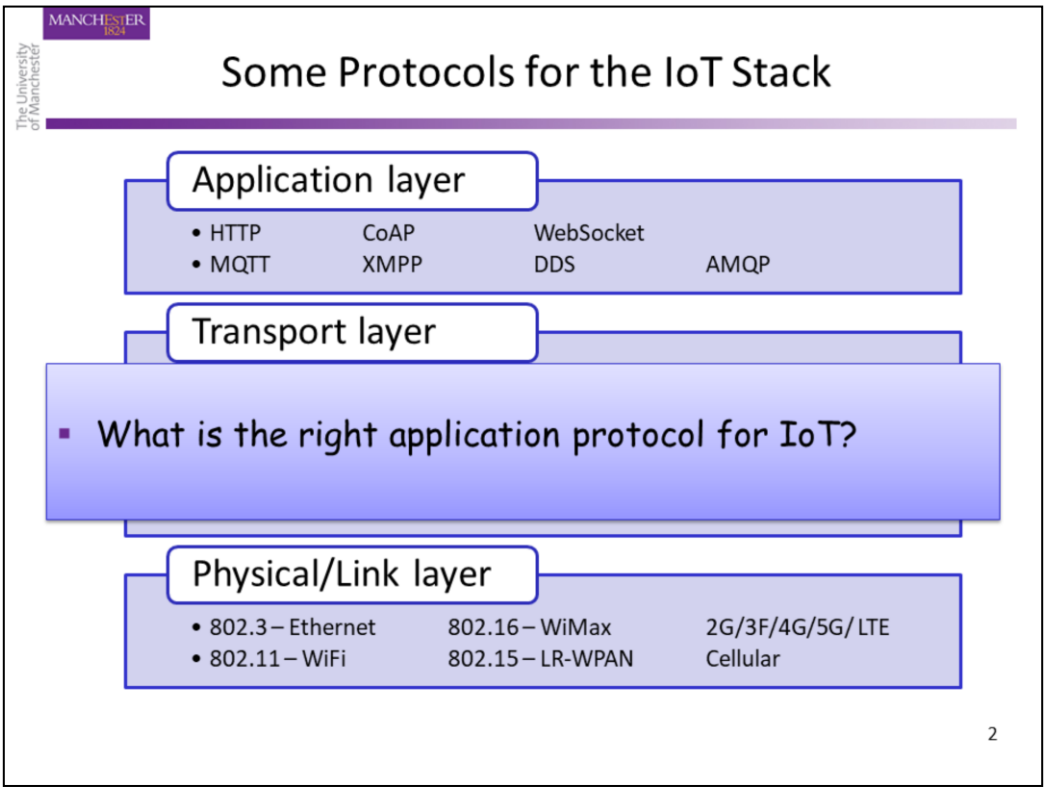
APPLICATION LAYER PROTOCOLS

- The material in this section was based on the following material:

1. J. Dizdarevic, F. Caprio, A. Jukan, X. Masip-Bruin, “A Survey of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration”, *ACM Computing Surveys*, April 2018.
2. V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, J. Alonso-Zarate, “A Survey on Application Layer Protocols for the Internet of Things”, *Transactions on IoT and Cloud Computing*, 2015.
3. N. Naik, “Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP”, IEEE International Systems Engineering Symposium (ISSE), 2017.
4. S. Mijovic, E. Shehu, and C. Buratti, “Comparing application layer protocols for the Internet of Things via experimentation”, IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016.



- The communication and networking field contains myriads of acronyms. To facilitate understanding, we provide where required definitions of these acronyms.
  - Message Queue Telemetry Transport (MQTT)
  - Constrained Application Protocol (CoAP)
  - Extensible Messaging and Presence Protocol (XMPP)
  - Advanced Message Queuing Protocol (AMQP)
  - Data Distribution Service (DDS) protocol
  - Representational State Transfer Hypertext Transfer Protocol (REST HTTP).

MANCHESTER 1824

The University of Manchester

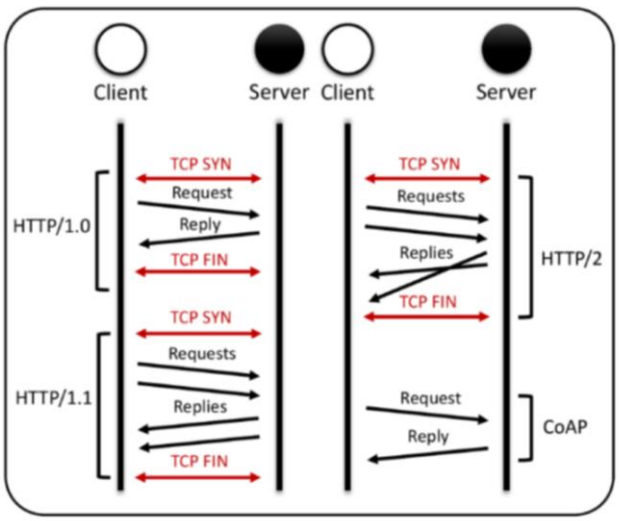
Interaction Models of Application Protocols

- Request/response -> client/server
  - What's the most well-known protocol that uses this model?
  - CoAP, DDS(?)
- Publish/subscribe -> publisher/subscriber
  - MQQT, AMQP, XMPP(?)
- Most of these protocols continually evolve and may support (to a different extent) both models
- Handshaking and data transfer
  - Specific to Websocket only!

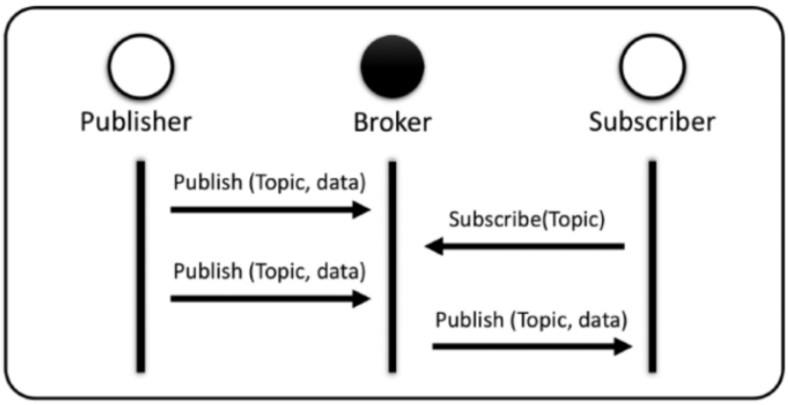
MANCHESTER 1824

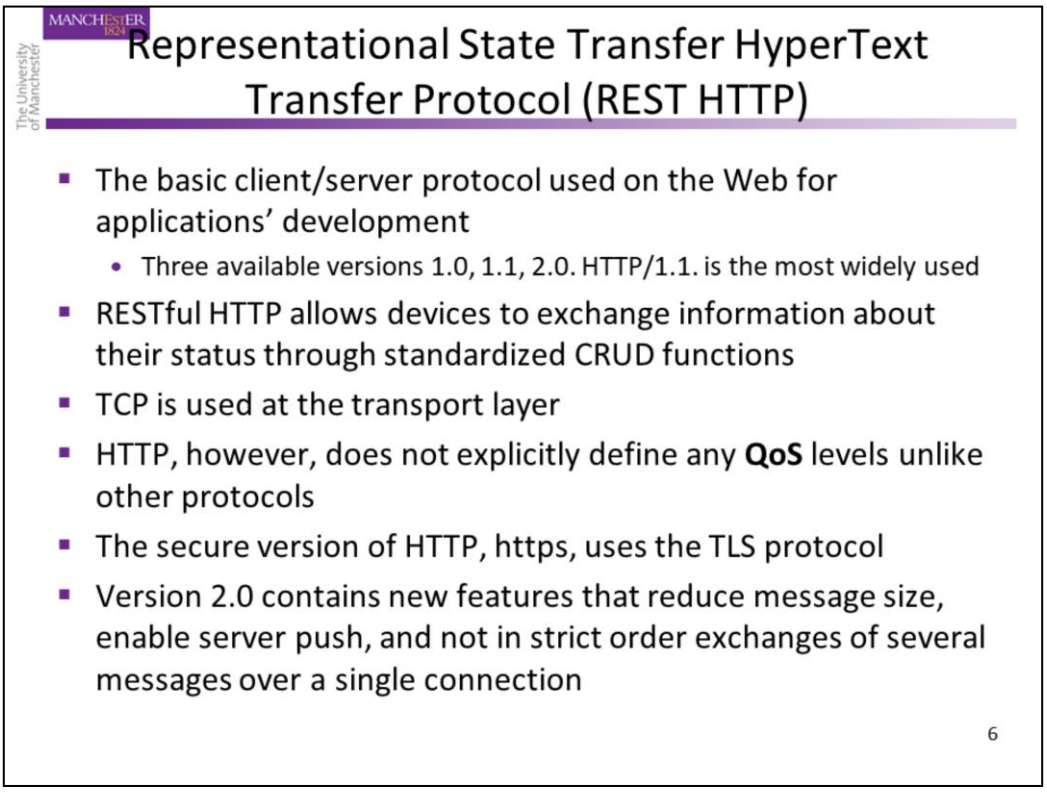
The University of Manchester

Request/Response Model

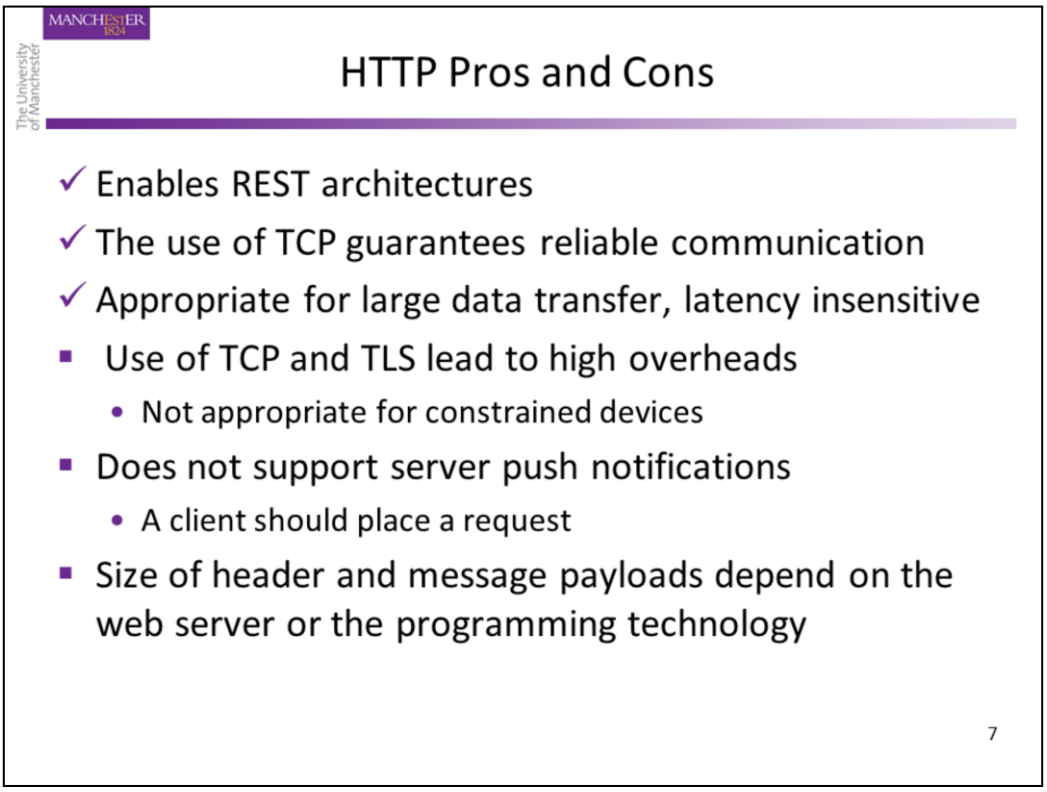


Publish/Subscribe Model





- CRUD stand for Create, Read, Update, and Delete. These abstract functions correspond perfectly to the respective methods in HTTP: POST, GET, PUT and DELETE.
- JSON is typically used to represent the data.
- In HTTP 1.0, the TCP connection is closed after a single HTTP request/reply pair.
- In HTTP 1.1, a keep-alive-mechanism was introduced, where a TCP connection could be reused for sending multiple requests to the server without waiting for a response (pipelining). Once the requests are all sent, the browser starts listening for responses and HTTP 1.1 specification requires that a server must send its responses to those requests in the same order that the requests were received.
- Its most widely used version is HTPP/1.1 and is based on client/server communication that follows the request/response model. Recently, HTPP has been associated with the REST architecture to facilitate interaction between different entities over web-based services. The combination of HTTP and REST enables devices to make their status readily available in terms of the standardized CRUD (create, read, update, delete) functions. The CRUD functions are mapped to the POST, GET, PUT and DELETE methods of HTTP, respectively. In this way, it is possible to develop a REST model for different IoT devices.



- While the use of JSON assures the transport of large volumes of data reliably, it is not optimized for resource constrained environments. One of the major issues is that in the context of IoT, nodes with limited resources send small amounts of data sporadically and installing a TCP connection every time results in high delay and significant overhead.
- HTTP/2.0 introduces a more efficient use of network resources and shorter total latency by using compressed headers and by exchanging multiple parallel messages over the same connection. Furthermore, it introduces the server push mode, in which the server can forward data to clients without waiting for requests. These features seem to be useful in IoT applications and remain to be realized and tested for their performance.

- The combination of HTTP protocol with REST is commendable, because the devices can make their state information easily available, due to a standardized way to create, read, update, and delete data (the so-called CRUD operations). IoT standardizes around JSON over HTTP.
- While using TCP provides reliable delivery of large amounts of data, which is an advantage in connections that do not have strict latency requirements, it creates challenges in resource constrained environments.

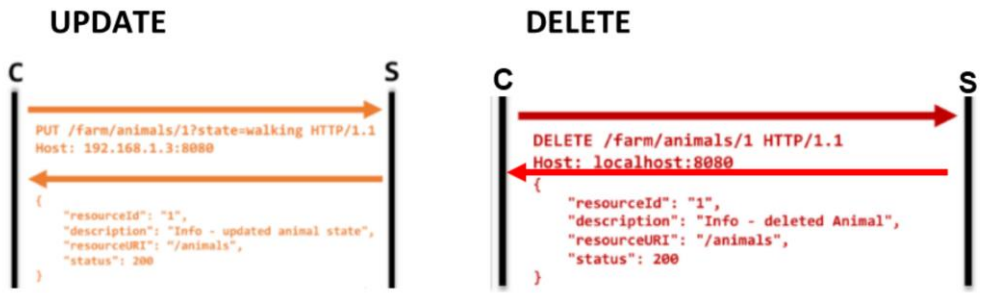
MANCHESTER 1824

The University of Manchester

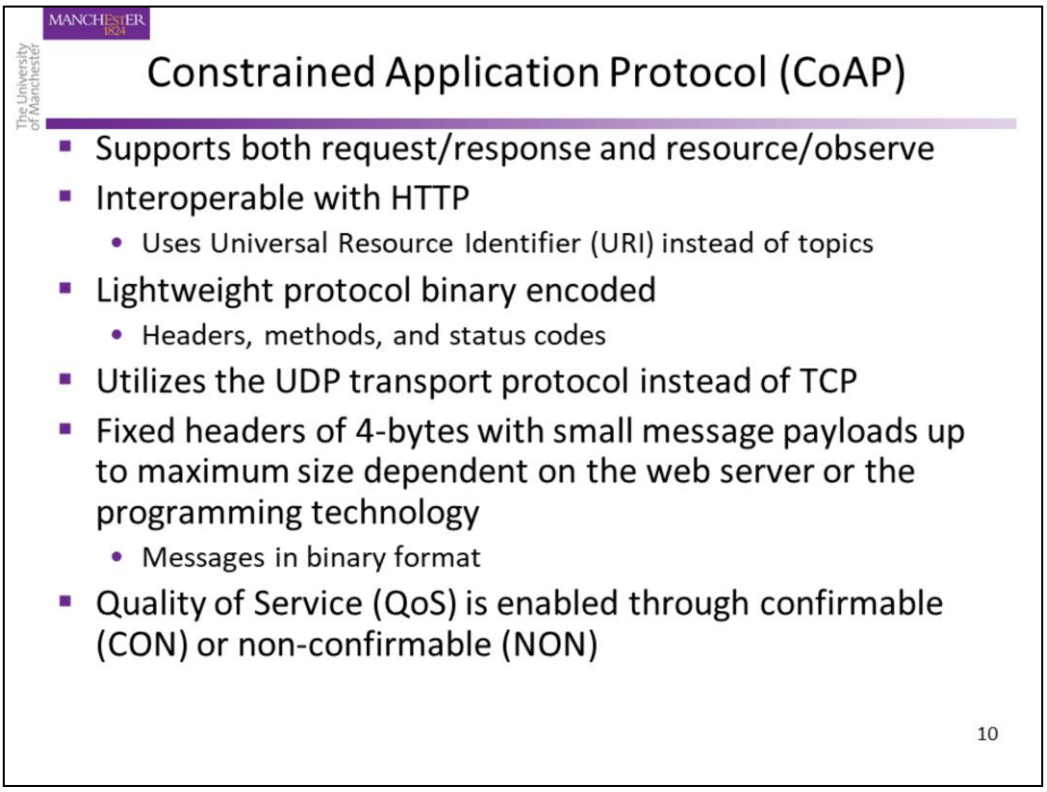
REST HTTP CRUD Methods (1)

8

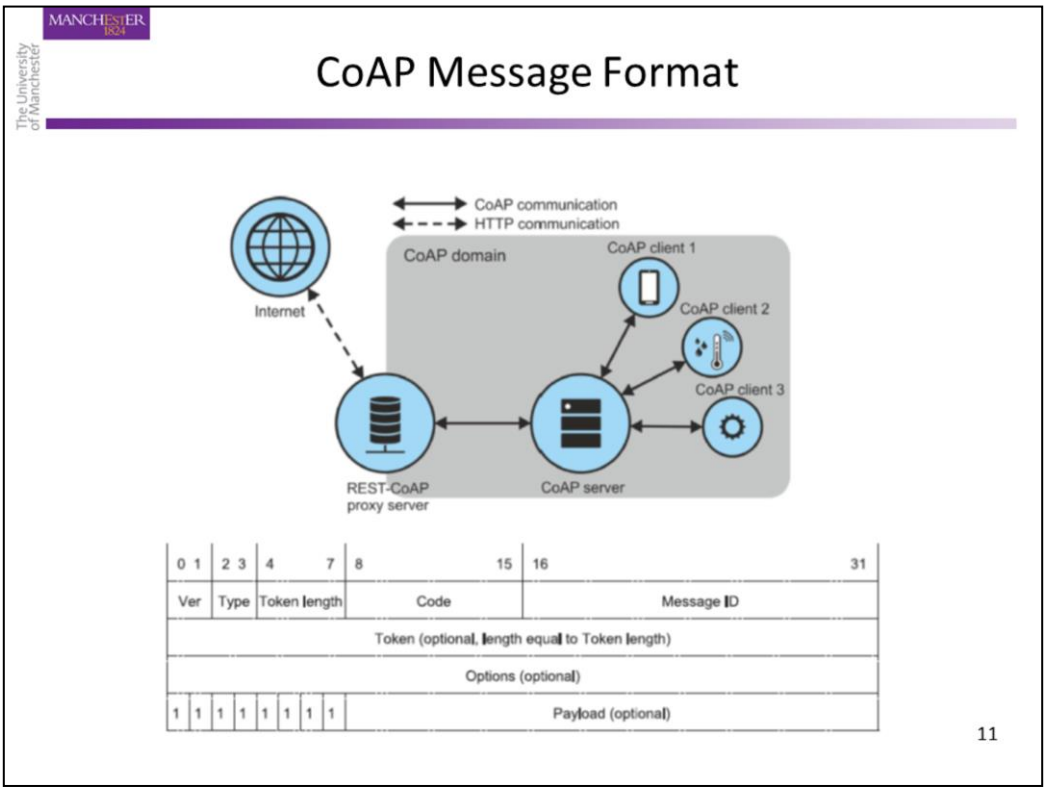
REST HTTP CRUD Methods (2)



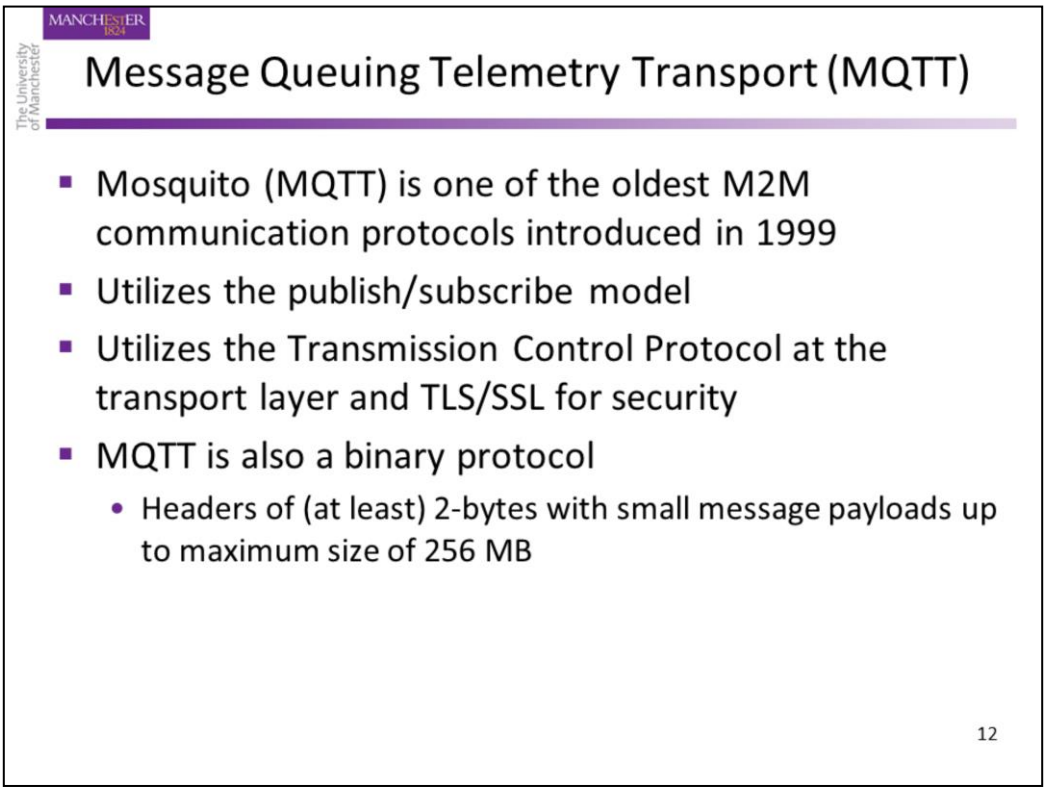
9



- CoAP was designed by the Constrained RESTful Environments (CoRE) working group of Internet Engineering Task Force (IETF) aiming constrained devices with limited processing capabilities.
- Z. Shelby, K. Hartke, and C. Bormann. 2014. The Constrained Application Protocol (CoAP). RFC 7252. RFC Editor. http://www.rfc-editor.org/rfc/rfc7252.txt.
- When a CoAP client sends one or multiple CoAP requests to the server and gets the response, this response is not sent over a previously established connection, but exchanged asynchronously over CoAP messages, which effectively reduces reliability.
- CoAP is structured into two logically different layers. One of these layers follows the request/response models, implementing the RESTful paradigm and enables CoAP clients to behave similar to HTTP when sending requests.
- The biggest difference is the second layer, which addresses the reliability issue as a result of the use of the User Datagram Protocol (UDP) at the transport layer.

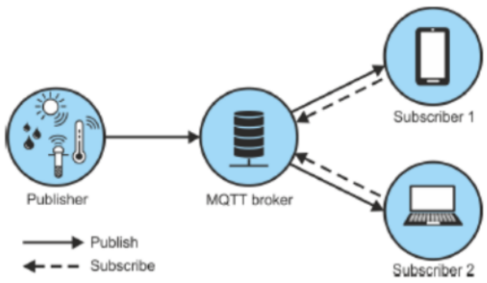


- This layer (called the message layer) defines four types of messages: CON (Confirmable), NON (non-confirmable), ACK (Acknowledgment), and RST (reset). CON messages require an acknowledgment from the receiver side with an ACK message, thereby ensuring reliable communication.
- The resource/observe model uses the option “observe” in the GET method. In this case the server puts the client in the list of observers of the specified quantity and the client receives updates when this quantity changes. In this way communication is particularly close to the publish/subscribe logic.



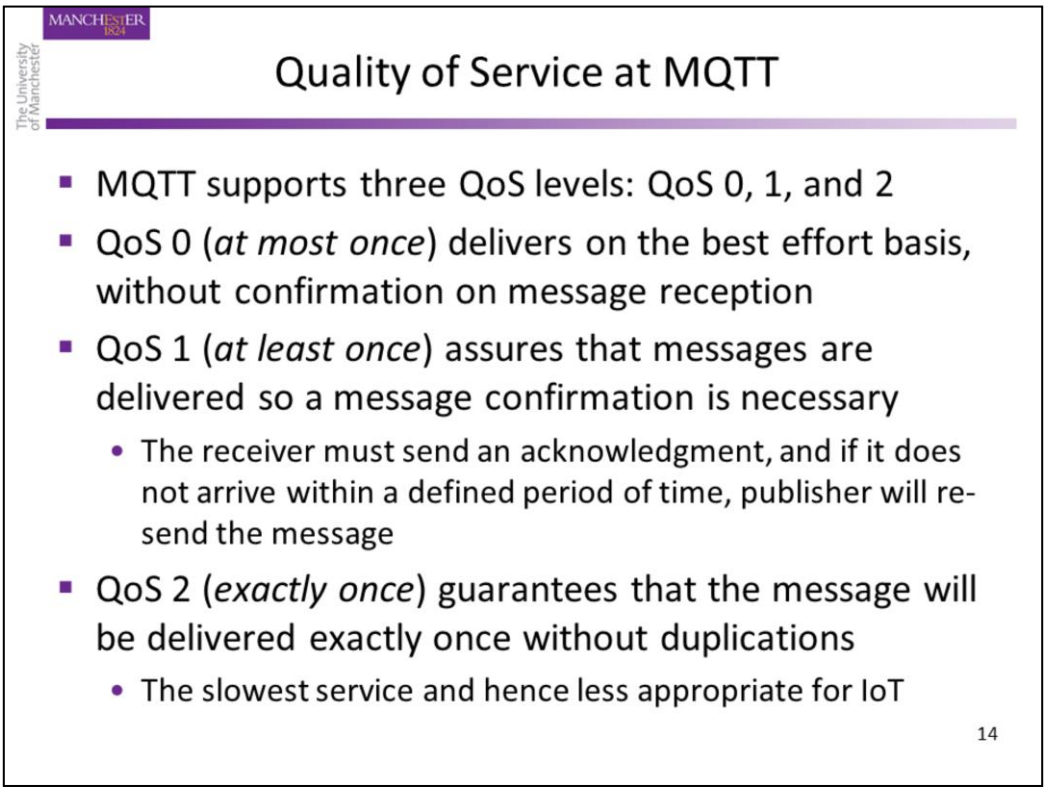
- MQTT was developed by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom Control Systems Ltd (Eurotech). MQTT was released by IBM, with its latest version MQTT v3.1 adopted for IoT by the OASIS (Andrew Banks and Rahul Gupta (Ed.). 29 October 2014. MQTT Version 3.1.1. OASIS Standard. http://docs.oasis-open. org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html).
- Die to its lightweight, MQTT is appropriate for resource constrained devices and for non-ideal network connectivity conditions, such as with low bandwidth.
- The key messages being exchanged are: CONNECT/CONNACK to connect a client with the broker; SUBSCRIBE/SUBACK and UNSUBSCRIBE/UNSUBACK to subscribe/unsubscribe a client to/from a topic; PUBLISH/[PUBACK, PUBREC, PUBREL, PUBCOMP] for sending a message from a publisher to the broker or from the broker to a client when the last has been subscribed to the topic together with the associated confirmations, according to the provided Quality of Service (QoS).

Interaction Model in MQTT



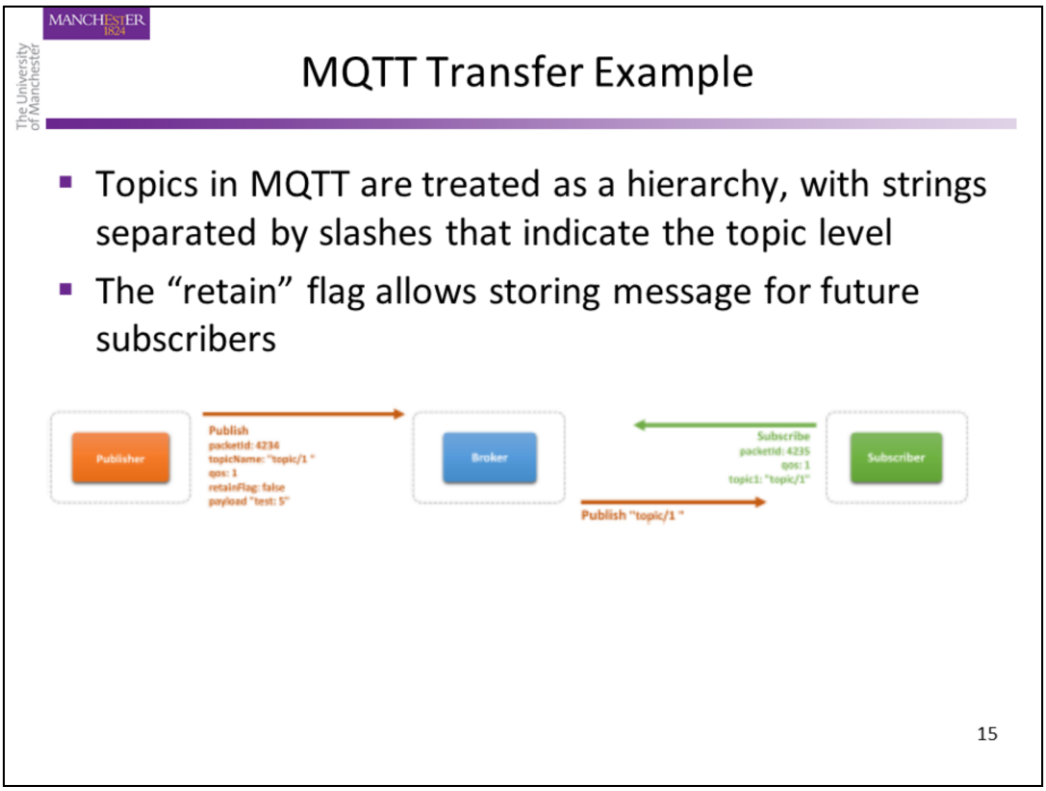
0	1	2	3	4	5	6	7
Message Type				DUP	QoS level		Retain
Remaining length (1-4 bytes)							
Variable length header (optional)							
Variable length message payload (optional)							

13

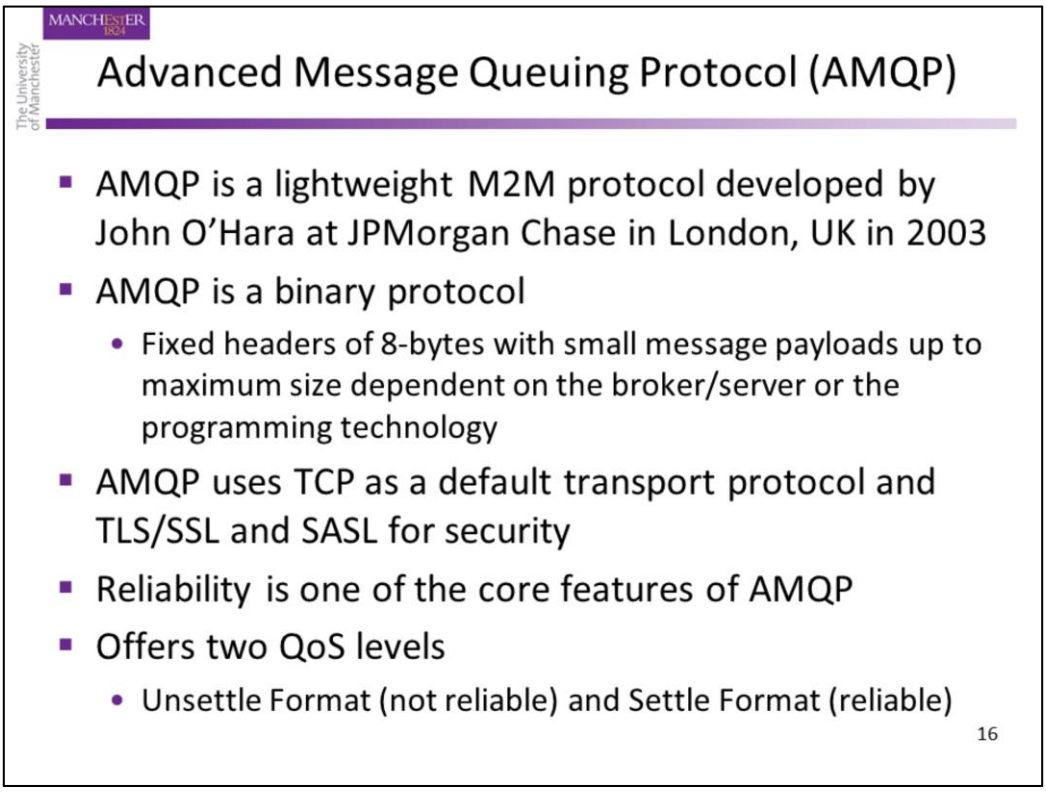


- The choice of the level can be defined both in the publish and the subscribe message body.
- No QoS is a choice in cases where some sensors gather telemetry information over a longer time period, and where the sensors values do not change significantly. It is then acceptable if sometimes the messages are missing, because the general sensor value is still known since most of the message updates have been received.

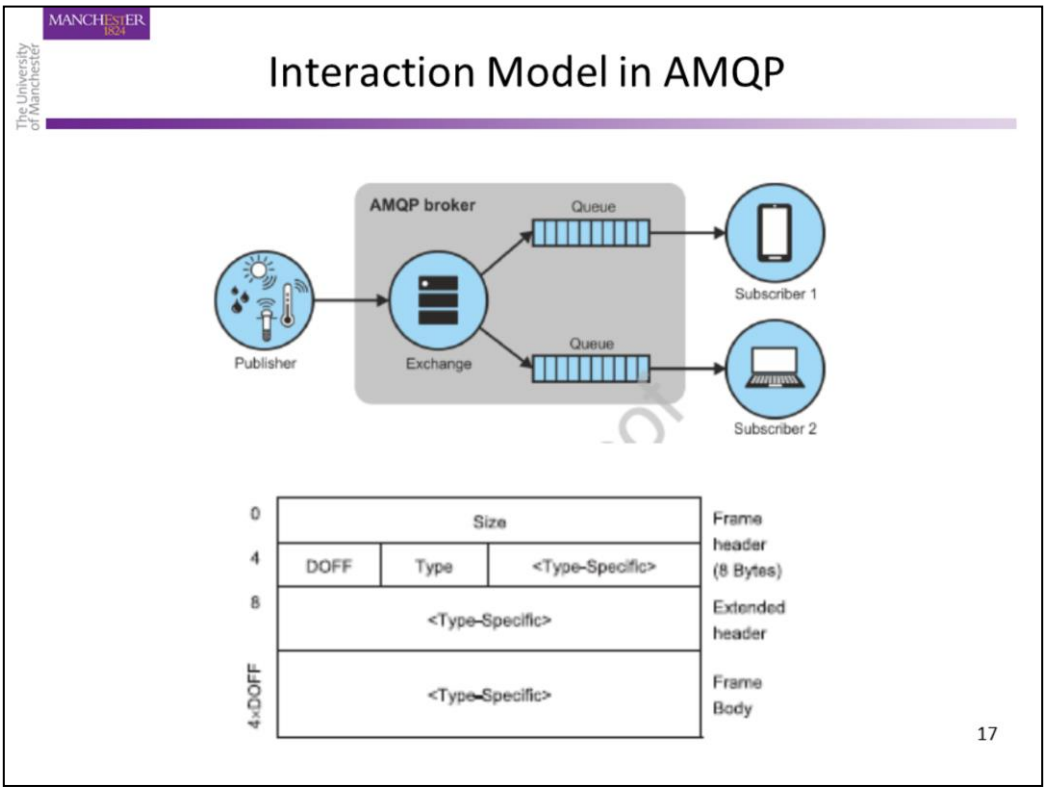
- QoS 0 is the lowest level at which the message is not confirmed or stored by the receiver nor it is sent again by the transmitter ("fire and forget").
- In QoS 1, the message will arrive one or more times at the receiver, which responds with a confirmation message.
- In QoS 2, each message is received once and only once by the receiver with confirmation. It is the safest but also the slowest level with a total of four messages being exchanged between the client and the broker.
- The amount of resources necessary to process MQTT packet increases with the higher chosen QoS level, so it is important to adjust the QoS choice to specific network conditions.



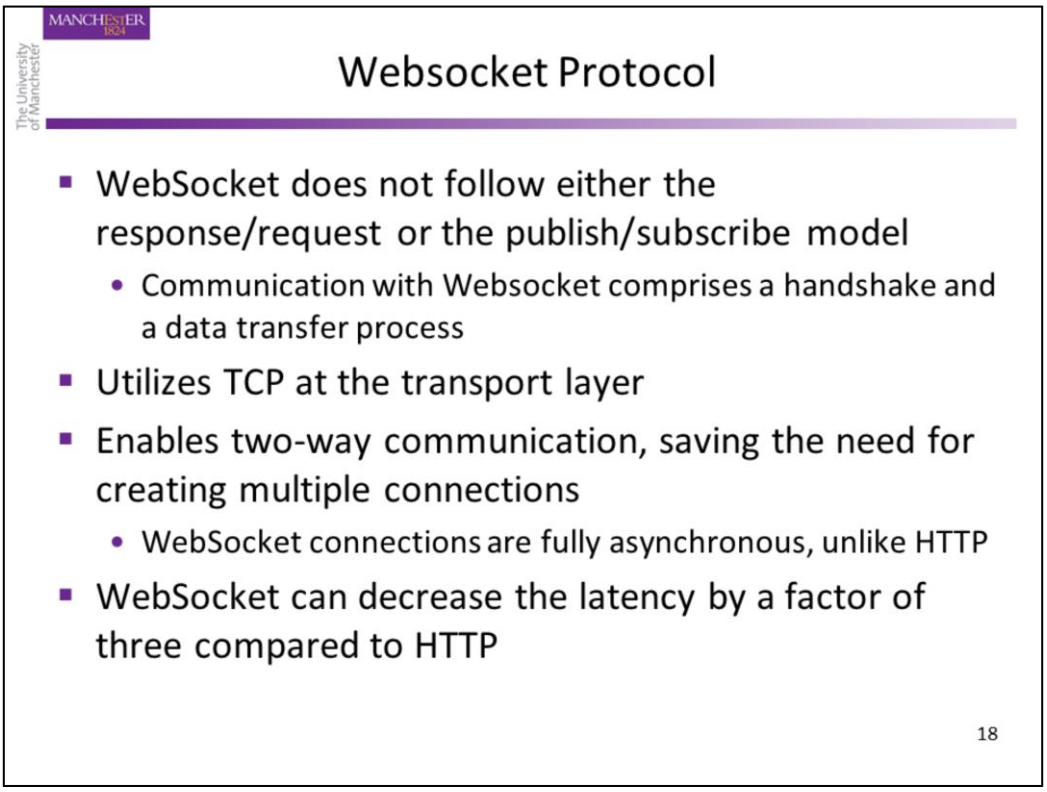
Devices in MQTT can behave as brokers by installing an appropriate MQTT broker library. Examples of such libraries include Emqttd, ActiveMQ, HiveMQ, IBM, MessageSight, JoramMQ, RabbitMQ, and VerneMQ. Note that each of these implementations differ from each other in the way MQTT is implemented. The clients are realized by installing corresponding MQTT client libraries.



- AMQP has been implemented in two very different versions, AMQP 0.9.1 and AMQP 1.0, each with a completely different messaging paradigm.
- AMQP communication systems require that either the publisher or consumer creates an "exchange" with a given name and then broadcasts that name. Publishers and consumers use the name of this exchange to discover each other. Subsequently, a consumer creates a "queue" and attaches it to the exchange at the same time.
- AMQP has relatively high power-, processing- and memory related requirements, making it a rather heavy protocol, which has been its biggest disadvantage in IoT-based ecosystems. This protocol is better suited in the parts of the system that are not bandwidth and latency restricted, with more processing power.



- AMQP 0.9.1 version follows the publish/subscribe model and it is realized with two basic entities in a broker: exchanges and message queues. Exchanges refer to that part of the broker used to route messages received from publishers to the appropriate queues, following predetermined rules and conditions. Message queues are the queues on which messages are routed and remain there until they are read by the corresponding subscriber.
- New AMQP implementations (AMQP 1.0) follow a peer-to-peer paradigm, and can be used without a broker in the middle. Broker is present only in the communication that needs to provide a store-and-forward mechanism, while in other cases direct messaging is possible. This option of supporting different topologies increases the flexibility for the possible AMQP-based solutions, enabling different communication patterns, such as client-to-client, client-to-broker, and broker-to-broker.
- Overall, AMQP offers many capabilities, but it is a heavy protocol in terms of the network resources and the computational power of the nodes it requires. Because of this, its use within the IoT framework is mainly confined between servers or strong autonomous nodes.



- The WebSocket protocol was developed by an initiative within the HTML 5 framework to facilitate communications over TCP.
- The handshake process in Websocket resembles that of HTTP protocol.
- After the handshake process, the HTTP headers are removed for the rest of the session and the client exchanges messages with the server in an asynchronous two-way (fullduplex) communication with an overhead of only 2 bytes (when masking is not used). The session terminates either from the server side or from the client when it is no longer required. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that do not rely on opening multiple HTTP connections.
- With WebSocket, the client and the server can both send frames at any time without any restriction. It is closer to TCP than any of the HTTP protocols. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. By default, the WebSocket Protocol uses port 80 for regular WebSocket connections.

MANCHESTER 1824
The University of Manchester

Summary of Protocol Features

	HTTP	CoAP	MQQT	AMQP	Websocket
Year	1997	2010	1999	2003	2008
Architecture	Client/Server	Client/Server or Client/Broker	Client/Broker	Client/Broker or Client/Server	
Abstraction	Request/Response	Request/Response or Publish/Subscribe	Publish/Subscribe	Publish/Subscribe or Request/Response	
Transport layer	TCP	UDP	TCP	TCP	
Header size (at least)	Undefined	4 bytes	2 bytes	8 bytes	6-14 bytes
QoS/ Reliability	–	NON/CON	3 levels	3 levels	–
Encoding	Text	Binary	Binary	Binary	Binary
Default port	80/ 443 (TLS/SSL)	5683 (UDP Port)/ 5684 (DLTS)	1883/ 8883 (TLS/SSL)	5671 (TLS/SSL), 5672	
Licensing model	Free	Open Source	Open Source	Open Source	N/A
Standards	IETF	IETF	OASIS	OASIS[(1)]	IETF

19

- (1) https://www.oasis-open.org/

MANCHESTER 1824

The University of Manchester

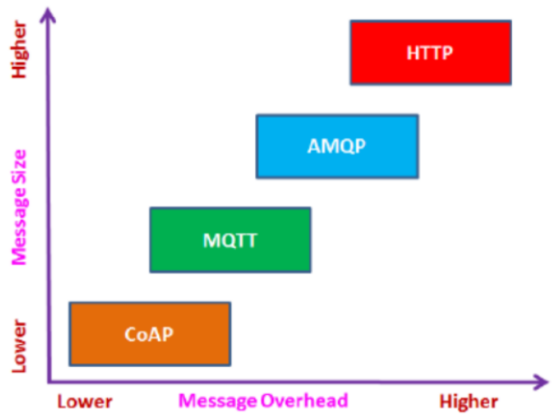
Summary of Protocol Challenges(?)

Protocol	Req.-Rep.	Pub.-Sub.	Standard	Transport	QoS	Security
REST HTTP	✓		IETF	TCP	-	TLS/SSL
MQTT		✓	OASIS	TCP	3 levels	TLS/SSL
CoAP	✓	✓	IETF	UDP	Limited	DTLS
AMQP	✓	✓	OASIS	TCP	3 levels	TLS/SSL
DDS		✓	OMG	TCP/UDP	Extensive	TLS/DTLS/DDS sec.
XMPP	✓	✓	IETF	TCP	-	TLS/SSL
HTTP/2.0	✓	✓	IETF	TCP	-	TLS/SSL

Protocol	Open challenges and efforts in constrained environments
REST HTTP	TLS version 1.3; HTTP/2.0 version
MQTT	TLS version 1.3; MQTT-SN (based on UDP)
CoAP	DTLS optimization
AMQP	not recommended for constrained devices
DDS	DDS security specification
XMPP	lightweight XMPP publish-subscribe scheme

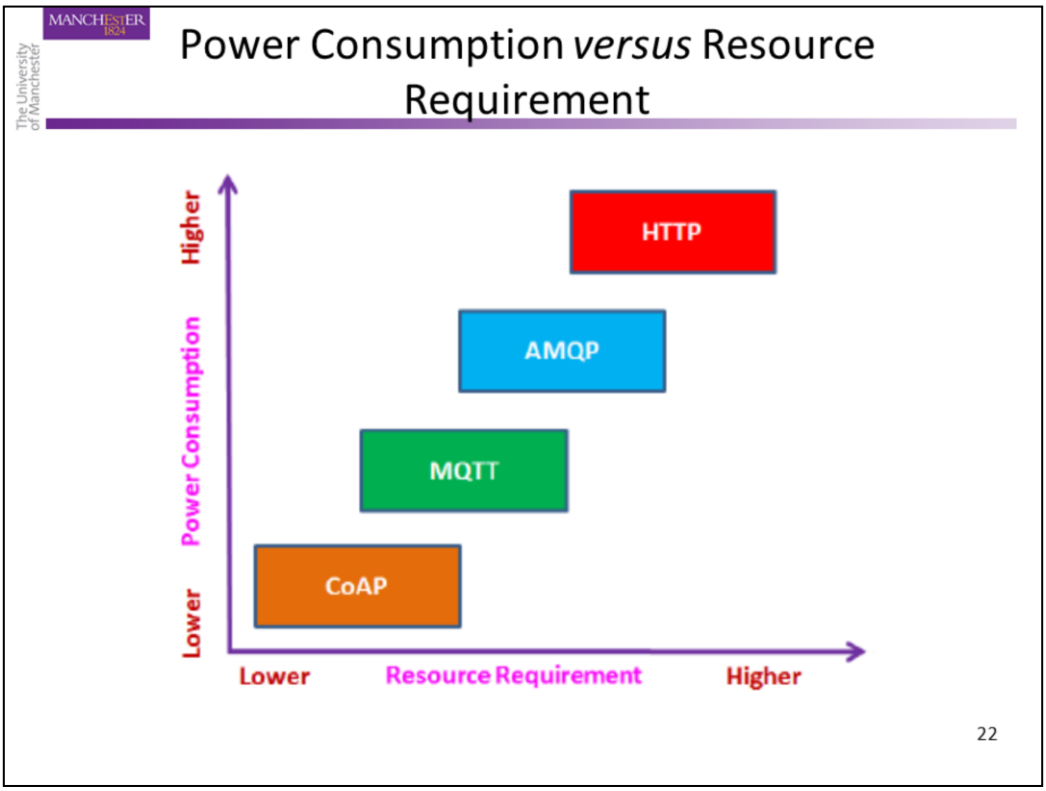
Message Size *versus* Message Overhead

- The dynamic network conditions (e.g., network congestion, noise) and re-transmission overheads are not considered

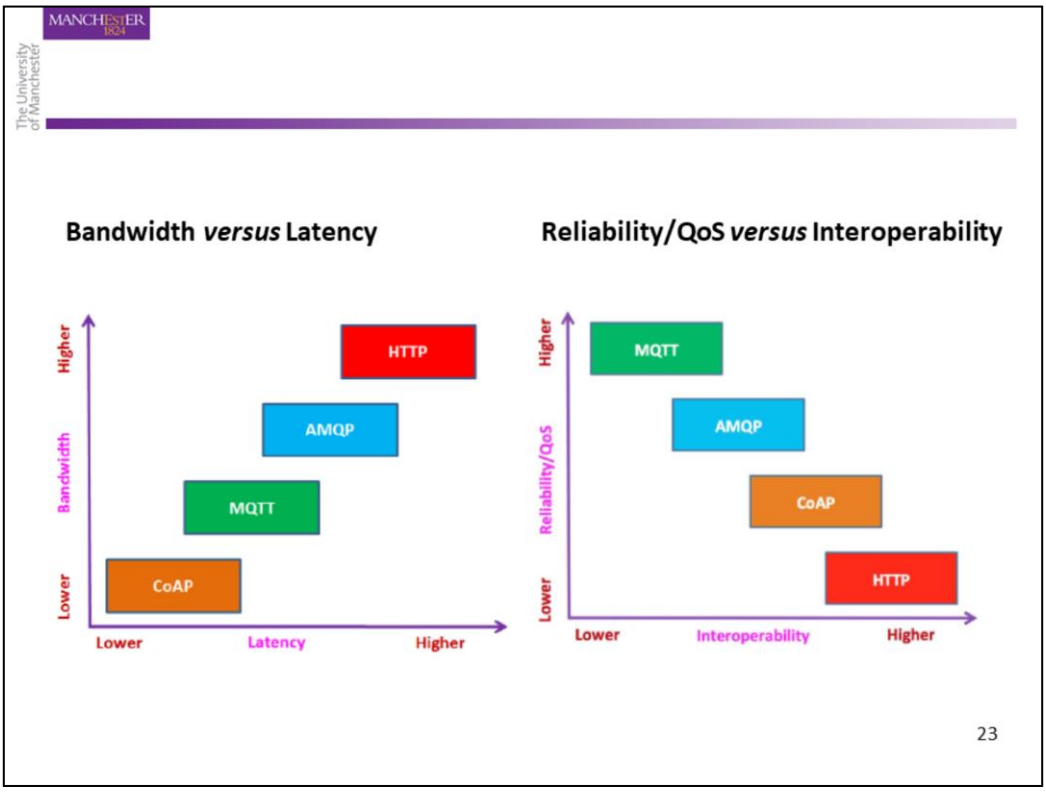


21

- MQTT, AMQP and HTTP run on TCP. Therefore, all these protocols entail TCP connection overheads for connection establishment and closing.
- MQTT is lightweight and has the least header size of 2-byte per message but its requirement of TCP connection increases the overall overhead, and thus the whole message size.
- CoAP runs on UDP that does not add connection overheads as UDP works in fire and forget basis. This situation decreases the overall overhead considerably, and thus the whole message size.
- HTTP among all four is the most verbose and heavy-weight protocol.

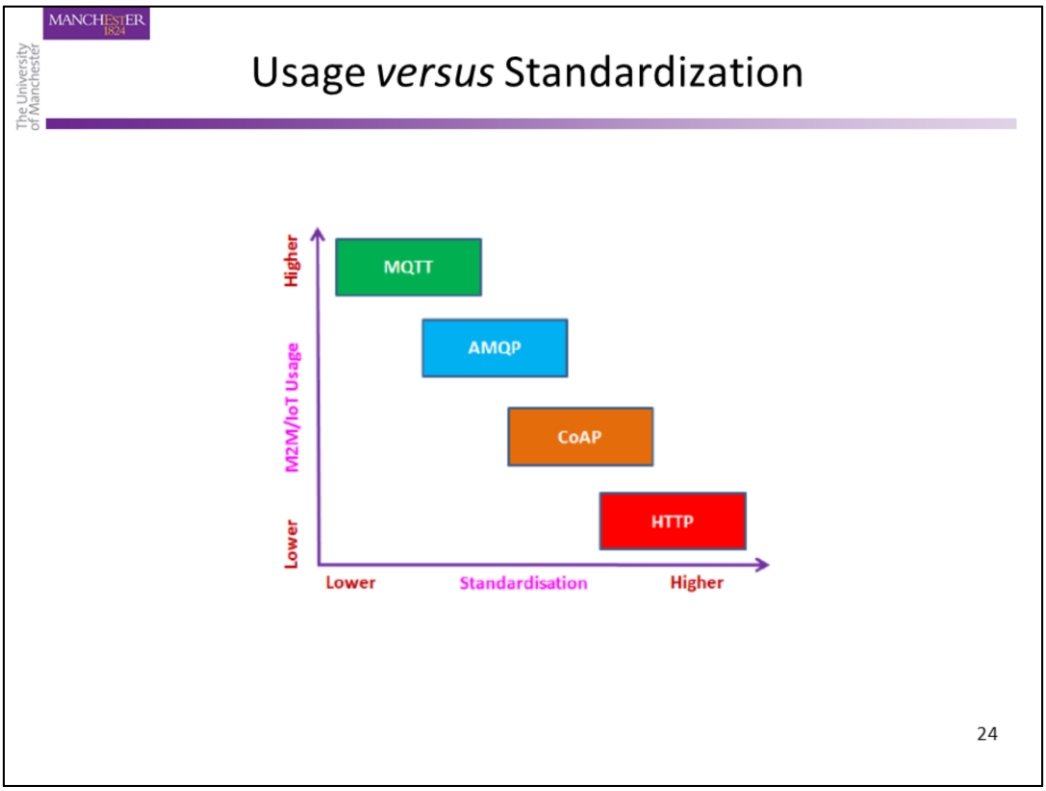


- CoAP and MQTT are designed for low bandwidth and resource-constrained devices and can be deployed on an 8-bit controller and 100's bytes of memory.
- Various experimental studies found that CoAP consumes slightly less power and resources in similar circumstances that is an unreliable transmission scenario (MQTT QoS 0 vs. CoAP NON), and a reliable transmission scenario (MQTT QoS 1 or 2 vs. CoAP CON), while assuming that no packet losses are happened.

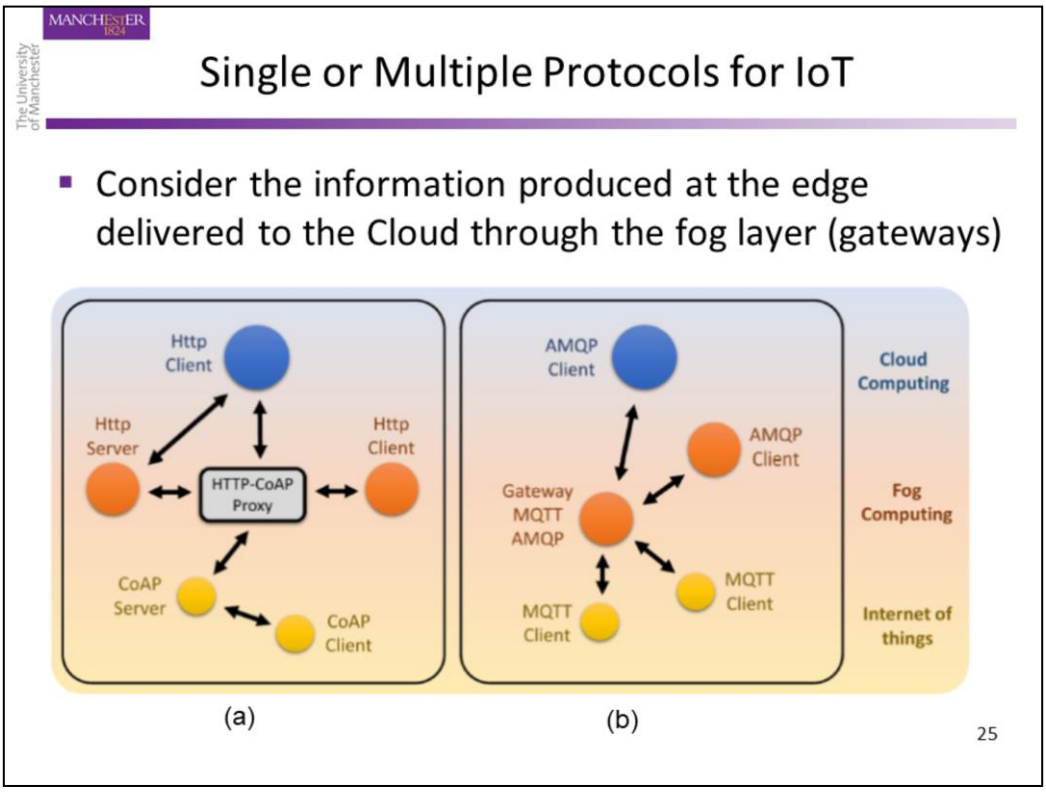


- MQTT, AMQP, and HTTP all employ TCP, which is a major factor in determining the latency and bandwidth requirement and its use does not help to improve latency.
- Comparing MQTT QoS 2 with CoAP CON, the bandwidth usage of MQTT was approximately double than CoAP. This result is due to the four-way handshake mechanism of QoS 2. AMQP's extra services demand moderately higher bandwidth and latency. HTTP takes significantly larger bandwidth and latency time.
- MQTT offers the highest level of quality of services with least interoperability among four, whereas HTTP was designed for greatest interoperability on the Web and did not include reliability as a core feature.
- AMQP defines two QoS levels: Settle Format (similar to MQTT QoS 0) and Unsettle Format (similar to MQTT QoS 1).
- Though CoAP does not provide explicit QoS, it facilitates the use of non-confirmable messages (NON) and confirmable messages (CON), which is very similar to MQTT QoS 0 and QoS 1. The QoS is not a default service of HTTP. Therefore, its default reliability is the TCP guarantee.
- Interoperability is the biggest issue among all IoT protocols. MQTT only supports the publish/subscribe pattern of communication, which barely covers all use cases within the IoT. In AMQP, it is common to use serialization formats such as Protocol Buffers, MessagePack, Thrift, and JSON to serialize structured data in order to publish it as the message

payload. CoAP is a part of the Web architecture and best suited for devices that support UDP or a UDP analogue, however, making it limited to a few special kinds of IoT devices. HTTP-based RESTful clients and servers are the most interoperable because all that is needed to support message exchanges, is an HTTP stack (either on the client or the server).



- MQTT has been employed by the large number of organisations but it is still not a global standard, while, HTTP is a global web standard but mostly not suitable and used in the IoT industry. MQTT is an established M2M protocol and has been used and supported by the large number of organisations such as IBM, Facebook, Eurotech, Cisco, Red Hat, M2Mi, Amazon Web Services (AWS), InduSoft and Fiorano.
- AMQP is the most successful IoT protocol that has been employed in the worlds biggest projects such as Oceanography's monitoring of the Mid-Atlantic Ridge, NASA's Nebula Cloud Computing.
- CoAP has been swiftly gaining momentum and supported by many large companies such as Cisco (Field Area Network), Contiki, Erika and IoTivit.
- AMQP is an OASIS adopted international standard ISO/IEC 19464:2014.
- HTTP is an IETF and W3C standard and already established as a global standard for the Web.



- In (a), it is necessary to deploy a proxy between them that will allow HTTP clients to request resources from CoAP servers and CoAP clients to request resources from HTTP servers. The reference information for implementing a proxy that performs a translation between HTTP and CoAP is given in a document by the CoRE working group. A lot of research effort is put into developing and analyzing HTTP-CoAP proxies and mappings between the two.
- In (b), MQTT was built as a lightweight protocol for devices with limited resources so it could be used for communication between IoT constrained nodes and fog nodes. AMQP is also a lightweight protocol; however, with additional support for security, reliability, provisioning, and interoperability the overhead and message size also increase, thus degrading its performance in nodes with limited processing power. For these reasons, AMQP is more suitable in the more powerful devices, which would position it ideally between fog and cloud nodes.

Case Study of Application Protocols

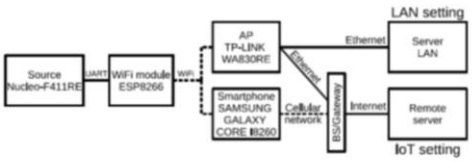
- STM32F411RE microcontroller
  - Up to 100 MHz clock speed
  - 512kB of flash memory and 128kB of RAM
  - Arduino and Morpho headers for easy interfacing with other hardware components
- Network connectivity
  - ESP8266 module
- Communication between STM µC and ESP through UART

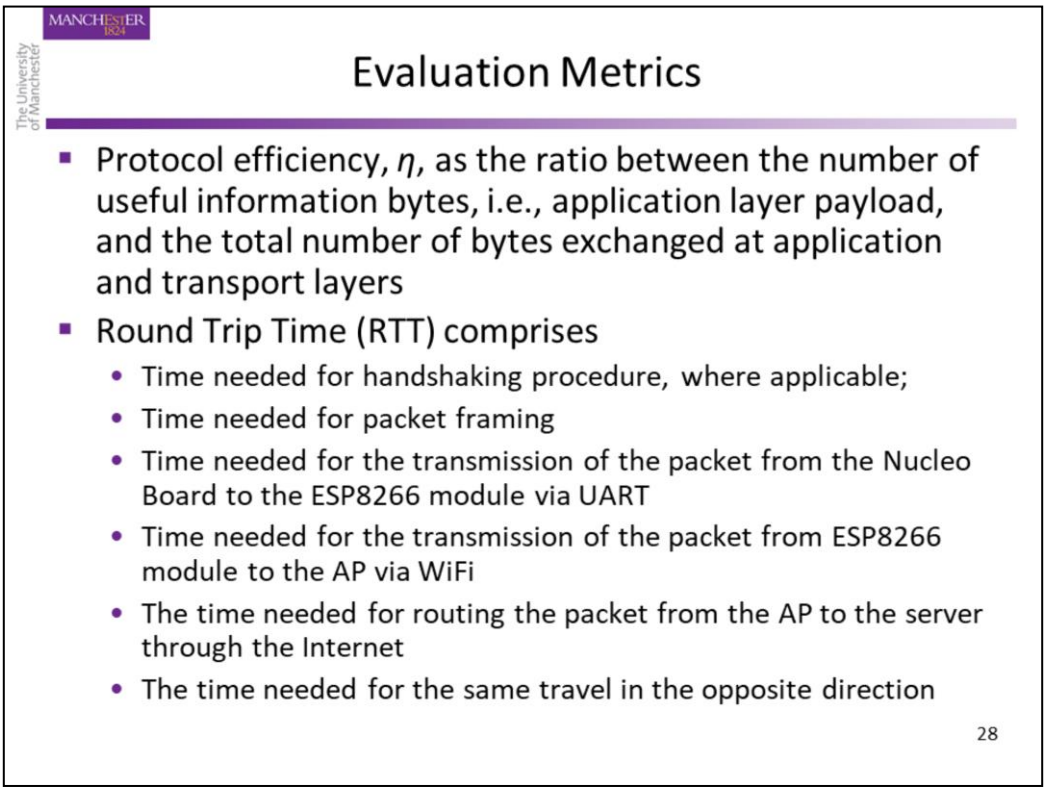
26

- S. Mijovic, E. Shehu, and C. Buratti, “Comparing application layer protocols for the Internet of Things via experimentation,” Proceedings of the IEEE International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016.

Network Configuration for the Experiments

- LAN configuration
  - AP and server in the same LAN
  - A reliable low-latency link
- IoT configuration
  - AP and server not in the same LAN, but connected via Internet
  - Two types of Internet connection
    - in this case ESP8266 module is connected to a home router
    - via cellular network: ESP8266 module is connected to a cellular phone in tethering mode

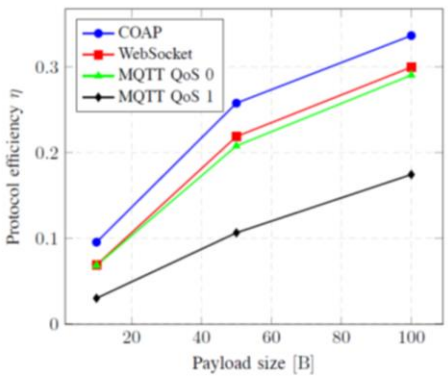




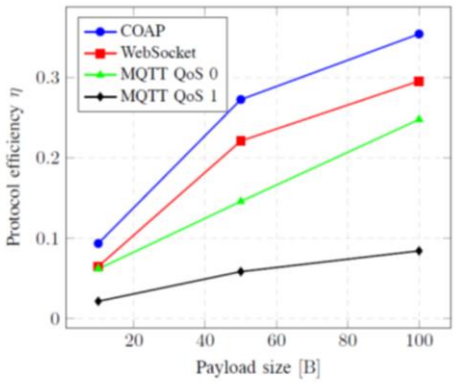
- The RTT of a packet is computed at the IoT device side, i.e., Nucleo board, by means of timestamps.
- One timestamp is taken at the packet generation instant at the application layer, while the second timestamp is taken at the instant of reception of the server response, again at the application layer.
- Since both timestamps are taken at the same device, there is no time reference problem and RTT at the application layer can be accurately estimated.

Protocol Efficiency

LAN scenario



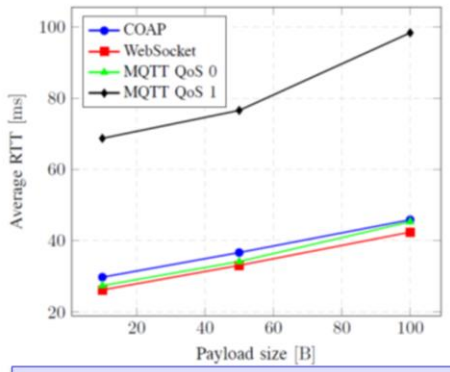
IoT scenario - ISP



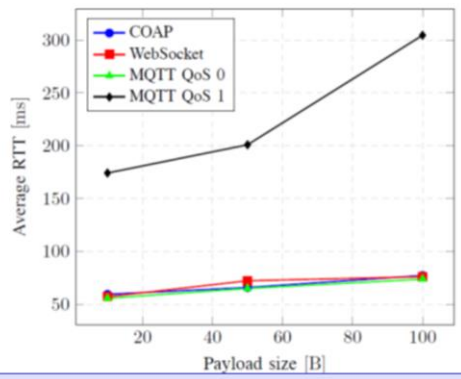
29

Round Trip Time

LAN scenario



IoT scenario - ISP



- What is the right application protocol for IoT?