Internet of Things

Lecture 5: IoT Transport and Application Protocols

Mehdi Rasti
Amirkabir University of Technology

Spring 2020

Contents

Contents

Review of Application and Transport Layer Protocols

- **The transport layer**: takes application messages and transmits those message segments into Layer 3, the networking layer
  - is responsible for end-to-end communication over a network.
  - It provides logical communication between application processes running on different hosts

- **The application layer:** is responsible for data formatting and presentation

Review of Application and Transport Layer Protocols

- **The Transport Layer:**
  - IP-based networks use either TCP or UDP.
  - However, the constrained nature of IoT networks requires a closer look at the use of these traditional transport mechanisms.

- **IoT Application Transport Methods:**
  - The application layer in the Internet is typically based on HTTP.
  - However, HTTP is not suitable in resource constrained environments because it is fairly verbose in nature and thus incurs a large parsing overhead.
  - Many alternate protocols have been developed for IoT environments such as CoAP (Constrained Application Protocol) and MQTT (Message Queue Telemetry Transport).

Review of Transport Layer Protocols

- Transport Layer
  - Transmission Control Protocol (TCP)
    - Connection-oriented protocol
    - It as an equivalent to a traditional telephone conversation, in which two phones must be connected and the communication link established before the parties can talk
  - User Datagram Protocol (UDP)
    - Connection-less protocol
    - This is analogous to the traditional mail delivery system
    - No guarantee of delivery

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

Review of TCP

- With the predominance of human interactions over the Internet, TCP is the main protocol used at the transport layer
- This is largely due to its inherent characteristics, such as its ability to transport large volumes of data into smaller sets of packets
- In addition, it ensures reassembly in a correct sequence, flow control and window adjustment, and retransmission of lost packets
- These benefits occur with the cost of overhead per packet and per session, potentially impacting overall packet per second performances and latency.

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

Review of UDP

- In contrast, UDP is most often used in the context of network services, such as
  - Domain Name System (DNS),
  - Network Time Protocol (NTP),
  - Simple Network Management Protocol (SNMP), and
  - Dynamic Host Control Protocol (DHCP), or
  - for real-time data traffic, including voice and video over IP.

- In these cases, performance and latency are more important than packet retransmissions because re-sending a lost voice or video packet does not add value.

- When the reception of packets must be guaranteed error free, the application layer protocol takes care of that function.

Review of UDP

- While the use of TCP may not strain generic compute platforms and high data-rate networks, it can be challenging and is often overkill on constrained IoT devices and networks.
  - This is particularly true when an IoT device needs to send only a few bytes of data per transaction.
  - When using TCP, each packet needs to add a minimum of 20 bytes of TCP overhead, while UDP adds only 8 bytes.
- TCP also requires the establishment and potential maintenance of an open logical channel

Review of HTPP

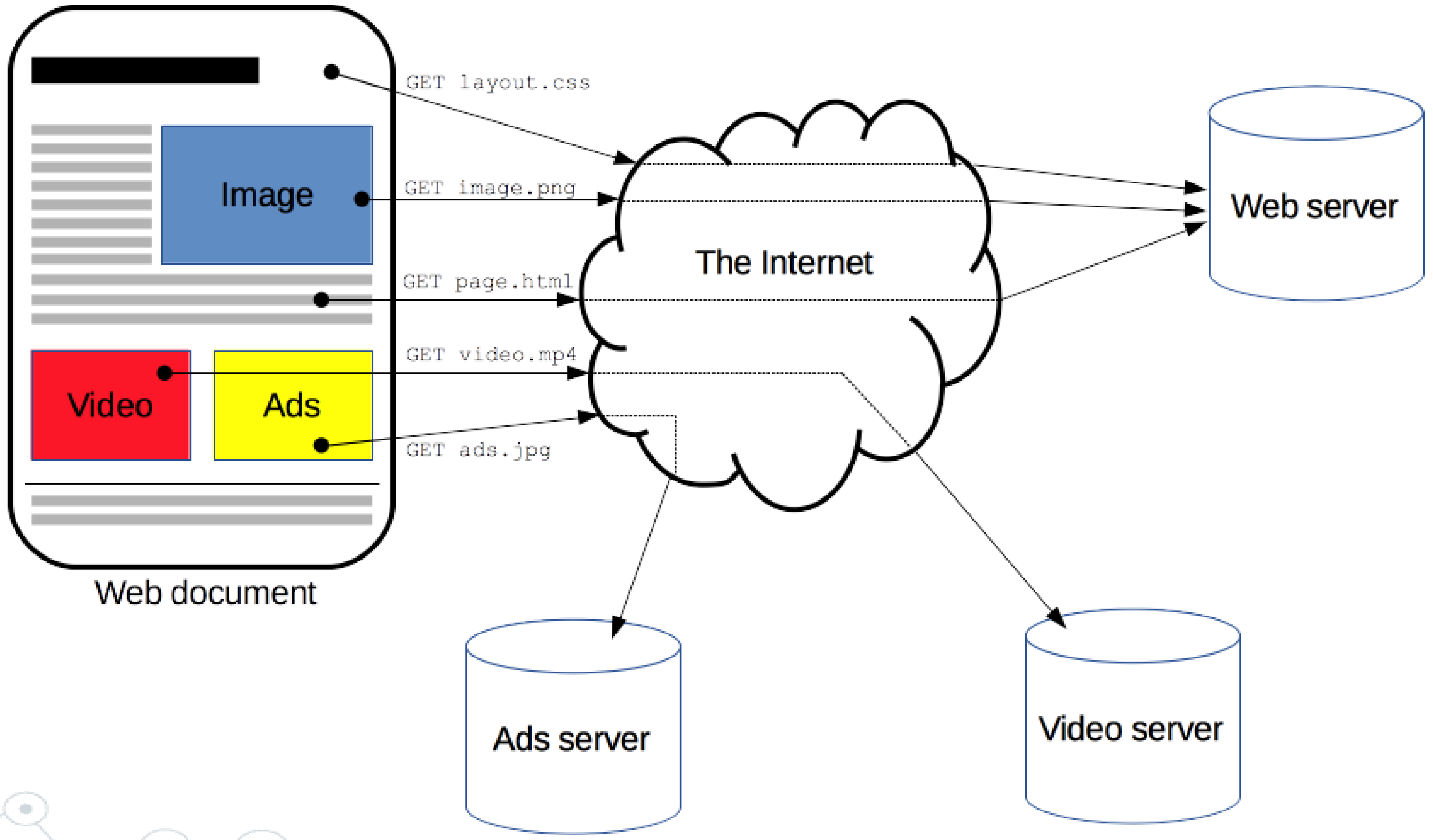
- **HTTP** is a protocol which allows the fetching of resources, such as HTML documents.

- It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser.

- A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more.

An overview of HTTP, Online Available, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

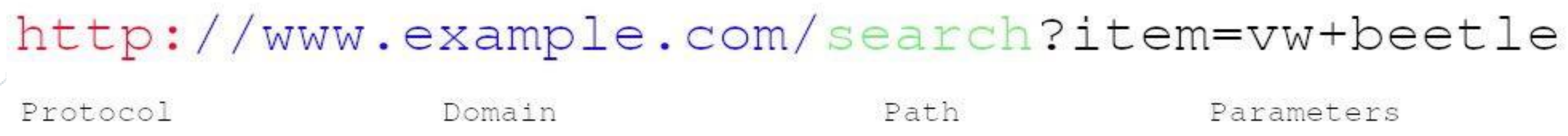
Review of HTPP



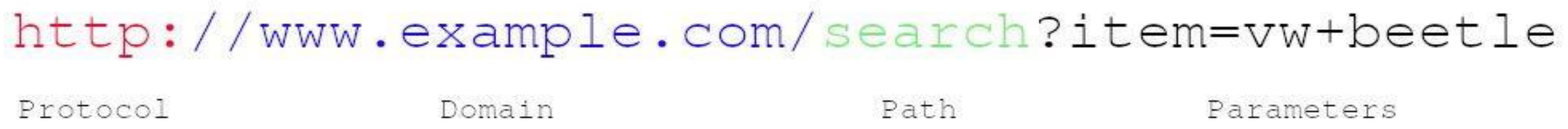
An overview of HTTP, Online Available, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Review of HTPP

- **URLs**
  - The URL (Uniform Resource Locator) is probably the most known concept of the Web.
  - A URL is a web address used to identify resources on the Web.
  - The idea of the web is structured around resources.
  - From its beginnings the Web was the platform for sharing text/HTML files, documents, images etc, and as such it can be considered a collection of resources.



Review of HTPP



- **Domain** — Name that is used to identify one or more IP addresses where the resource is located.
- **Path** —Specifies the resource location on the server. It uses the same logic as a resource location used on the device where you are reading this article (i.e. /search/cars/VWBeetle.pdf or C:/my cars/VWBeetle.pdf).
- **Parameters** — Additional data used to identify or filter the resource on the server.

Review of HTPP

- **HTTP Requests**
  - In HTTP, every request must have an URL address.
  - Additionally, the request needs a method.
  - The four main HTTP methods are:
  - GET
  - PUT
  - POST
  - DELETE

Review of HTPP

- **HTTP flow**
  - When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:
    - Open a TCP connection:
      - The TCP connection is used to send a request, or several, and receive an answer.
      - The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.
    - Send an HTTP message:

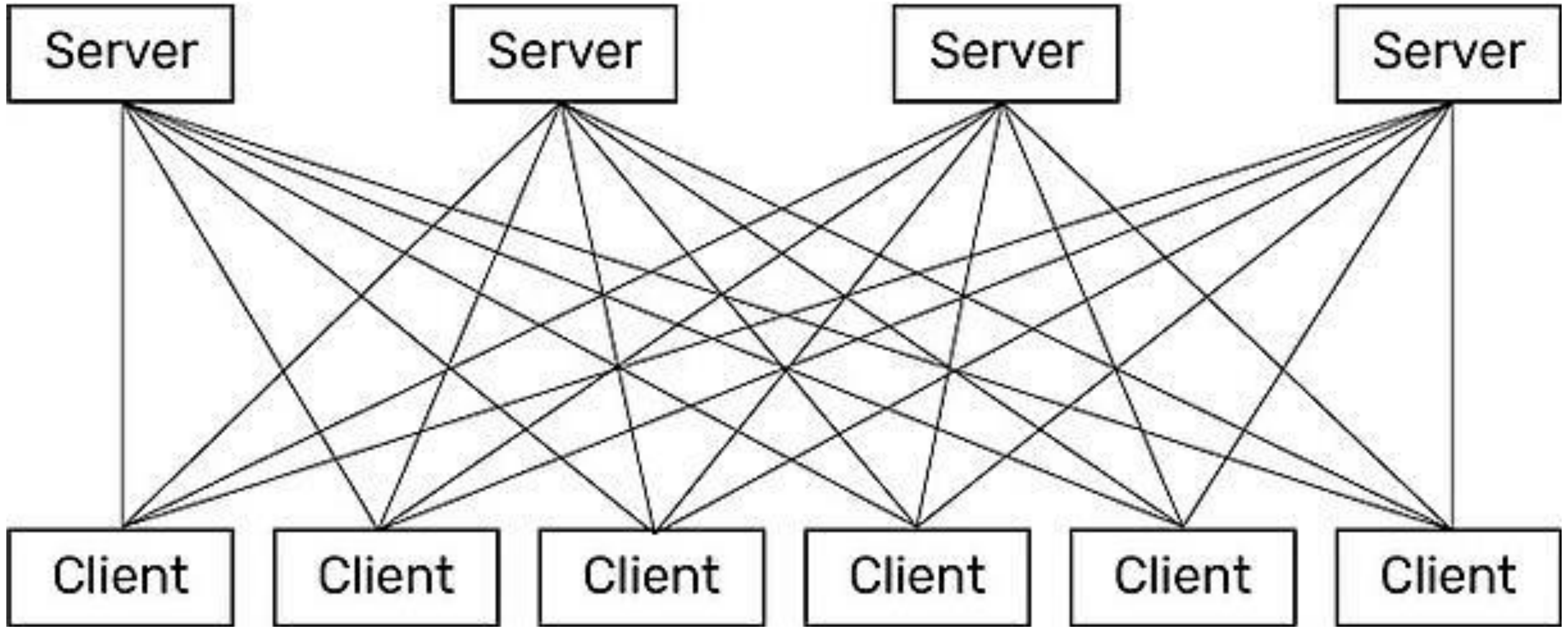
An overview of HTTP, Online Available, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Review of HTPP

- HTTP functions as a request–response protocol in the client–server computing model.
- A web browser, for example, may be the *client* and an application running on a computer hosting a website may be the *server*.
- The client submits an HTTP *request* message to the server.
- The server, which provides *resources* such as HTML files and other content, or performs other functions on behalf of the client, returns a *response* message to the client.

An overview of HTTP, Online Available, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Review of HTPP



* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

Review of HTTP

- HTTP is based on TCP, the data stream-based transmission method in the TCP/IP stack.
  - Among other things, TCP requires that data packets are equipped with counters in order to be able to correct the order of incoming packets in case of doubt.
  - TCP devices need buffers to store and manage packets between them.
- Many small devices therefore use UDP, which delivers datagrams individually and does not guarantee delivery or the actual order.
  - This makes the implementation leaner and more resource-efficient.
  - Unfortunately, HTTP does not pay attention to UDP, newer protocols like QUIC rely on UDP, but are not yet so widespread.

Application protocols for the Internet of Things, online available, https://www.wespeakiot.com/application-protocols-for-the-internet-of-things/

Contents

IoT Application Transport Methods

- As an example, consider the Device Language Message Specification/Companion Specification for Energy Metering (DLMS/COSEM) application layer protocol
  - a popular protocol for reading smart meters in the utilities space, is the de facto standard in Europe.
- Adjustments or optimizations to this protocol should be made depending on the IoT transport protocols that are present in the lower layers.

- For example, if you compare the transport of DLMS/COSEM over a cellular network versus an LLN deployment, you should consider the following:
  - Select TCP for cellular networks because these networks are typically more robust and can handle the overhead.
  - For LLNs, where both the devices and network itself are usually constrained, UDP is a better choice and often mandatory.

IoT Application Transport Methods

- DLMS/COSEM can reduce the overhead associated with session establishment by offering a “long association” over LLNs.
  - *Long association* means that sessions stay up once in place because the communications overhead necessary to keep a session established is much less than is involved in opening and closing many separate sessions over the same time period.
  - Conversely, for cellular networks, a short association better controls the costs by tearing down the open associations after transmitting.

- When transferring large amounts of DLMS/COSEM data, cellular links are preferred to optimize each open association.

- Smaller amounts of data can be handled efficiently over LLNs.
  - Because packet loss ratios are generally higher on LLNs than on cellular networks, keeping the data transmission amounts small over LLNs limits the retransmission of large numbers of bytes.

IoT Application Transport Methods

- The following categories of IoT application protocols and their transport methods are explored in the following sections:
  - **Application layer protocol not present:**
    - In this case, the data payload is directly transported on top of the lower layers.
    - No application layer protocol is used.
  - **Generic web-based protocols:**
    - Generic protocols, are found on many consumer- and enterprise-class IoT devices that communicate over non-constrained networks.
    - A web protocol like HTTP can also be used on small devices.
      - In the Arduino boards area, for example, there are many examples of smart sensors that have their own web server.
    - The user can direct his browser to the IP address of the device and receive data directly in the form of a web page
  - **IoT application layer protocols:**
    - IoT application layer protocols are devised to run on constrained nodes with a small compute footprint and are well adapted to the network bandwidth constraints on cellular or satellite links or constrained 6LoWPAN networks.
    - Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP), covered later in this chapter, are two well-known examples of IoT application layer protocols.

Application Layer Protocol Not Present

- As introduced before, IETF RFC 7228 devices defined as class 0 send or receive only a few bytes of data
- For myriad reasons, such as processing capability, power constraints, and cost, these devices do not implement a fully structured network protocol stack, such as IP, TCP, or UDP, or even an application layer protocol.
- Class 0 devices are usually simple smart objects that are severely constrained.
- Implementing a robust protocol stack is usually not useful and sometimes not even possible with the limited available resources.

Application Layer Protocol Not Present

- For example, consider low-cost temperature and relative humidity (RH) sensors sending data over an LPWA LoRaWAN infrastructure.

- Temperature is represented as 2 bytes and RH as another 2 bytes of data. Therefore, this small data payload is directly transported on top of the LoRaWAN MAC layer, without the use of TCP/IP.

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

Application Layer Protocol Not Present

- Following Example shows the raw data for temperature and relative humidity and how it can be decoded by the application.

```
Temperature data payload over the network: Tx = 0x090c
Temperature conversion required by the application
T = Tx/32 - 50  to  T = 0x090c/32 - 50  to  T = 2316/32 - 50 =
22.4°
RH data payload over the network: RHx = 0x062e
RH conversion required by the application:
100RH = RHx/16-24  to 100RH = 0x062e/16-24 = 74.9  to RH = 74.9%
```

Application Layer Protocol Not Present

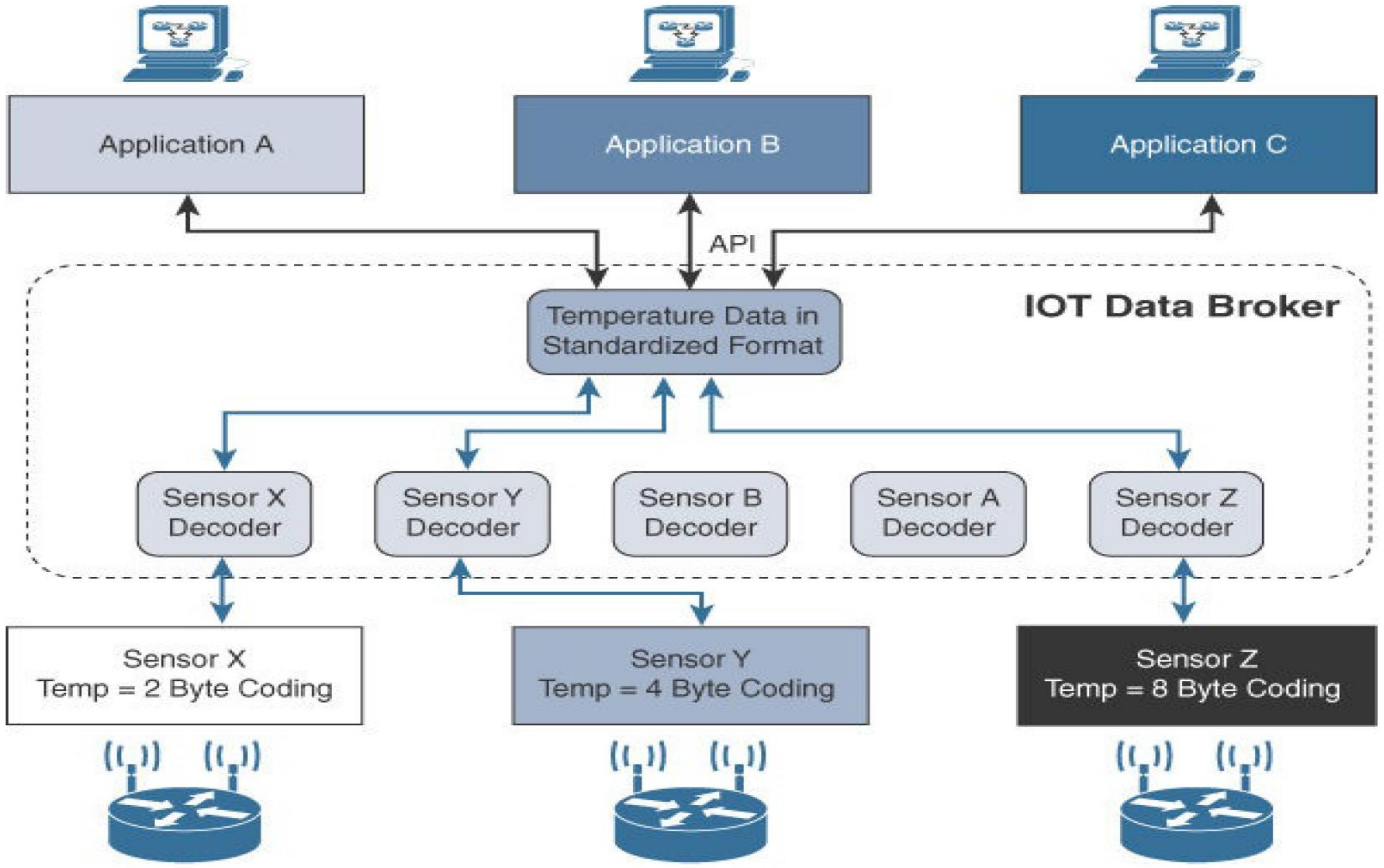
- While many constrained devices, such as sensors and actuators, have adopted deployments that have no application layer, this transportation method has not been standardized.
- This lack of standardization makes it difficult for generic implementations of this transport method to be successful from an interoperability perspective.

Application Layer Protocol Not Present

- Consider different kinds of temperature sensors from different manufacturers.
  - These sensors will report temperature data in varying formats.
  - A temperature value will always be present in the data transmitted by each sensor, but decoding this data will be vendor specific.
- If you scale this scenario out across hundreds or thousands of sensors, the problem of allowing various applications to receive and interpret temperature values delivered in different formats becomes increasingly complex.
- The solution to this problem is to use an IoT data broker, as detailed in Figure of next slide
  - An IoT data broker is a piece of middleware that standardizes sensor output into a common format that can then be retrieved by authorized applications.

Application Layer Protocol Not Present

- IoT Data Broker



* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

Application Layer Protocol Not Present

- You should note that IoT data brokers are also utilized from a commercial perspective to distribute and sell IoT data to third parties.
  - Companies can provide access to their data broker from another company's application for a fee.
- This makes an IoT data broker a possible revenue stream, depending on the value of the data it contains.

Application Layer Protocol Not Present

- Directly transporting data payload without a structured network stack clearly optimizes data transmission over low-data-rate networks,
- However, the lack of a data model implies that each application needs to know how to interpret the data-specific format.
  - This becomes increasingly complex for larger networks of devices with different data payload formats.
- Furthermore, it makes the IoT application environment challenging in terms of evolution, development, interoperability, and so on, and often calls for structured data models and data broker applications.

Web Protocol for the Internet of Things

- A web protocol like HTTP can also be used on small devices.
  - In the Arduino boards area, for example, there are many examples of smart sensors that have their own web server.
  - The user can direct his browser to the IP address of the device and receive data directly in the form of a web page.

- Why don’t all small devices automatically become “web services”?

Web Protocol for the Internet of Things

- Let’s take a closer look at the way HTTP works
  - The protocol is used to interact with web resources (e. g. web pages or forms) on a server.
  - Several methods are available for this purpose.
    - Most inquiries are made to request data (“GET”),
    - Other inquiries data can be transferred, e. g. when an order form has to be filled out in the online shop (method “POST”).

- In principle, HTTP methods can also be used for interaction with devices:
  - Getting data from the device (“GET /temperature”), or
  - Control things/actuators (“POST /fan/control”).

- Most developers are already familiar with HTTP web services. So, why not just have IoT devices connect to web services?

Application protocols for the Internet of Things, online available, https://www.wespeakiot.com/application-protocols-for-the-internet-of-things/

HTTP: Not necessarily suitable for small devices

- There are several reasons why HTTP is not well suited to interacting with constrained devices:
  - HTTP is a heavy weight protocol with many headers and rules
    - HTTP is a chatty protocol from the point of view of resource restriction.
      - For example, meta information such as the accepted formats or the desired language are transmitted to the server in plain text form.
    - The header of an HTTP request can already be large, and the header of an HTTP response can easily contain several hundred bytes.
    - For the query of a web page with several megabytes of size this is not bad, but for the query of a temperature value in the range of approx. 5 bytes it is significant whether the transmission protocol is slim or rather broad.

Application protocols for the Internet of Things, online available, https://www.wespeakiot.com/application-protocols-for-the-internet-of-things/

HTTP: Not Necessarily Suitable for Small Devices

- HTTP is a synchronous protocol.
  - The client waits for the server to respond.
  - That is a requirement for web browsers, but it comes at the cost of poor scalability.
  - In the world of IoT, the large number of devices and most likely an unreliable / high latency network have made synchronous communication problematic.
  - An asynchronous messaging protocol is much more suitable for IoT applications. The sensors can send in readings, and let the network figure out the optimal path and timing for delivery to its destination devices and services.

Getting to know MQTT, online available, https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/

HTTP: Not Necessarily Suitable for Small Devices

– HTTP is one-way.
  - The client must initiate the connection.
  - In an IoT application, the devices or sensors are typically clients, which means that they cannot passively receive commands from the network.

– HTTP is a 1-1 protocol.
  - The client makes a request, and the server responds.
  - It is difficult and expensive to broadcast a message to all devices on the network, which is a common use case in IoT applications.

Getting to know MQTT, online available, https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/

IoT Protocol Stack- Application Layer Protocols

- When considering constrained networks and/or a large-scale deployment of constrained nodes, verbose web-based and data model protocols, as discussed in the previous section, may be too heavy for IoT applications.
- To address this problem, the IoT industry is working on new lightweight protocols that are better suited to large numbers of constrained nodes and networks.
- Two of the most popular protocols are CoAP and MQTT.
- Before introducing CoAP and MQTT, we review two data exchange models

Request-Response vs. Publish-Subscribe

- How does a spreadsheet get to the printer, a YouTube video get to your smartphone, or—most important for automation engineers—a value from a sensor get to your HMI?

- Two Communication (Data Exchange) Models:
  - Request and Response
  - Publish and Subscribe

Request-Response vs. Publish-Subscribe

- Request and Response Model
  - A client computer or software requests data or services, and a server computer or software responds to the request by providing the data or service
  - Examples:
    - Send a spreadsheet to the network printer
      - your spreadsheet program is the client and print server, responds to the request and allocates resources for printers on the network.
    - Watching the YouTube video on your smartphone
      - your web browser or YouTube app is the client, YouTube's web server receives the request and responds by serving the video page to you

Request-response vs. publish-subscribe, part 1: What's the diff? **, online available,** https://blog.opto22.com/optoblog/request-response-vs-pub-sub-part-1

Request-Response vs. Publish-Subscribe

- Publish and Subscribe
  - A central source called a broker (also sometimes called a server) which receives and distributes all data.
  - Pub-sub clients can publish data to the broker or subscribe to get data from it—or both.
  - Clients that publish data send it only when the data changes (report by exception, or RBE).
  - Clients that subscribe to data automatically receive it from the broker/server, but again, only when it changes.
  - The broker does not store data; it simply moves it from publishers to subscribers.
    - When data comes in from a publisher, the broker promptly sends it off to any client subscribed to that data.

Request-response vs. publish-subscribe, part 1: What's the diff? **, online available,** https://blog.opto22.com/optoblog/request-response-vs-pub-sub-part-1

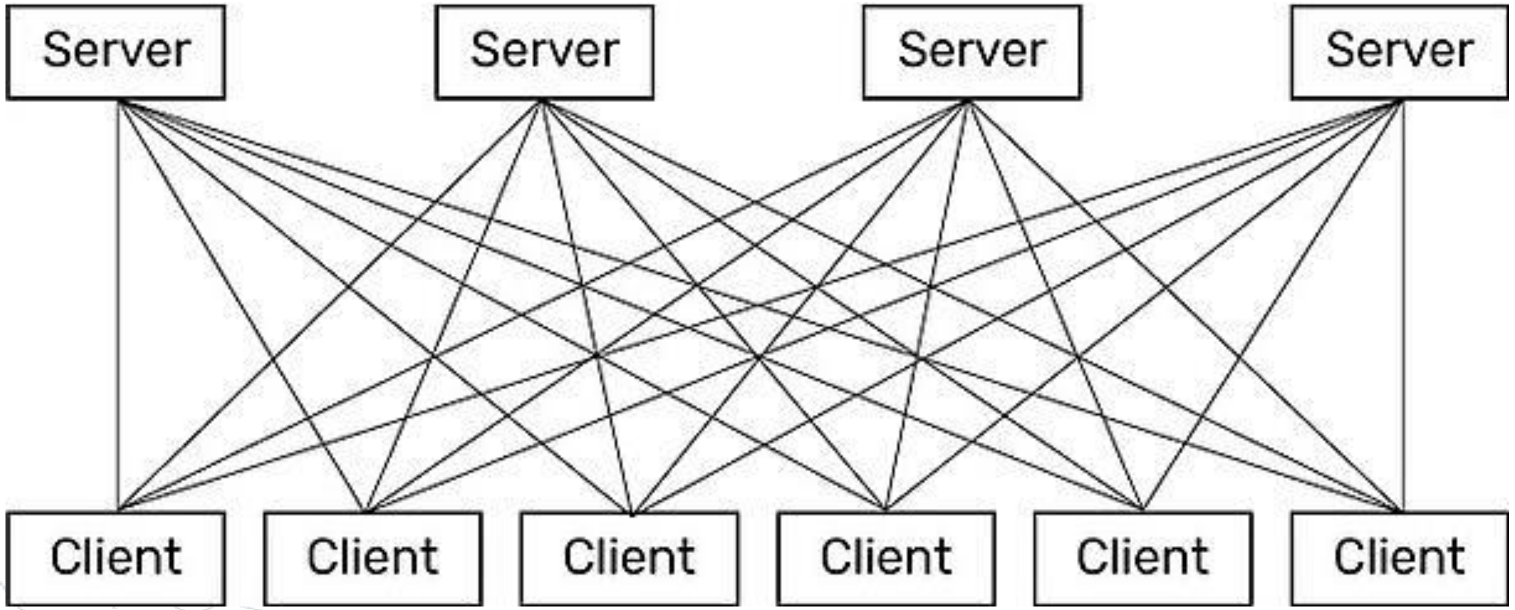
Request-Response vs. Publish-Subscribe-Which to Use?

- In a request-response architecture, each client opens a direct connection to each server, because the client requests data directly from the server.
  - In automation, as fast as multiple times per second—and servers repetitively respond:
    - Q: What’s the sensor value? A: 10
      Q: What’s the sensor value? A: 10
      Q: What’s the sensor value? A: 10
      Q: What’s the sensor value? A: 10
      Q: What’s the sensor value? A: 10
      Q: What’s the sensor value? A: 9
      Q: What’s the sensor value? A: 9

- If your network is robust and has few servers, request-response model works very well.
  - As long as the server has the capacity to respond to client demands and the network can handle the volume of traffic, request-response is a proven, reliable communication method. It's particularly useful for communications over a secure internal network.

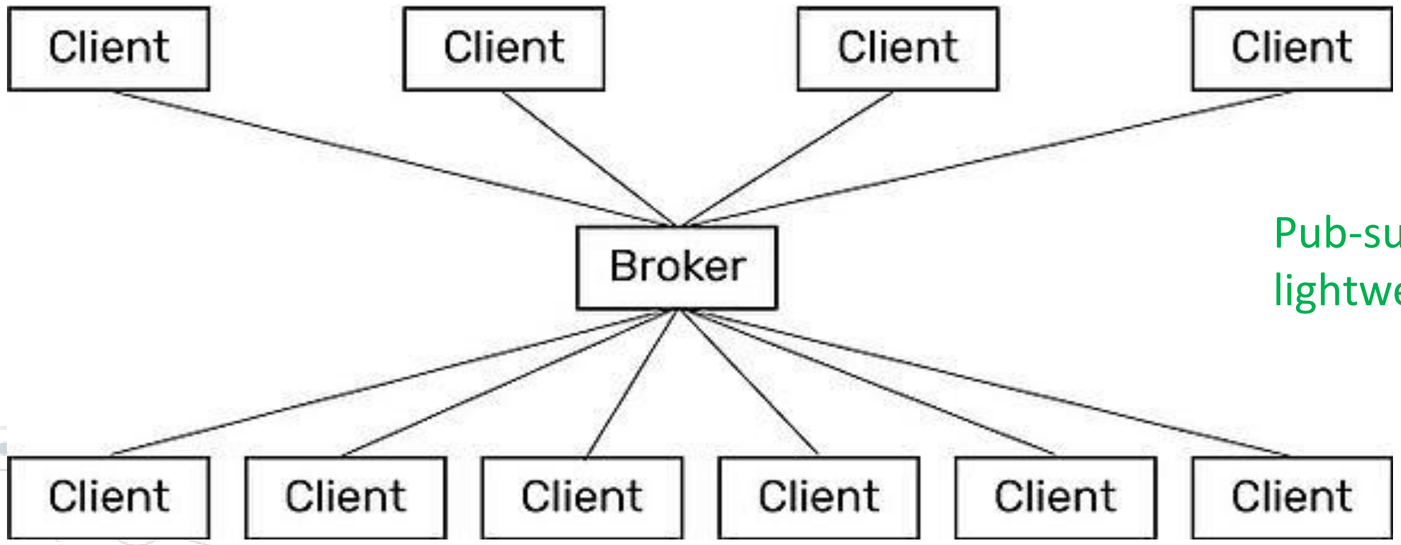
Request-Response vs. Publish-Subscribe-Which to Use?

- What about traffic volume?
  - If you have multiple servers with multiple clients, however, the volume of traffic in a request-response model can quickly become a problem.



Request-Response vs. Publish-Subscribe-Which to Use?

- In contrast, a pub-sub architecture simplifies communications.
  - Direct connections and repetitive requests for data are not needed.
  - The web of links is replaced by a single link from each device to the broker (also called a server).
- The connection between client and broker is kept open and is incredibly lightweight.
  - Only two things travel over this connection: changed data, and a tiny heartbeat to let the broker know that the client is still there.



Pub-sub: good for heavy traffic and lightweight networks

Request-Response vs. Publish-Subscribe-Which to Use?

- So a pub-sub model can make sense if you have many servers and many clients that need to share data and services.

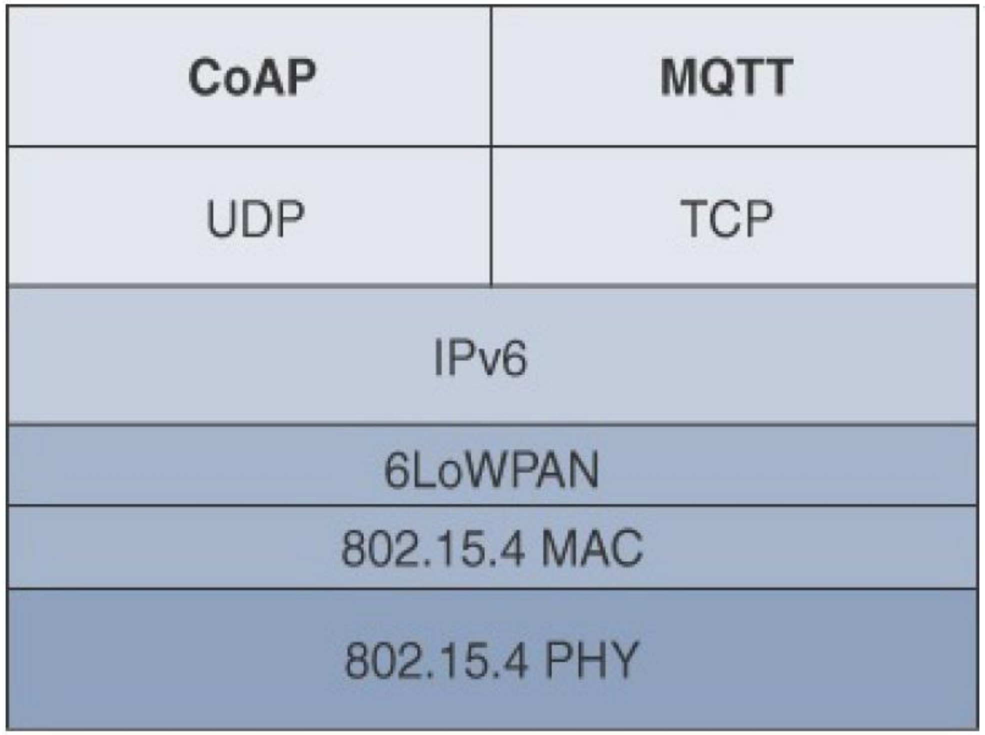
- Since the broker is the central clearinghouse for data, individual servers don’t have to strain to serve multiple clients, and clients don’t have to connect to multiple servers.

- In addition, network traffic is reduced overall, because data is published and sent on a report-by-exception (RBE) basis.
  - that is, only when the data changes—rather than at regular intervals.

- Pub-sub can also make sense when it's difficult to set up a direct connection between a client and a server, or when the network is low-bandwidth, expensive, or unreliable—for example, when monitoring equipment in remote locations.

IoT Application Layer Protocols

- Constrained Application Protocol (CoAP)
  - Uses both client-server and pub-Sub methods
  - UDP-based
- Message Queuing Telemetry Transport (MQTT)
  - Based on pub-sub
  - TCP-based

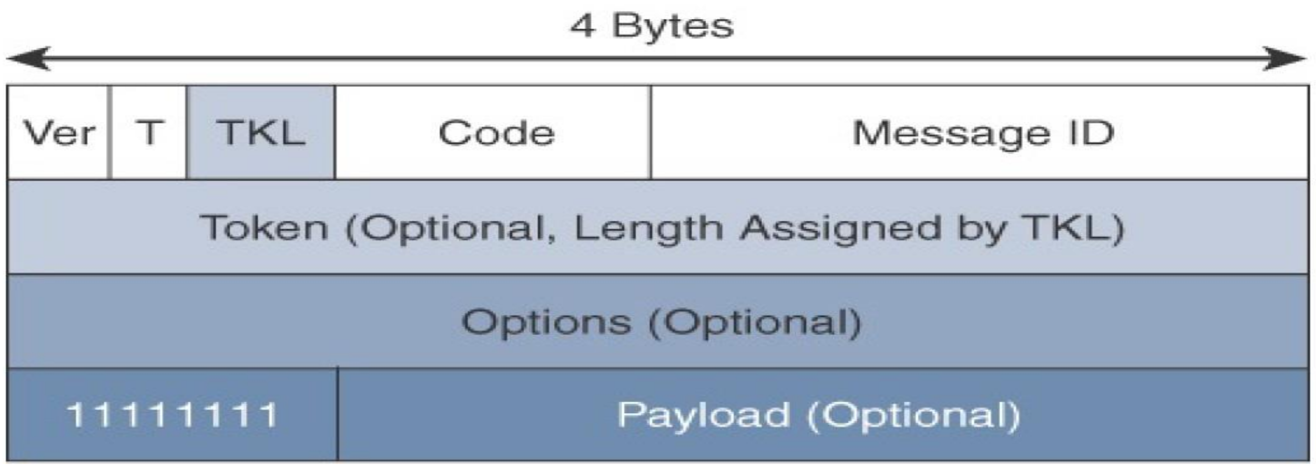


IoT Application Layer Protocols-CoAP

- CoAP has been created close to HTTP in several aspects:
  - It is basically a request/response protocol,
  - It integrates the URIs known from the web world to name resources.
  - It also adopts some of the query methods (such as "GET","PUT","POST", etc.) and defines response codes similar to those of HTTP (e. g. 4.04 "Not found" if a resource could not be found).
- Web developers who have grown up with HTTP can quickly find their way around CoAP.

IoT Protocol Stack- Application Layer Protocols

- Application Layer Protocols- CoAP
  - The CoAP framework defines simple and flexible ways to manipulate sensors and actuators for data or device management
  - A CoAP message is composed of
    - a short fixed length Header field (4 bytes),
    - a variable-length but mandatory Token field (0–8 bytes),
    - Options fields if necessary, and the Payload field



* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

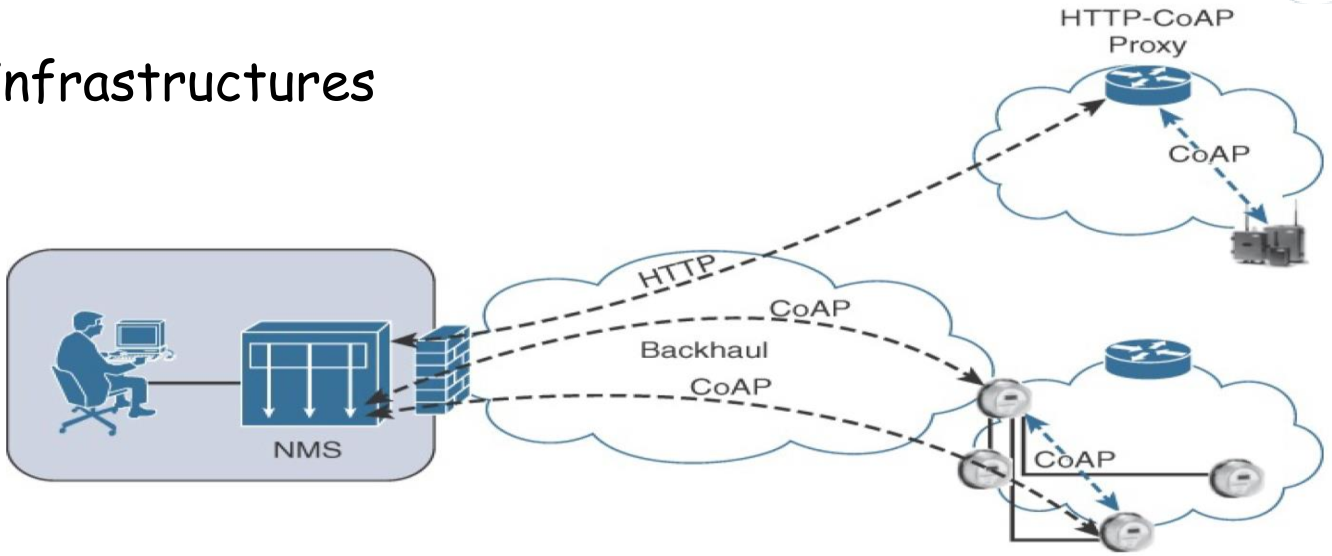
CoAP Message Fields

CoAP Message Field	Description
Ver (Version)	Identifies the CoAP version.
T (Type)	Defines one of the following four message types: Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK), or Reset (RST). CON and ACK are highlighted in more detail in Figure 6-9.
TKL (Token Length)	Specifies the size (0–8 Bytes) of the Token field.
Code	Indicates the request method for a request message and a response code for a response message. For example, in Figure 6-9, GET is the request method, and 2.05 is the response code. For a complete list of values for this field, refer to RFC 7252.
Message ID	Detects message duplication and used to match ACK and RST message types to Con and NON message types.
Token	With a length specified by TKL, correlates requests and responses.
Options	Specifies option number, length, and option value. Capabilities provided by the Options field include specifying the target resource of a request and proxy functions.
Payload	Carries the CoAP application data. This field is optional, but when it is present, a single byte of all 1s (0xFF) precedes the payload. The purpose of this byte is to delineate the end of the Options field and the beginning of Payload.

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols-CoAP

- Application Layer Protocols- CoAP
  - CoAP Communications in IoT Infrastructures

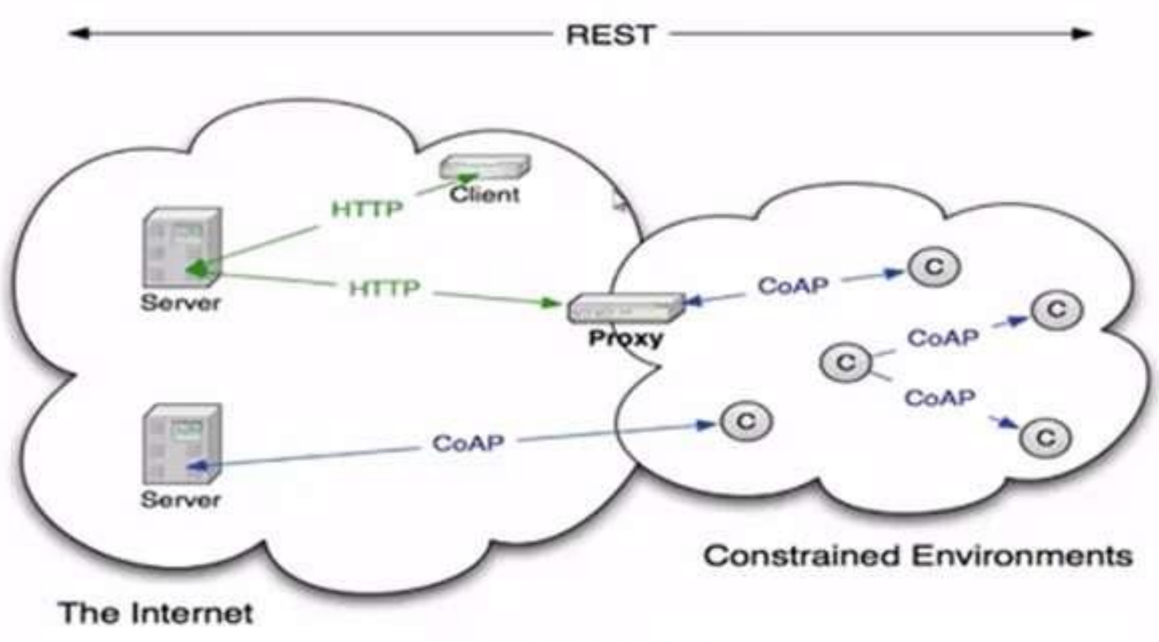


Connections can be between devices located on the same or different constrained networks or between devices and generic Internet or cloud servers, all operating over IP.

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols-CoAP

- Application Layer Protocols- CoAP
  - CoAP Architecture



CoAP Architecture

* https://www.rfwireless-world.com/Terminology/Difference-between-CoAP-and-HTTP.html

IoT Application Layer Protocols-CoAP

- Through the exchange of asynchronous messages, a client requests an action via a method code on a server resource.
- A uniform resource identifier (URI) localized on the server identifies this resource.
- The server responds with a response code that may include a resource representation.
- The CoAP request/response semantics include the methods GET, POST, PUT, and DELETE.

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols-CoAP

- CoAP Messages Type

  1. Confirmable

     - a reliable transmission of messages on UDP protocol
     - basic congestion control with a default time-out
     - simple stop and wait retransmission with exponential back-off mechanism
     - detection of duplicate messages through a message ID
     - while running over UDP, CoAP offers a reliable transmission of messages when a CoAP header is marked as "confirmable."

  2. Non-Confirmable

     - not require reliable transmission

IoT Application Layer Protocols-CoAP

- **CoAP Messages Type**

  3. Acknowledgement

     – the recipient must explicitly either acknowledge or reject the confirmable message using the same message ID

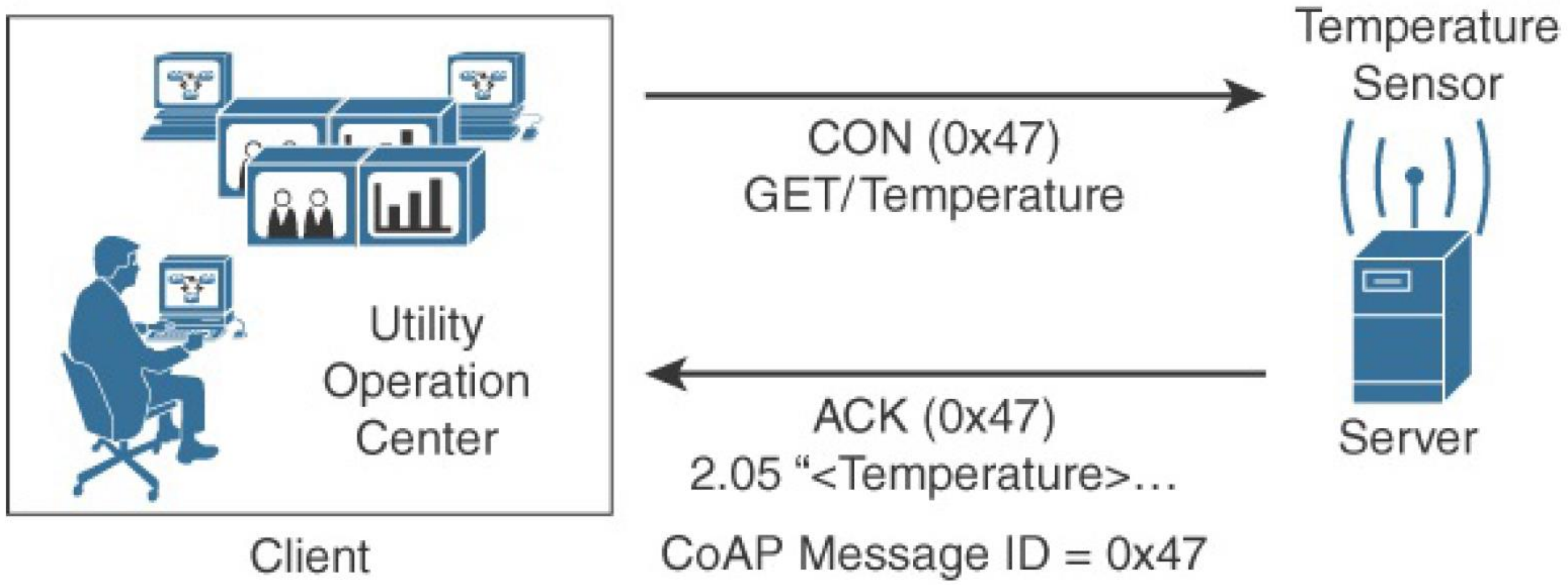
  4. Reset

     – a recipient sends a reset message when can't process a confirmable or non-confirmable message

IoT Application Layer Protocols-CoAP

- CoAP Messages semantics
  1. Get
  2. Post
  3. Put
  4. Delete
- Method codes and response codes included in some of these four-types messages make them carry requests or responses

IoT Application Layer Protocols-CoAP



* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols-CoAP

- CoAP vs HTTP
  - CoAP is upgraded version of HTTP.
    - It is designed for resource constrained applications such as IoT/WSN/M2M etc.
  - CoAP is based on UDP.
    - It uses ACK messages so that it will become reliable like TCP.
  - CoAP has low latency and consumes lesser power compare to HTTP.

* https://www.rfwireless-world.com/Terminology/Difference-between-CoAP-and-HTTP.html

IoT Application Layer Protocols-CoAP

- CoAP vs HTTP
  - Differences of CoAP with HTTP result mainly in the amount of data used.
    - In contrast to the plain text format of HTTP, especially the header format, CoAP code words are packed in bytes or individual bits of a binary representation.
    - There are messages that require a response (“Confirmable”), and messages that do not need a response (“Non-Confirmable”).
      - This means that the information about the reliability of a transmission is lifted to the application layer and can be distinguished by the application in individual cases.

Application protocols for the Internet of Things, online available, https://www.wespeakiot.com/application-protocols-for-the-internet-of-things/

IoT Application Layer Protocols-CoAP

- For example:
  - If a temperature sensor transmits the temperature 1x per minute, a failure of a single message can be merged, it could be mapped as a non-confirmable.
  - If a smart door lock is supposed to lock the door, the message behind it is surely to be classified as “Confirmable”.

IoT Application Layer Protocols-CoAP

- CoAP vs HTTP
  - The asynchronicity of message exchange via UDP also gives CoAP a device the possibility of responding to a request in a resource-saving and time-delayed manner, as well as of transmitting a response in several small (partly "smallest") blocks.
    - The latter is particularly useful for lossy networks with small packet sizes such as those in the WPAN and LPWAN.

Application protocols for the Internet of Things, online available, https://www.wespeakiot.com/application-protocols-for-the-internet-of-things/

IoT Application Layer Protocols-CoAP

- CoAP vs HTTP

Feature	CoAP	HTTP
Protocol	It uses UDP.	It uses TCP.
Network layer	It uses IPv6 along with 6LoWPAN.	It uses IP layer.
Multicast support	It supports.	It does not support.
Architecture model	CoAP uses both client-Server & Publish-Subscribe models.	HTTP uses client and server architecture.
Synchronous communication	CoAP does not need this.	HTTP needs this.
Overhead	Less overhead and it is simple.	More overhead compare to CoAP and it is complex.
Application	Designed for resource constrained networking devices such as WSN/IoT/M2M.	Designed for internet devices where there is no issue of any resources.

* https://www.rfwireless-world.com/Terminology/Difference-between-CoAP-and-HTTP.html

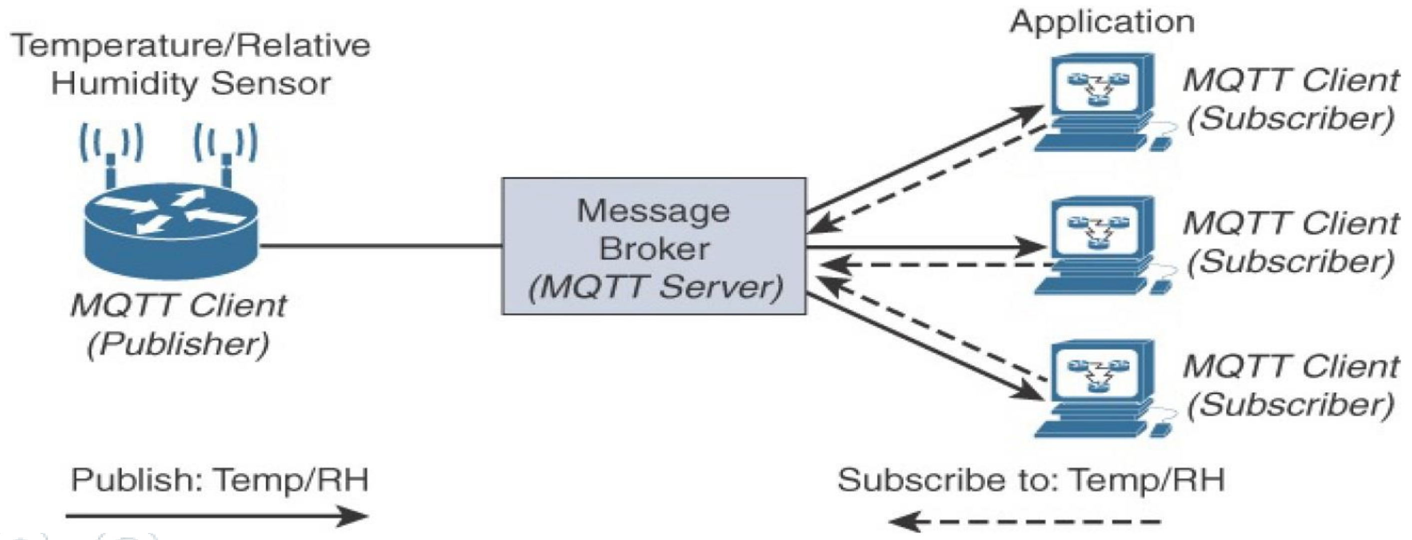
IoT Application Layer Protocols-MQTT

- At the end of the 1990s, engineers from IBM and Arcom (acquired in 2006 by Eurotech) were looking for a reliable, lightweight, and cost-effective protocol to monitor and control a large number of sensors and their data from a central server location, as typically used by the oil and gas industries.

- Their research resulted in the development and implementation of the Message Queuing Telemetry Transport (MQTT) protocol that is now standardized by the Organization for the Advancement of Structured Information Standards (OASIS).

IoT Application Layer Protocols-MQTT

- Application Layer Protocols- MQTT
  - MQTT Publish/Subscribe Framework



  - Is like Twitter

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols-MQTT

- The MQTT protocol defines two types of entities in the network:
  - a message broker
  - a number of clients.
- The broker is a server that receives all messages from the clients and then routes those messages to relevant destination clients.
- A client is anything that can interact with the broker to send and receive messages.
  - A client could be an IoT sensor in the field or an application in a data center that processes IoT data.
  - Both publishers and subscribers are called as clients

Getting to know MQTT, **online available,** developer.ibm.com/articles/iot-mqtt-why-good-for-iot/+&cd=10&hl=en&ct=clnk&gl=ir

IoT Application Layer Protocols-Terminology of MQTT

- **Client**:
  - Clients can be persistent or transient.
    - Persistent clients maintain a session with the broker
    - transient clients are not tracked by the broker.
  - Clients often connect to the broker through libraries and SDKs.
    - There are over a dozen libraries available for C, C++, Go, Java, C#, PHP, Python, Node.js, and Arduino.

Get to Know MQTT: The Messaging Protocol for the Internet of Things, online available, https://thenewstack.io/mqtt-protocol-iot/

IoT Application Layer Protocols-Terminology of MQTT

- **Broker**
  - The broker is the software that receives all the messages from the publishing clients and sends them to the subscribing clients.
  - It holds the connection to persistent clients.
  - Since the broker can become the bottleneck or result in a single point of failure, it is often clustered for scalability and reliability.
    - It is up to the implementers to decide how to create a scalable broker layer.
  - MQTT brokers include
    - Commercial: HiveMQ, Xively, AWS IoT, and Loop,
    - an open source: Mosquitto.

IoT Application Layer Protocols-MQTT

- With MQTT, clients can subscribe to all data (using a wildcard character) or specific data from the information tree of a publisher.
- In addition, the presence of a message broker in MQTT decouples the data transmission between clients acting as publishers and subscribers.
  - In fact, publishers and subscribers do not even know (or need to know) about each other.
  - A benefit of having this decoupling is that the MQTT message broker ensures that information can be buffered and cached in case of network failures.
    - This also means that publishers and subscribers do not have to be online at the same time.

IoT Application Layer Protocols-Terminology of MQTT

- **Topic**
  - A topic in MQTT is an endpoint to that the clients connect.
  - It acts as the central distribution hub for publishing and subscribing messages.
  - A topic is a well-known location for the publisher and subscriber.
  - Topics are simple, hierarchical strings, encoded in UTF-8, delimited by a forward slash.
    - For example, building1/room1/temperature and building1/room1/humidity are valid topic names. Subscribers can choose to subscribe to a specific topic or all the subtopics through wildcards.
    - A subscription to building1/+/temperature will automatically subscribe to the temperature topic of all the rooms in the building.
    - Similarly, building1/#/ will match all the topics available under building1.

IoT Application Layer Protocols-Terminology of MQTT

- **Topic**
  - The pound sign (#) is a wildcard character that matches any number of levels within a topic. The multilevel wildcard represents the parent and any number of child levels.
    - For example, subscribing to **adt/lora/adeunis/#** enables the reception of the whole subtree, which could include topic names such as the following:
      - adt/lora/adeunis/0018B20000000E9E
      - adt/lora/adeunis/0018B20000000E8E
      - adt/lora/adeunis/0018B20000000E9A
  - The plus sign (+) is a wildcard character that matches only one topic level.
    - For example, **adt/lora/+** allows access to **adt/lora/adeunis/** and **adt/lora/abeeway** but not to **adt/lora/adeunis/0018B20000000E9E**.
  - Topic names beginning with the dollar sign ($) must be excluded by the server when subscriptions start with wildcard characters (# or +).
  - Often, these types of topic names are utilized for message broker internal statistics. So messages cannot be published to these topics by clients. For example, a subscription to **+/monitor/Temp** does not receive any messages published to **$SYS/monitor/Temp**. This topic could be the control channel for this temperature sensor.

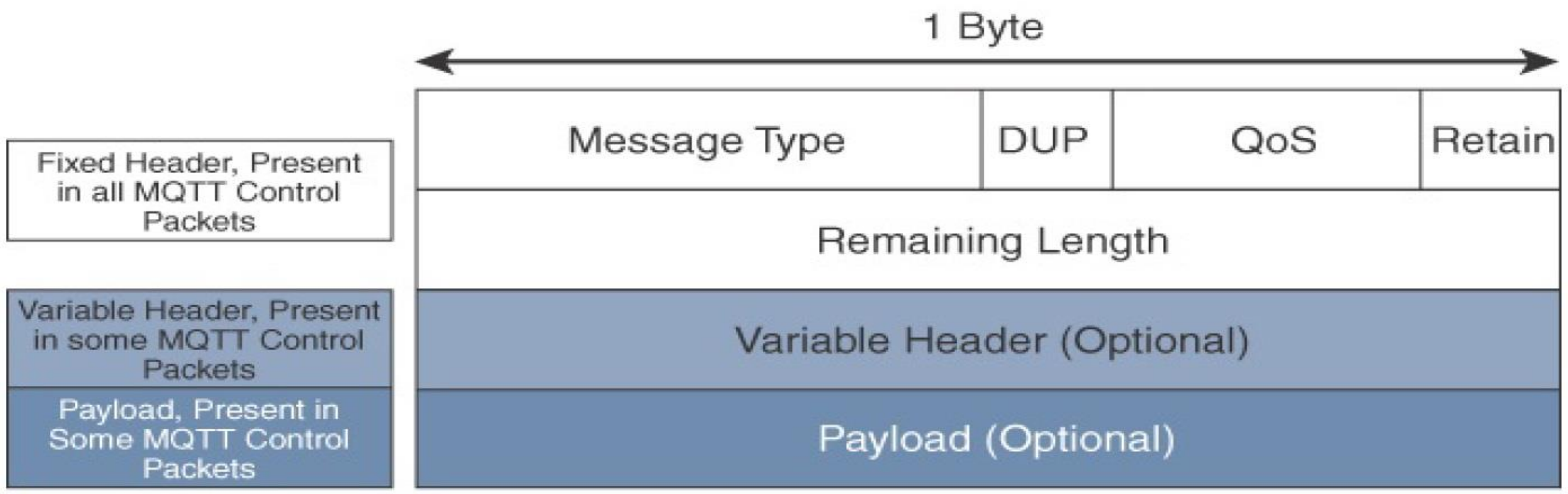
IoT Application Layer Protocols-MQTT

- MQTT is extremely lightweight
  - it takes up almost no space in a device, so that even small devices with very little computing power can use it.

- Each control packet consists of a 2-byte fixed header with optional variable header fields and optional payload.
  - Note that a control packet can contain a payload up to 256 MB.

IoT Application Layer Protocols-MQTT

- MQTT Message Format
  - Compared to the CoAP message format, MQTT contains a smaller header of 2 bytes compared to 4 bytes for CoAP


1 Byte
Fixed Header, Present in all MQTT Control Packets
Variable Header, Present in some MQTT Control Packets
Payload, Present in Some MQTT Control Packets
Message Type
DUP
QoS
Retain
Remaining Length
Variable Header (Optional)
Payload (Optional)

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

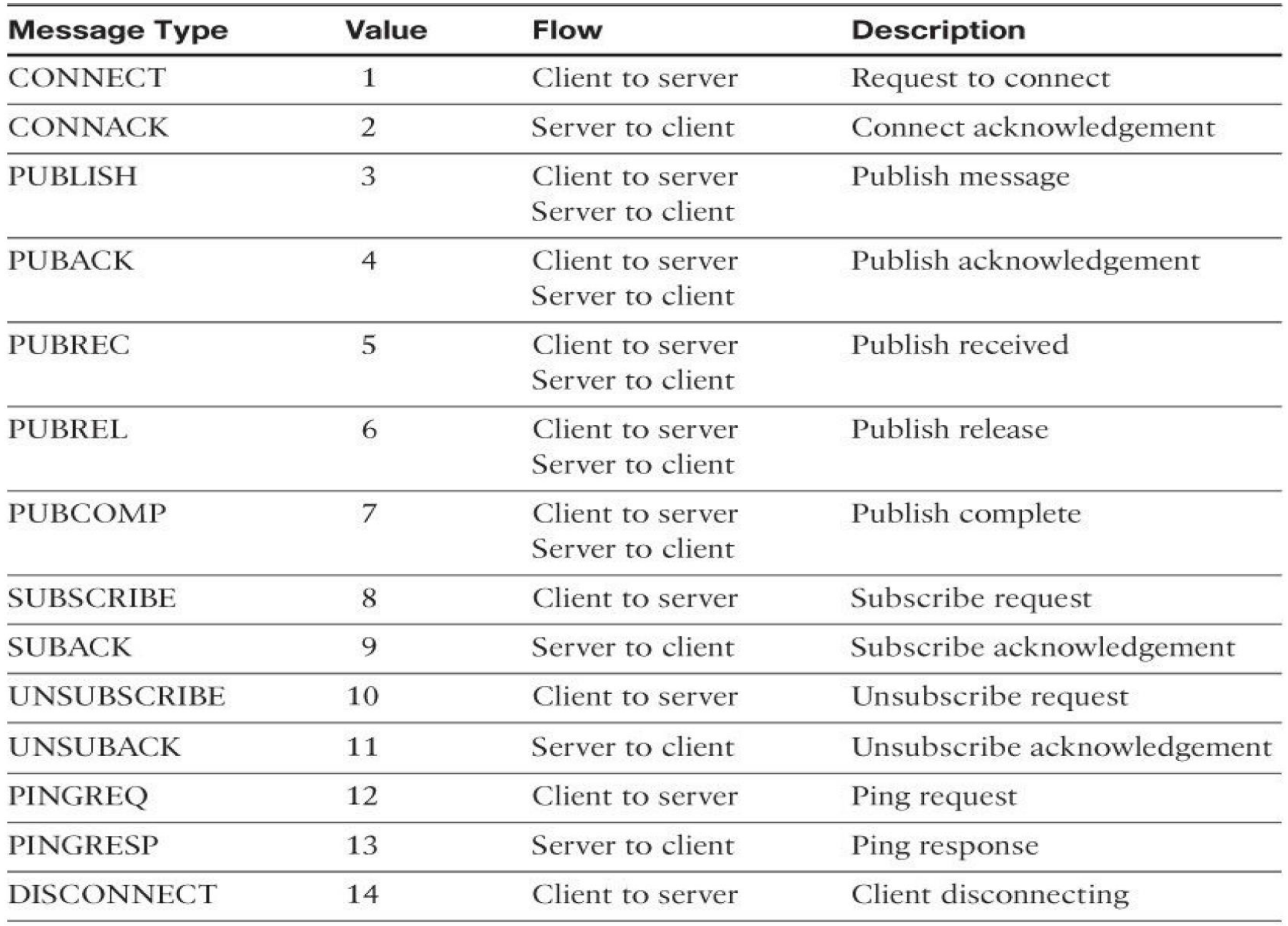
IoT Application Layer Protocols-MQTT

- Message Type
  - identifies the kind of MQTT packet within a message
  - Fourteen different types of control packets are specified in MQTT version 3.1.1.
  - Each of them has a unique value that is coded into the Message Type field.

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols-MQTT

- *MQTT Message Types*



Message Type	Value	Flow	Description
CONNECT	1	Client to server	Request to connect
CONNACK	2	Server to client	Connect acknowledgement
PUBLISH	3	Client to server Server to client	Publish message
PUBACK	4	Client to server Server to client	Publish acknowledgement
PUBREC	5	Client to server Server to client	Publish received
PUBREL	6	Client to server Server to client	Publish release
PUBCOMP	7	Client to server Server to client	Publish complete
SUBSCRIBE	8	Client to server	Subscribe request
SUBACK	9	Server to client	Subscribe acknowledgement
UNSUBSCRIBE	10	Client to server	Unsubscribe request
UNSUBACK	11	Server to client	Unsubscribe acknowledgement
PINGREQ	12	Client to server	Ping request
PINGRESP	13	Server to client	Ping response
DISCONNECT	14	Client to server	Client disconnecting

IoT Application Layer Protocols-MQTT

- *MQTT Message Types*
  - PINGREQ/PINGRESP control packets are used to validate the connections between the client and server.
    - Similar to ICMP pings that are part of IP, they are a sort of keep alive that helps to maintain and check the TCP session.

IoT Application Layer Protocols-MQTT

- DUP (Duplication Flag) field:
  - This flag, when set, allows the client to notate that the packet has been sent previously, but an acknowledgement was not received.
- The QoS header field:
  - allows for the selection of three different QoS levels.
- Retain flag field:
  - Only found in a PUBLISH message, the Retain flag notifies the server to hold onto the message data.
  - This allows new subscribers to instantly receive the last known value without having to wait for the next update from the publisher.

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols-MQTT

- Remaining Length:
  - Specifies the number of bytes in the MQTT packet following this field.

IoT Application Layer Protocols-MQTT

- **QoS 0:**
  - This is a best-effort and unacknowledged data service referred to as “at most once” delivery.
  - The publisher sends its message one time to a server, which transmits it once to the subscribers.
  - No response is sent by the receiver, and no retry is performed by the sender.
  - The message arrives at the receiver either once or not at all.

- **QoS 1:**
  - This QoS level ensures that the message delivery between the publisher and server and then between the server and subscribers occurs at least once.
  - In PUBLISH and PUBACK packets, a packet identifier is included in the variable header.
  - If the message is not acknowledged by a PUBACK packet, it is sent again.
  - This level guarantees “at least once” delivery.

IoT Application Layer Protocols-MQTT

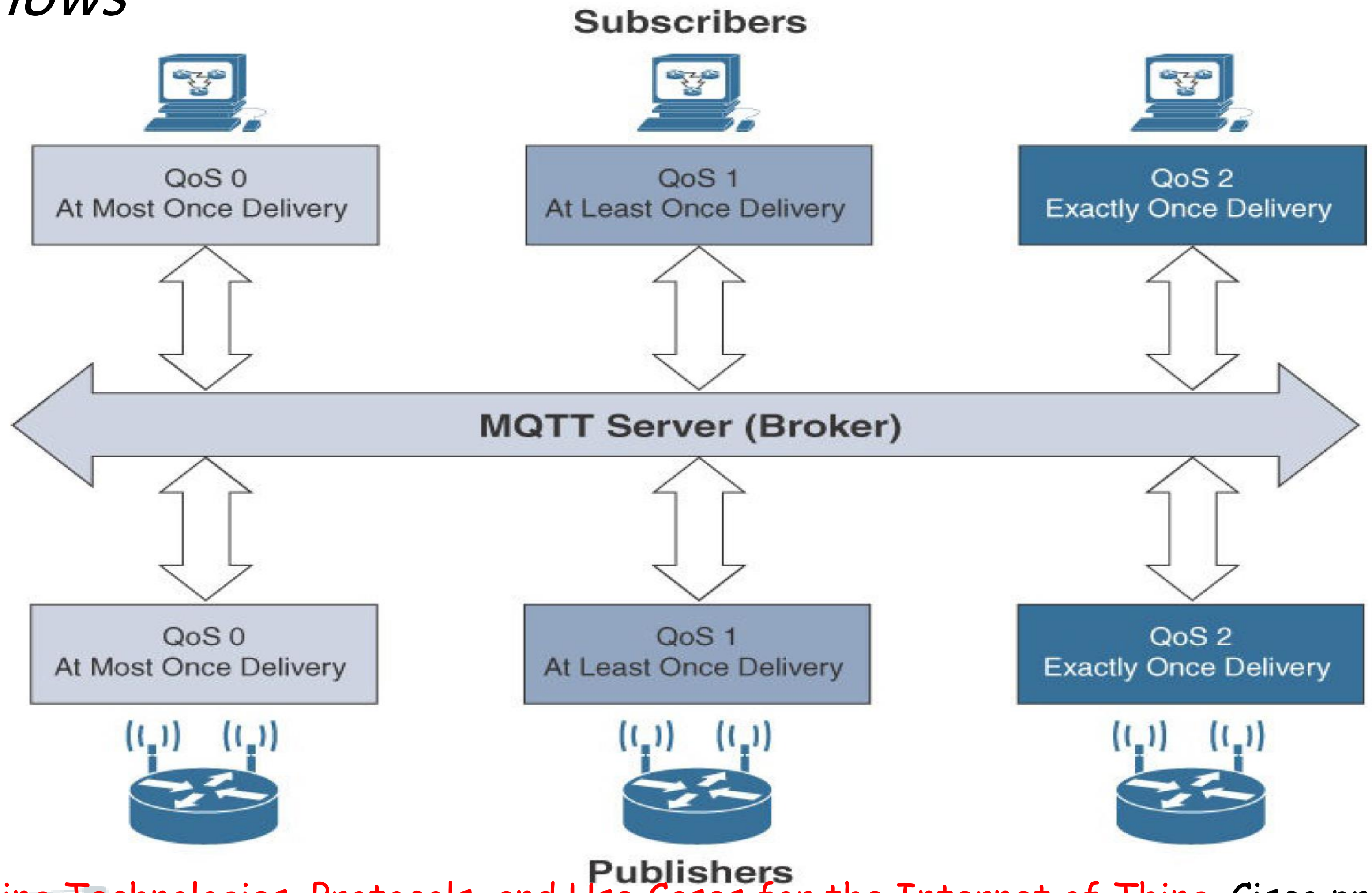
- **QoS 2:**
  - This is the highest QoS level, used when neither loss nor duplication of messages is acceptable.
  - There is an increased overhead associated with this QoS level because each packet contains an optional variable header with a packet identifier.
  - Confirming the receipt of a PUBLISH message requires a two-step acknowledgement process.
    - The first step is done through the PUBLISH/PUBREC packet pair, and the second is achieved with the PUBREL/PUBCOMP packet pair.
    - This level provides a "guaranteed service" known as "exactly once" delivery, with no consideration for the number of retries as long as the message is delivered once.

IoT Application Layer Protocols-MQTT

- The QoS process is symmetric in regard to the roles of sender and receiver, but two separate transactions exist.
  - One transaction occurs between the publishing client and the MQTT server,
  - and the other transaction happens between the MQTT server and the subscribing client.
- The publishing client side sets the QoS level for communications to the MQTT server.
- On the other side, the client subscriber sets the QoS level through the subscription with the MQTT server.
  - In most cases, QoS remains the same between clients and broker end to end.
  - However, you should be aware that in some scenarios, QoS levels change and are not the same end to end.

IoT Application Layer Protocols-MQTT

- *MQTT QoS Flows*



* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols-MQTT

- MQTT sessions between each client and server consist of four phases:
  - session establishment,
  - authentication,
  - data exchange, and
  - session termination.

- Each client connecting to a server has a unique client ID, which allows the identification of the MQTT session between both parties.
  - When the server is delivering an application message to more than one client, each client is treated independently.

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017

IoT Application Layer Protocols- MQTT Security

- Securing MQTT connections through TLS is considered optional because it calls for more resources on constrained nodes.

- When TLS is not used, the client sends a clear-text username and password during the connection initiation.

- MQTT server implementations may also accept anonymous client connections (with the username/password being “blank”).
  - When TLS is implemented, a client must validate the server certificate for proper authentication.
  - Client authentication can also be performed through certificate exchanges with the server, depending on the server configuration.

IoT Application Layer Protocols

- MQTT vs HTTP
  - MQTT is data centric whereas HTTP is document-centric.
  - HTTP is request-response protocol for client-server computing and not always optimized for mobile devices. Whereas MQTT is lightweight protocol (MQTT transfers data as a byte array) and publish/subscribe model, which makes it perfect for resource-constrained devices and help to save battery.
  - MQTT has pretty short specification. There are only CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE and DISCONNECT types that are significant for developers. Whereas HTTP specifications are much longer.
  - MQTT has a very short message header and the smallest packet message size of 2 bytes

* https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105

IoT Application Layer Protocols

- MQTT vs HTTP
  - According to measurements in 3G networks, throughput of MQTT is 93 times faster than HTTP's.
  - Besides, in comparison to HTTP, MQTT Protocol ensures high delivery guarantees. There are 3 levels of Quality of Services:
    - at most once: guarantees a best effort delivery.
    - at least once: guaranteed that a message will be delivered at least once. But the message can also be delivered more than once.
    - exactly once: guarantees that each message is received only once by the counterpart

* https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105

IoT Application Layer Protocols

- MQTT vs HTTP

Features	MQTT	HTTP
Full form	Message Queue Telemetry Transport	Hyper Text Transfer Protocol
Architecture	It has publish/subscribe architecture. Here devices can publish any topics and can also subscribe for any topics for any updates.	It has request/response means Client/Server architecture.
Upper layer protocol	It runs over TCP.	It runs over TCP and UDP.
message size	small, .	Large,
Message format	binary with 2Byte header	ASCII format.
Data distribution	1 to 0/1/N	one to one only , more POST
Data security	Yes, It uses SSL/TLS for security	NO, hence HTTPS is used to provide data security
Complexity	Simple	Client more complex (ASCII parser)
Encryption	It encrypts payload i.e. it is payload agnostic	data are not encrypted before transmission
When to use	if your project is to let the fridge to communicate with the thermometer to adapt the engine pump, you can use the MQTT easily	if you need to collect big data from around the world, then you can think to use HTTP

* https://iotdunia.com/mqtt-and-http/

IoT Application Layer Protocols

- CoAP vs MQTT

Factor	CoAP	MQTT
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/Subscribe
Effectiveness in LLNs	Excellent	low
Security	DTLS	SSl/TLS
Communication model	one-to-one	Many-to-many
Strengths	Lightweight, fast, low overhead, support for multicasting messages	Robust communication, simple management, scalability
Weakness	Not as reliable as MQTT	Higher overhead, no multicasting support

* IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Thing, Cisco press, 2017