

Prepared by Dr. Wesam Ahmed

IT402
Internet of Things

Lecture 6



IoT Networking Protocols – Part 2

Lecture 6

IT402

Internet of Things

Prepared by
Dr. Wesam Ahmed

06

Learning Objectives:

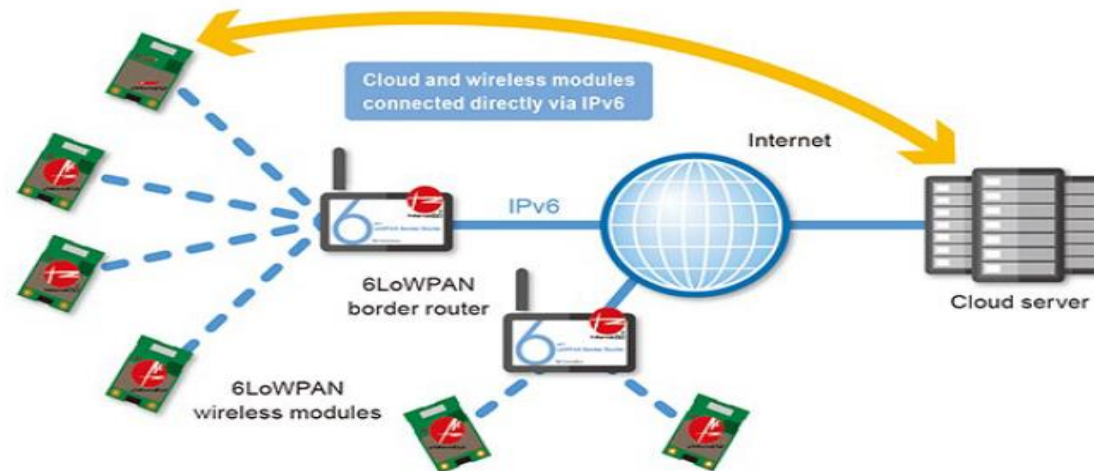
- 1.1 Explain IoT networking application layer
- 1.2 Machine-to-Machine (M2M)
- 1.3 One Machine-to-Machine (M2M)

06

IoT Networking Protocols

Internet Layer:

The Internet Layer in IoT is responsible for **addressing, routing, and transferring data packets** between devices across different networks — essentially **ensuring that data from IoT devices can travel to cloud servers, gateways, or other devices reliably.**



06

IoT Networking Protocols

Internet Layer:

- It allows **IPv6 packets** to be transmitted efficiently over **low-power, low-data-rate** wireless networks.
- Because IoT devices have **limited power, memory, and bandwidth**, standard **IPv6 packets** (which have large headers) need optimization — that's where **IPv6 over Low power Wireless Personal Area Networks (6LoWPAN)** helps.
- **IPv6 over Low power Wireless Personal Area Networks (6LoWPAN)** provides three main functions:
- **6LoWPAN compresses IPv6 headers** to just a few bytes to save bandwidth and energy.

06

IoT Networking Protocols

Internet Layer:

- IPv6 header compression: 6LoWPAN introduces three headers for each of the three functions that it provides. Those headers are: Compression header, Fragment header, and Mesh header.
- IPv6 packet segmentation and reassembly
- Layer 2 forwarding (also referred to as Mesh Under).

◆ Headers used for compression:

Header Type	Function
Compression Header	Reduces the IPv6 header size using shared network information (like shared prefixes).
Fragment Header	Used when a packet is too large for one frame; it splits the packet into smaller fragments.
Mesh Header	Supports multi-hop forwarding at the link layer (Layer 2), allowing devices to route data without IP-level routing.

06

IoT Networking Protocols

Internet Layer:**IPv6 Packet Segmentation and Reassembly**

- Since **IEEE 802.15.4** frames are small, large IPv6 packets must be **fragmented** into smaller pieces before transmission.
- Each fragment is labeled with a **Fragment Header**.
- At the destination, the fragments **are reassembled** into the **full IPv6 packet**.

➤ Example:

If an IPv6 packet = 1000 bytes, and the frame can carry only 100 bytes →

It is split into **10 fragments**, sent individually, and reassembled by the receiver.

06

IoT Networking Protocols

Internet Layer:**Layer 2 Forwarding (Mesh Under)**

- Also known as **Mesh Under routing**.
- It enables **multi-hop communication** at the **data link layer (Layer 2)** — meaning packets can hop across multiple devices **without IP routing** at every hop.
- This is useful in **mesh networks**, where small IoT nodes act as both transmitters and routers.
- **Example:**
A packet can travel from **Sensor A → Sensor B → Gateway**,
even if **A and Gateway** are not directly connected — B helps forward the packet.

06

IoT Networking Protocols

Internet Layer:

- IEEE 802.15.4 TSCH (Time Slotted Channel Hopping) is a **Data Link Layer** technology used in low-power, reliable, and time-synchronized IoT networks, such as **industrial IoT (IIoT)**.
- However, **IPv6** was designed for traditional Internet networks — it doesn't natively understand the timing, synchronization, or limited resources of TSCH networks.
- An **adaptation layer is required in order to run the IPv6 stack** on top of IEEE 802.15.4 TSCH .
- It handles functions such as **network formation, maintenance, scheduling, and security**, which are essential for reliable IoT networking.

06

IoT Networking Protocols

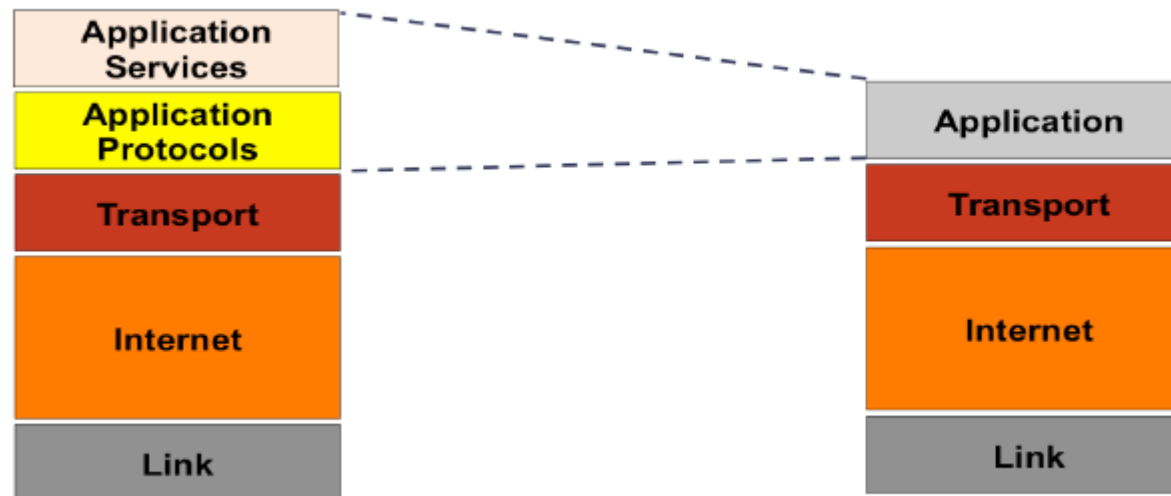
➤ Goals are to address the following issues:

Function	Description
Network Formation	Defines how new nodes securely join the TSCH network and how existing nodes advertise the network's presence.
Network Maintenance	Ensures ongoing operation, handling node mobility, link quality changes, and synchronization.
Topology and Schedule Mapping	Maps the network's logical topology (who communicates with whom) to the TSCH time-slot schedule.
Resource Management	Manages limited bandwidth, time slots, and energy resources across nodes.
Flow Control	Controls data transmission rates to avoid congestion and ensure smooth traffic flow.
Determinism	Guarantees predictable data delivery (low latency and reliability), crucial for industrial IoT.
Scheduling Mechanisms	Allocates communication time slots dynamically or statically for each node's transmissions.

06

IoT Networking Protocols

Application Protocols Layer:



06

IoT Networking Protocols

Application Protocols Layer:

- The **Application Protocols Layer** in the **Internet of Things (IoT)** is responsible for managing the **communication between IoT devices (Things, Gateways) and Applications or Cloud Services**.
- It ensures that **data collected by sensors** can be properly sent, received, interpreted, and acted upon.
- They typically support **the flow of data (e.g. readings or measurements) from Things to Applications** and the flow of command or **control information (e.g. to trigger or actuate end-devices) in the reverse direction**.

06

IoT Networking Protocols

Application Protocols Layer:

- ✓ **From devices to applications**
 - This is the **data flow** from IoT **sensors or devices** sending information to the **cloud, server, or application** for processing, analysis, or storage.
- ✓ The data typically includes:
 - ✓ Sensor readings (temperature, humidity, pressure, etc.)
 - ✓ Device status updates (battery level, location, etc.)
 - ✓ Event notifications or alerts
- ✓  **Example:**

A smart thermostat sends the current room temperature to a mobile app or cloud server every minute.

06

IoT Networking Protocols

✓ From applications to devices

This is the **control flow** in the **reverse direction**, where the application sends **commands or control messages** to IoT devices to perform certain actions.

- The commands may include:
 - Turning a device ON/OFF
 - Changing configurations (e.g., sampling rate, thresholds)
 - Triggering actuators (e.g., open a valve, start a motor)
- **Example:**

A user changes the target temperature from a mobile app → the command is sent to the thermostat → the device adjusts the heater accordingly.

06

IoT Networking Protocols

- Application protocols define **the semantics and mechanisms** for message exchanges between the communicating endpoints.
- In simple terms, this means that **application protocols** specify **how devices and applications talk to each other** both **what** their messages mean (**semantics**) and **how** they are sent and received (**mechanisms**).
- The landscape of the Application Protocols Layer in IoT is currently **crowded with competing protocols and standards**, each **having its own set of strengths and weaknesses** and with no clear path towards convergence being agreed upon by the industry yet.
- **IoT systems are very diverse — devices vary in:**
 - **Power consumption** (battery vs. mains-powered)
 - **Network connectivity** (Wi-Fi, cellular, Zigbee, LoRa, Bluetooth, etc.)
 - **Data needs** (small sensor readings vs. large data streams)
 - **Latency and reliability requirements** (industrial vs. consumer IoT)

Because of these differences, **no single protocol works best for every case** — so multiple standards coexist.

06

IoT Networking Protocols

- Applications Protocols vary in **the data serialization formats** used to encode information into messages.
- **Data serialization** means converting structured data (**like sensor readings or commands**) into a format that can be **easily transmitted** over a network and then **reconstructed** by the receiver.

Challenges in IoT data serialization formats:

- ✓ Mapping between the **formats used in constrained devices and those used by applications** in the World Wide Web. **Popular data serialization formats on the Web include XML and (JavaScript Object Notation) JSON.**

Example:

- A sensor sends data as CBOR (compact binary).
- The gateway converts it to JSON for a web dashboard.
- This translation uses time, CPU, and energy.

06

IoT Networking Protocols

Challenges in IoT data serialization formats:

- ✓ The impact data serialization formats have on device resource utilization, socially energy consumption.
- Serialization requires **CPU cycles and memory**, and larger data formats mean more **energy** to transmit.
- IoT devices are often **battery-powered**, so efficient serialization is crucial.
- ✓ The impact data serialization formats have on network bandwidth utilization.
- The **size** of serialized data affects **network traffic** and **bandwidth usage**.

06

IoT Networking Protocols

Quality of Service (QoS):

- Application protocols should provide **control over the real-time behavior** and performance **of IoT applications by means of a rich set of QoS Policies.**
- QoS ensures that the **right data** reaches the **right place**, at the **right time**, using the **right amount of resources.**



QoS Control Areas:**➤ Control of Local Resources**

Protocols must efficiently manage **local device resources** such as:

- **Memory buffers**
 - **CPU processing power**
 - **Transmission queues**
 - Ensures that a device doesn't overload or crash due to excessive communication.
- ◆ **Policies:**
- **Resource Limits Policy:**
Defines how much memory or buffer space the protocol can use for storing messages.
 - **Time Filter Policy:**
Sets the **minimum interval** between consecutive samples (data messages), preventing flooding or redundant data.
 - **Example:**
A temperature sensor sends data only **every 10 seconds** instead of every second → saves CPU and bandwidth.



QoS Control Areas:

➤ End-to-End Data Transfer Properties

- QoS also defines how messages behave **from sender to receiver**, across the entire IoT system.

- **Example:**

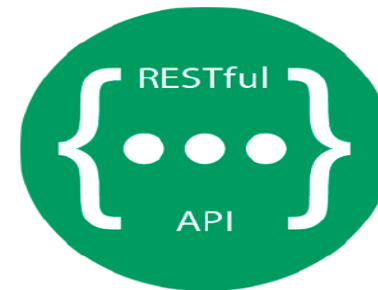
An industrial robot controller must receive motion commands **without delay** — this requires **low-latency and reliable** data transfer.

➤ **Resource Utilization in QoS** : QoS policies also influence **how much energy and processing power** are consumed by the Application protocol for data transmission and reception.

- ☐ Limiting message rates and buffer sizes reduces **power use**.
- ☐ Scheduling or delaying transmissions optimizes **CPU and network load**.
- ☐ Proper QoS configuration ensures **system stability** and **longer battery life**.

RESTfull Model:

- Some Application protocols follow a set of constraints defined by the **REpresentationalState Transfer (REST)** architectural model.
- **REST** is an **architectural style** for designing **networked applications**. It defines a set of **constraints** and **principles** that make communication between clients (like IoT devices or apps) and servers (like cloud platforms) **simple, scalable, and reliable**.
- In IoT, REST is used to allow **Things** (sensors, gateways, or actuators) to communicate with web applications and services in a **lightweight and standardized way**.



RESTfull Model:

- REST is a distributed client-server software architecture is an ideal model for IoT. The REST architectural is simple, scalable and reliable.
- Key Features of REST in IoT:
 - **Client/Server Communication Model:** This allows for separation of concerns where the server focuses on functions such as data storage whereas clients focus on the user interface and user state. Uniform interfaces separate the clients from the servers.
 - ✓ This separation of concerns allows:
 - **Servers** to handle data storage, processing, and logic.
 - **Clients** to handle user interface and device interaction.

➤ Key Features of REST in IoT:

- **Stateless Communication:**

- Each **request** from a client to a server must contain **all necessary information** to understand and process it.
- ✓ The server must not store any client context that continues between requests.
- The client maintains its own state (e.g., authentication, session data). In other words, requests are self-contained from a server perspective.

- ✓ **Example:**

When a smart light sends a request to turn **ON**, it includes all required details (like device ID, authentication token, and command).

The server doesn't need to remember previous requests.

The Constrained Application Protocol (CoAP):

- CoAP is a lightweight alternative to HTTP, targeted for constrained nodes in Low-power and Lossy Networks (LLNs).
- CoAP is designed for **constrained nodes** (limited CPU, memory, and energy) and **LLNs** (networks with high packet loss, low bandwidth, or intermittent connectivity).

How CoAP Works

- **Simple Example: Read a Temperature Sensor**
- Client sends a **CoAP GET request** to the sensor resource.
- Sensor responds with the **temperature value**.
- Optionally, messages can be **acknowledged** to ensure delivery.
- **All of this happens with fewer messages and smaller headers compared to HTTP.**



The Constrained Application Protocol (CoAP):

- Similar to HTTP, CoAP is a RESTful protocol. It supports the Create, Read, Update and Delete (CRUD) verbs but in addition provides built-in support for the publish/subscribe paradigm via the new Observe verb.
- meaning it uses standard **CRUD operations** to interact with resources:
 - **Create (POST)**: Add a new resource or data
 - **Read (GET)**: Retrieve data from a resource
 - **Update (PUT)**: Modify existing data
 - **Delete (DELETE)**: Remove a resource

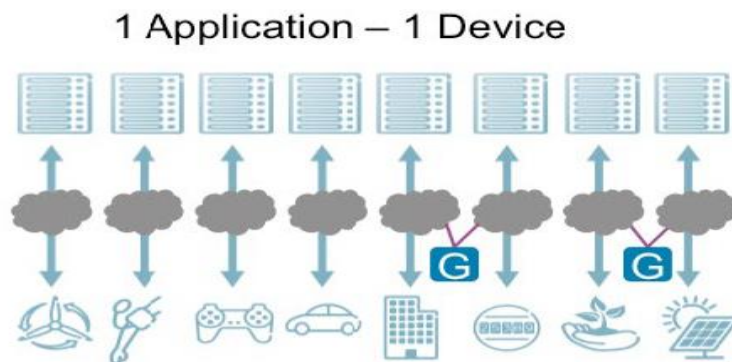
Example: A smart light bulb exposes a “status” resource:

- **GET** /light/status → Returns ON/OFF state
- **PUT** /light/status → Changes the state



Application Services Layer:

- 1 Application – 1 Device: **traditional M2M deployments.**
- Any Application – Any Device: Represents **modern IoT architecture** using an **Application Services Layer (ASL)** or **Common Abstraction Layer.**
- This Figure 6.1 illustrates the **evolution from fragmented M2M systems** to an **interoperable IoT architecture** enabled by a **Common Application Services Layer.**



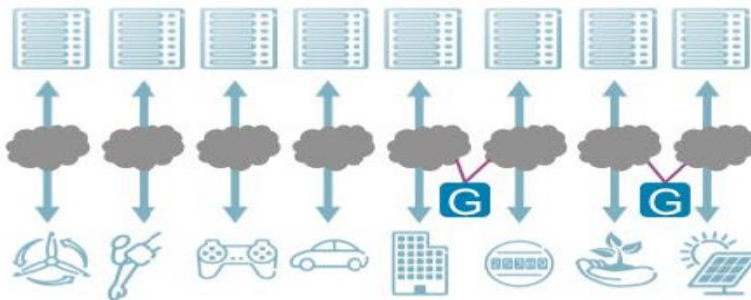
Any Application – Any Device



- **1 Application – 1 Device**

- ✓ Represents **traditional M2M deployments**.
- ✓ Each device (e.g., wind turbine, vehicle, sensor) is connected to a **dedicated application** through its **own proprietary communication stack**.
- ✓ Systems are **isolated** — data cannot be easily shared or reused.
- ✓ Integration across industries or domains is difficult.

1 Application – 1 Device

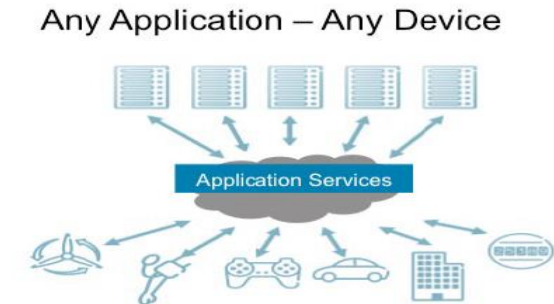
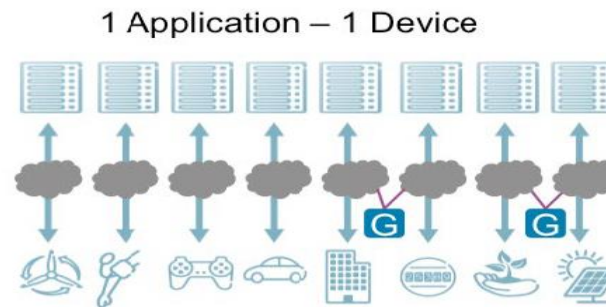


Any Application – Any Device



Any Application – Any Device

- Represents **modern IoT architecture** using an **Application Services Layer (ASL)** or **Common Abstraction Layer**.
- Devices communicate through **standardized APIs and data models** provided by the ASL.
- Applications can access data from **any device**, regardless of vendor or technology.
- Result: The system shifts from “one application per device” to “any application can use any device,” enabling true IoT interoperability.



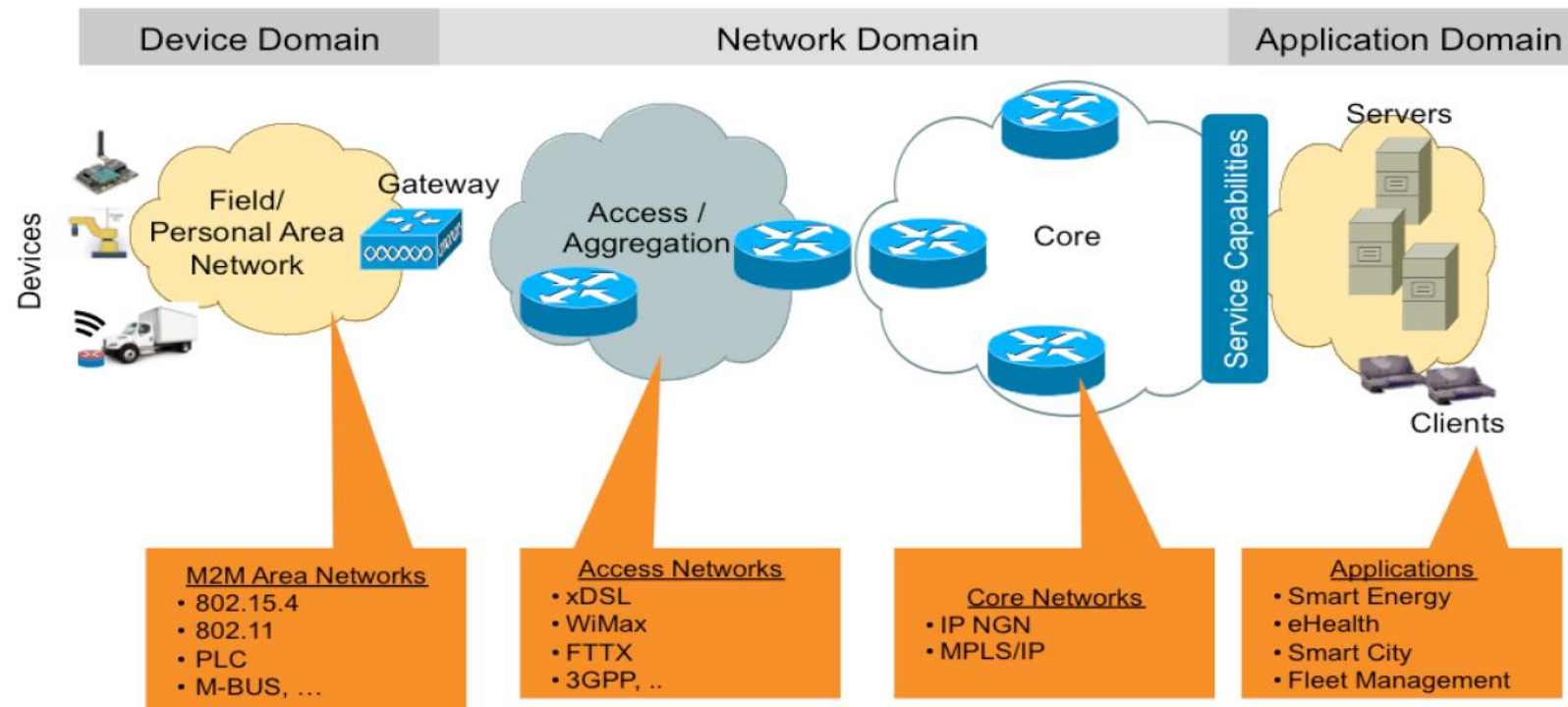
Machine-to-Machine (M2M) deployments have existed for over two decades. M2M deployments have been used across industries such as manufacturing, energy, transportation, and utilities. These systems enabled devices to communicate with centralized applications for monitoring and control. However, these early M2M solutions were typically built in isolation — each system developed its own vertical stack, with proprietary protocols and data models. This led to fragmentation across the ecosystem.

- ❑ **Proprietary Protocols:** non-standard communication methods developed by a specific vendor or organization for their own devices and systems. Example: A smart meter might use a vendor-specific TCP/IP message format, while a vehicle tracking system uses a custom XML-over-SMS protocol.
- ❑ **Proprietary data models** refer to custom or vendor-specific ways of structuring and naming data generated by devices.

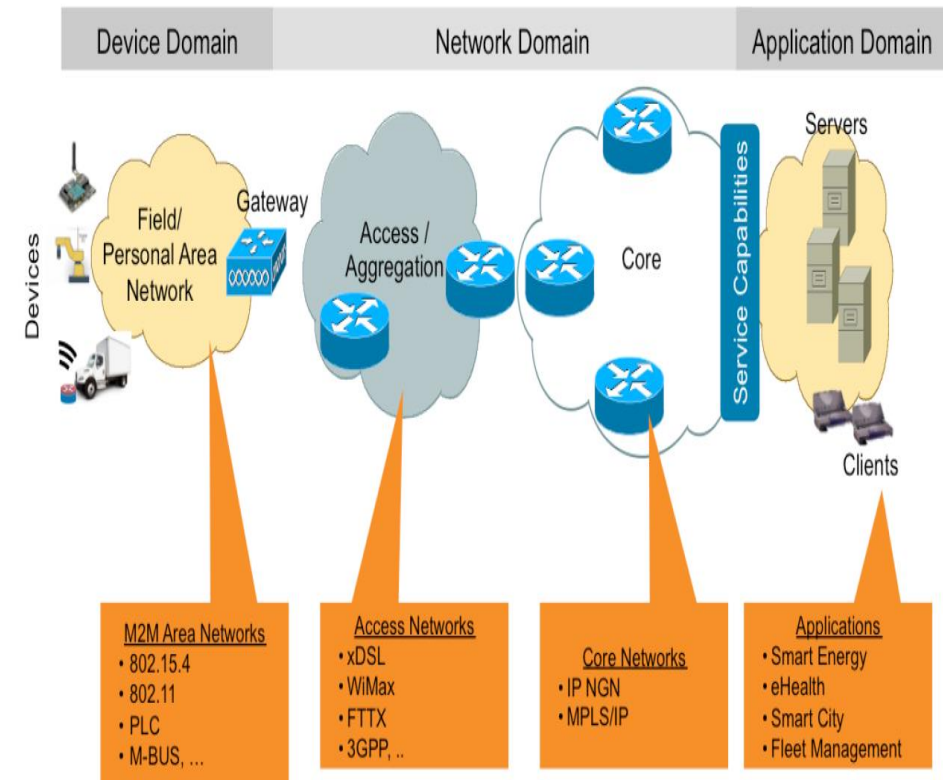
Application Services Layer

- To overcome this fragmentation, a **Common Abstraction Layer**—often called the **Application Services Layer (ASL)**—is introduced between devices and applications.
- **What It Does:**
 - Acts as a **middleware** or **service enablement layer** between devices and applications.
 - Provides **standardized interfaces and data models** so that any application can communicate with any device.
 - the **Common Abstraction Layer transforms** fragmented M2M ecosystems into **open, flexible IoT environments**—where **any application can work with any device** through a shared, standardized platform.

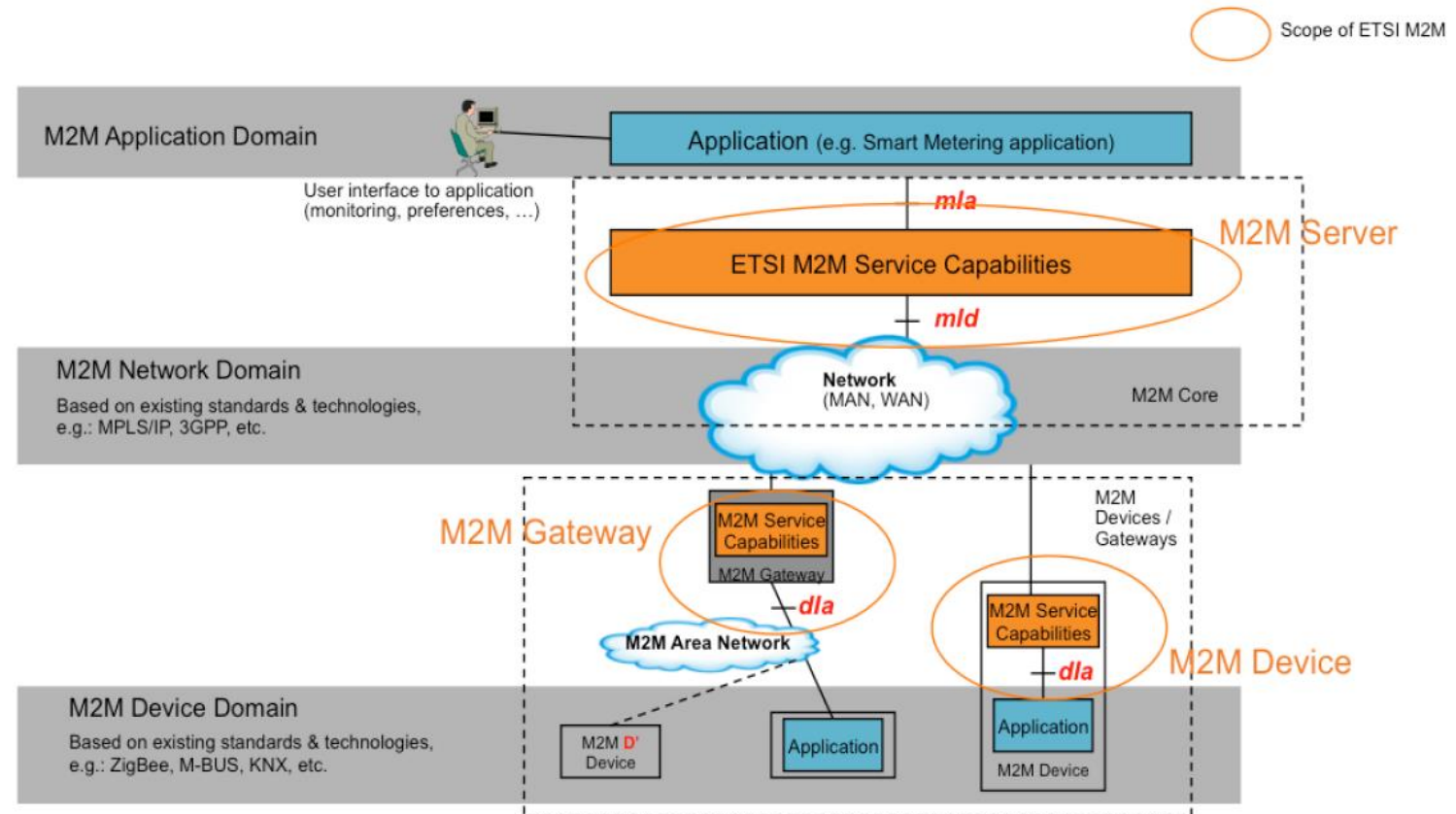
Machine-to-Machine (M2M)



- **How M2M Works**
- **Devices (machines)** collect data from the environment (e.g., temperature, speed, pressure).
- **M2M communication** allows these devices to share this data directly with each other or through a gateway.
- The **Gateway** aggregates data and forwards it to higher layers (network and application domains).
- In the **Application Domain**, data is analyzed and used in systems like **Smart Energy, Smart Cities, eHealth**, etc.

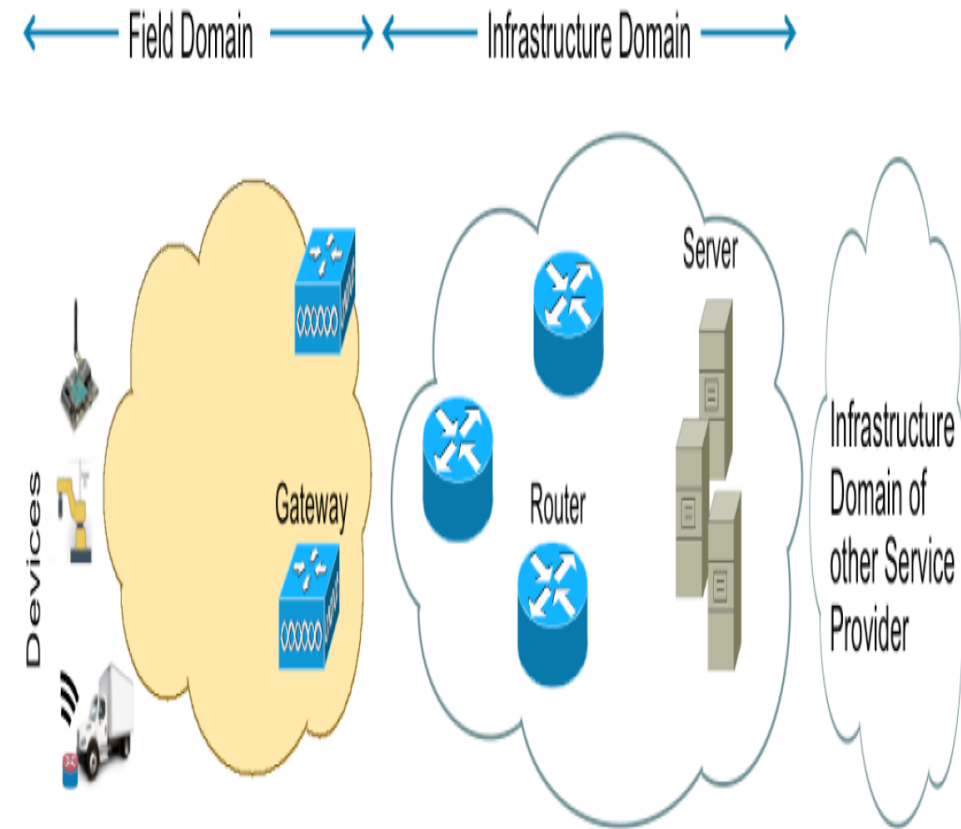


M2M Protocols



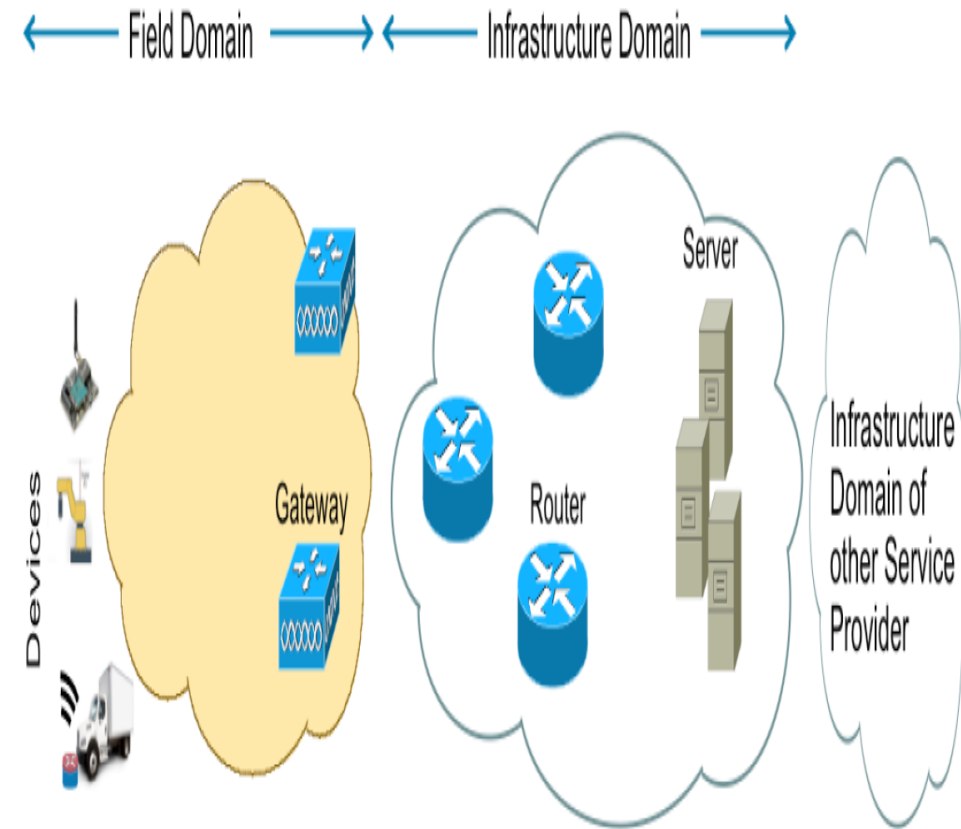
One M2M

- The **oneM2M standard** defines a **common framework** for IoT deployments, so that **devices, networks, and applications can interoperate seamlessly** — regardless of vendor or technology.



One M2M

- OneM2M standards consider any IoT deployment to be comprised of two domains:
- **Field Domain:** includes the Things (e.g. sensors, actuator) and gateways that collect or act on data.
- **Infrastructure Domain:** includes the **communication networks** (aggregation, core) and data centers that support large-scale IoT operations.



One M2M

Each of these domains includes three flavors of entities:

- Application Entity,
- Common Services Entity
- Network Services Entity.

➤ **Application Entity (AE):**

- **Purpose:** Implements the **application logic** for a specific IoT use case.
- **Functions:** Performs **domain-specific tasks** (e.g., monitoring, control, analytics).
- **Examples:**
 - **Home automation** AE (controls lights, thermostat)
 - **Smart parking** AE (detects free parking spaces)

One M2M

➤ Common Services Entity (CSE)

- **Purpose:** Acts as a **middleware layer** between applications (AEs) and the underlying network. It provides **common service functions** that **every application can use** — instead of **each AE having to implement them separately**.
- The Common Services Entity (CSE) provides the following set of common functions to applications:
 - Identity Management: Identification of Applications Entities and CSEs.
 - Data Exchange: Supports **storing & sharing of data between applications and devices**, in addition to event notification.
 - Security & Access control: **Provides control over access to data** (who can access what and when, etc.).

One M2M

➤ Network Services Entity (NSE)

- **Purpose:** Provides **network-level support and value-added services** to the Common Services Entity.
- **Example Providers:**
- Telecom operators (e.g., 4G/5G network services)

Entity	Main Role	Typical Location	Examples / Functions
AE (Application Entity)	Implements IoT application logic	Device, Gateway, Cloud	Smart Home, Smart Parking
CSE (Common Services Entity)	Middleware that provides core IoT functions	Gateway, Cloud	Registration, Security, Data Storage, Discovery
NSE (Network Services Entity)	Provides network-related services	Telecom / Internet network	QoS, Device Triggering, Location Services

06

Summary

1. IoT networking layers
2. M2M



Thank You