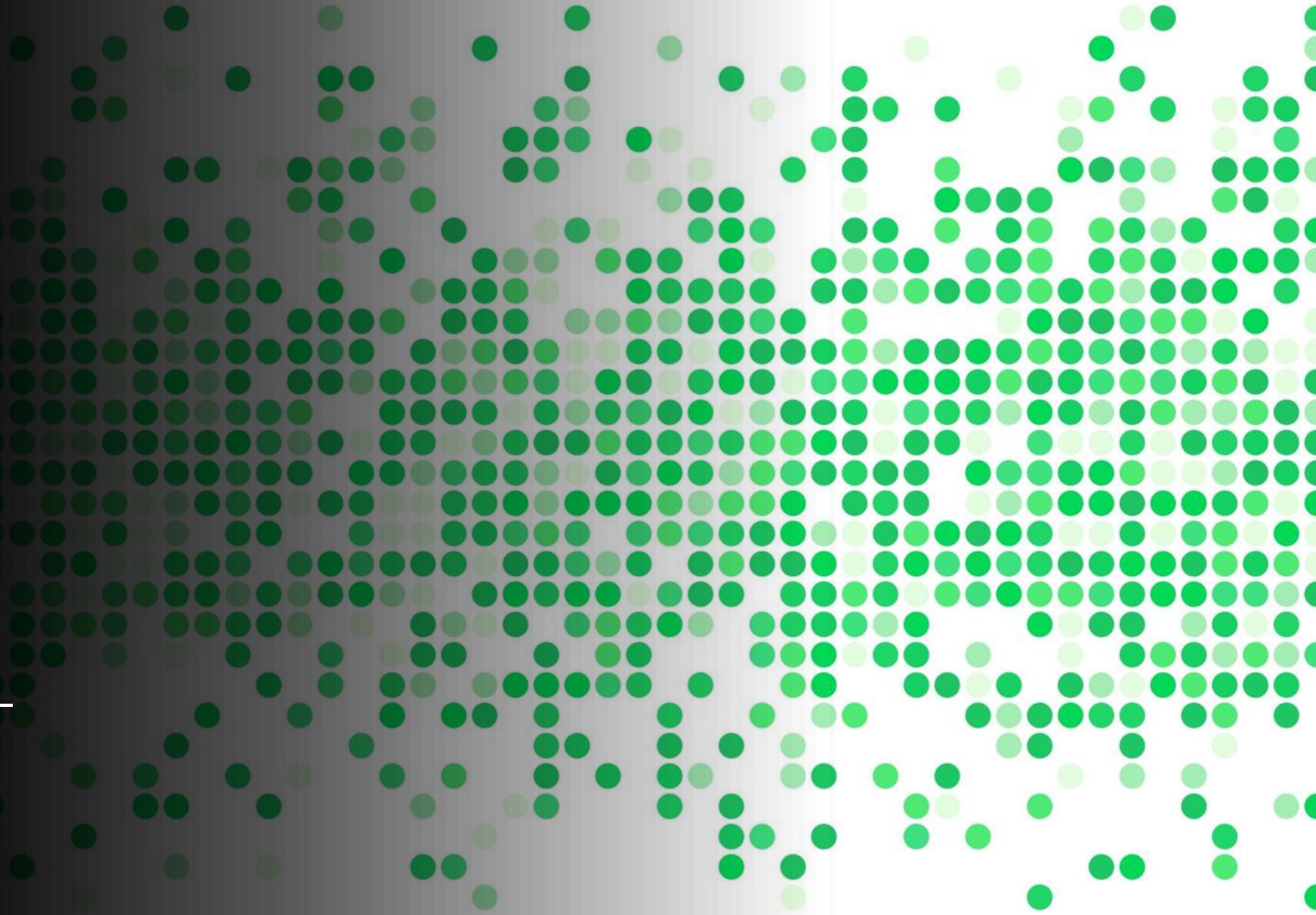
Integration Technologies and Tools for IoT Environments

Course Outcome:

Students will be able to Integrate IoT application with Cloud



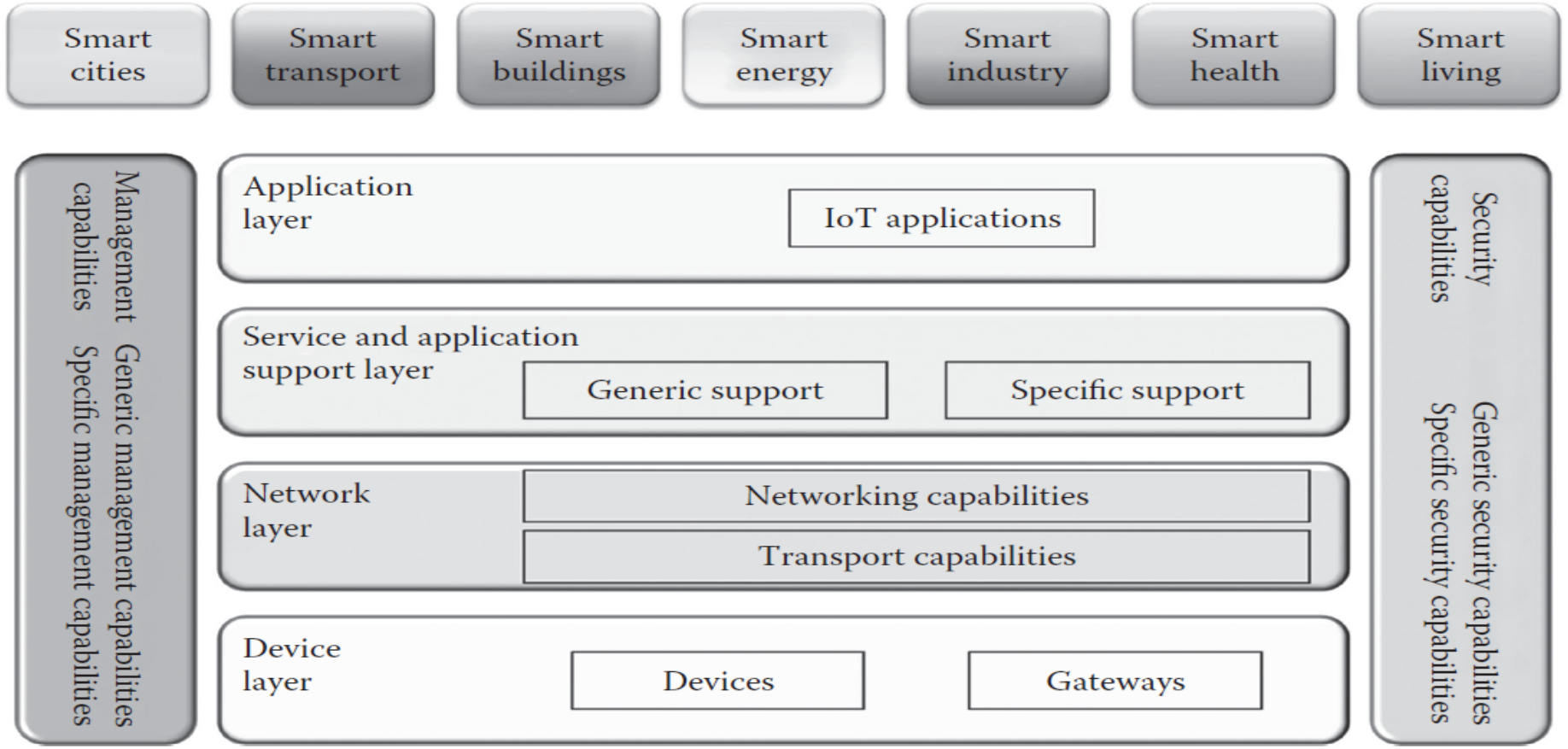
Topics to be Covered

- Sensor and Actuator Networks
- Sensor-to-Cloud Integration
- IoT Device Integration Concepts, Standards, Implementations
- Service Oriented Device Architecture, Device Profile for Web Services
- Open Service Gateway Initiative (OSGi), REST Paradigm
- Message Queue Telemetry Transport (MQTT)
- Advanced Message Queuing Protocol (AMQP)
- Constrained Application Protocol (CoAP).

What are IoT Devices Networking Needs?

IoT End Network Requirements	*Networking Style Impact*
Self-healing/scalable	Mesh capable
Secure	Scalable to no, low, medium, and high security without overburdening clients
End-node addressability	Device-specific addressing scalable to thousands of nodes
Device Requirements	*Messaging Protocol Impact*
Low power/battery-operated	Lightweight connection, preamble, packet
Limited memory	Small client footprint, persistant state in case of overflow
Low cost	Ties to memory footprint

IoT Reference Architecture

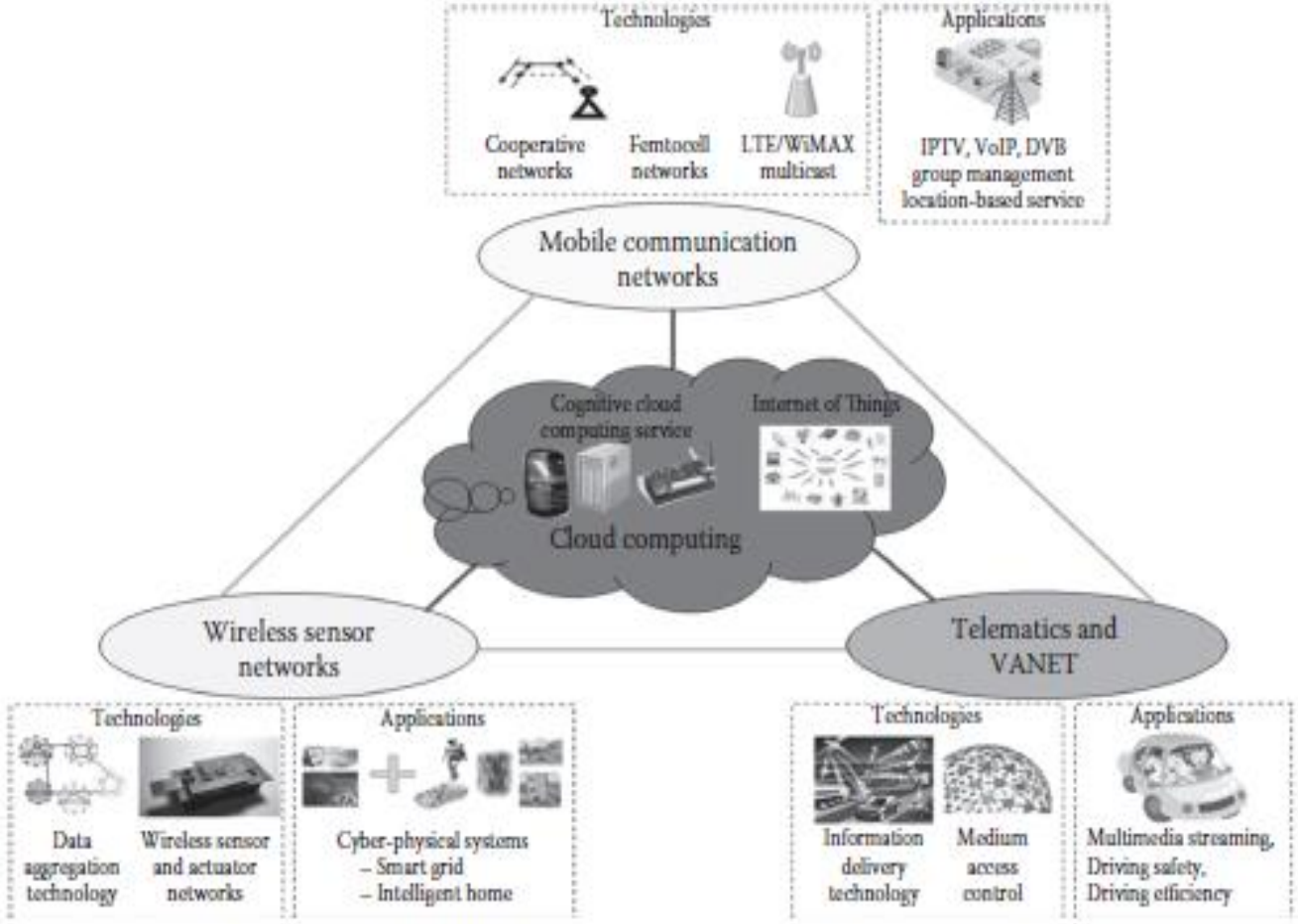


Source: From ITU Telecommunication Standardization Sector

Sensor and Actuator Network (S2AN)

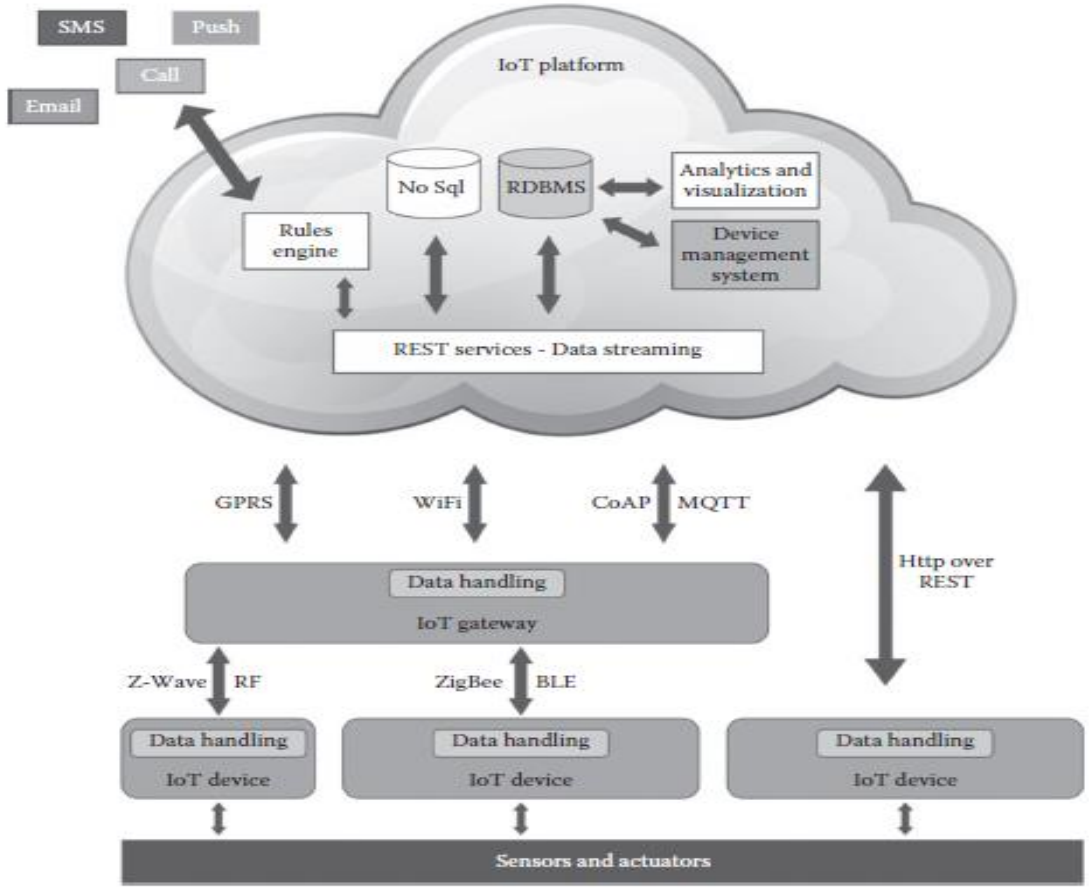
- Sensor gateways, middleware, brokers, adaptors, connectors, drivers, and controllers are being leveraged which use Sensor Data Fusion Algorithms

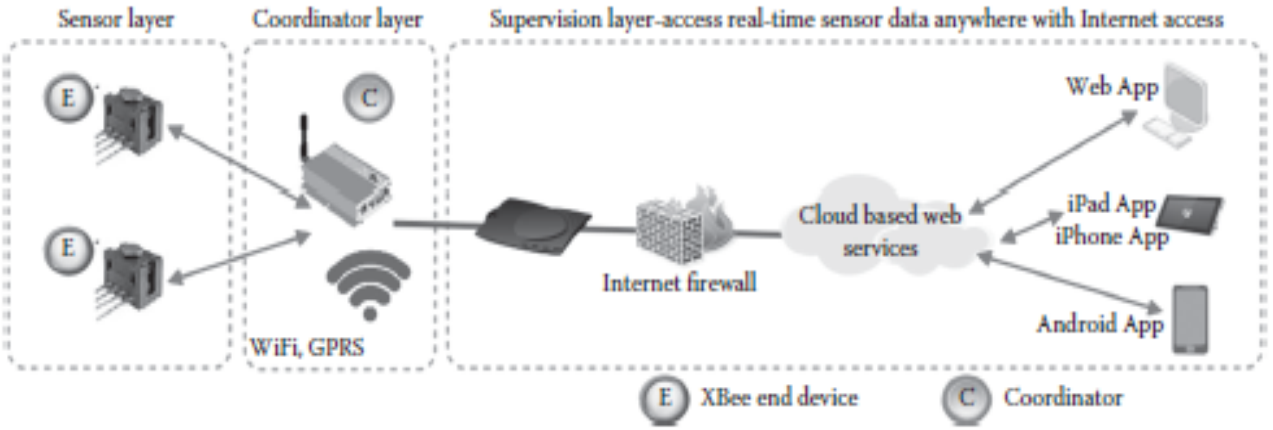
- Actuators are the ones that accomplish the execution based on the sensor findings.



Sensor to Cloud Network (S2CN)

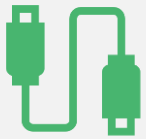
- Multiple cloud options
  - Off-premise
  - On-premise
  - Edge or Fog clouds
- Passaging Architecture of S2CN
- Layered Architecture of S2CN



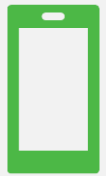


IoT Device Integration Concepts, Standards, and Implementations

M2M Communication: 5G, UWB, NFC etc



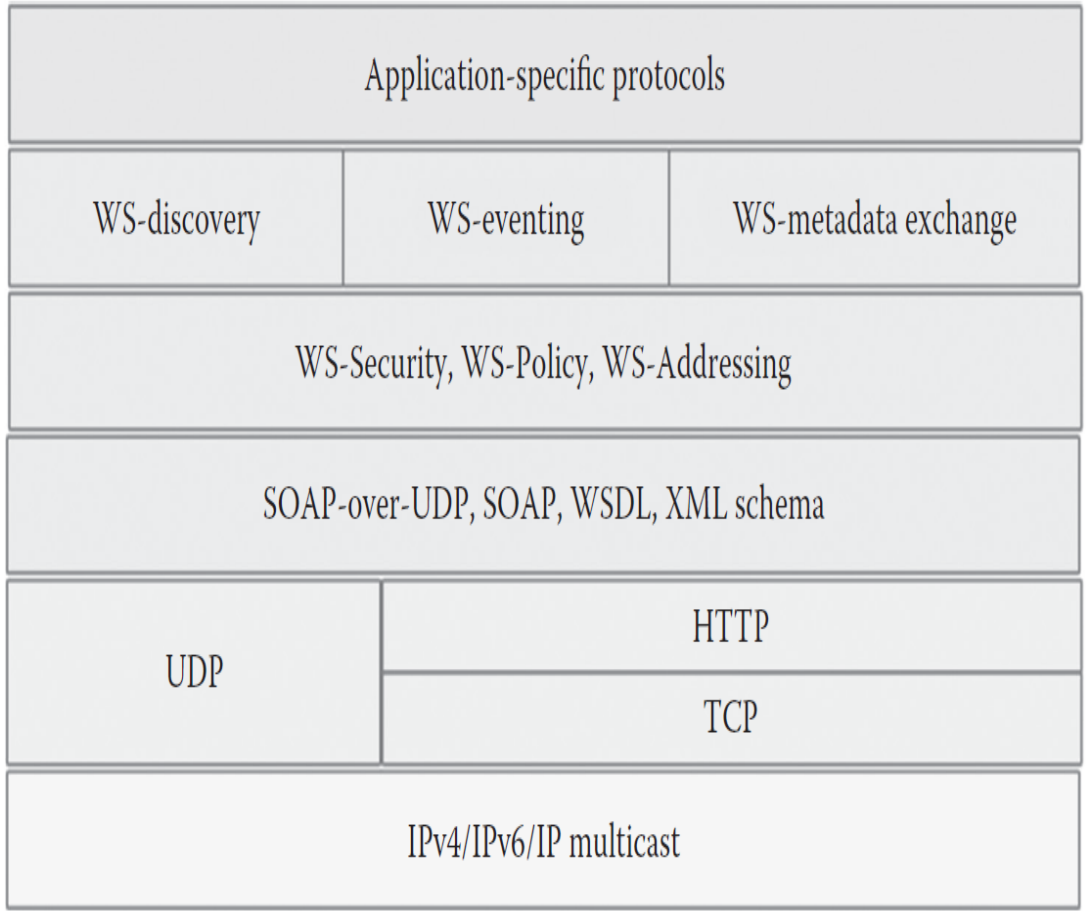
SODA for Device Integration: To enable service-based device integration.



Device Profile for Web Services: Using REST-less or SOAP messages.

Device Profile Web Service (DPWS)

- **DPWS is a widespread standard for service design, development, deployment & integration for all kinds of devices that are much deviated from typical compute machines.**
- **DPWSim toolkit to support building device applications that in turn use the distinctive services being offered by low as well as high-end devices.**



DPWS OSI Model

http://sourceforge.net/projects/dpwsim/

https://github.com/sonhan/dpwsim

Open Gateway System Interconnect (OGSi)

- An independent, non-profit corporation defined and promoted open specifications for the delivery of ***managed services*** to networked environments, such as homes and automobiles.
- OSGi framework sits on top of a ***JVM*** and is the execution environment for services
- OSGi brings in the built-in support for describing and ***handling modules and their dependencies***.
- Every single functional module can be updated to a newer version ***without restarting*** the application.
- Services need to be executed as Java bundles on the ***central gateway***.
- The bundles then communicate to the devices using ***proprietary*** means. Thus, the proxy approach is mandatory in OSGi
- The topology, which describes the network setup of a device-centric SOA is **Star topology**.

Environments to Explore

- Eclipse  is an Eclipse IoT project that provides a platform for building IoT gateways
- It is a smart application container that enables remote management of gateways and provides a wide range of APIs for allowing you to write and deploy your own IoT application.

-  is a software for integrating different home automation systems and technologies into one single solution that allows overarching automation rules and that offers uniform user interfaces

https://www.eclipse.org/kura/
https://www.openhab.org/download/

Device Integration Protocols and Middleware

- MQTT: A protocol for collecting device data and communicating it to servers (D2S)
- XMPP: A best protocol for connecting devices to people, a special case of the D2S pattern, since people are connected to the servers
- AMQP: A queuing system designed to connect servers to each other (S2S)
- CoAP (Shelby et al. 2013): An optimized protocol

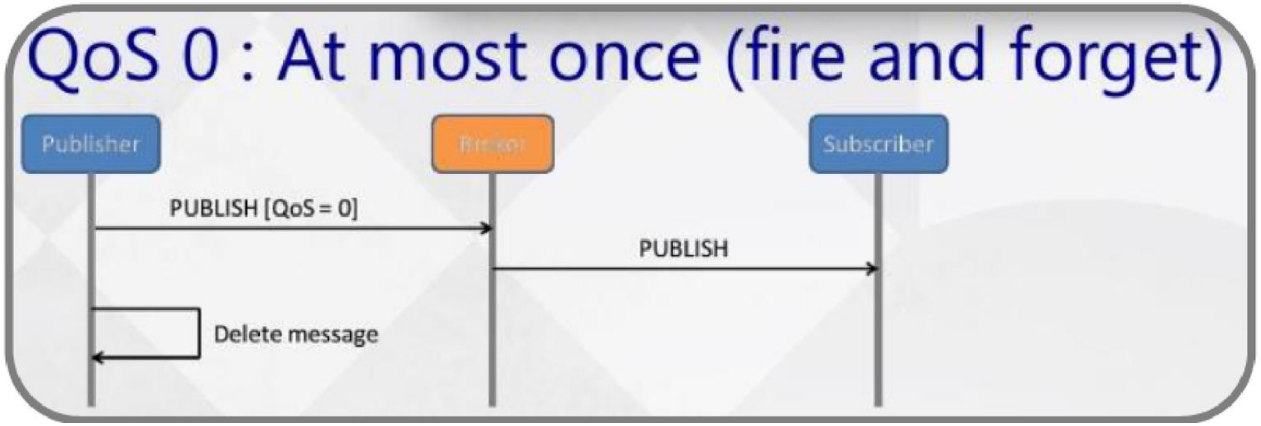
Message Queue Telemetry Transport

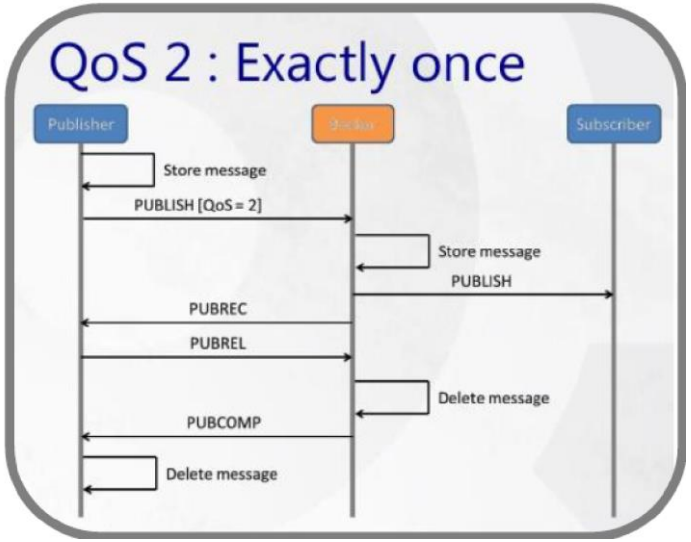
- Publish-Subscribe based lightweight messaging protocol for use in conjunction with the TCP/IP protocol
- Designed to provide connectivity (mostly embedded) between applications and middle-wares on one side and networks and communications on the other side
- Components:
  - Publisher: Lightweight Sensors
  - Subscriber: Applications awaiting for sensor data
  - Broker: Interface between Pub-Sub & classifies sensor based data using Topics

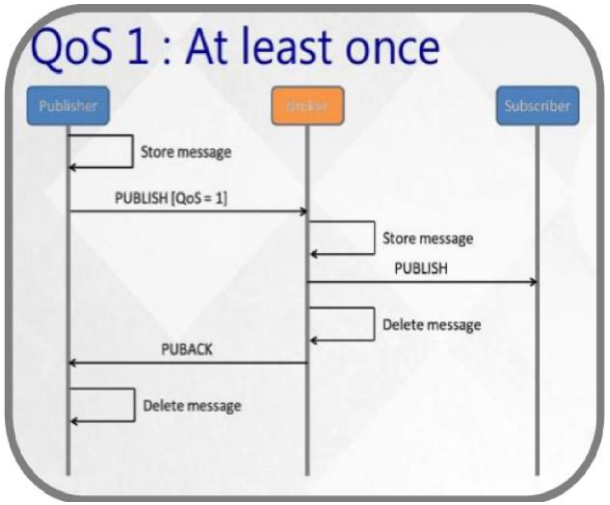
Message Queue Telemetry Transport

- Methods:
  - **Connect**: Waits for a connection to be established with the server
  - **Disconnect**: Waits for the MQTT client to finish any pending task and closes the TCP session
  - **Subscribe**: Request the server to subscribe the client to one or more topics,
  - **Unsubscribe**: Request the server to subscribe the client to one or more topics
  - **Publish**: Updates a topic with data

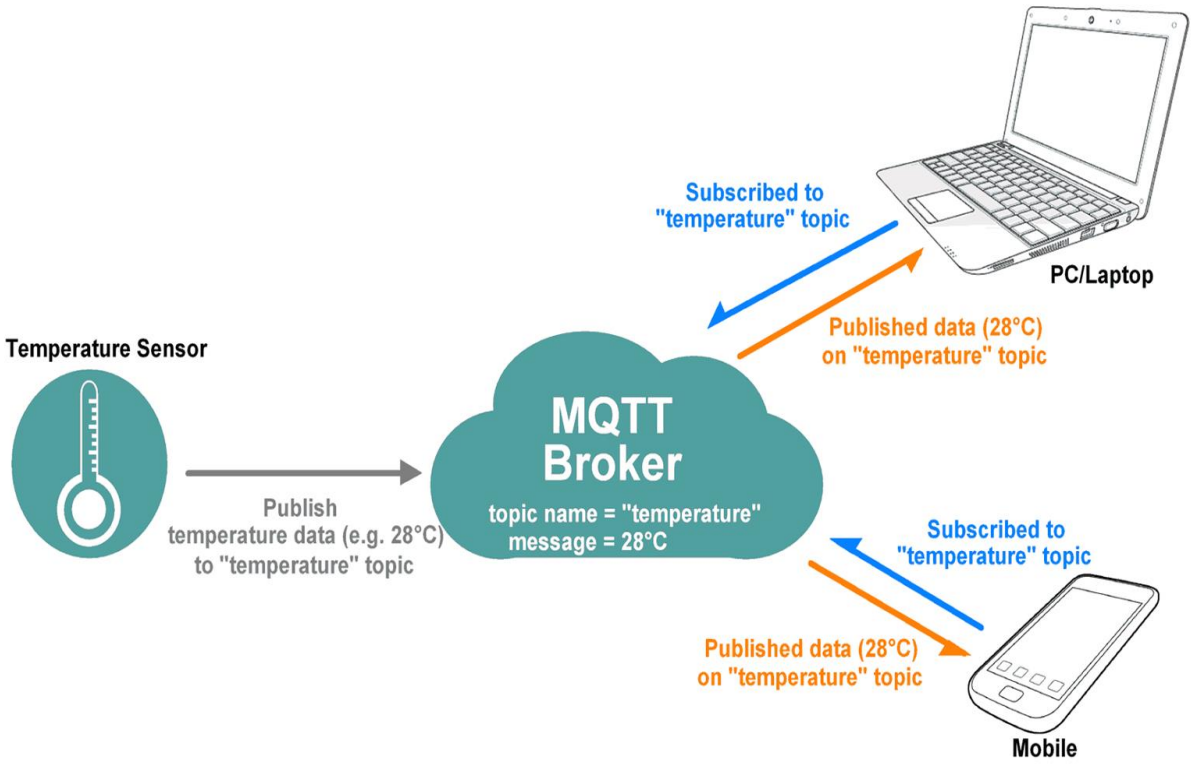
MQTT Quality of Service







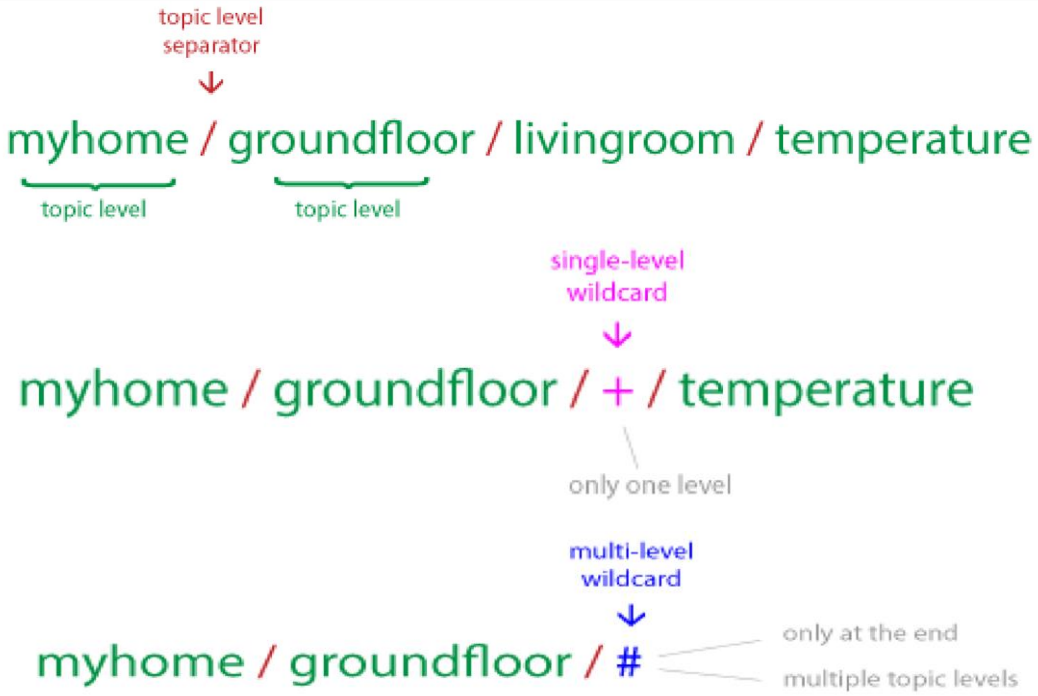
MQTT Example



Source: medium.com/@Jaidev.Dave93

MQTT Example

- A **topic** is a simple string that can have more hierarchy levels, which are separated by a '/'
- Three wild cards **+, #, $**
- If a subscriber listens to the topic myhome/+/temperature ==> Looks for temperature in any room in that house
- If a subscriber listens to the topic myhome/# ==> Looks for any sensor included in any room of this house
- Topics starting with $ are special and are reserved for the broker statistics



$SYS/broker/clients/connected
$SYS/broker/clients/disconnected
$SYS/broker/clients/total
$SYS/broker/messages/sent
$SYS/broker/uptime

MQTT Demo using Node-Red

- MOSCA is a Nodejs MQTT Broker & Server
- Available as a node for Node-Red
- Useful for simple testing and automation
- Not comparable to Mosquitto but is simple

MQTT Demo using Node-Red

- Install MOSCA in Node-Red as a Pallet node
- Configure MQTT Broker using MOSCA, MQTT Publisher and Subscriber.
- Include the inject and debug nodes for giving input and recording output
- Deploy the project and enable debug on debug nodes
- See Pub-Sub communication using MQTT

Source

Applications of MQTT

- EVERYTHNG IoT platform uses MQTT as an M2M protocol for millions of connected products.
- Adafruit launched a free MQTT cloud service for IoT experimenters called Adafruit IO.

Advanced Message Queueing Protocol (AMQP)

- Designed as an open replacement for existing proprietary messaging middleware
- Important uses: reliability and interoperability
- Plenty of fine-grained control possible with such a rich feature set.
- AMQP is a binary wire protocol, which was designed for interoperability between different vendors.
- JP Morgan use it to process one billion messages a day.
- NASA uses it for Nebula cloud computing
- Google uses it for complex event processing
- India uses is as a messaging platform for Aadhar project
- Ocean observatories initiative—an architecture that collects 8 TB/day

AMQP vs MQTT

AMQP	MQTT
Offers a wealthier range of messaging circumstances	Implemented mostly in embedded systems
Uses TCP for asynchronous transfer of messages with varied use of N/W & Infra	Uses TCP for asynchronous transfer of messages with minimum bandwidth over the N/W
Uses Buffering to increase performance of servers	Executes frames for minimum memory devices
Supports different transactions, use cases along with the message queues, providing ack's.	Doesn't support any kind of transaction and provide only general purpose ack's
Unified with TLS	Doesn't act to any security issues
Supports SASL for authentication	Supports basic authentication with small usernames & passwords

Representational State Transfer (REST)

- An architectural pattern for ***uniformly accessing and modifying*** a resource.
- The current popular REST model uses ***URIs to identify objects*** (*/lamp/1234*), ***HTTP verbs*** to specify an action, and ***JSON*** to represent the object.
- RESTful systems are ***loosely-coupled*** systems that follow the above principles to exchange application states as resource representations.
- This kind of ***stateless*** interactions improves the resources consumption and the scalability of the system.
- Distinct advantages over arbitrary SOAP are ***less overhead***, low parsing-complexity, statelessness, and tighter integration with HTTP.

RESTFul vs RESTless

RESTFUL WEB SERVICE	RESTLESS WEB SERVICE
An application that conforms to the REST architectural style that provides interoperability between computer systems on the internet	An application that is not based on the principles of REST
Use REST	Use SOAP
Support various data formats such as HTML, JSON, text etc.	Support XML format
Use URL to expose business logic	Use the service interface to expose business logic

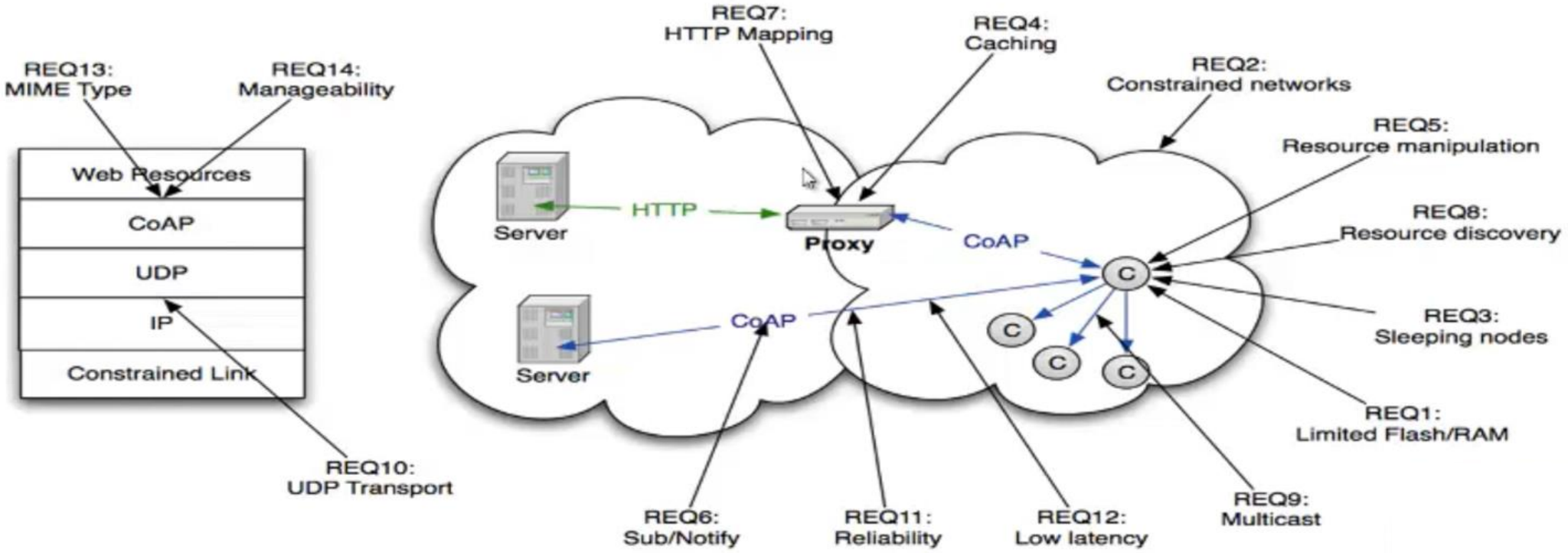
RESTFUL WEB SERVICE	RESTLESS WEB SERVICE
Easier and flexible	Not as easy or flexible
Inherits security measures from the underlying transport protocols; therefore, it is less secure	Defines its own security layer and is more secure
Consume less bandwidth and resource	Consume more bandwidth and resources

Constrained Application Protocol (CoAP)

- **Web transfer protocol** for use with constrained nodes and networks.
- **Designed for Machine to Machine** (M2M) applications such as smart energy and building automation.
- Based on **Request-Response model** between end-points
- Client-Server interaction is **asynchronous over a datagram oriented transport protocol** such as UDP
- **Session layer protocol** designed by IETF Constrained RESTful Environment (CoRE) working group to provide lightweight RESTful (HTTP)interface.
- Designed to **enable low-power sensors** to use RESTful services while meeting their power constraints.
- **Built over UDP**, instead of TCP (which is commonly used with HTTP) and has a light mechanism to provide reliability

Constrained Application Protocol (CoAP)

CoAP Design Requirements

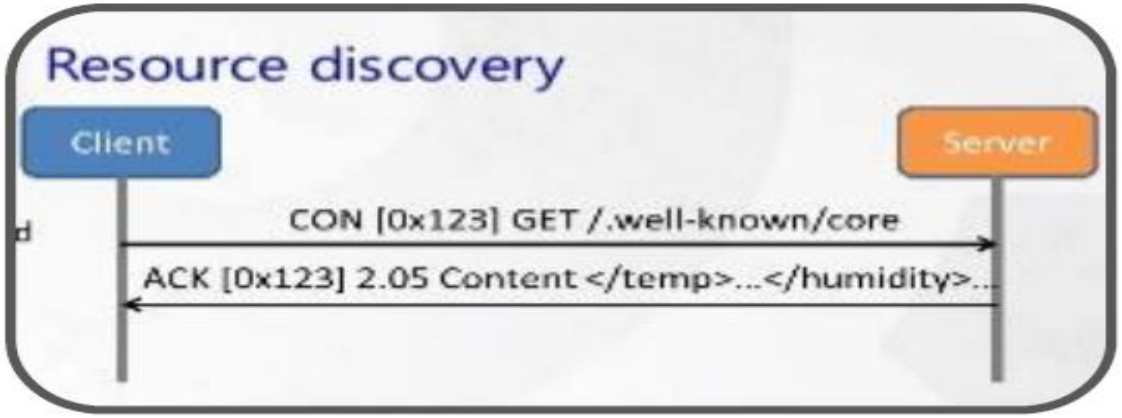


ARM

Constrained Application Protocol (CoAP)

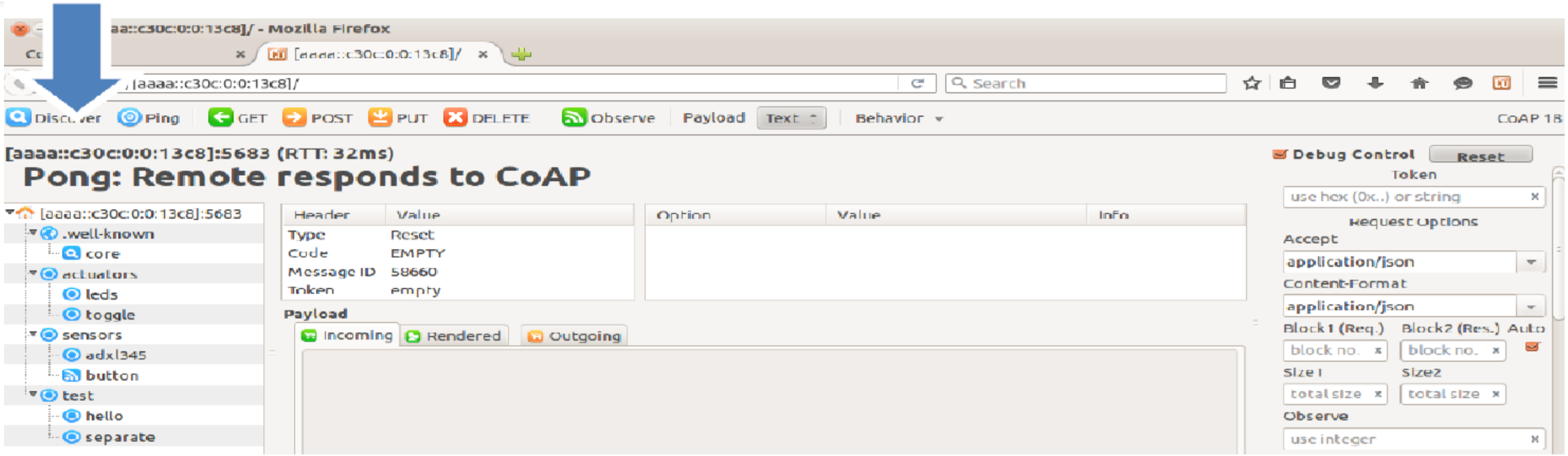
- CoAP architecture is divided into two main sub-layers:
  - Messaging
  - Request/response.
- The messaging sub-layer is responsible for reliability and duplication of messages, while the request/response sub-layer is responsible for communication.
- CoAP has four messaging mode
  - Confirmable
  - Separate
  - Non-confirmable
  - Piggyback

Constrained Application Protocol (CoAP)



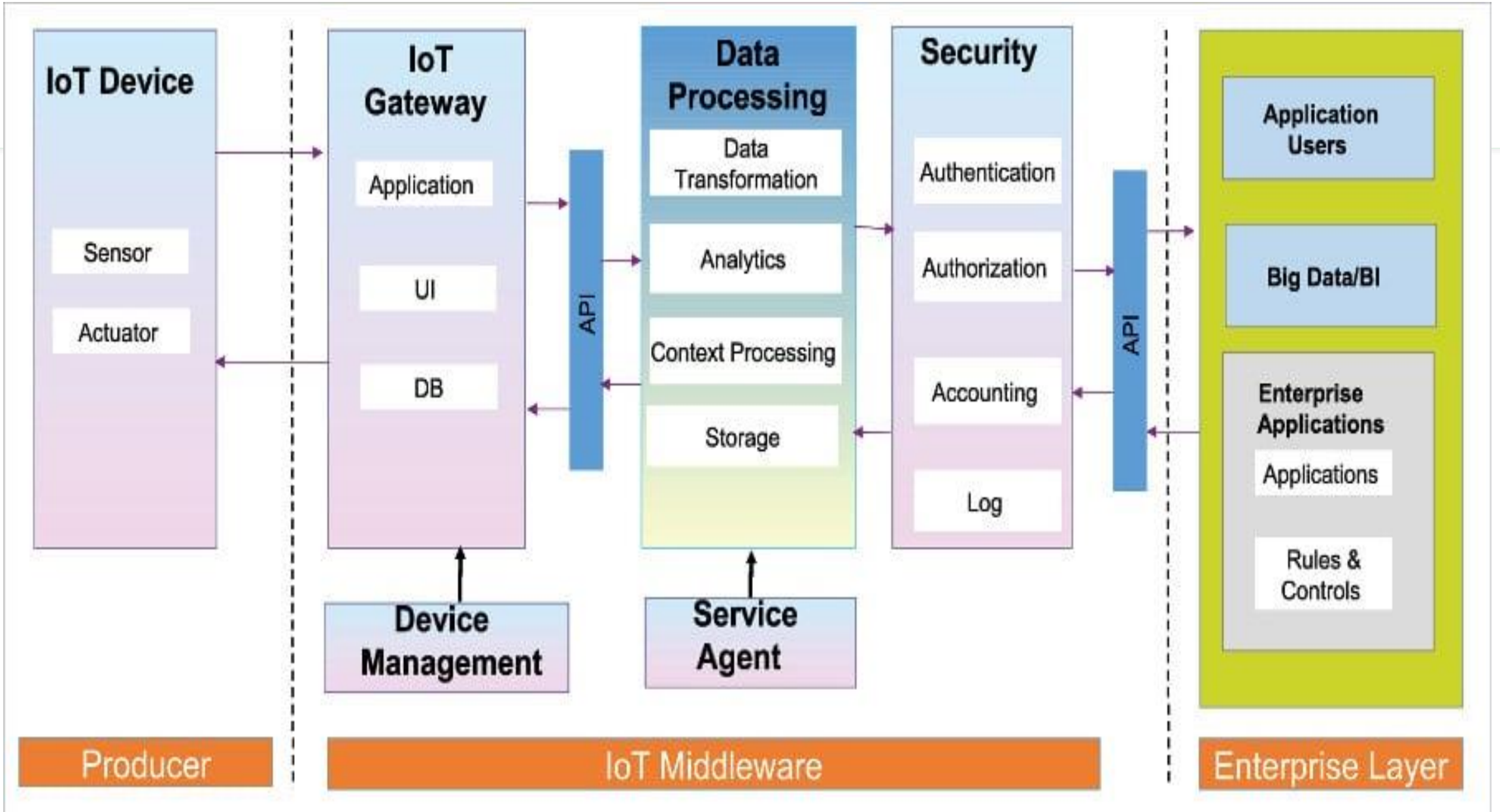
Constrained Application Protocol (CoAP)

Discover – learn the Resources the CoAP server has



examples/zolertia/tutorial/03-coap/resources/er-example-server.c

IoT Middleware



Functions of Middleware

- Device abstraction, discovery, management, and control: It includes interoperation among the heterogeneous connected devices/things using different standards. Application programming interfaces (APIs) are used for abstracting the communication with the physical layer and also for disseminating data and services to the different connected backend applications, hiding all details and complexities.
- Data management and dissemination: It provides the different data preprocessing functionalities, such as filtering, duplicate removal, aggregation.
- Context detection and processing
- Security, privacy, and business rules processing

Popular IoT Middleware

EclipseIoT (Kura)

Kaa is platform-centric middleware.

SiteWhere

IoTSyS

DeviceHive

Zetta

OpenIoT

ThingsBoard

NATS.io