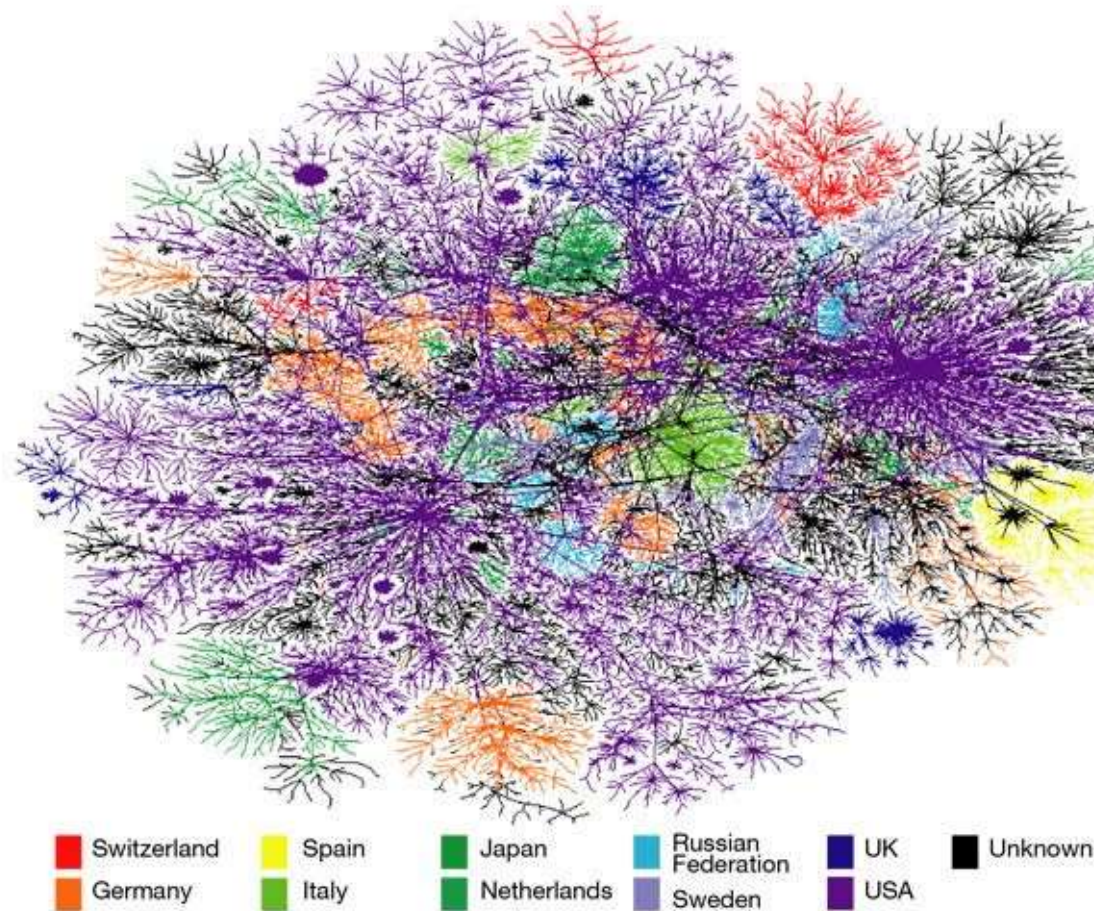


# Internet of Things

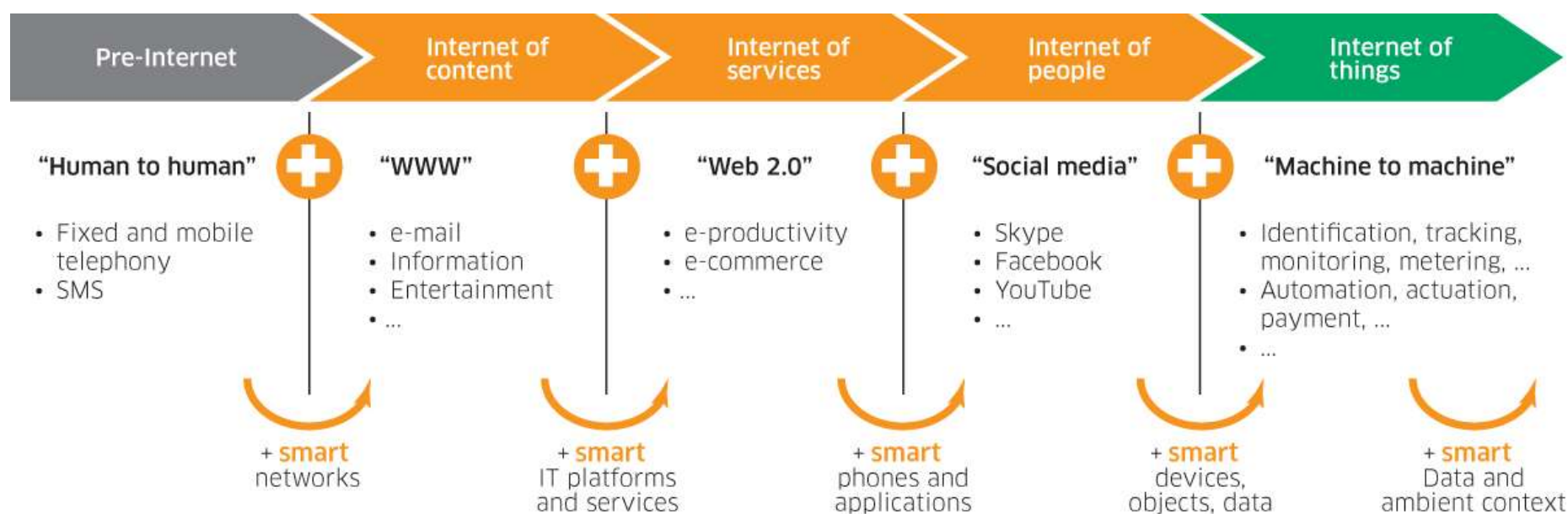
Photo by Mathew Schwartz on Unsplash

# The Internet



Yuhai Tu, "How robust is the Internet?", <https://www.nature.com/articles/35019222>

# IoT Evolution



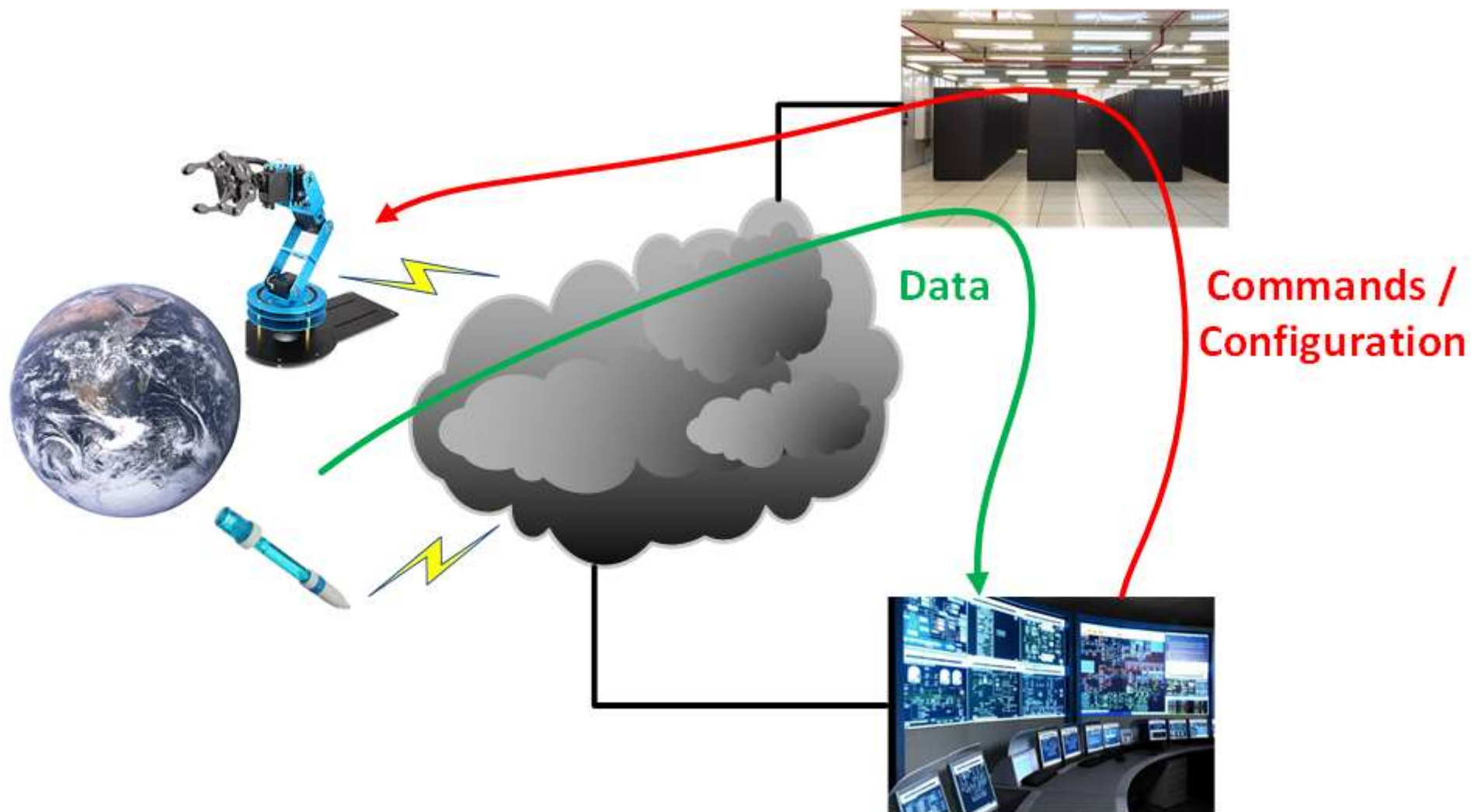
# What is IoT?

---

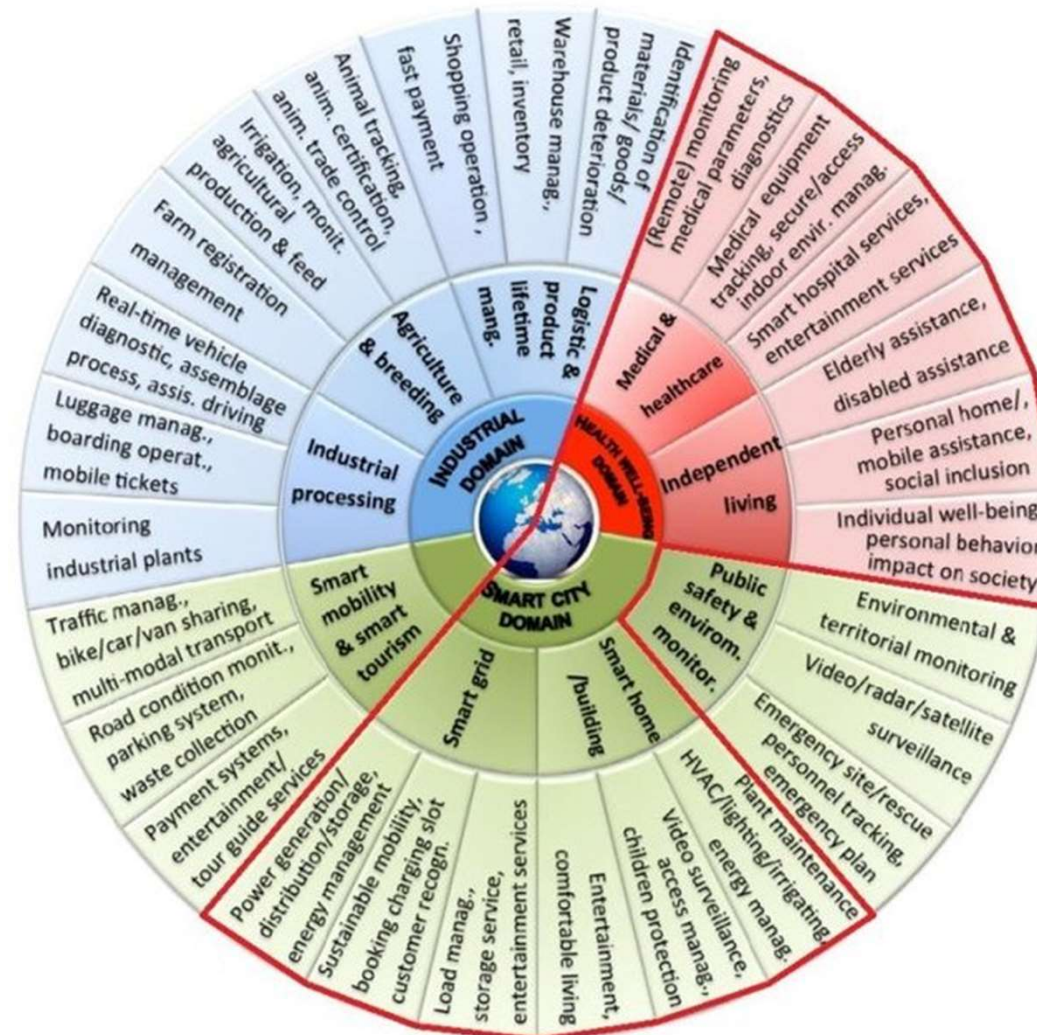
- 📶 Internet of Things (IoT) comprises **things** that have unique identities and are connected to the Internet
- 📶 The focus on IoT is in the monitoring, configuration, control and networking via the Internet of all sort of devices or “Things”
  - 📶 Eg: smartphone sensors, pump, utility meter, car engine
- 📶 IoT is a new revolution in the capabilities of the **endpoints that are connected to the internet**



# IoT: A Global Cyberphysical System

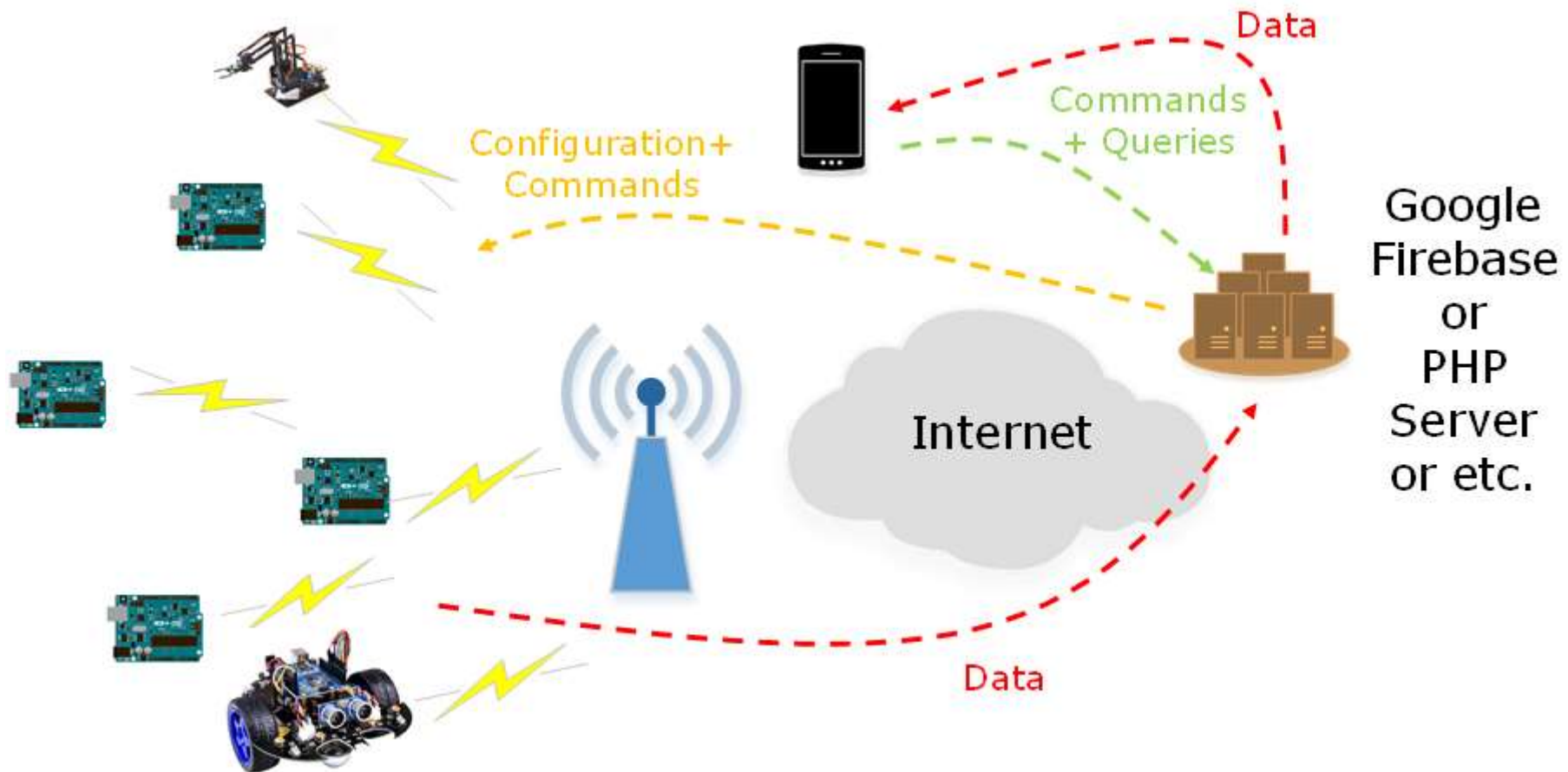


# IoT: Application Areas

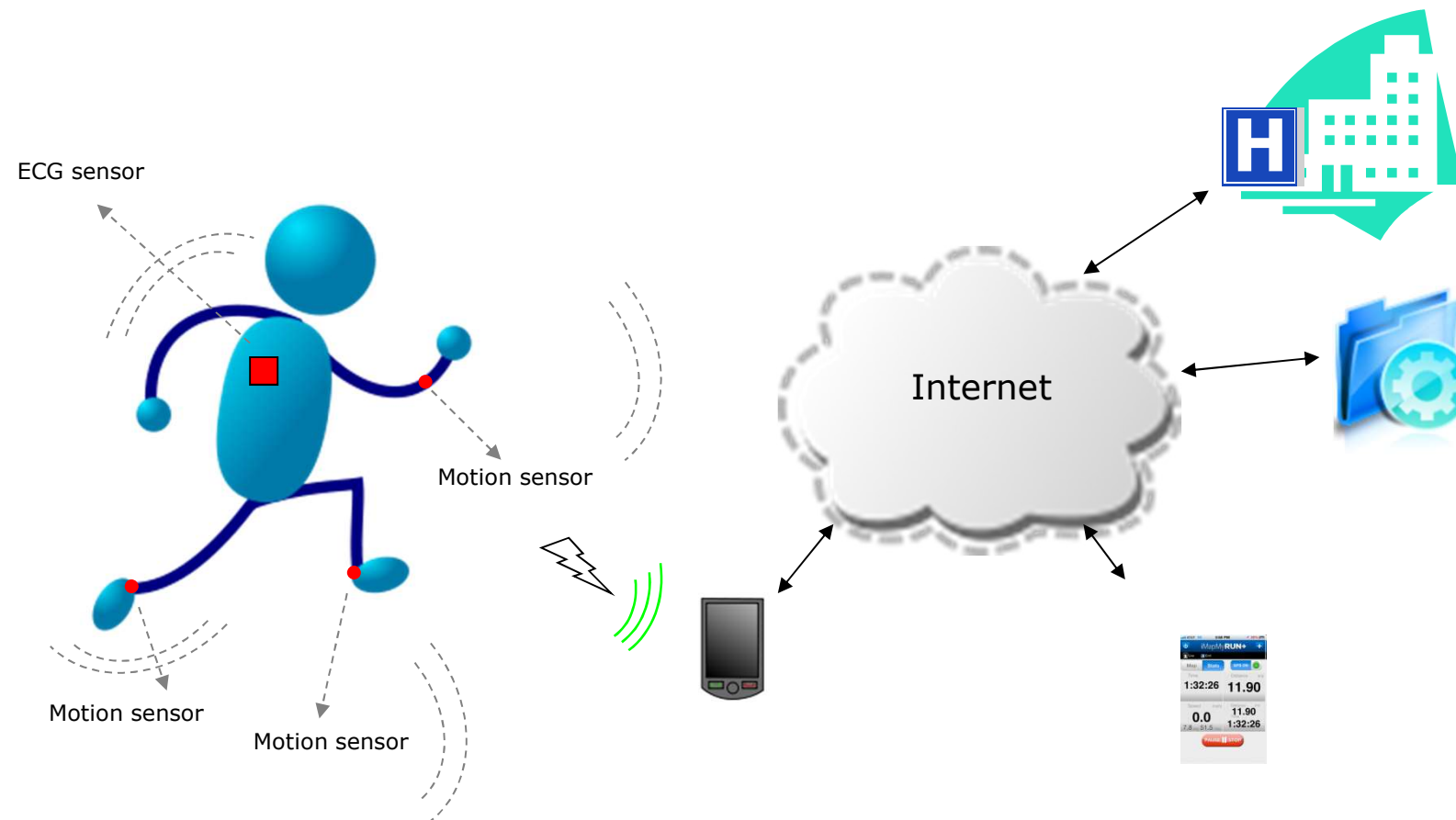


B. Alhafidh, W. Allen, "Design and Simulation of a Smart Home managed by an Intelligent Self-Adaptive System", Int. Journal of Engineering Research and Application, ISSN : 2248-9622, Vol. 6, Issue 8, ( Part -1) August 2016, pp.64-90.

# IoT: Typical Application Architecture



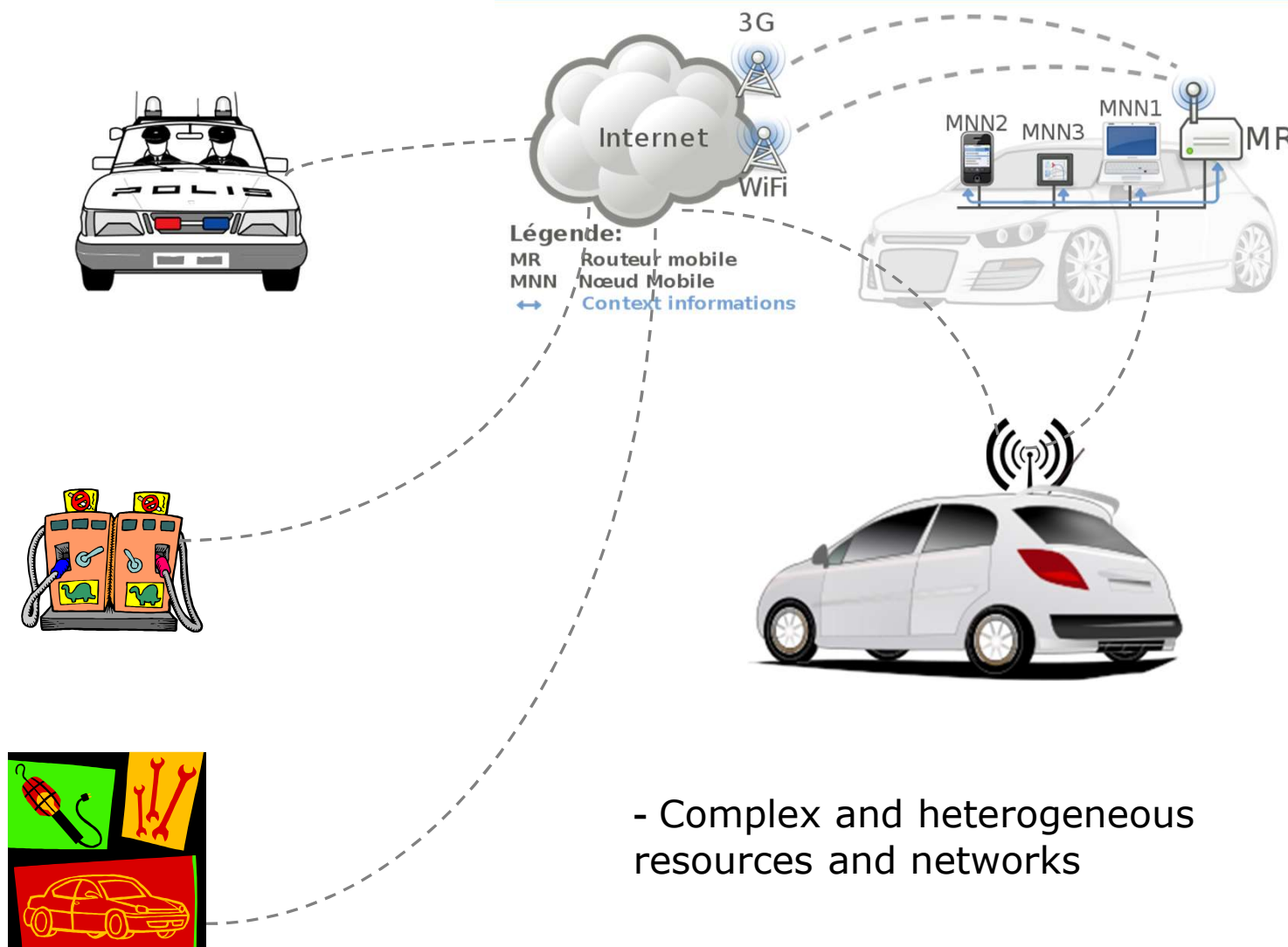
# IoT: People Connecting with Things



Dr.Kayarvizhy, "Internet of Things", <http://studyslide.com/doc/20337/iot---dr-kayarvizhy>



# IoT: Things Connecting with Things



- Complex and heterogeneous resources and networks

Dr.Kayarvizhy, "Internet of Things", <http://studyslide.com/doc/20337/iot---dr.-kayarvizhy>

# IoT Architecture

Integrated  
Application



Smart Grid

Green  
Building

Smart Transport

Env.  
Monitor

Information  
Processing



Data Center Search Engine Smart Decision Info. Security Data Mining

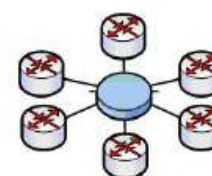
Network  
Construction



WWAN



WPAN



Internet



WMAN



WLAN

Sensing &  
Identification



GPS



Smart  
Device



RFID

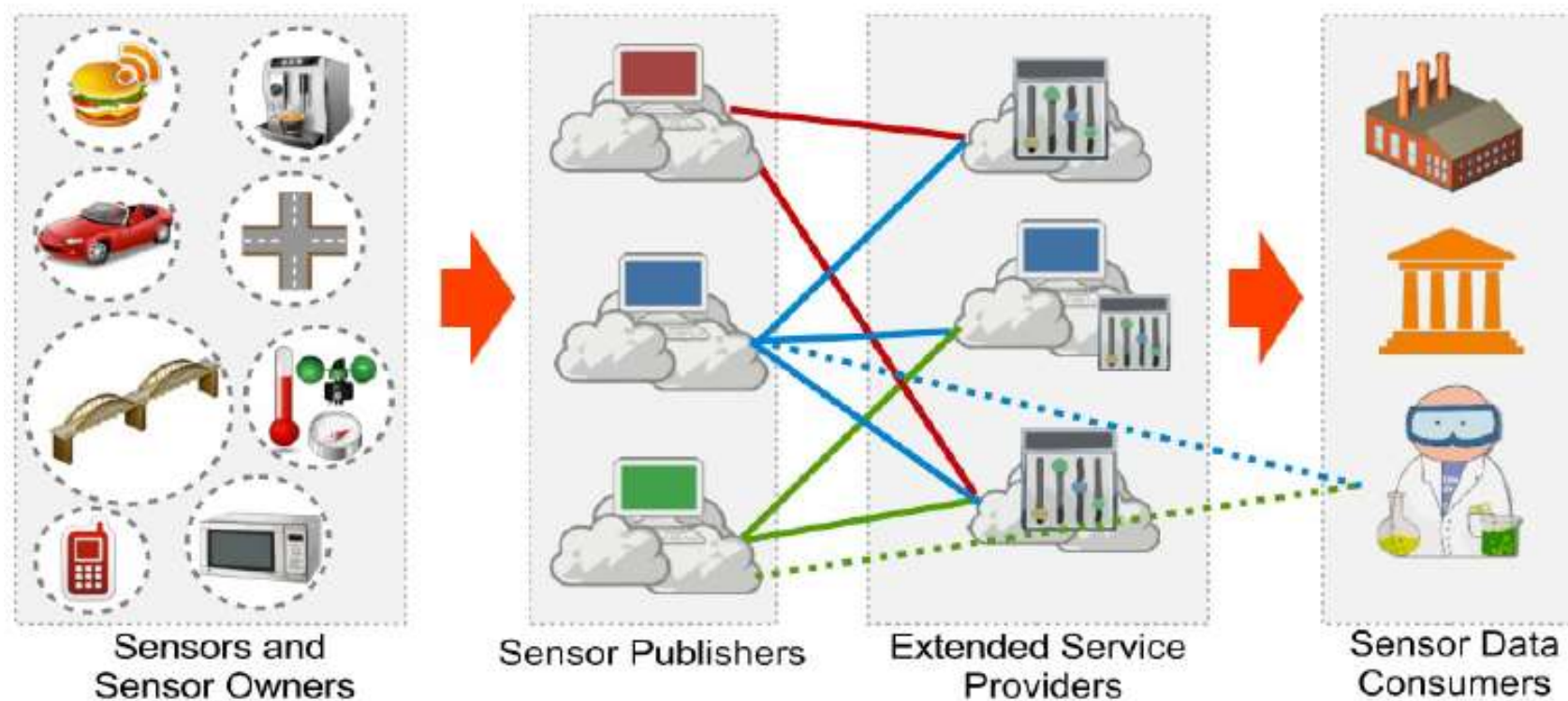


Sensor

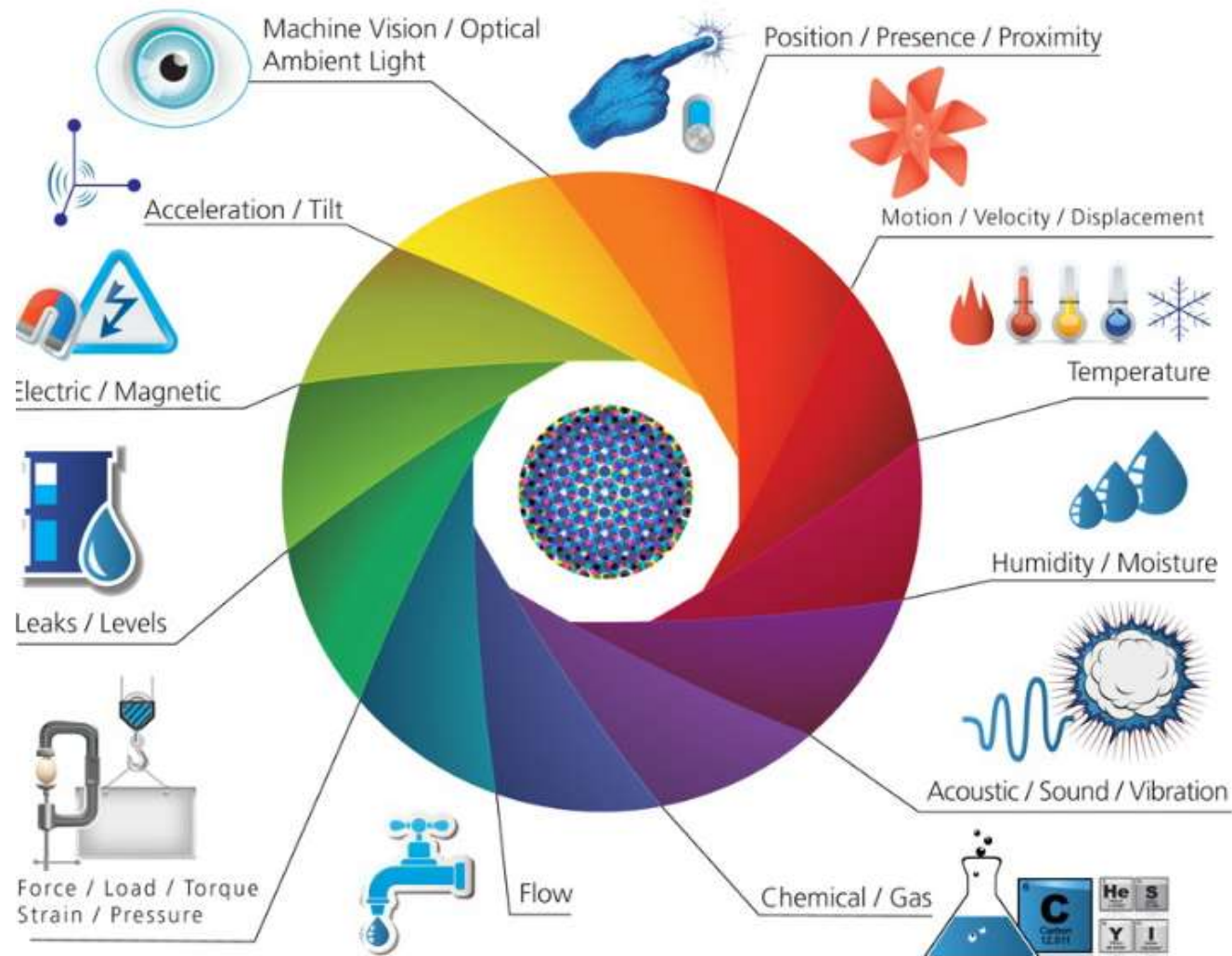


Sensor

# IoT Cloud: Sensing-As-A-Service Model



# IoT: Sensors and Actuators



Dr.Kayarvizhy, "Internet of Things", <http://studyslide.com/doc/20337/iot---dr.-kayarvizhy>

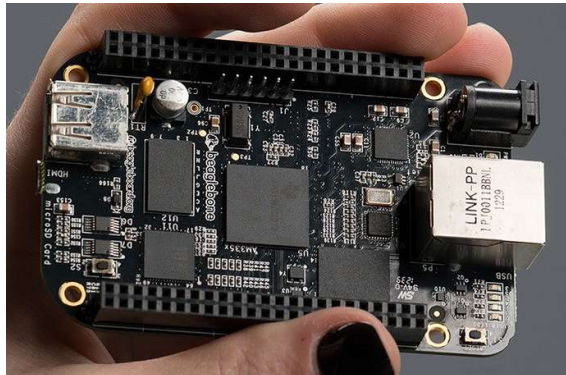


# IoT: Sensors Available in the Market (examples)

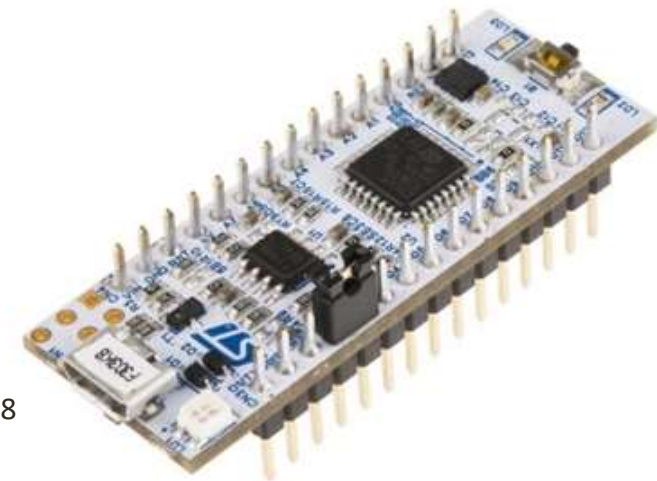


Dr.Kayarvizhy, "Internet of Things", <http://studyslide.com/doc/20337/iot---dr.-kayarvizhy>

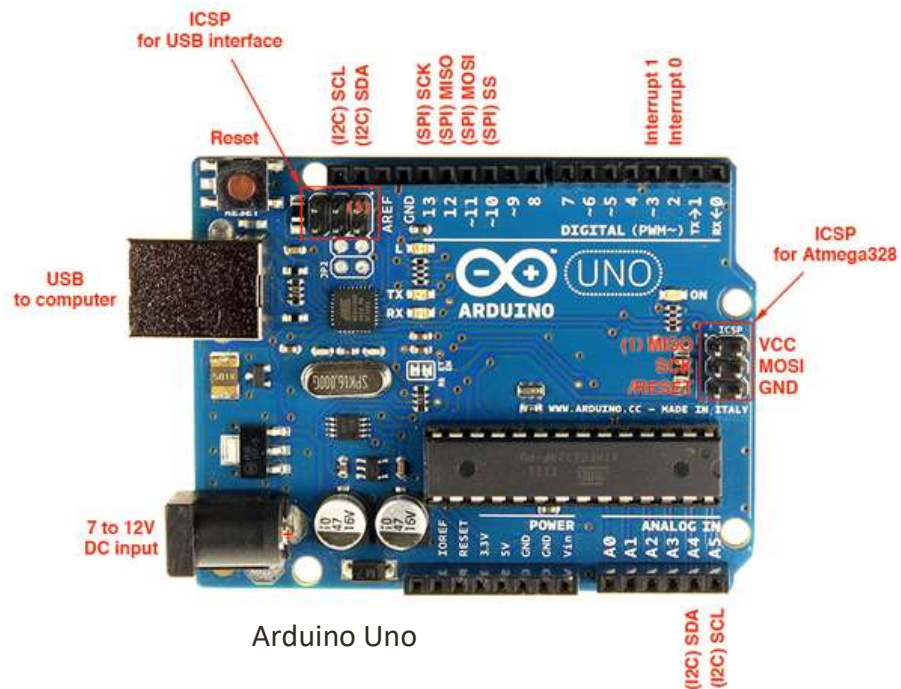
# IoT: Smart Objects (examples)



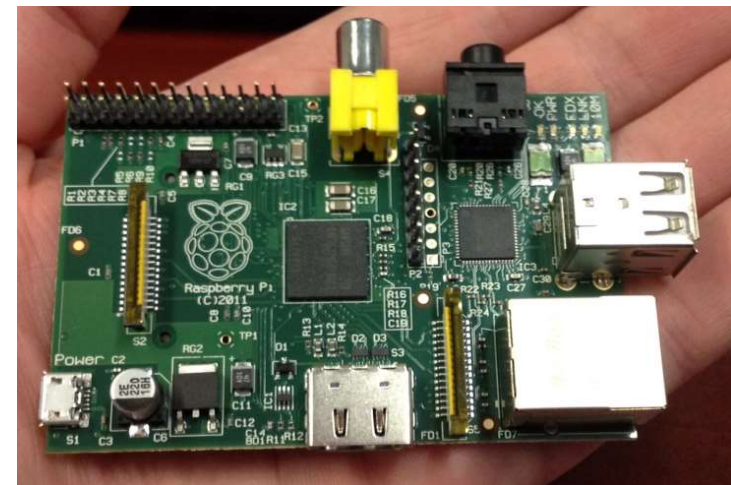
Beaglebone black



Nucleo-F303K8



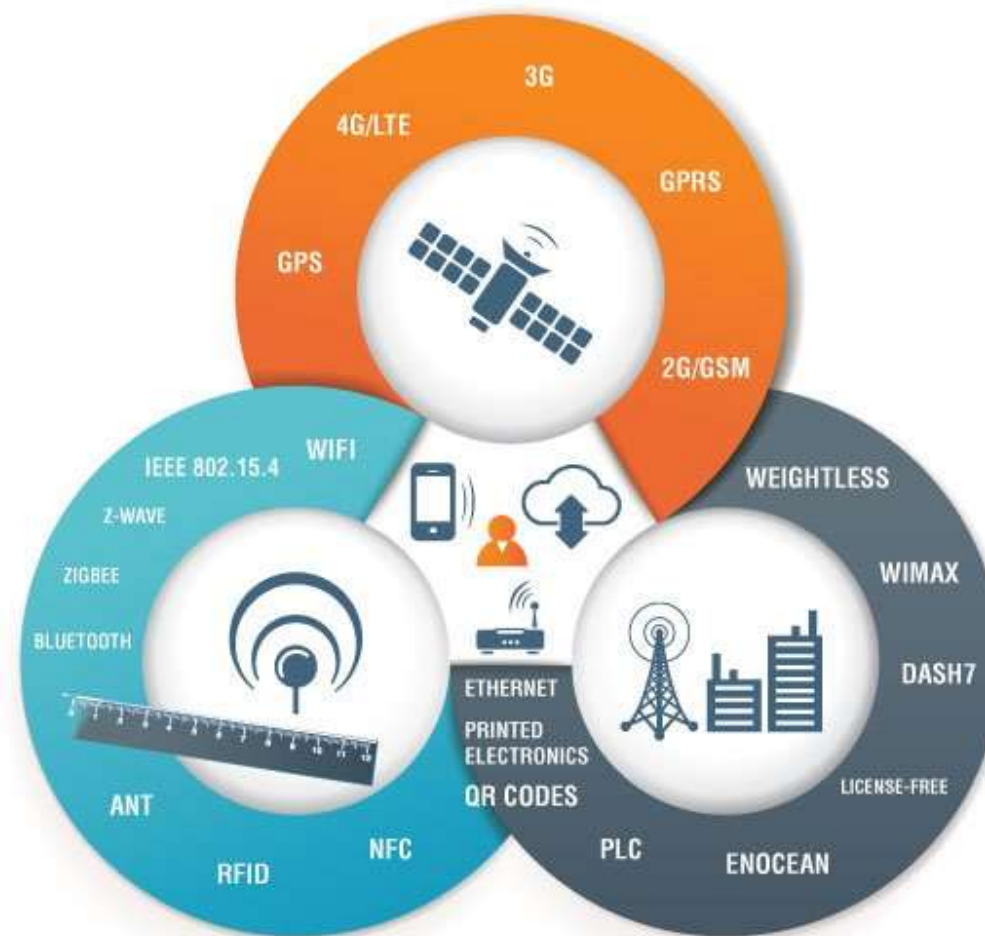
Arduino Uno



Raspberry Pi



# IoT Communication Technologies



Dr.Kayarvizhy, "Internet of Things", <http://studyslide.com/doc/20337/iot---dr.-kayarvizhy>

## Unified & Horizontal IoT Platform

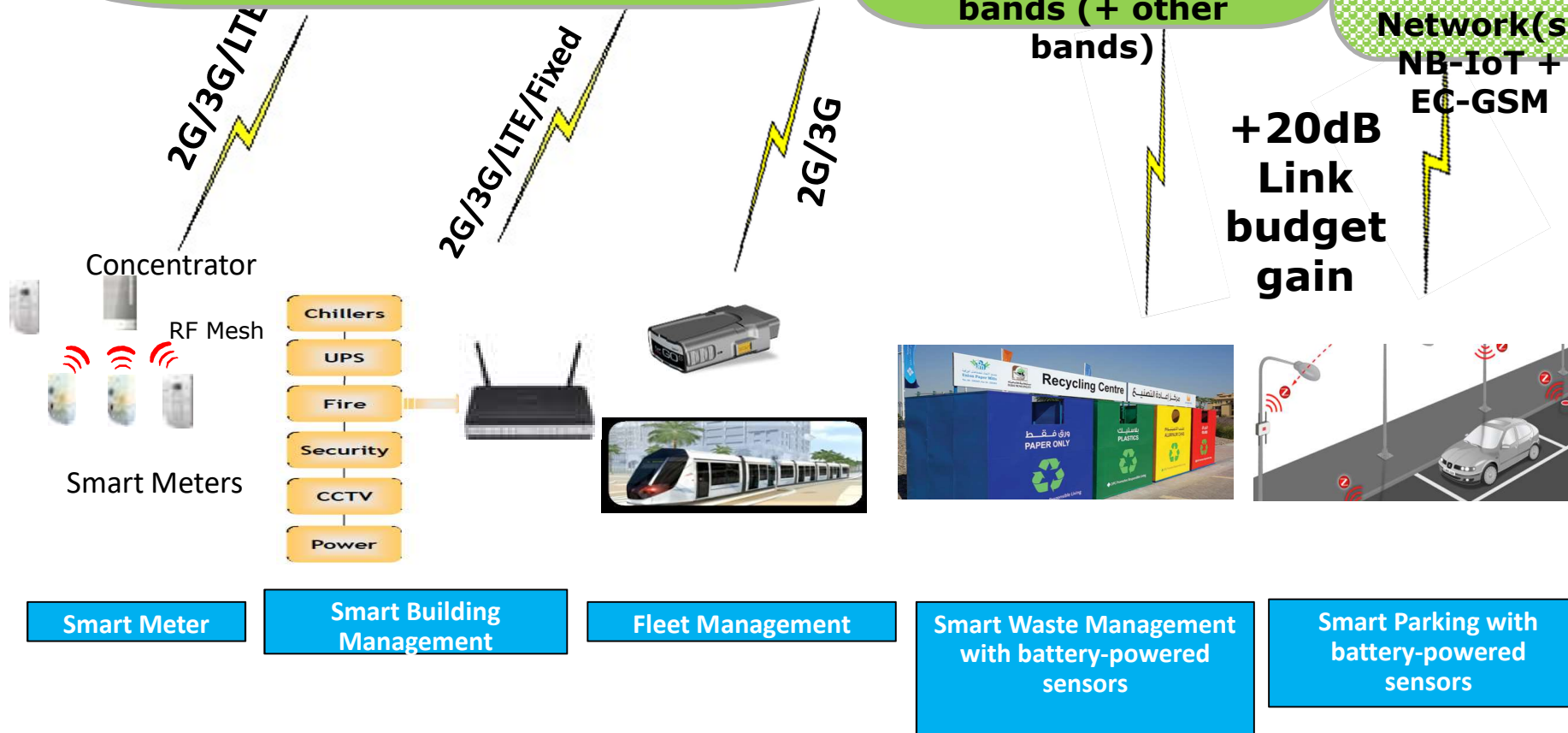
Device Management / Cloud

2G/3G/LTE/WiFi/Fixed

Unlicensed LPWA  
Networks in ISM  
bands (+ other  
bands)

3GPP  
Licensed  
LPWA  
Network(s)  
NB-IoT +  
EC-GSM

+20dB  
Link  
budget  
gain





# IoT Protocols

---

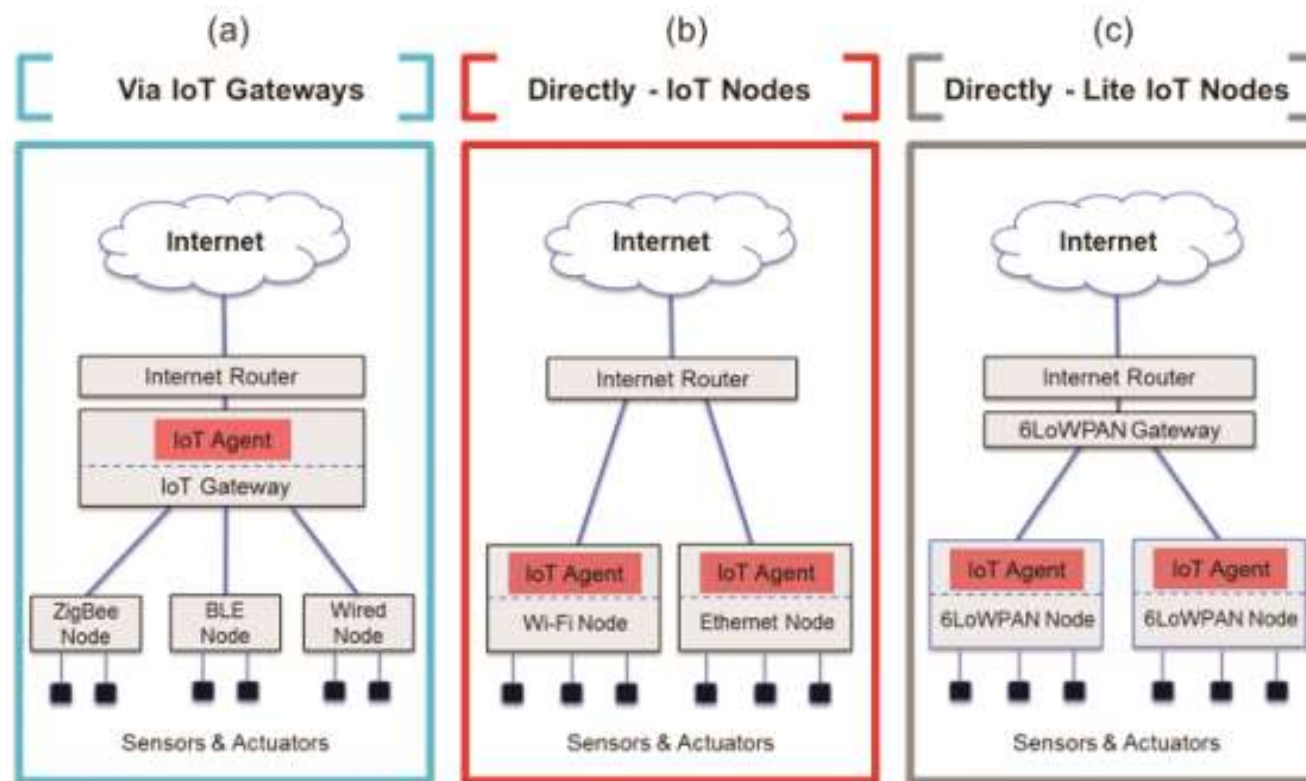
 CoAP ( Constrained Application Protocol)

 MQTT (Message Queue Telemetry Transport)

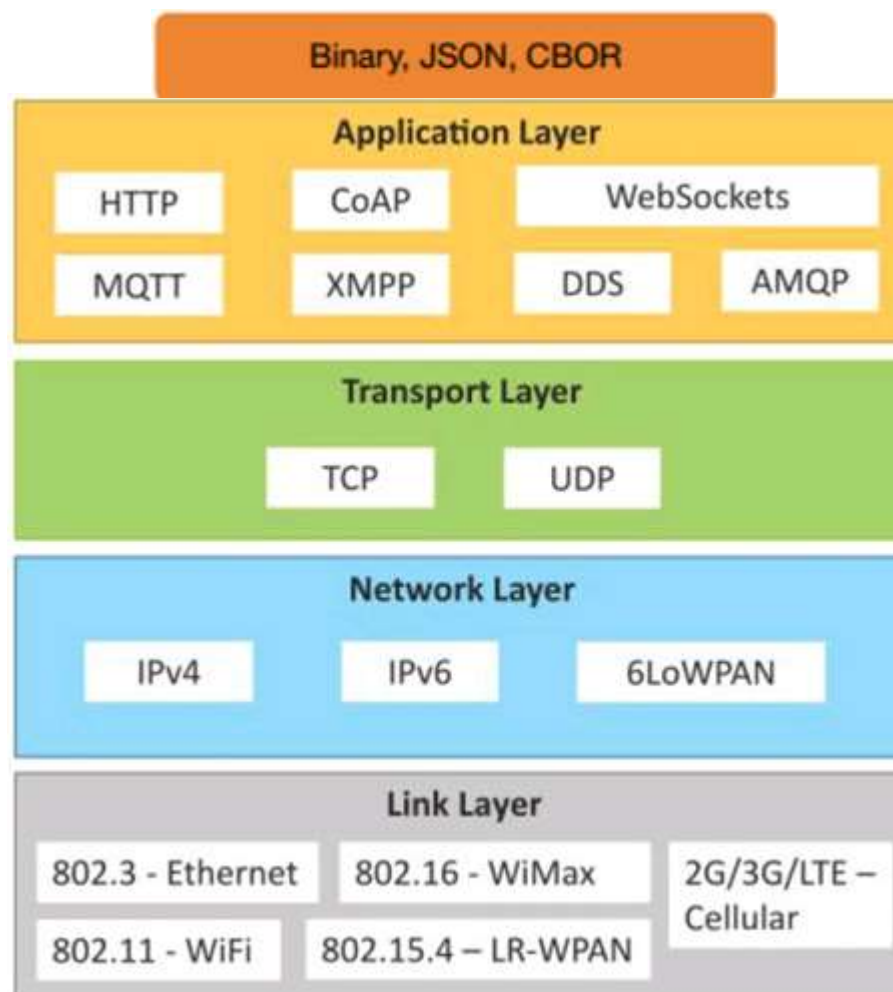
 XMPP (Extensible Messaging and Presence Protocol)

 6LoWPAN (Low power Wireless Personal Area Networks)

# IoT Protocol Architectures

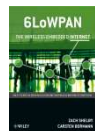
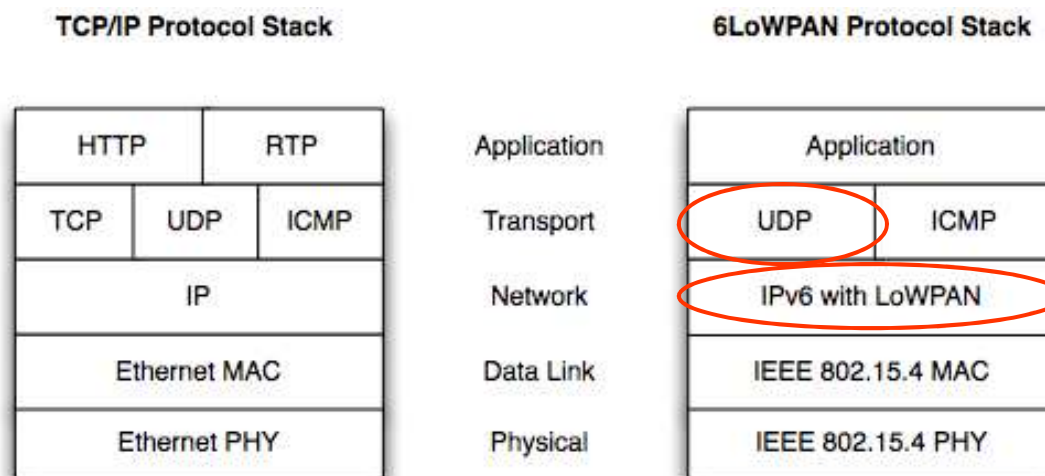


# IoT Protocol Stack



# The 6LoWPAN Format

- 6LoWPAN is an adaptation header format
  - Enables the use of IPv6 over low-power wireless links
  - IPv6 header compression
  - UDP header compression
- Format initially defined in RFC 4944
- Updated by RFC 6282





- TCP for Smart Objects

- **Advantages**

- Built-in reliability
      - Mechanism to recover lost packets
    - Control of the maximum size of its packets
      - Use of the TCP MSS (Maximum Segment Size) option

- **Drawbacks**

- Many TCP mechanisms e.g., sliding-window, congestion avoidance are not needed in smart object networks
    - Large header size introduces a significant overhead.
    - Unicast only

- UDP for Smart Objects

- **Advantages**

- Low overhead for header size and protocol logic
      - Less energy for packet transmission and reception
      - More space for application data
      - Small code footprint
    - Well suited for traffic with low reliability demand.
    - Allows multicast

- **Drawbacks**

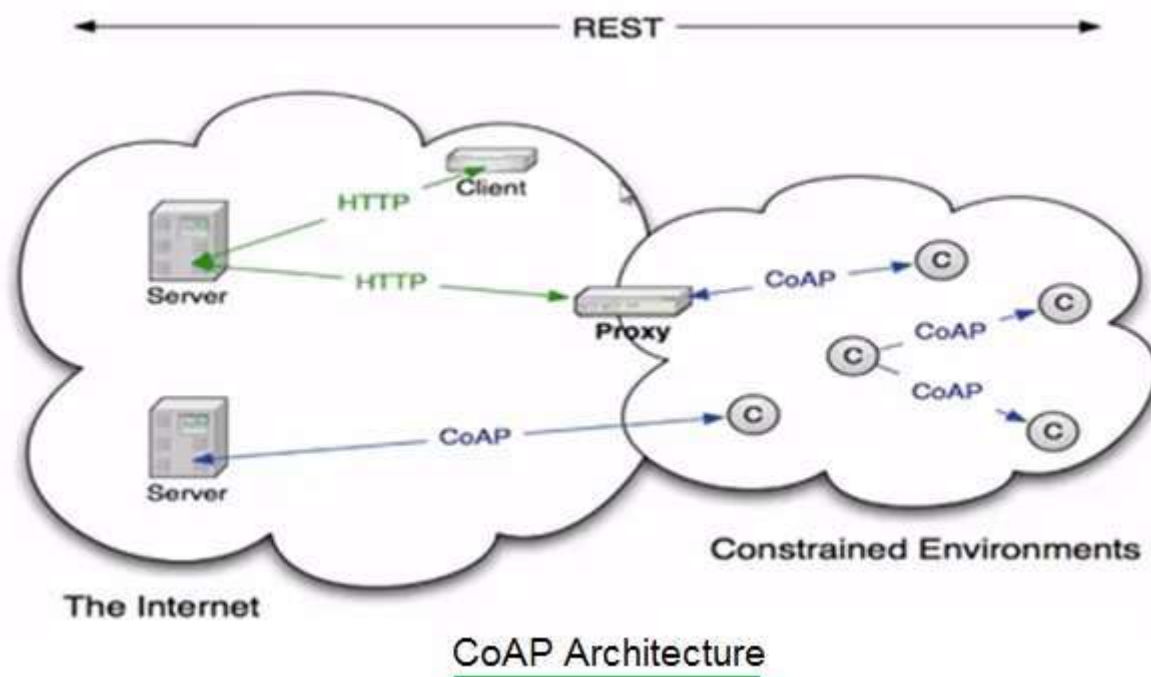
- No provision of recovery mechanism for lost packets (application has to recover them)
    - No mechanism for splitting application data into appropriate packet sizes.
      - Usually, smart object networks deal w/ small packet sizes.

# Constrained Application Protocol (CoAP)

- IoT oriented and less complex alternative to HTTP
- Open IETF standard (RFC 7252)
- Datagram Transport Layer Security (DTLS)
- Easy proxy to/from HTTP: GET, POST, PUT, DELETE
- URIs supported (e.g., `coap://hostname:port/leds/red?q=state&on`)
- RESTfull client-server model
- Implements reliable unicast over UDP
- Supports best effort multicast
- Client-Server model.



# CoAP and HTTP Interworking



## CoAP Message Layer Model

---

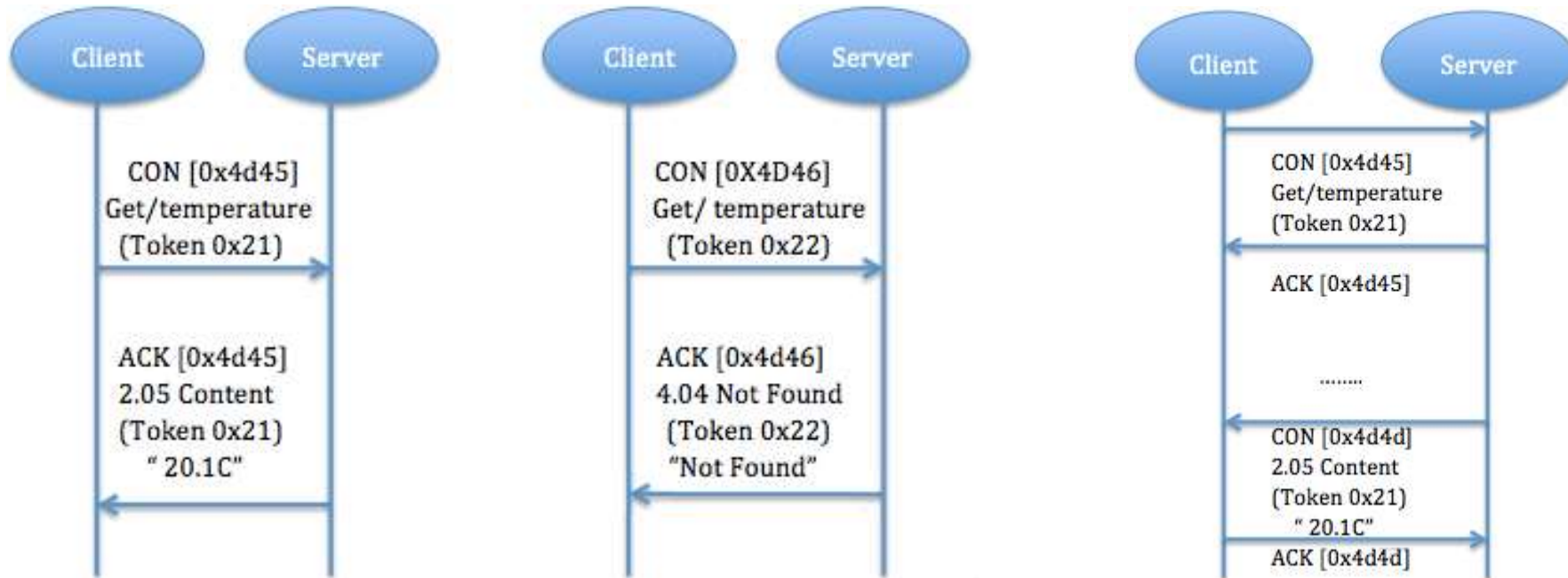
- Confirmed and non-confirmed message exchange models





# CoAP Request/Response Layer Model

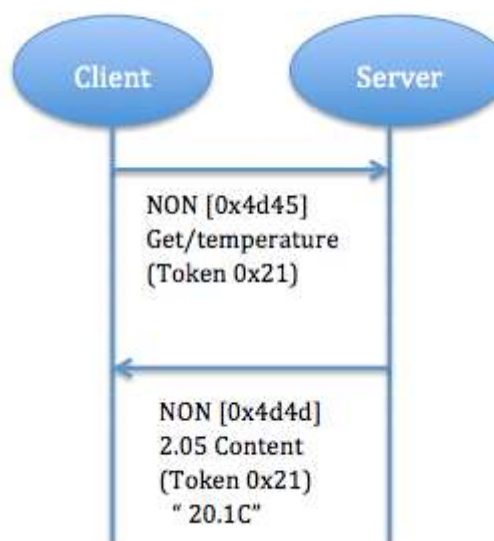
- Piggy-backed Confirmed Response
- Separate Confirmed Response



# CoAP Request/Response Layer Model

---

- Non-confirmed Response



# CoAP Message Format

Ver(2)	Type(2)	Tkl(4)	Req/Resp code(8)	Message ID (16)
Token (0-8) Tk1 byte				
Option if (any)				
Payload Marker (0xff)		Payload (if any)		

CoAP message header	Description
Ver	It is 2 bit unsigned integer. It mentions CoAP version number. Set to one.
Type	It is 2 bit unsigned integer. Indicates message type viz. confirmable (0), non-confirmable (1), ACK (2) or RESET(3).
TKL	It is 4 bit unsigned integer, Indicates length of token (0 to 8 bytes).
Req/Resp Code	It is 8 bit unsigned integer, It is split into two parts viz. 3 bit class (MSBs) and 5 bit detail (LSBs).
Message ID	16 bit unsigned integer. Used for matching responses. Used to detect message duplication.
Options	Zero or more option fields may follow a token. A few options are Content Format, Accept, Max-Age, Etag, Uri-Path, Uri-Query, etc.

# CoAP Req/Resp Code

0	1	2	3	4	5	6	7
Class				Code			

Code Range	Type	Code	Name / Description
0.00	Empty message		
0.01-0.31	Request Methods	0.01 0.02 0.03 0.04	GET POST PUT DELETE
1.00-1.31	Reserved		
2.00-5.31	Success Response	2.01 2.02 2.03 2.04 2.05	Created Deleted Valid Changed Content
		3.0-3.31	Reserved
	Client Response Error	4.00 4.01 4.02 4.03 4.04 4.05 4.06 4.12 4.13 4.15	Bad Request Unauthorized Bad Option Forbidden Not Found Method Not Allowed Not Acceptable Precondition Failed Request Entity Too Large Unsupported Content-Format
		5.00 5.01 5.02 5.03 5.04 5.05	Internal Server Error Not Implemented Bad Gateway Service Unavailable Gateway Timeout <u>Proxying</u> Not Supported
	Server Response Error		
	Reserved		
	Reserved		
	Reserved		
	Reserved		
	Reserved		
6.00-7.31	Reserved		

# Message Queuing Telemetry Transport (MQTT)

- Lightweight messaging protocol designed for sensors and devices with
  - Flaky network connectivity
  - Low computing power
  - Connections where bandwidth is at a premium
- Works on top of TCP
- Transport Layer Security (TLS)
- Protocol specification is open source
- Applications:
  - A way to obtain real world data
    - Information is gathered by an increasing number of sensors and devices deployed all over
  - A way to provide real time information
    - E.g. Locate an item in a supply chain
    - Accurate current load of any system (e.g. electricity meters)
    - Current status of a system (level of liquid in a container, temperature, pressure etc.)
    - A way to connect all the devices and sensors directly to your messaging infrastructure



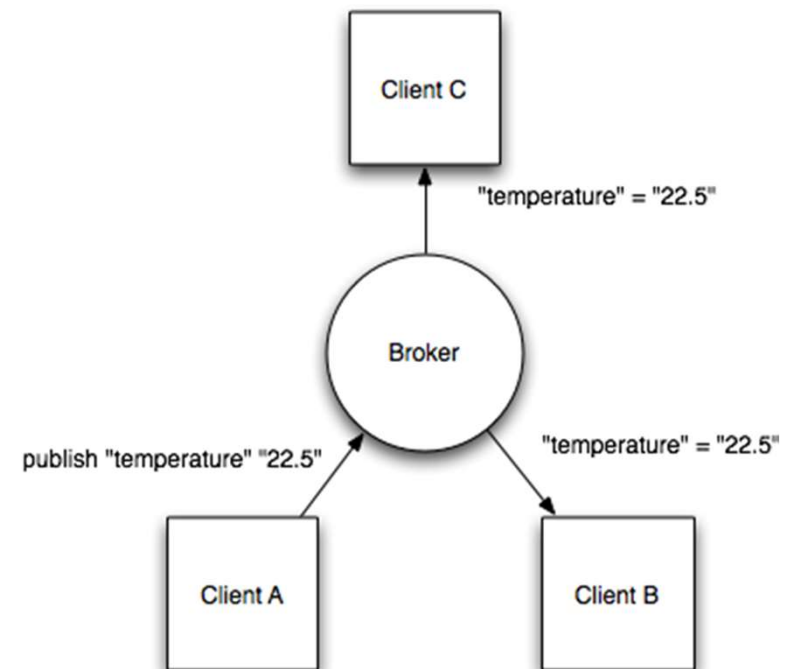
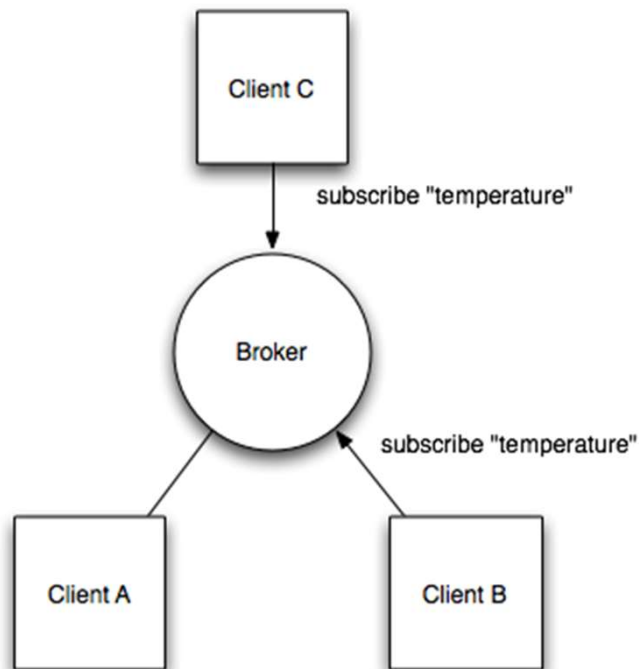
# MQTT Features

---

- Client/Server model with Clients and Brokers
- Publish and subscribe to topics
  - Managed by the broker
- 3 qualities of service
  - 0 Best effort to deliver a message
  - 1 Deliver at least once
  - 2 Deliver exactly once
- Supports persistent messages (only most recent per topic)
- Minimal transport overhead to reduce network traffic
  - As little as 2 bytes
- Last Will and Testament
  - MQTT clients can register a custom “last will and testament” message to be sent by the broker if they disconnect.
  - These messages can be used to signal to subscribers when a device disconnects.

# MQTT Architecture

- All three clients open TCP connections with the broker. Clients B and C subscribe to the topic temperature .
- At a later time, Client A publishes a value of 22.5 for topic temperature. The broker forwards the message to all subscribed clients.

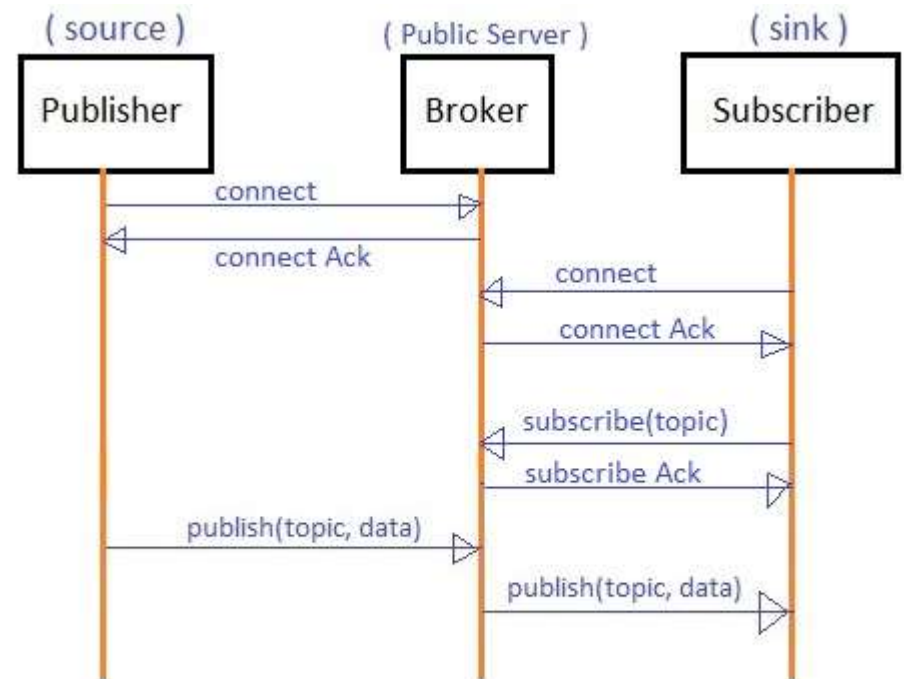


## MQTT Topics and Topic Matching

---

- In MQTT, topics are hierarchical, like a filing system (e.g., kitchen/oven/temperature).
- Wildcards are allowed when registering a subscription (but not when publishing) allowing whole hierarchies to be observed by clients.
- The wildcard + matches any single directory name, # matches any number of directories of any name.
- Examples:
  - kitchen/+/temperature matches kitchen/foo/temperature but not kitchen/foo/bar/temperature
  - kitchen/# matches kitchen/fridge/compressor/valve1/temperature

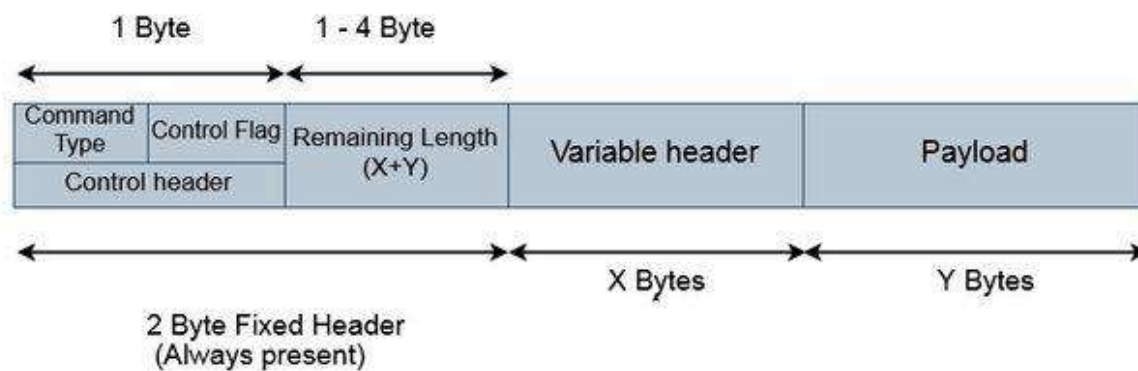
# MQTT Protocol



Broker based MQTT Protocol

# MQTT Message Format

- Message Format



# MQTT Message Format

- Fixed header

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAI N
byte 2	Remaining Length							

Mnemonic	Enumeration	Description
Reserved	0	Reserved
CONNECT	1	Client request to connect to Server
CONNACK	2	Connect Acknowledgment
PUBLISH	3	Publish message
PUBACK	4	Publish Acknowledgment
PUBREC	5	Publish Received (assured delivery part 1)
PUBREL	6	Publish Release (assured delivery part 2)
PUBCOMP	7	Publish Complete (assured delivery part 3)
SUBSCRIBE	8	Client Subscribe request
SUBACK	9	Subscribe Acknowledgment
UNSUBSCRIBE	10	Client Unsubscribe request
UNSUBACK	11	Unsubscribe Acknowledgment
PINGREQ	12	PING Request
PINGRESP	13	PING Response
DISCONNECT	14	Client is Disconnecting
Reserved	15	Reserved

Bit position	Name	Description
3	DUP	Duplicate delivery
2-1	QoS	Quality of Service
0	RETAIN	RETAIN flag

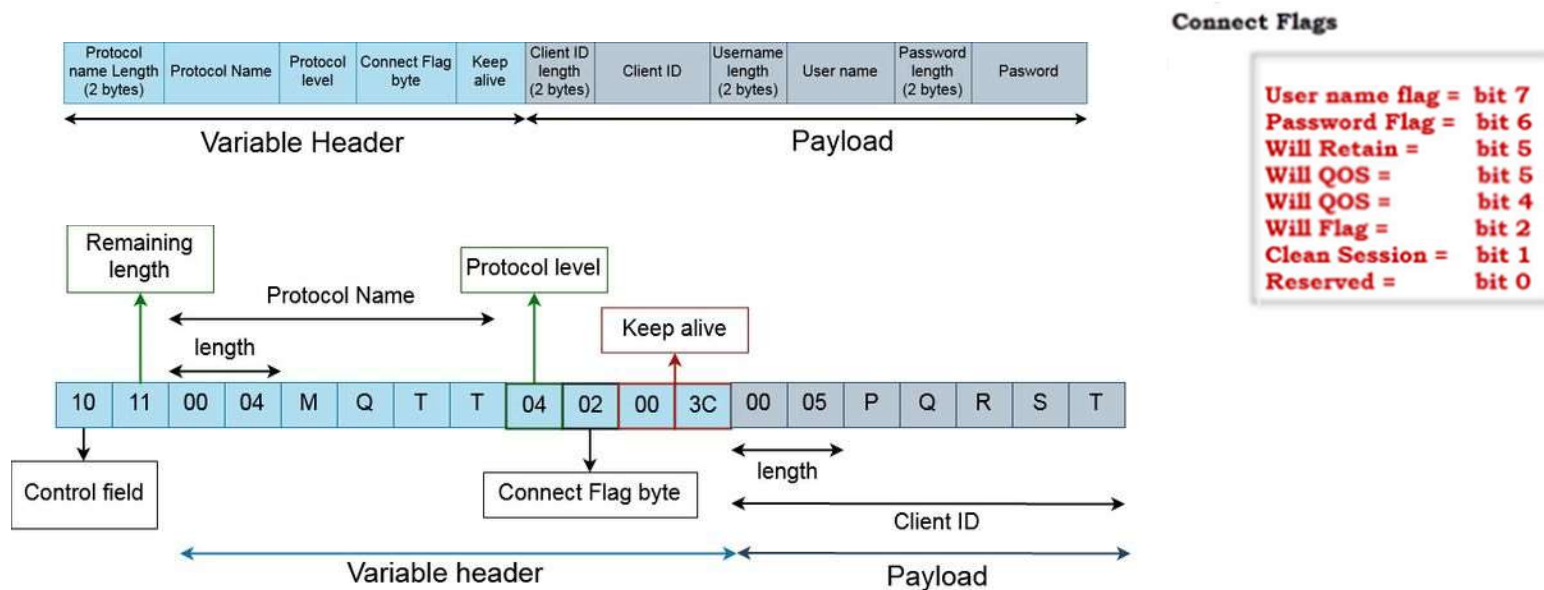
QoS value	bit 2	bit 1	Description		
0	0	0	At most once	Fire and Forget	<=1
1	0	1	At least once	Acknowledged delivery	>=1
2	1	0	Exactly once	Assured delivery	=1
3	1	1	Reserved		

Digits	From	To
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

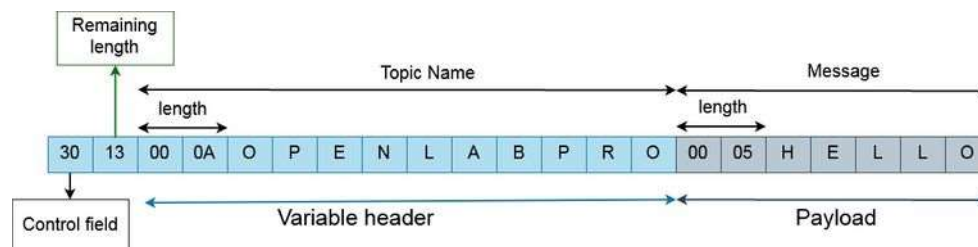


# MQTT Message Examples

- CONNECT

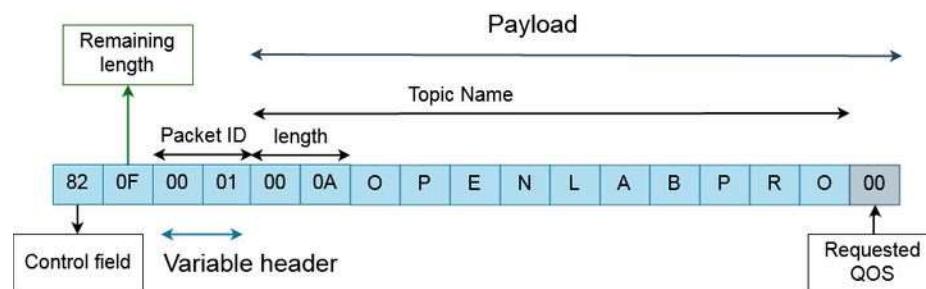


- PUBLISH "HELLO" to topic "OPENLABPRO" (QoS=0)



# MQTT Message Examples

- SUBSCRIBE to topic "OPENLABPRO"

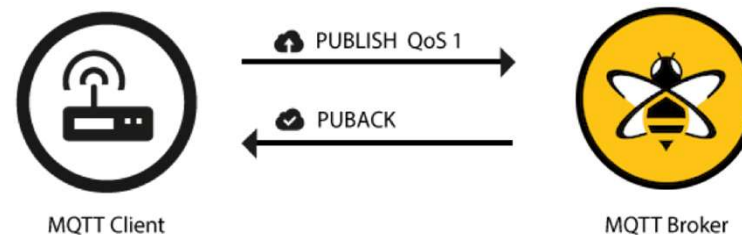


# MQTT QoS

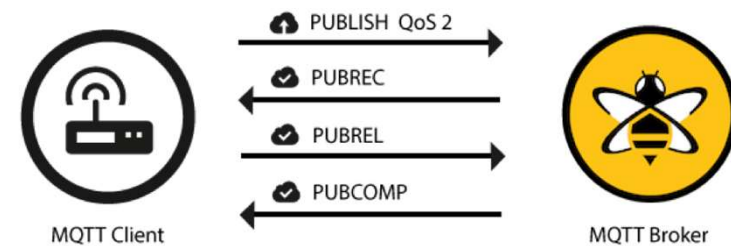
- QoS 0 – at most once



- QoS 1 – at least once

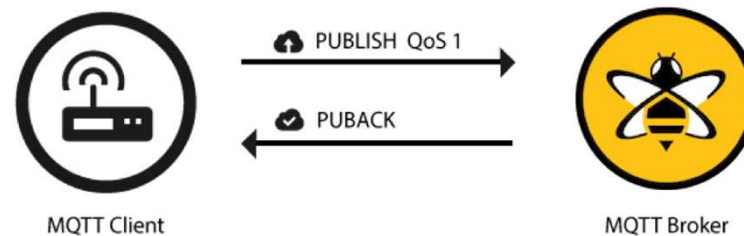


- QoS 2 – exactly once



# MQTT QoS

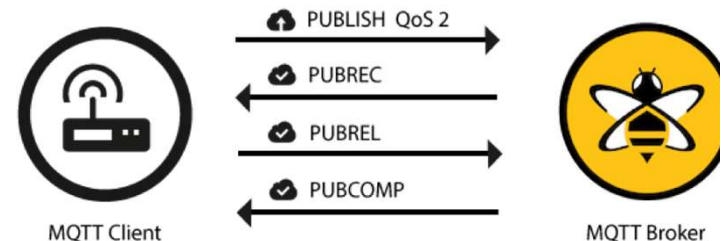
- QoS 1 – at least once



- QoS level 1 guarantees that a message is delivered at least one time to the receiver. The sender stores the message until it gets a **PUBACK** packet from the receiver that acknowledges receipt of the message. It is possible for a message to be sent or delivered multiple times.
- The sender uses the packet identifier in each packet to match the PUBLISH packet to the corresponding PUBACK packet. If the sender does not receive a PUBACK packet in a reasonable amount of time, the sender resends the PUBLISH packet. When a receiver gets a message with QoS 1, it can process it immediately. For example, if the receiver is a broker, the broker sends the message to all subscribing clients and then replies with a PUBACK packet.
- If the publishing client sends the message again it sets a duplicate (DUP) flag. In QoS 1, this DUP flag is only used for internal purposes and is not processed by broker or client. The receiver of the message sends a PUBACK, regardless of the DUP flag.

# MQTT QoS

- QoS 2 – exactly once



- QoS 2 is the highest level of service in MQTT. This level guarantees that each message is received only once by the intended recipients. QoS 2 is the safest and slowest quality of service level. The guarantee is provided by at least two request/response flows (a four-part handshake) between the sender and the receiver. The sender and receiver use the packet identifier of the original PUBLISH message to coordinate delivery of the message.
- When a receiver gets a QoS 2 PUBLISH packet from a sender, it processes the publish message accordingly and replies to the sender with a **PUBREC** packet that acknowledges the PUBLISH packet. If the sender does not get a PUBREC packet from the receiver, it sends the PUBLISH packet again with a duplicate (DUP) flag until it receives an acknowledgement.
- Once the sender receives a PUBREC packet from the receiver, the sender can safely discard the initial PUBLISH packet. The sender stores the PUBREC packet from the receiver and responds with a **PUBREL** packet.
- After the receiver gets the PUBREL packet, it can discard all stored states and answer with a **PUBCOMP** packet (the same is true when the sender receives the PUBCOMP). Until the receiver completes processing and sends the PUBCOMP packet back to the sender, the receiver stores a reference to the packet identifier of the original PUBLISH packet. This step is important to avoid processing the message a second time. After the sender receives the PUBCOMP packet, the packet identifier of the published message becomes available for reuse.
- When the QoS 2 flow is complete, both parties are sure that the message is delivered and the sender has confirmation of the delivery..
- If a packet gets lost along the way, the sender is responsible to retransmit the message within a reasonable amount of time. This is equally true if the sender is an MQTT client or an **MQTT broker**. The recipient has the responsibility to respond to each command message accordingly.



## MQTT for Sensor Networks (MQTT-SN)

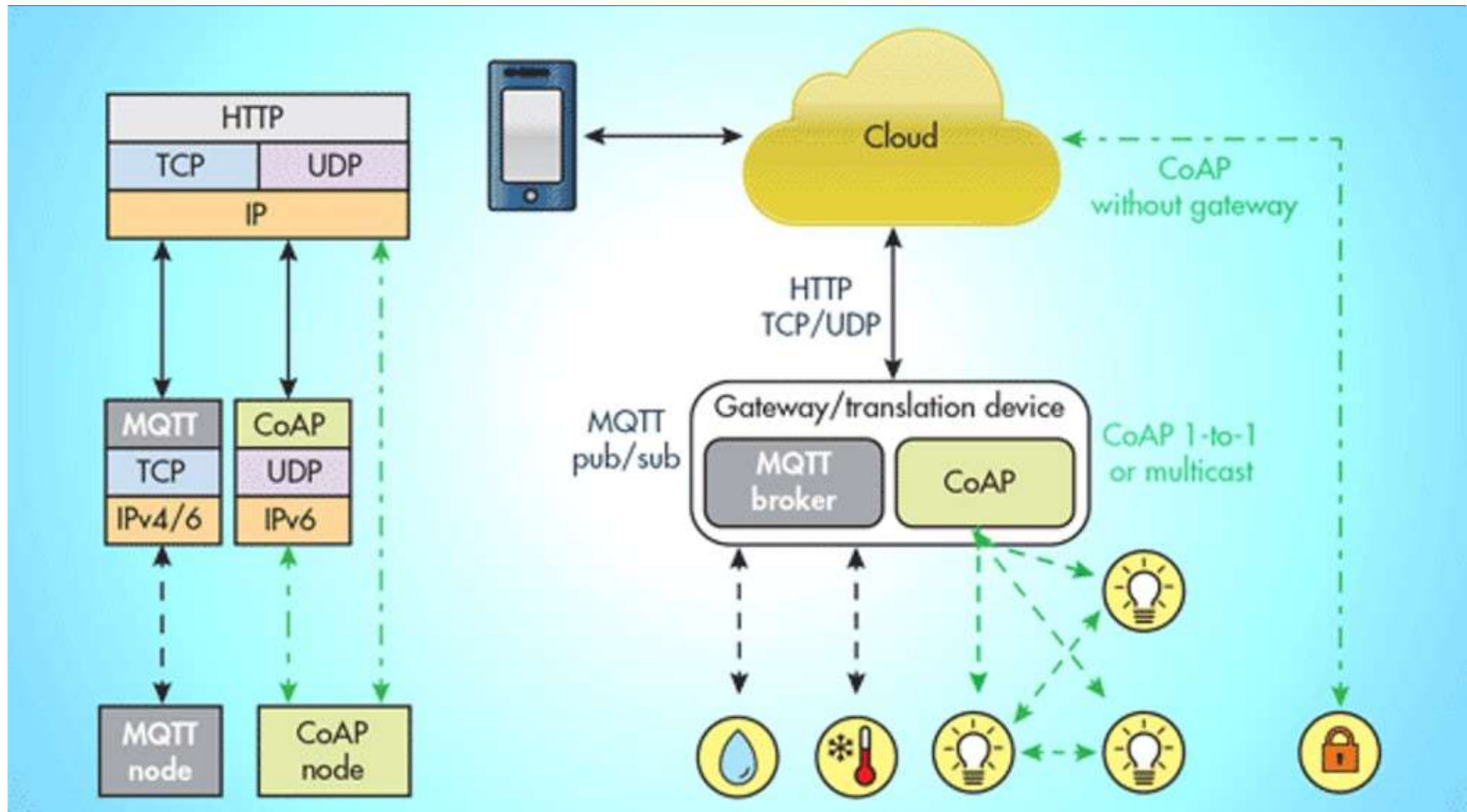
---

- Even though MQTT is designed to be lightweight, it has two drawbacks for very constrained devices:
  - Every MQTT client must support TCP and will typically hold a connection open to the broker at all times. For some environments where packet loss is high or computing resources are scarce, this is a problem.
  - MQTT topic names are often long strings which make them impractical for 802.15.4 and other low bitrate small packet protocols.
- Both of these shortcomings are addressed by the MQTT-SN protocol:
  - MQTT-SN does not require TCP (can use UDP or serial link)
  - Broker support for indexing topic names (short topic IDs).
- Requires MQTT-SN to MQTT gateway.

# MQTT vs CoAP

Features	MQTT	CoAP
Full Form	Message Queue Telemetry Transport	Constrained Application Protocol
Messages used	Connect, connect ack, publish, publish ack, subscribe, subscribe ack, disconnect etc.	GET, PUT, POST and DELETE
Architecture	Publish/Subscribe	Request/Response
Need of centralized broker	required, end devices communicate via broker	not required, end devices directly communicate
Transport protocol	TCP/IP	UDP/IP
Security protocol	TLS	DTLS
fault tolerance	broker is SPoF	server is SPoF
scope	device to cloud cloud to cloud	device to cloud cloud to cloud

# MQTT & CoAP: Wrap-Up



<https://www.electronicdesign.com/technologies/iot/article/21800998/silicon-labs-mqtt-and-coap-underlying-protocols-for-the-iot>

# Acknowledgements

---

1. Dr.Kayarvizhy, "Internet of Things",  
<http://studyslide.com/doc/20337/iot---dr.-kayarvizhy>
2. Augusto Casaca, "Internet of Things", IFIP TC6 LATIN AMERICA  
TUTORIALS IN NETWORKING
3. Xi Chen, "Constrained Application Protocol for Internet of Things",  
<https://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/index.html>
4. MQTT Essentials  
<https://www.hivemq.com/tags/mqtt-essentials/>
5. Toby Jaffey, "MQTT and CoAP, IoT Protocols",  
[https://www.eclipse.org/community/eclipse\\_newsletter/2014/february/article2.php](https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php)
6. "MQTT vs REST", <http://www.rfwireless-world.com/Terminology/MQTT-vs-REST.html>