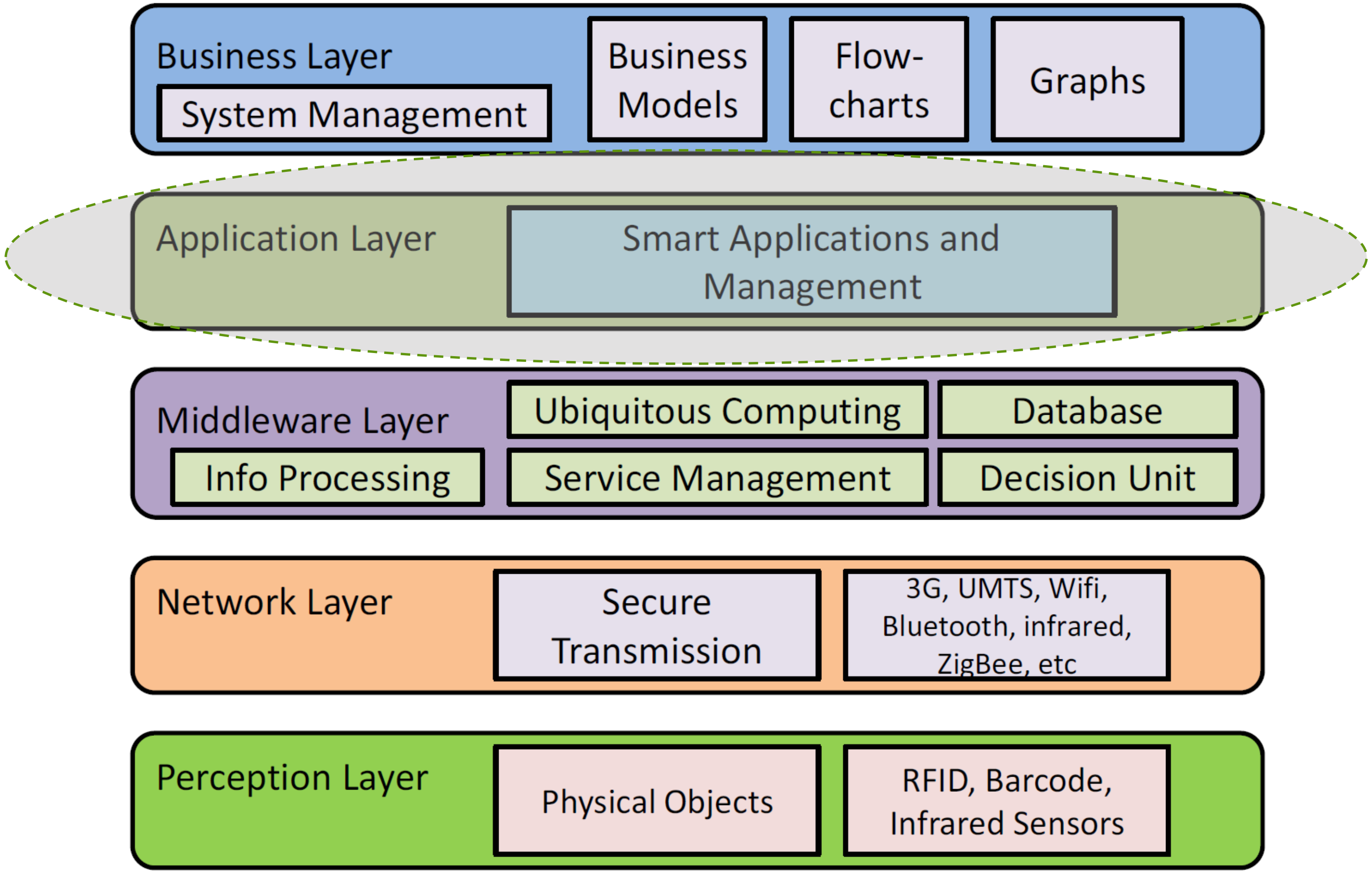
IoT Edge to Cloud Protocols

Instructor: Deepak Gangadharan

IoT Architecture



Source- Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges, 2012

Why other protocols?

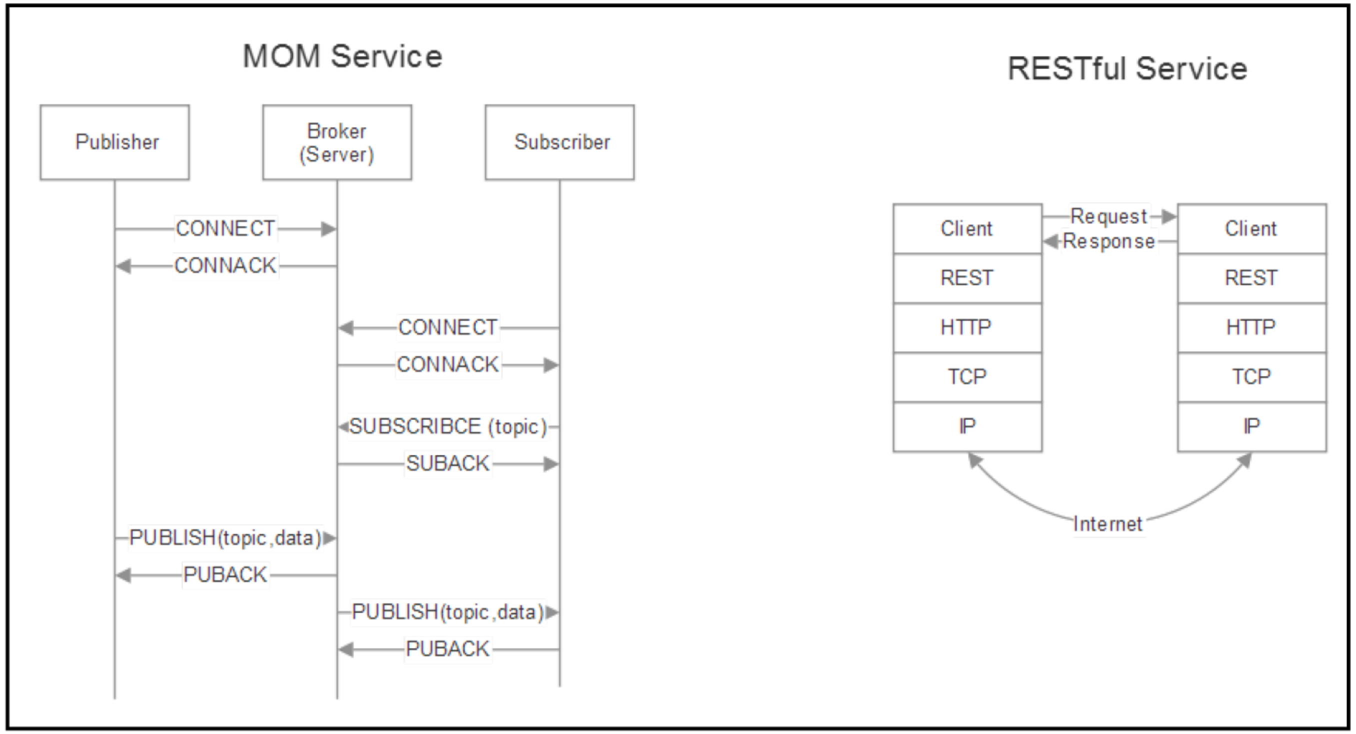
- Why are there other protocols other than HTTP to transport data across WAN?
- HTTP was designed for general purpose computing in client/server models
- IoT devices are **very constrained**, **remote** and **bandwidth limited** → More efficient, secure and scalable protocols are necessary
- Many protocols are Message Oriented Middleware (MOM) implementations → Communication between 2 devices using distributed message queues
- Some devices produce data and add to queue, others consume data from queue
- Some implementations require a broker

Alternative to MOM implementation

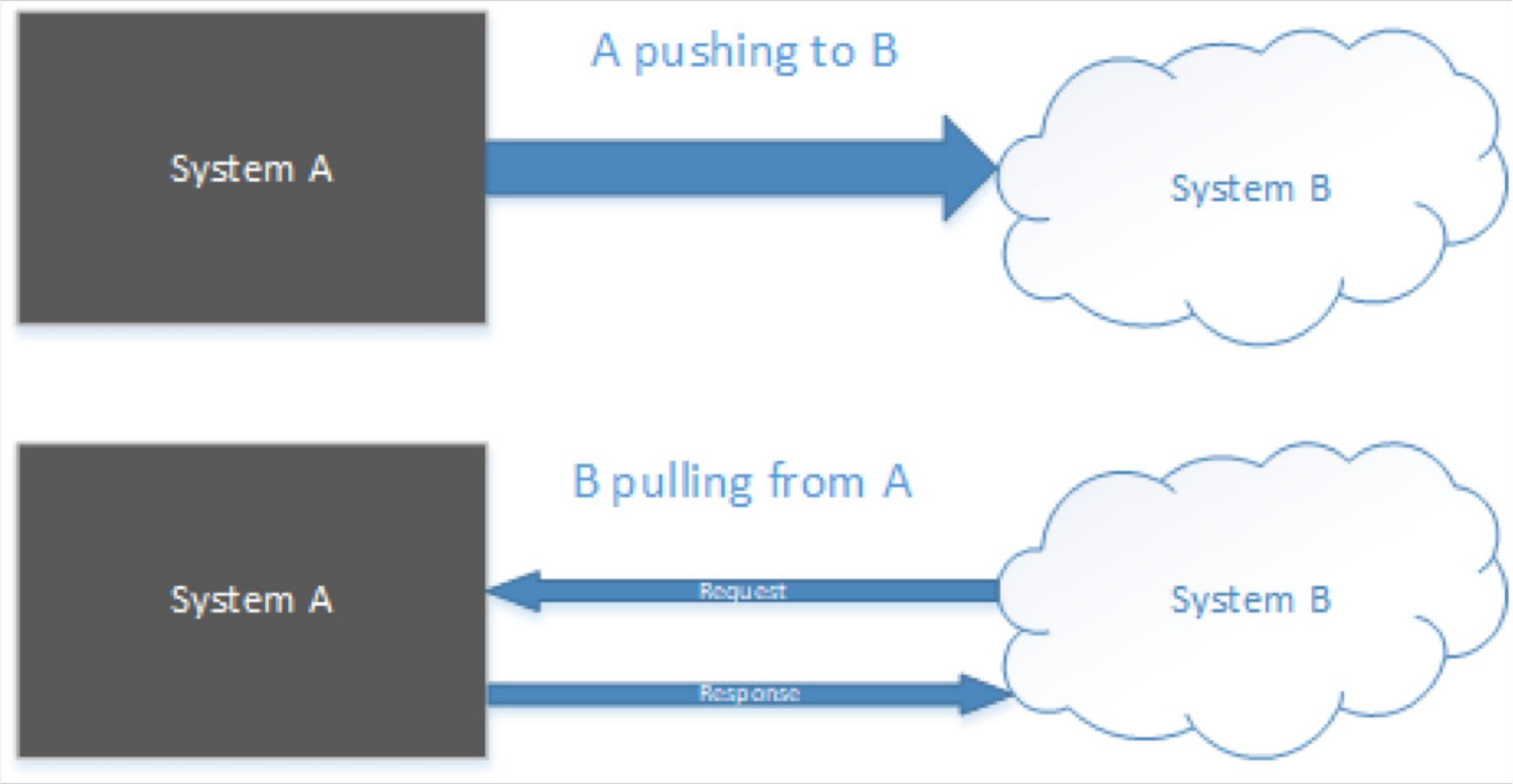
- RESTful model
  - A server owns the state of a resource but state is not transferred in a message from the client to the server
  - Use HTTP methods such as GET, PUT, POST and DELETE to place requests on a resource's Universal Resource Identifier (URI)
  - No broker in this architecture
  - Clients initiate access to resources through synchronous request-response patterns

Alternative to MOM implementation

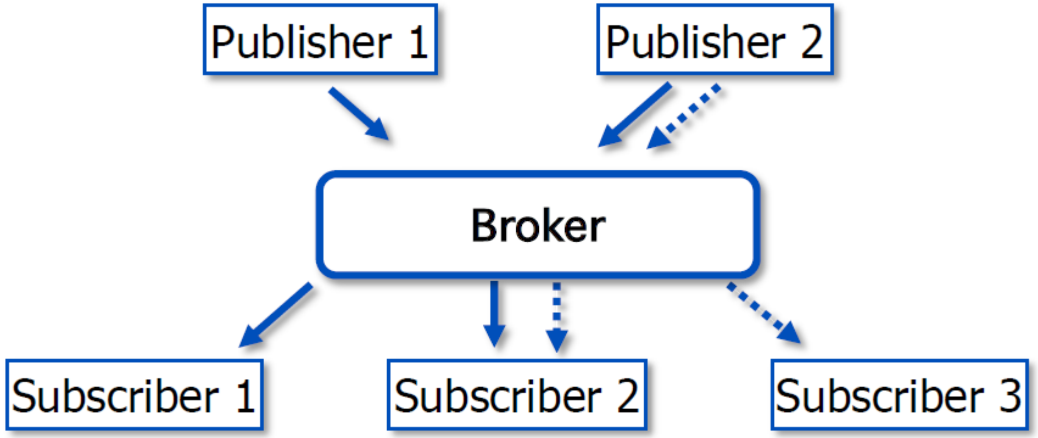
- RESTful model
  - Clients are responsible for errors, even if the server fails



IoT Protocols – Ways to interchange messages



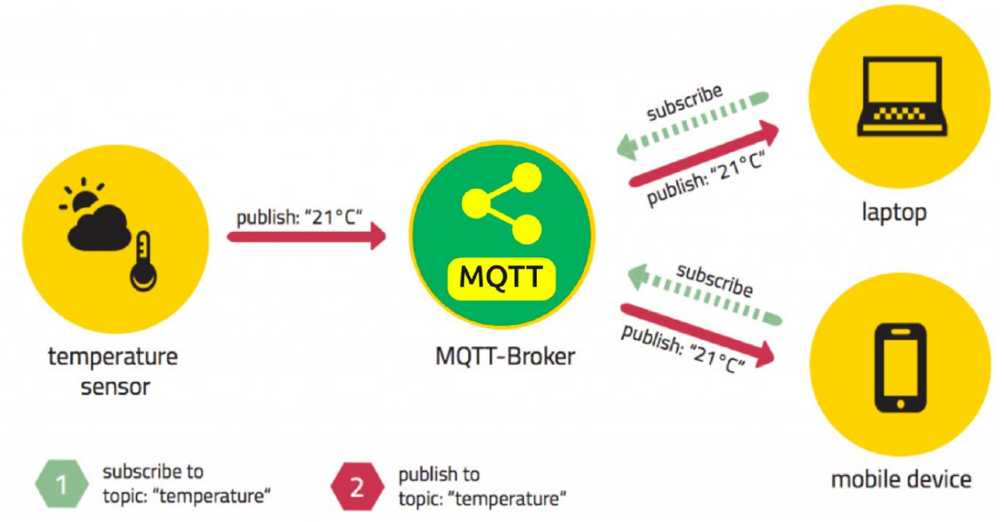
IoT Protocols – Pub/Sub Approach



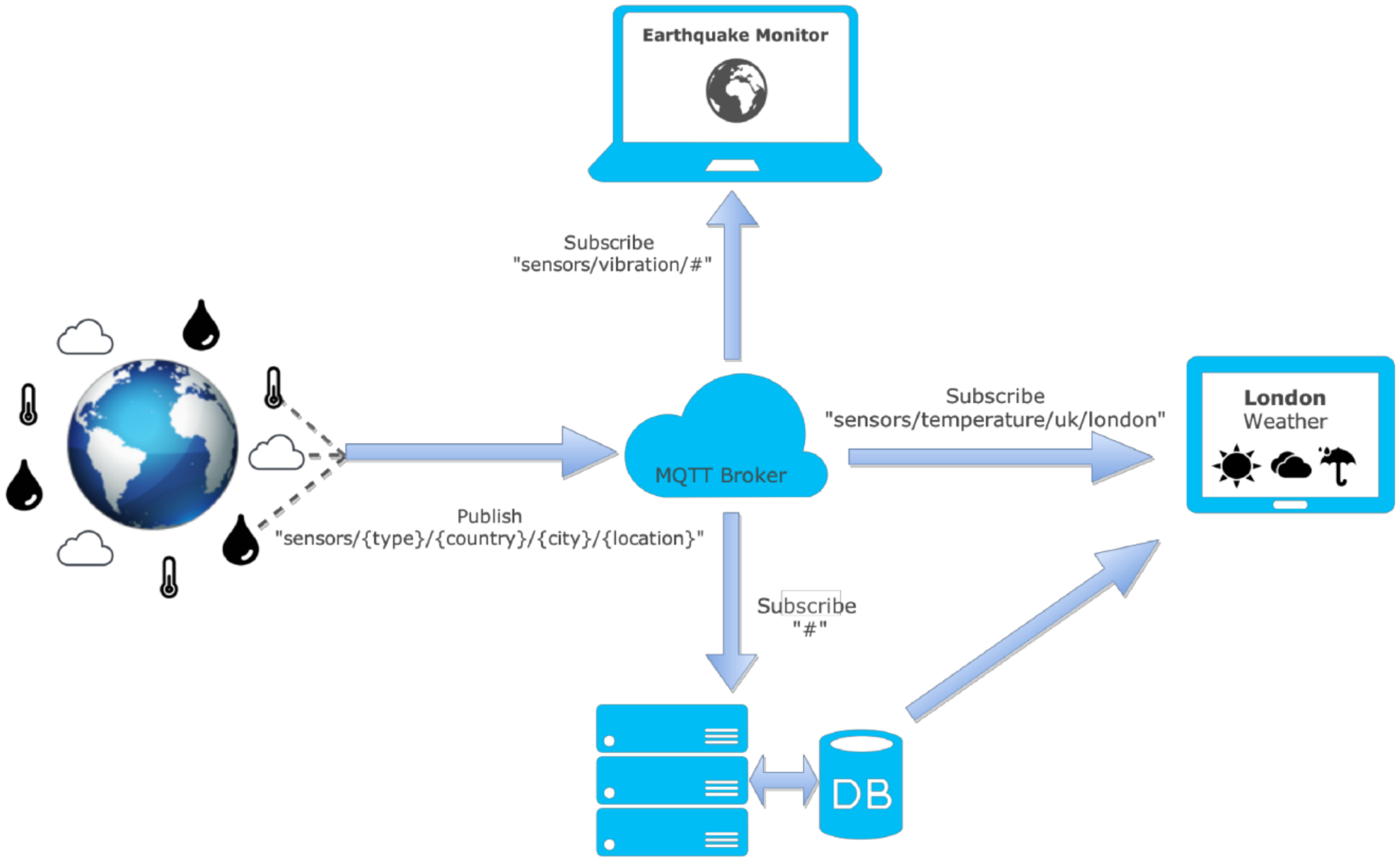
- Publisher/Subscriber (Producer/Consumer)
- Various protocols:
  MQTT (MQ Telemetry Transport), AMQP, XMPP

IoT Protocols – Pub/Sub Approach

- Pub/Sub separate a client, who is sending a message about a specific topic, called **publisher**, from another client (or more clients), who is receiving the message, called **subscriber**
- Unlike client/server model, clients not aware of any physical identifiers such as IP address or port
- A third component, called the **broker**, which is known both by the publisher and subscriber, filters all incoming messages and distributes them accordingly



Pub/Sub Approach – An example



Source: https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot

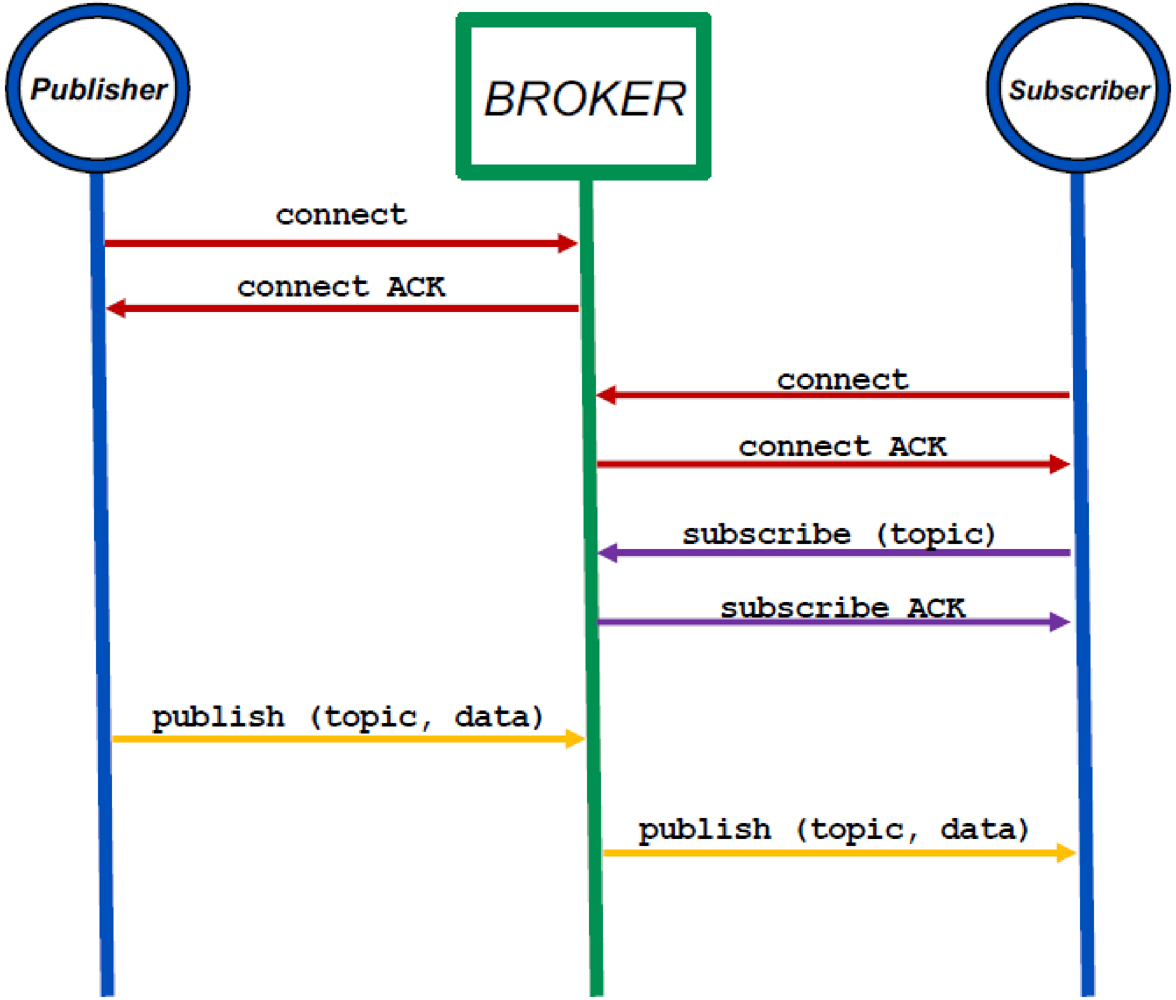
Goals of MQTT

- Must be simple to implement
- To provide a form of quality of service
- To be very lightweight and bandwidth efficient
- To be data agnostic
- To have continuous session awareness
- To address security issues

IoT Protocols - MQTT

- A lightweight publish-subscribe protocol that can run on embedded devices and mobile platforms → http://mqtt.org/
- A machine to machine/ IoT connectivity protocol, which is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium
- It has been used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers, and in a range of home automation and small devices scenario
- Ideal for mobile applications because of – small size, low power usage, minimized data packets and efficient distribution of information to one or many receivers
- References:
  MQTT community wiki: https://github.com/mqtt/mqtt.github.io/wiki
  A good tutorial: http://www.hivemq.com/mqtt-essentials/

Publish/Subscribe Interactions Sequence



MQTT Architecture

- MQTT is an asymmetric protocol
- MQTT can retain a message on a broker indefinitely → controlled by a flag in normal message transmission
- MQTT defines an optional facility called Last Will and Testament (LWT) → message a client specifies during the connect phase
- LWT contains the Last Will topic, QoS and the actual message
- If a client disconnects from a broker ungracefully, the broker is obligated to broadcast LWT message to all other subscribed clients of that topic.
- keep-alive packet used to retain the connection between client and broker if the client device is disconnected

MQTT topics

- MQTT topics are structured in a hierarchy similar to folders and files in a file system using forward slash (/) as a delimiter
- Topic names are case sensitive and must consist of atleast one character to be valid

- Topic subscriptions can have wildcards, which enable nodes to subscribe to groups of topics that don't exist yet
  '+' matches anything at a given tree level
  '#' matches a whole sub-tree

MQTT broker

- Responsibility of connecting clients and filtering data
- Filters provide
  - **Subject filtering**: Clients subscribe to topics and certain topic branches → Broker responsible for re-broadcasting the message to subscribed clients or ignoring it
  - **Content filtering**: Have the ability to inspect and filter published data → Any data not encrypted can be managed
  - **Type filtering**: A client can apply their own filters to subscribed data stream
- No need to directly identify a publisher or consumer based on physical aspects
- Cloud-managed MQTT brokers can ingest millions of messages per hour and support tens of thousands of publishers
- Maximum allowable packet size in MQTT is 256 MB, however data payload size is cloud and broker-dependent.

MQTT – Quality of Service

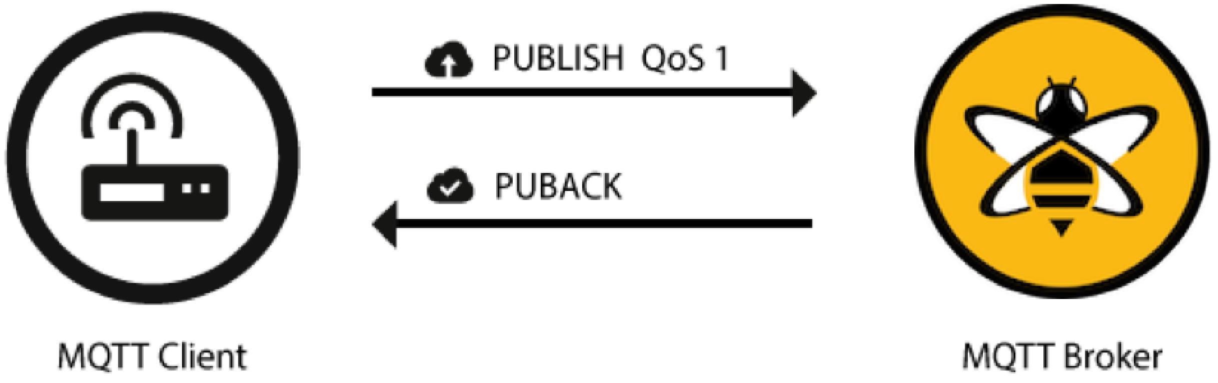
- Messages are published with a QoS level, which specifies delivery requirements
- It also gives the client the power to choose a level of service that matches its network reliability and application logic
- Communication in unreliable networks is lot easier as MQTT manages the retransmission of messages and guarantees delivery

MQTT – Quality of Service

- QoS 0 – at most once
  - ➢ Guarantees best-effort delivery
  - ➢ No guarantee of delivery
  - ➢ Recipient does not acknowledge receipt of the message and message is not stored and retransmitted by the sender
  - ➢ Also called "fire and forget" – Same guarantee as underlying TCP protocol

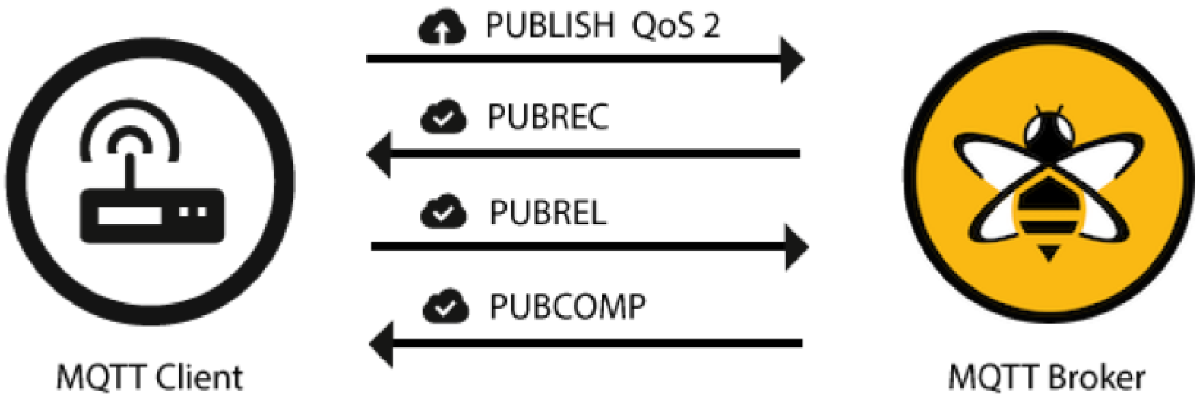
MQTT – Quality of Service

- QoS 1 – at least once
  - Guarantees that a message is delivered at least one time to the receiver
  - Sender stores the message until it gets a PUBACK packet from the receiver
  - Possible for a message to be sent or delivered multiple times
  - Uses packet identifier to match the PUBLISH packet to the corresponding PUBACK packet



MQTT – Quality of Service

- QoS 2 – exactly once
  - Highest level of service in MQTT
  - Guarantees that each message is received only once by the intended recipients
  - Guarantee provided by at least two request/response flows (a four part handshake)

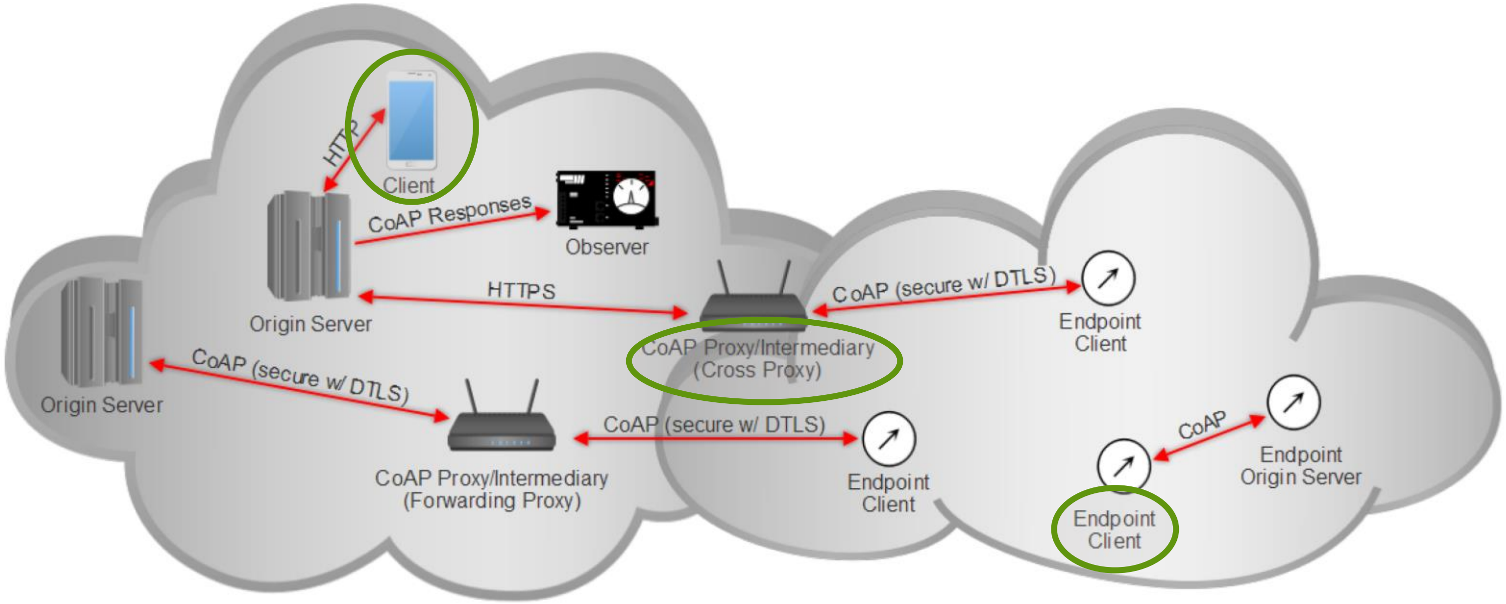


Constrained Application Protocol (CoAP)

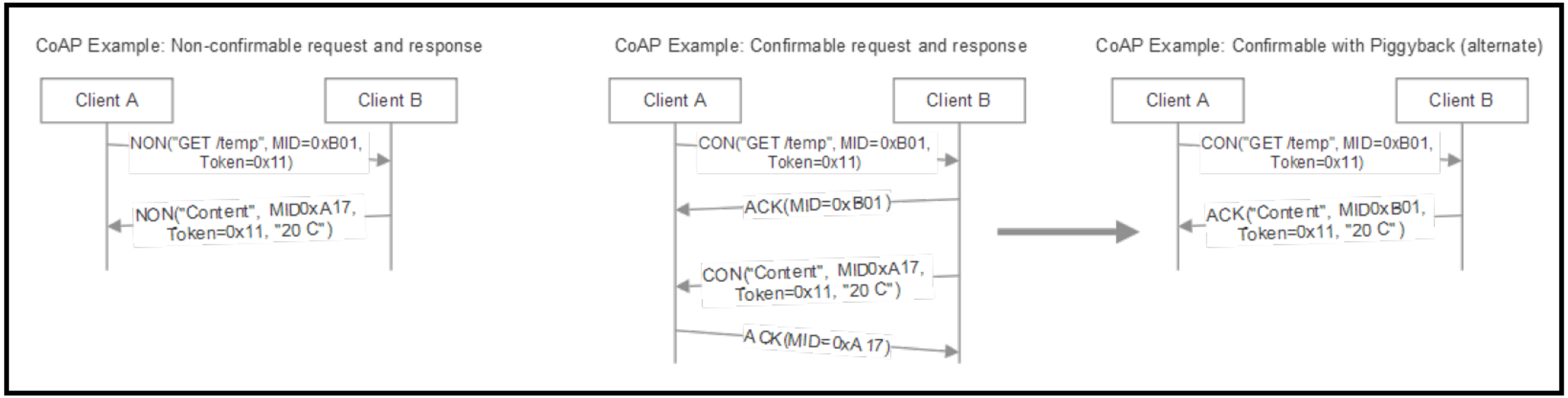
- IETF Constrained RESTful Environments (CoRE) working group created the first draft of the protocol
- Intended as communication protocol for constrained devices
- Tailored for M2M communication between edge nodes
- Provides a similar and easy structure of resource addressing familiar while using the web but with reduced resources and bandwidth
- Some implementations of CoAP perform up to 64x better than HTTP equivalents on similar hardware (power of 0.744 mW compared to 1.333 mW for HTTP)

CoAP Architecture

- Based on concept of mimicking and replacing heavy HTTP abilities and usage with a lightweight equivalent for IoT
- Not a HTTP replacement



CoAP Messaging



Protocol Summary and Comparison

	MQTT	MQTT-SN	CoAP	AMQP	STOMP	HTTP/RESTful
Model	MOM pub/sub	MOM pub/sub	RESTful	MOM	MOM	RESTful
Discovery protocol	No	Yes (via gateways)	Yes	No	No	Yes
Resource demands	Low	Very Low	Very Low	High	Medium	Very High
Header Size (bytes)	2	2	4	8	8	8
Average power usage	Lowest	Low	Medium	High	Medium	High
Authentication	No (SSL/TLS)	No (/TLS)	No (DTLS)	Yes	No	Yes (TLS)
Encryption	No (SSL/TLS)	No (SSL/TLS)	No (DTLS)	Yes	No	Yes (TLS)
Access controls	No	No	No (proxy)	Yes	No	Yes
Communication overhead	Low	Very Low	Very Low	High	High, verbose	High
Protocol complexity	Low	Low	Low	High	Low	Very High
TCP/UDP	TCP	TCP/UDP	UDP	TCP/UDP	TCP	TCP
Broadcasting	Indirect	Indirect	Yes	No	No	No
Quality of Service	Yes	Yes	With CON messages	Yes	No	No

Thank You