics of the data being shared
- Eg: Units of measurement, compatible data structures

OSI model

NOTE:
This is a loose interpretation of OSI model and interoperability for understanding.

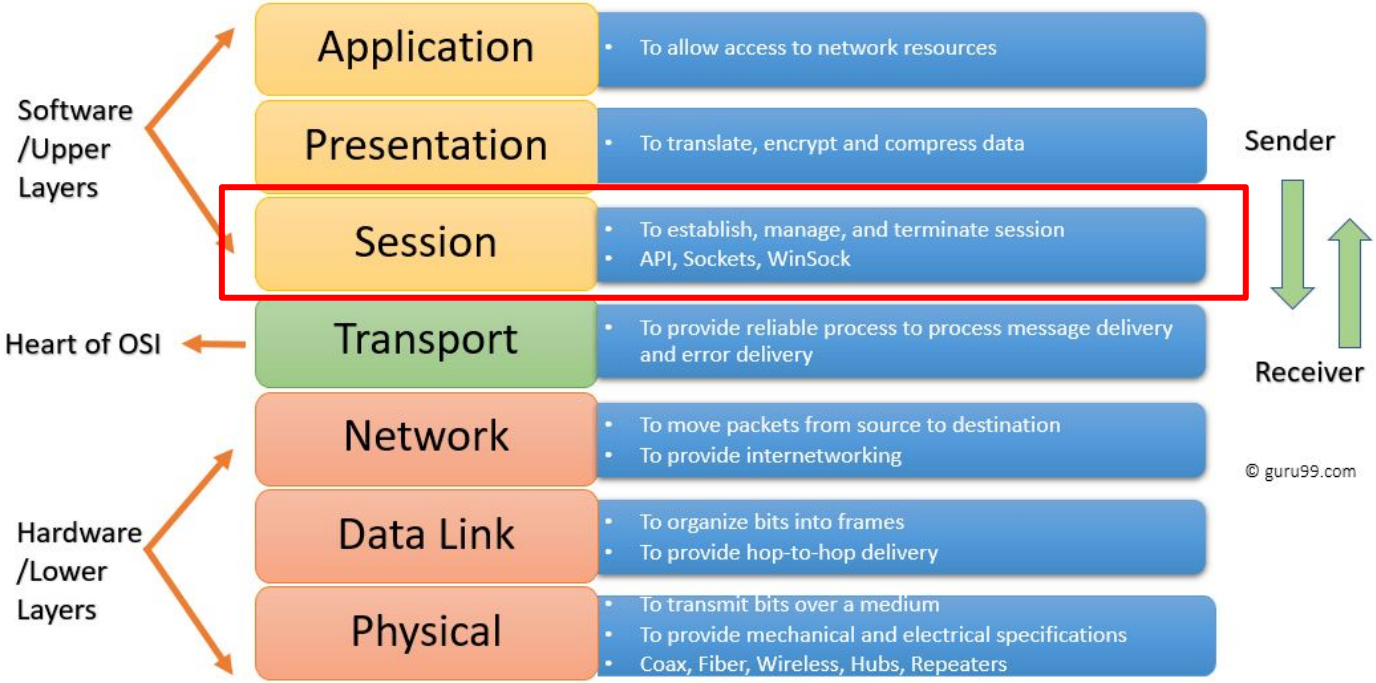


Syntactic interoperability

- Standardization of *Presentation layer*
- Standardized rules encrypt the data in the same way.
- In short, various applications should be able to decrypt and translate the data in the same way
- Eg: Using different encryption techniques doesn't allow understanding of data

OSI model

NOTE:
This is a loose interpretation of OSI model and interoperability for understanding.

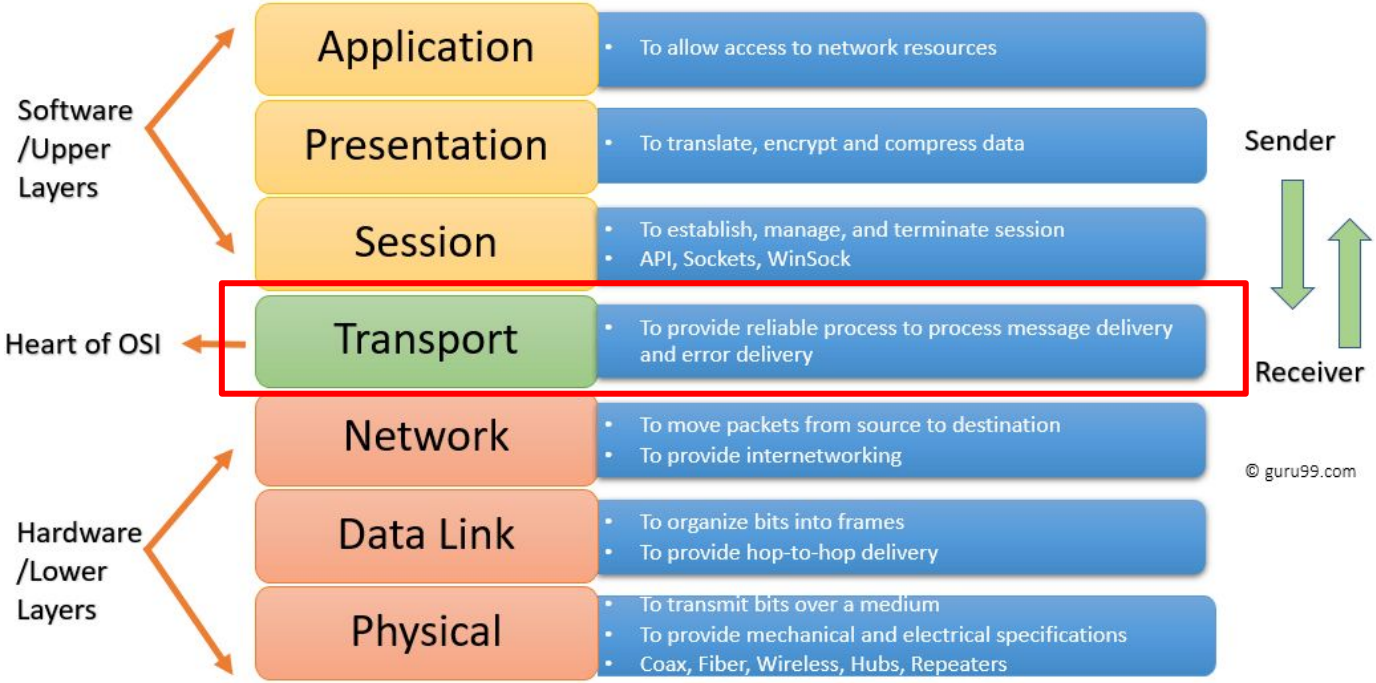


Platform interoperability

- Standardization of *Session layer*
- Rules on how to manage data.
- Eg: APIs of various verticals are different. With standardisation, it enables developers with ease of access to data and develop applications rather than spend time understanding the APIs of each vertical

OSI model

NOTE:
This is a loose interpretation of OSI model and interoperability for understanding.

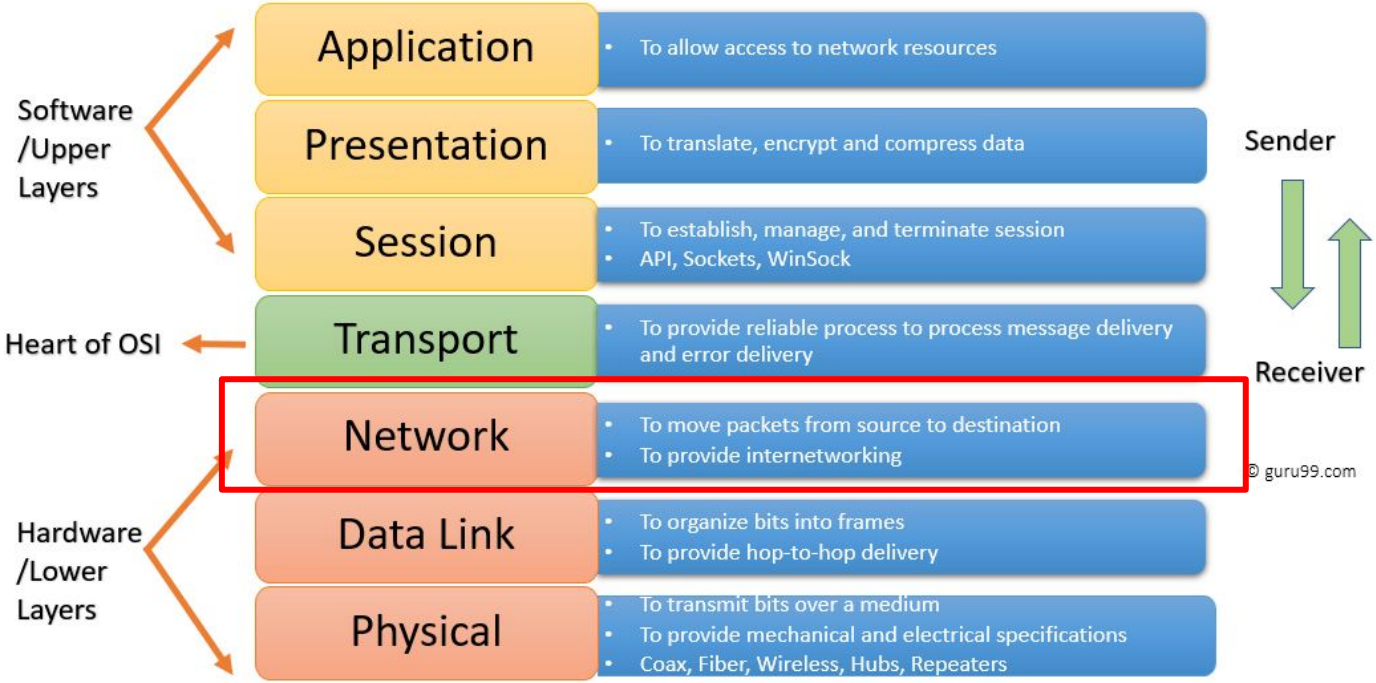


Transport interoperability

- Standardization of *Transport layer*
- OneM2M started with focus to standardize this layer
- In short, a horizontal layer to enable communications between various hardware and software solutions
- Will look into detail later

OSI model

NOTE:
This is a loose interpretation of OSI model and interoperability for understanding.

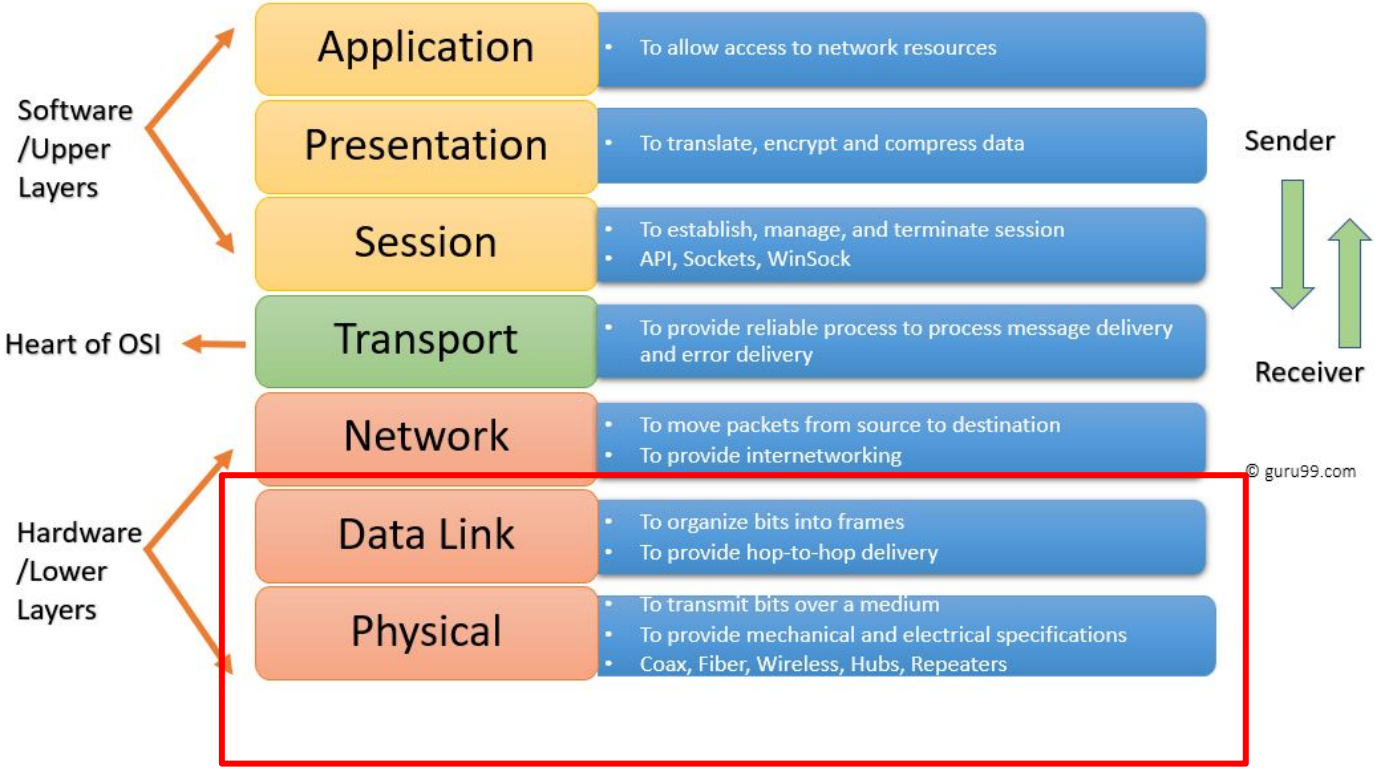


Network interoperability

- Standardization of *Network layer*
- Ability for data to travel across various kinds of networks
- Eg: Data transfer from Zigbee to bluetooth networks

OSI model

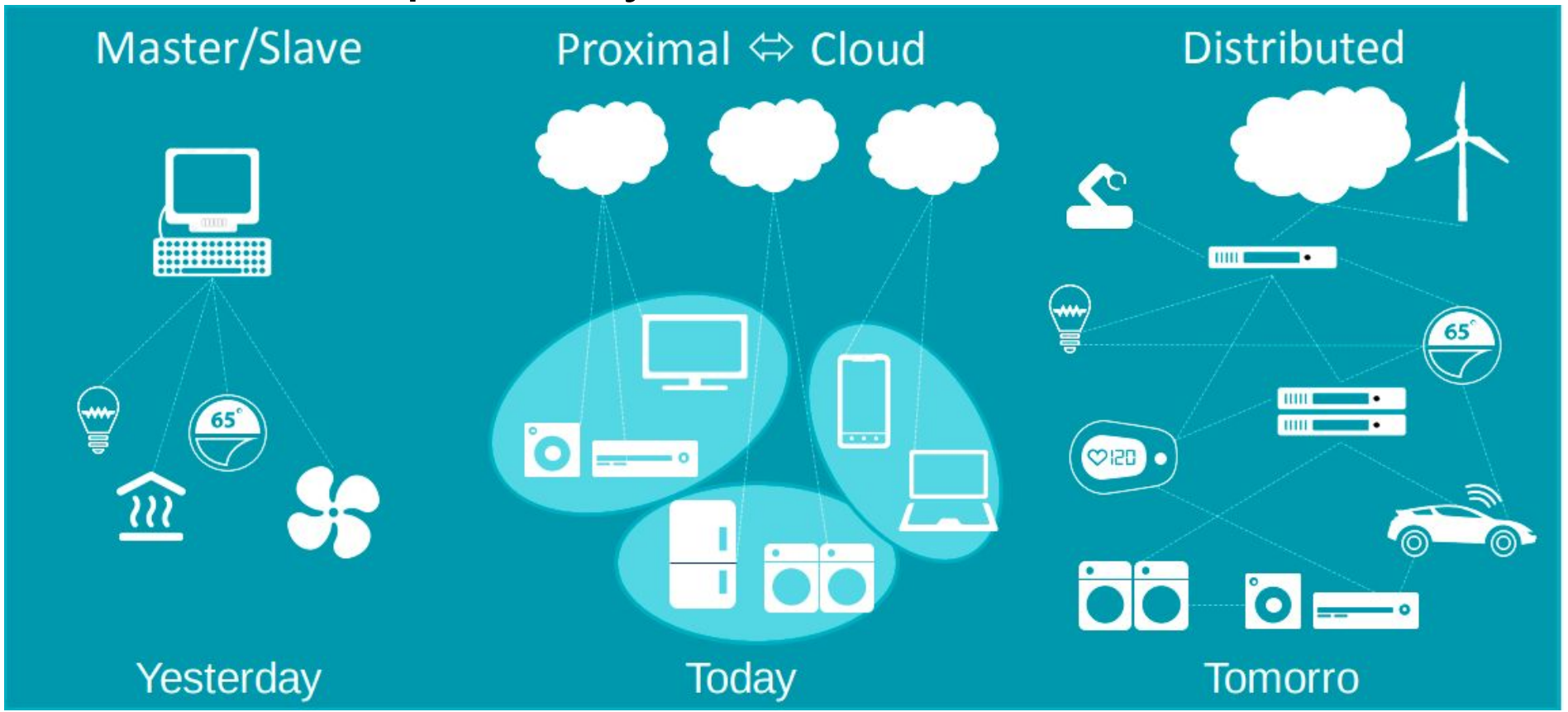
NOTE:
This is a loose interpretation of OSI model and interoperability for understanding.



Device interoperability

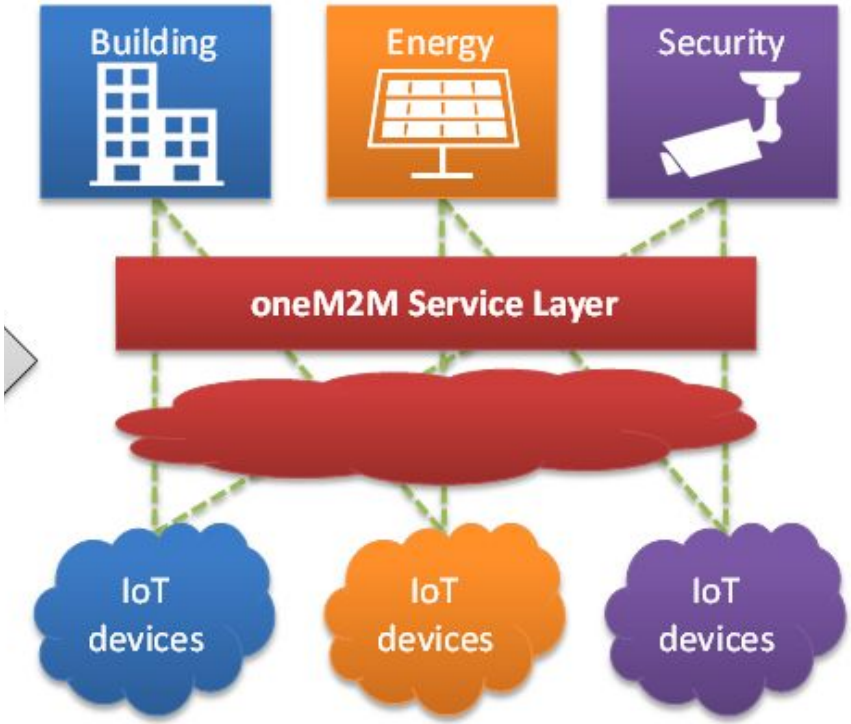
- Standardization of *Data link and Physical layer*
- Enables Cross-vendor support for devices/hardware platforms
- Eg: Using sensors and boards from various manufacturers. Servers able to accept data from various sources and route them accordingly

Aim of interoperability



Source

With oneM2M



Source : http://www.onem2m.org/images/app_dev_guide/tr-0057/sillos.png

- 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

Today’s topics

1. Case study : Existing IoT Solutions
2. The OSI model
3. General look at interoperability
4. **Introduction to OneM2M**
5. Why OneM2M
6. Ontology (or Nomenclature)
7. Using OneM2M
8. Open Source opportunities

What is oneM2M

- Global standardization for M2M and IoT
- Provides a software framework by creating a **horizontal** layer across domains.
- This service layer is located between applications and hardware infrastructure
- Enables reusability
- Members consist of various standard bodies, ICTs and companies.
- Work was initiated in 2008

Today’s topics

1. Case study : Existing IoT Solutions
2. The OSI model
3. General look at interoperability
4. Introduction to OneM2M
5. **Why OneM2M**
6. Ontology (or Nomenclature)
7. Using OneM2M
8. Open Source opportunities

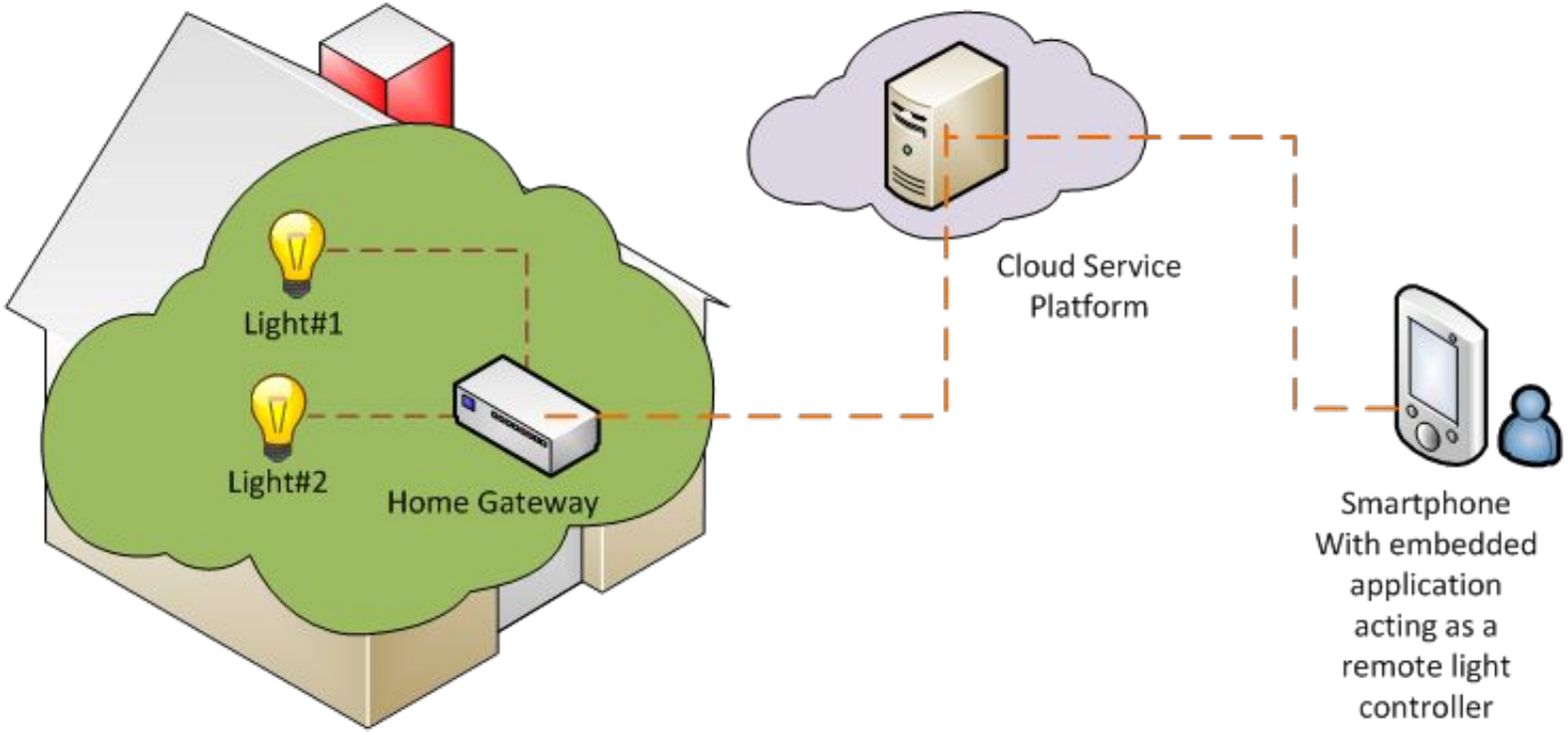
Need for standardization

- To provide scalability and flexibility
- Improves functionality-cost-quality trade off
- Set of APIs communicating with the service layer reduces:
  - Time-to-market
  - Development and on-boarding costs
  - Management of devices and applications

Today’s topics

1. Case study : Existing IoT Solutions
2. The OSI model
3. General look at interoperability
4. Introduction to OneM2M
5. Why OneM2M
6. **Ontology (or Nomenclature)**
7. Using OneM2M
8. Open Source opportunities

Ontology

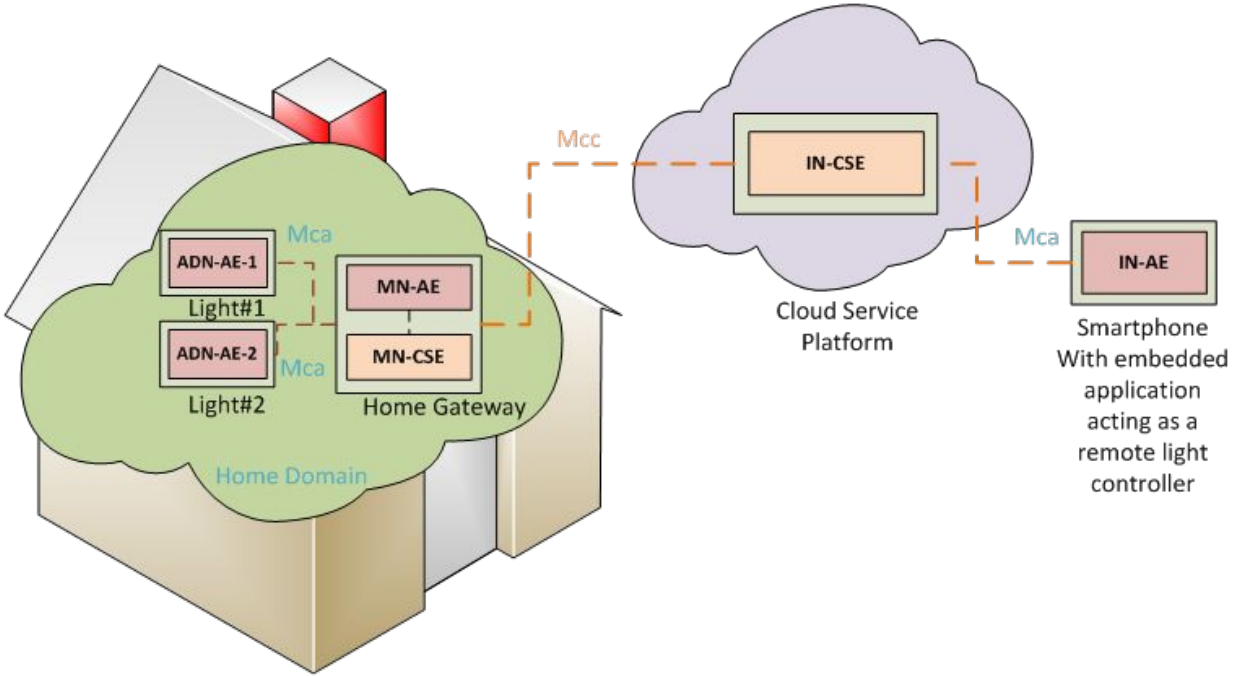


http://www.onem2m.org/images/app_dev_guide/UseCase.png

IN : Infrastructure Node
MN : Middle Node

CSE : Common service entity
AE : Application Entity

ADN : Application Device Node



http://www.onem2m.org/images/app_dev_guide/UseCase.png

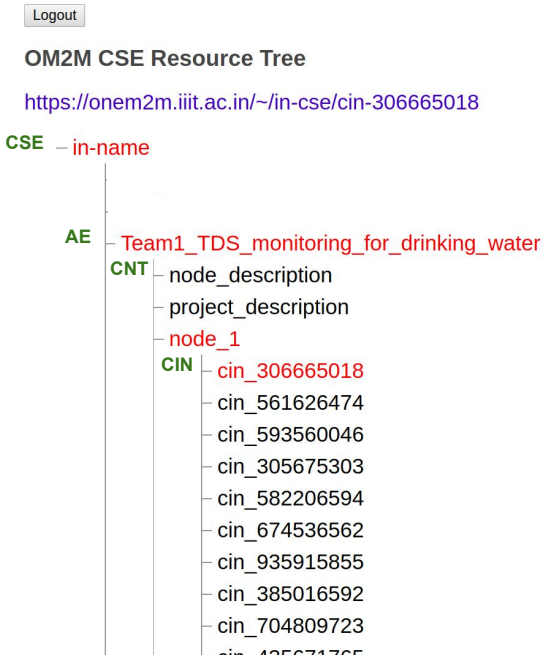
Today’s topics

1. Case study : Existing IoT Solutions
2. The OSI model
3. General look at interoperability
4. Introduction to OneM2M
5. Why OneM2M
6. Ontology (or Nomenclature)
7. **Using OneM2M**
8. Open Source opportunities

For the purpose of this demo...

Build the header and payload using Python libraries on my local system.

- Registering AE
- Creating Container
- Create content instances
- Retrieve data



OM2M Connecting things

Attribute	Value
rn	cin_306665018
ty	4
ri	/in-cse/cin-306665018
pi	/in-cse/cnt-808290327
ct	20191017T182755
lt	20191017T182755
st	0
cnf	text/plain:0
cs	13
con	test_instance

Protocols to communicate with OneM2M

- **HTTP**
- MQTT
- CoAP

HTTP Messages

The hardware reads the sensor data and encapsulates a HTTP message by forming the *header* and the *payload.*

- Header:
  - Consists of meta information, credentials, etc…
- Payload:
  - Actual data and other relevant parameters
  - Can specify payload in either XML or **JSON** format
- This is then sent to the IoT server using appropriate HTTP methods.

HTTP Request can be made from any device

Condition : Hardware should have a TCP/IP stack

- A plain arduino UNO or Mega doesn’t have TCP/IP stack
  (should use a WiFi shield)
- RaspberryPi, BeagleBone, etc.. have the TCP/IP stack.
- Considering the power consumption and footprint, ESP has been a good choice
  (ESP32, ESP8266, NodeMCU)

For the purpose of this demo...

- **Registering AE**
- Create Container
- Create content instances
- Retrieve data

Registering AE

Things to note:

- Run once per AE
- Type:2 (specifies we are creating an AE)
- rn : resource name (name of the AE)
- lbl : labels (used to filter)
- Header + payload = HTTP message
- Then execute a HTTP POST method (sends the HTTP message to the server)

```
def register_ae(uri_cse, ae_name, labels="", fmt_ex="json"):
    """
        Method description:
        Registers an application entity(AE) to the OneM2M framework/tree
        under the specified CSE

        Parameters:
        uri_cse : [str] URI of parent CSE
        ae_name : [str] name of the AE
        labels : [str] labels for the AE
        fmt_ex : [str] payload format
    """

    headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/{};ty=2'.format(fmt_ex)}

    payload = {
        "m2m:ae": {
            "rn": "{}".format(ae_name),
            "api": "tap",
            "rr": "true",
            "lbl": labels
        }
    }

    response = requests.post(uri_cse, json=payload, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
```

For the purpose of this demo...

- Registering AE
- **Creating Container**
- Create content instances
- Retrieve data

Creating Container

Things to note:

- Run once per CNT
- Type:3 (specifies we are creating an CNT)
- rn : resource name (name of the CNT)
- mni : Maximum number of Instances
- Header + payload = HTTP message
- Then execute a HTTP POST method (sends the HTTP message to the server)

```
def create_cnt(uri_ae, cnt_name="", fmt_ex="json"):
    """
        Method description:
        Creates a container(CON) in the OneM2M framework/tree
        under the specified AE

        Parameters:
        uri_ae : [str] URI for the parent AE
        cnt_name : [str] name of the container (DESCRIPTOR/DATA)
        fmt_ex : [str] payload format
    """

    headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/{};ty=3'.format(fmt_ex)}

    payload = {
        "m2m:cnt": {
            "rn": "{}".format(cnt_name),
            "mni": -1
        }
    }

    response = requests.post(uri_ae, json=payload, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
```

For the purpose of this demo...

- Registering AE
- Creating Container
- **Create content instances**
- Retrieve data

Creating Content instances

Things to note:

- Type:4 (specifies we are creating an CIN)
- con : content (sensor/actuator state)
- Header + payload = HTTP message
- Then execute a HTTP POST method (sends the HTTP message to the server)

```
def create_data_cin(uri_cnt, value, fmt_ex="json"):
    """
        Method description:
        Creates a data content instance(data_CIN) in the OneM2M framework/tree
        under the specified DATA CON

        Parameters:
        uri_cnt : [str] URI for the parent DATA CON
        fmt_ex : [str] payload format (json/XML)
    """

    headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/{};ty=4'.format(fmt_ex)}

    payload = {
        "m2m:cin": {
            "con": "{}".format(value)
        }
    }

    response = requests.post(uri_cnt, json=payload, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
```

For the purpose of this demo...

- Registering AE
- Creating Container
- Create content instances
- **Retrieve data**

Retrieve

- Using the GET method by specifying the URI of the resource

Things to note:

- Type: not necessary to specify
- No payload
- Just Header = HTTP message
- Then execute a HTTP GET method (get the data that we want)
- Append “/la” to the URI to get the latest content instance from the specified container

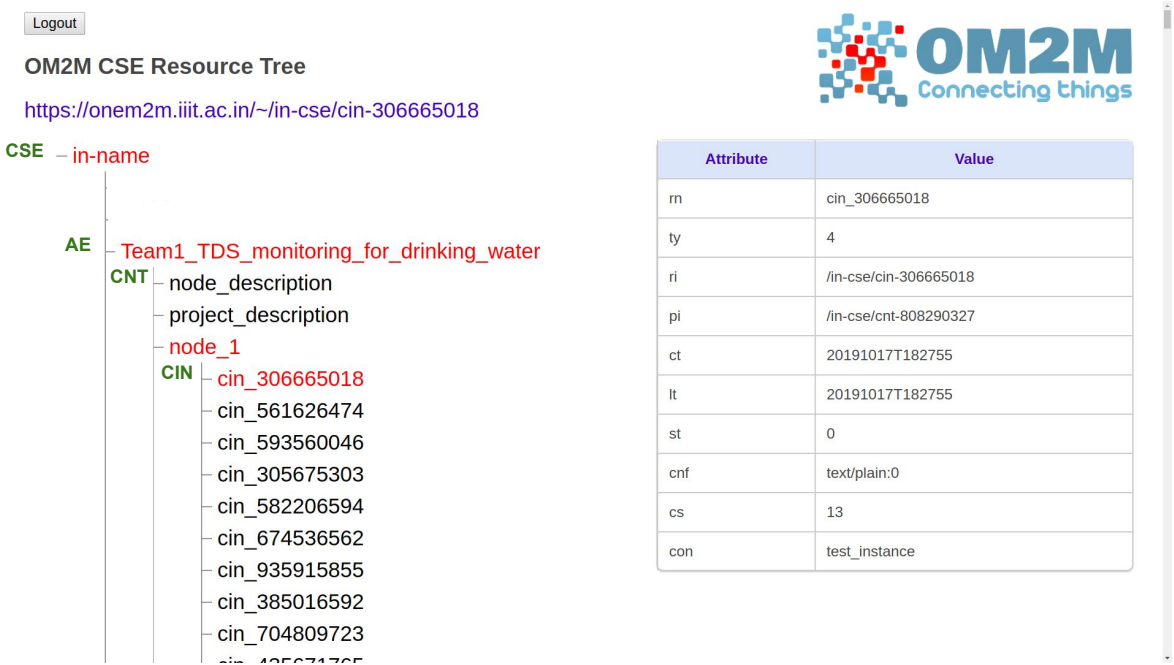
```
def get_data(uri, format="json"):
    """
        Method description:
        Gets data from the specified container(data_CIN)
        in the OneM2M framework/tree

        Parameters:
        uri : [str] URI for the parent DATA CON appended by "la" or "ol"
        fmt_ex : [str] payload format (json/XML)
    """
    headers = {
        'X-M2M-Origin': 'admin:admin',
        'Content-type': 'application/json'}

    response = requests.get(uri, headers=headers)
    print('Return code : {}'.format(response.status_code))
    print('Return Content : {}'.format(response.text))
    _resp = json.loads(response.text)
    return response.status_code, _resp["m2m:cin"]["con"]
```

OneM2M Resource tree

CSE : Common service entity
AE : Application entity
CNT : Container
CIN : Container Instance



- ae: Application Entity(Sensor/actuators)
- cnt: Container(For holding various kinds of data under the same AE)
- cin: Content Instance(For holding various instances of the same data type)
- sub: Subscription
- rn: Resource Name
- ty: Type
- ri: Resource ID
- pi: Parent Id
- Acpi: Access Control Policies IDs
- uril: URI List
- ct: Creation Time
- et: Expiration Time
- lt: Last Modified Time
- lbl: Label
- cnf: Content Format
- con: Content
- mni: Maximum Number of Instance
- api: Application Id
- poa: Point of Access
- rr: Request Reachability
- sur: Subscription URI

To summarize

- IoT devices need to make HTTP requests to communicate with the OneM2M resource tree:
  - To send data, we use the POST method using HTTP request
  - To get data, we use the GET method
  - There are other methods like UPDATE and DELETE

- HTTP Requests can be made from any kind of device given it has TCP/IP hardware stack.
  - Arduino boards with wifi shields
  - ESP32, ESP8266
  - Raspberry Pi

Advanced Features

- Grouping
- Filtering
- Subscription
- Security and permissions

- Can pass various parameters for attaining flavours of functionality
  - Eg: Get data after a particular date, get data between 2 time intervals, etc..

Kinds of URI

- Direct URI
- Indirect URI
- Other kinds can be found in the OneM2M documentation

Open Source Contribution

- Make presentations on Technical sheets
- Develop Python/C++ codes for advanced features
- Developing IIIT’s implementation of OneM2M

Github repo https://github.com/suraj2596/OneM2M-IIIT-H

References

- http://www.onem2m.org/getting-started/onem2m-overview/introduction/service-layer
- http://www.onem2m.org/technical/published-drafts/release-3
- http://www.onem2m.org/developer-guides
- Om2m.org