Application Protocols for IoT – Chapter 6

MUGOYA DIHFAHSIH
2021/HD05/2353U

Chapter Summary

Introduction

Transport Layer

IoT Application Transport Methods

- Application Layer Not Present
- SCADA;
  - Adapting SCADA for IP
  - Tunneling Legacy SCADA over IP Networks
  - SCADA Protocol Translation
  - SCADA Transport over LLNs with MAP-T
- Generic Web Based Protocols
- IoT Application Layer Protocols
- CoAP
- MQTT
- Comparing CoAP with MQTT

Introduction

- IoT application protocols selected should be contingent on the use cases and vertical industries they apply to.
- IoT application protocols are dependent on the Characteristics of lower layers themselves.
- IoT applications protocolls focus on:
  - Transport Layer(TCP and UDP)
  - IoT application Transport Methods
- There are many solutions for IoT application Transport Methods because IoT is growing and changing.

The Transport Layer

It is supported by TCP/IP architecture

- The two Main protocols specified;

1.Transmission Control Protocol (TCP):

  - Connection-oriented protocol
  - Requires an established session between source and destination before exchanging data

2. User Datagram Protocol (UDP):

  - Data sent to destination with no guarantee of delivery
  - Acknowledgment is only sent back if request for acknowledgment is sent.

TCP is the main protocol used at the transport layer due to ;

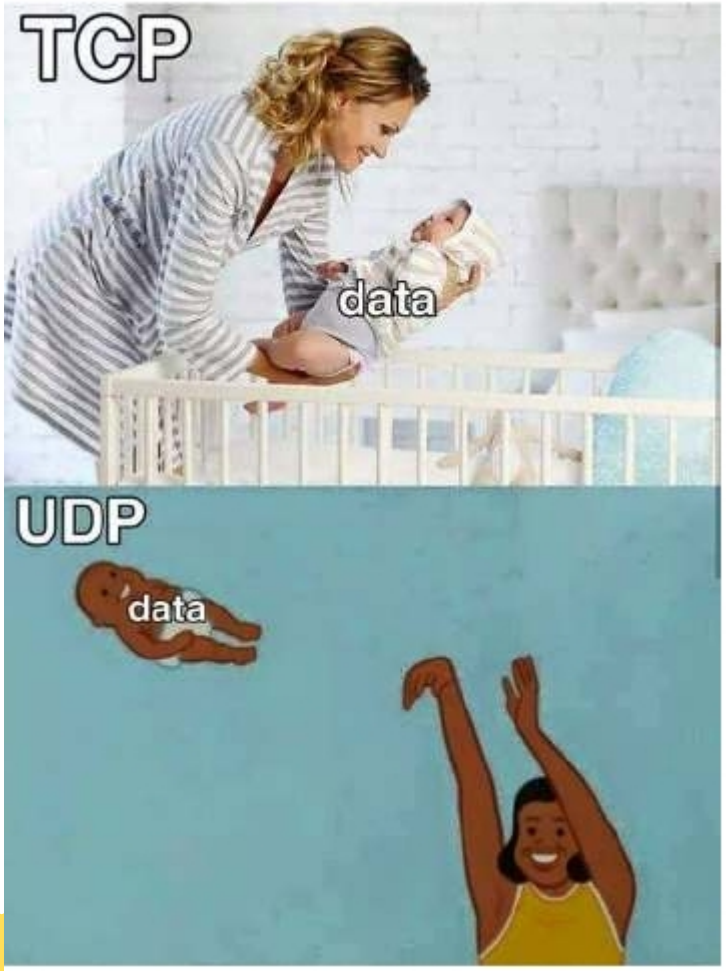
- Ability to transport large data in small sets of packets
- Ability to ensure correct reassembly of data
- Ability to control the flow and window adjustment
- Ability to retransmit lost packets

The cost of overhead per packet and per session increases with TCP, increasing latency and packet per second performance

- **UDP:** Is the main protocol used for network services
  - E.g Domain Name System, Network Time Protocol, Simple Network Management Protocol and Dynamic Control Protocol.
  - Also used for real time data traffic I.e voice and video over IP
  - Performance and latency is more important than retransmissions
  - When the reception of packets must be guaranteed error free, the application layer protocol takes care of the function
- To choose the transport Layer, Consider both the lower and upper layers of the stack.

- Most Industrial applications protocols are TCP
  - Due to older times where error protection and reliability of data link layers were needed.

  TCP may not strain generic compute platforms and high-data-rate networks but can be challenging and is often overkill on the constrained IoT devices and networks.



Reasons why TCP is preferred to UDP

- 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.
- It ensures reassembly in a correct sequence, flow control and window adjustment, and retransmission of lost packets.
- Why IoT Constrained nodes use UDP over TCP
  - TCP adds 20bytes even on a few bytes transaction while UDP adds only 8 bytes
  - TCP also requires the establishment and potential maintenance of an open logical channel.
  - IoT nodes may also be limited by the intrinsic characteristics of the data link layers. Forexample, low-power and lossy networks (LLNs),may not cope well with supporting large numbers of TCP sessions.

IoT Application Transport Methods

Various means for transporting application protocols across a network

- Common Issues are legacy utility and industrial IoT protocols
- To make the decision of which method to use, categorize the common IoT Application Protocols and focus on the transport methods

  The following categories of IoT application protocols and their transport methods

  - Application layer protocol not present:
  - Supervisory control and data acquisition (SCADA)
  - Generic web-based protocols:
  - IoT application layer protocols:

Method 1: Application Layer Protocol Not Present

In this method, data payload is directly transported on top of the lower layers. An IoT data broker is needed to scale this method of transporting application data

IoT 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.

Data broker is piece of middleware that standardizes sensor output into a common format that can then be retrieved by authorized applications.

An IoT data broker is needed to scale this method of transporting application data.

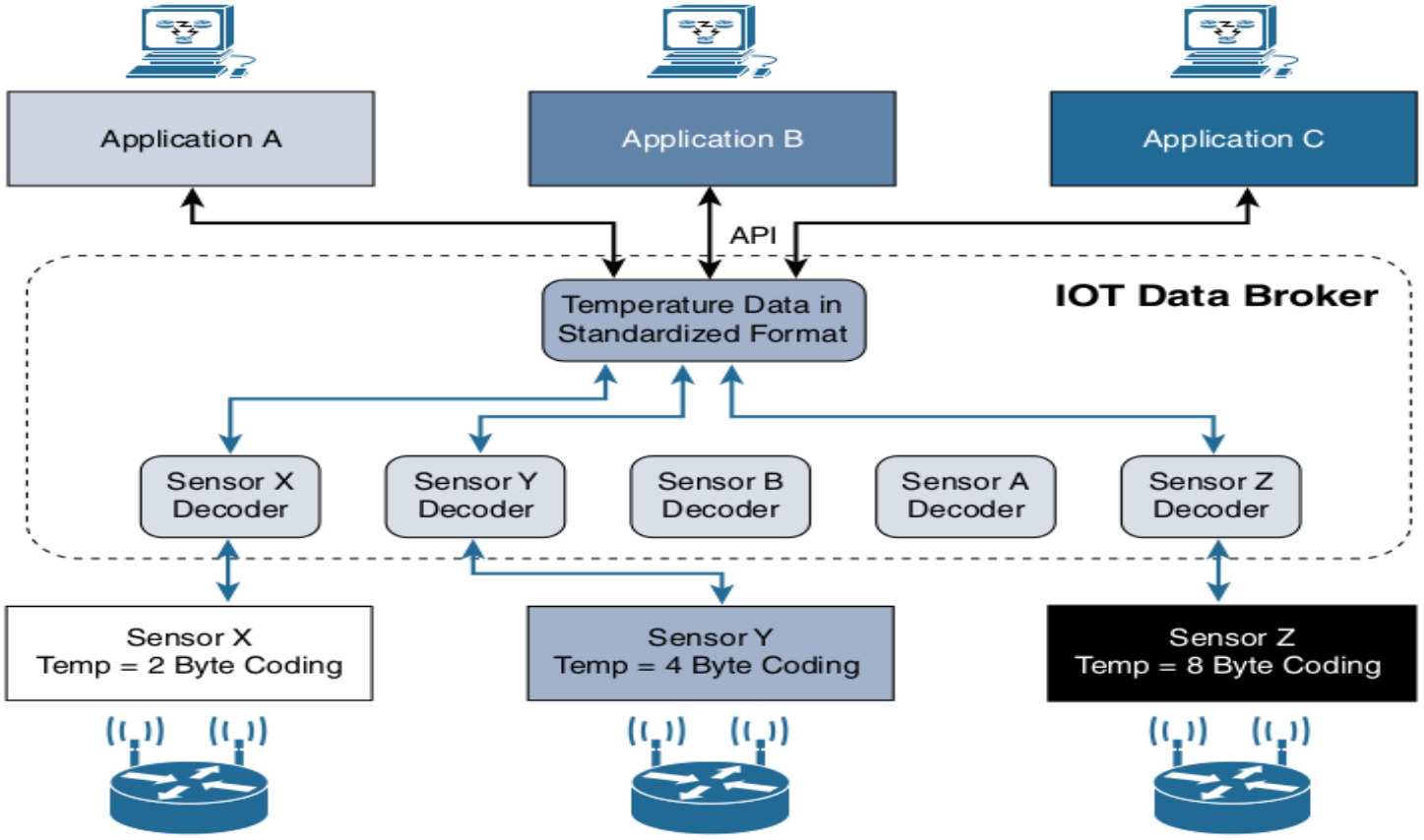
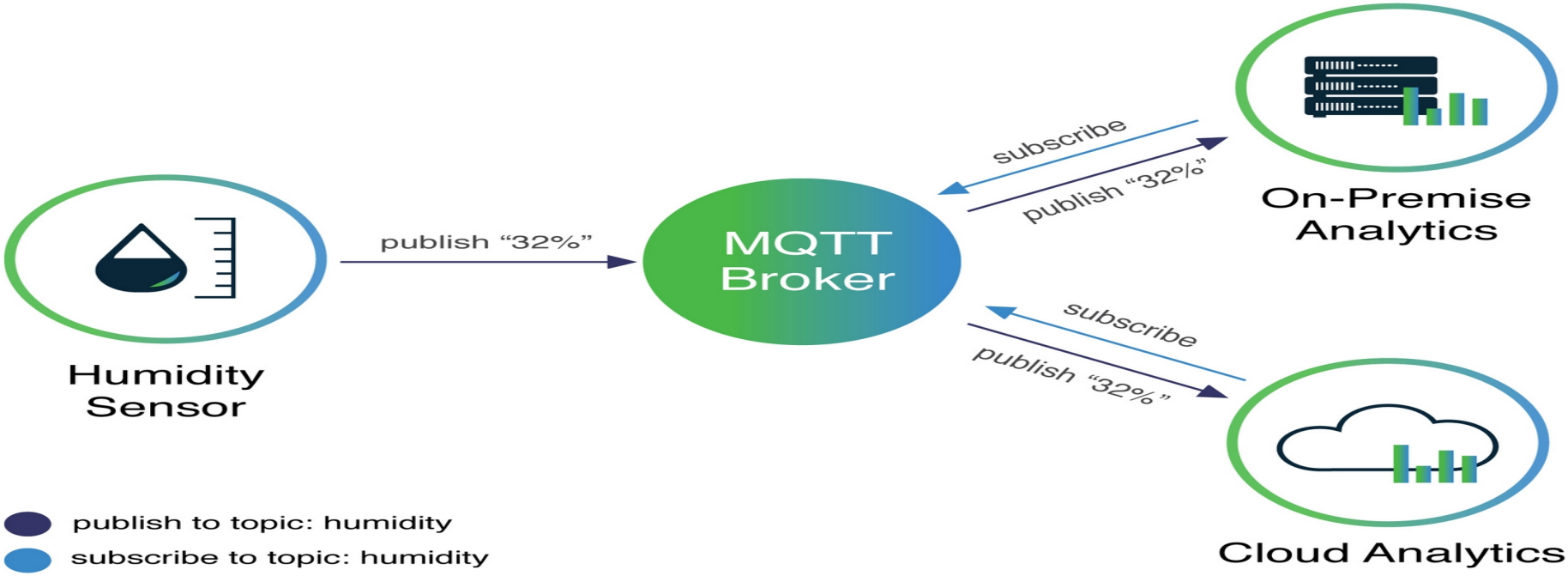


Figure 6-1 *IoT Data Broker*

- Basically the IoT broker is a middleman that can interprete various sensor data types into one format for easy picking by the applications
- IoT data brokers are also utilized from a commercial perspective to distribute and sell IoT data to third parties



SCADA

- The evolution of older protocols that connected sensors and actuators have adapted themselves to utilize IP and this evolution is known as **Supervisory Control And Data Acquisition** (SCADA)
- SCADA is an automation control system that was initially implemented without IP over serial links, before being adapted to Ethernet and IPv4.
  - Is an automation control system
  - Initially implemented without IP over Serial Links
  - Now adapted into Ethernet and IPv4
  - Collects sensor data and telemetry from remote devices while also providing ability to control them at a high level
  - Mainly concentrated in the utilities and manufacturing industrial Verticals
  - SCADA systems allow global, real-time, data-driven decisions to be made about how to improve business process

Adapting SCADA for IP

- To have the ability to leverage existing equipment and standards while integrating seamlessly the SCADA subnetworks to the corporate WAN infrastructures.
- Assigning TCP/UDP port numbers to the protocols such as
    - DNP3 (adopted by IEEE 1815-2012) specifies the use of TCP or UDP on port 20000
    - The Modbus messaging service utilizes TCP port 502.
    - IEC 60870-5-104 on port 2404.
    - DLMS User Association specified a communication profile based on TCP/IP

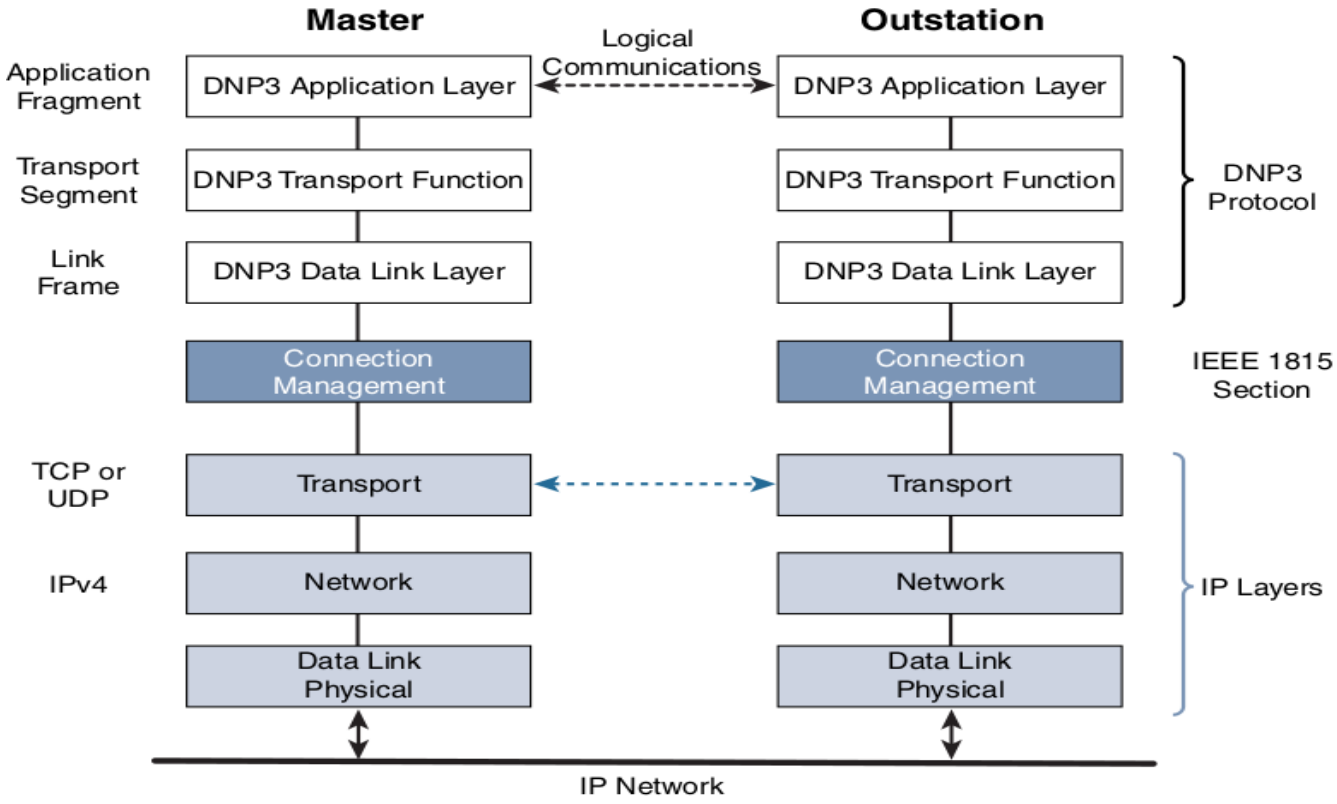
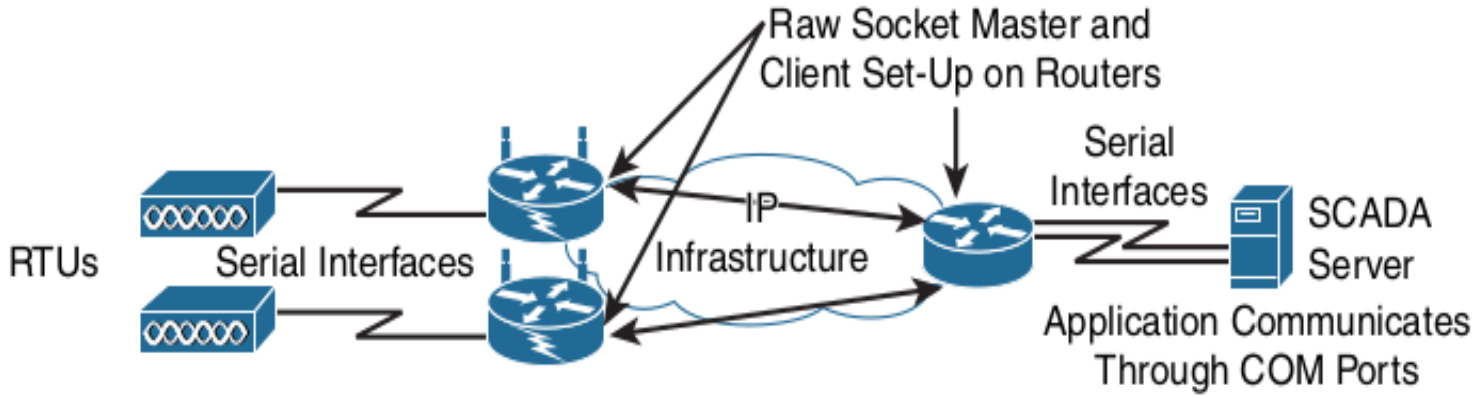


Figure 6-2 *Protocol Stack for Transporting Serial DNP3 SCADA over IP*

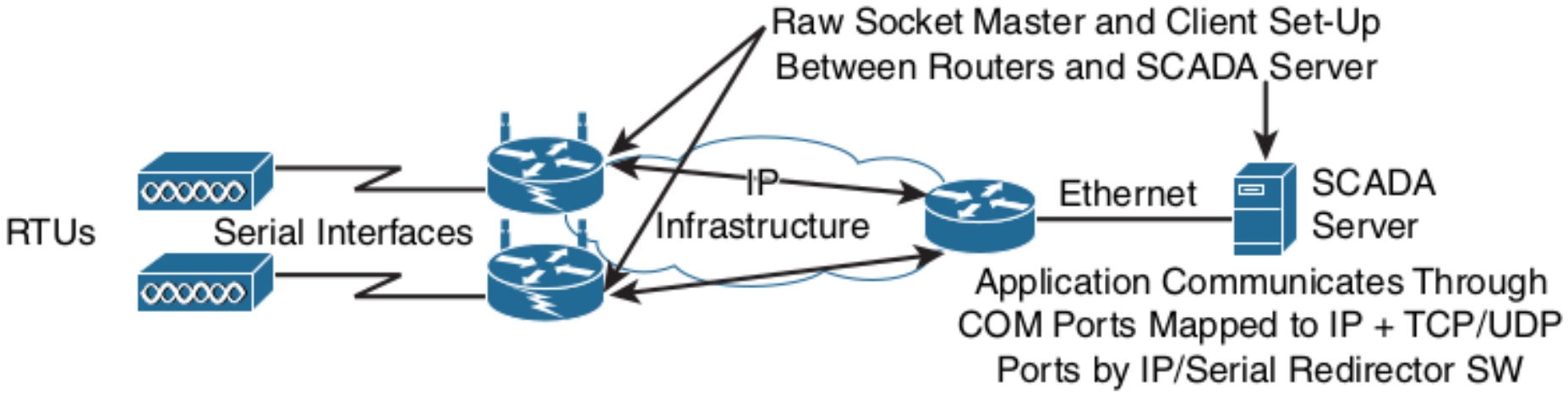
Tunneling Legacy SCADA over IP Networks

- Transport of the original serial protocol over IP can be achieved either by:
  - Tunneling using raw sockets over TCP or UDP
  - Installing an intermediate device that performs protocol translation between the serial protocol version and its IP implementation.
- Raw socket connection - the serial data is being packaged directly into a TCP or UDP transport.
  - Socket in this instance is a standard API composed of an IP address and a TCP or UDP port that is used to access network devices over an IP network.
  - Old application servers may need a device/software to convert pure serial data to serial over IPModern application servers are already capable of this



Scenario A: Raw Socket between Routers – no change on SCADA server

In this scenario - Raw serial data is sent with help from the router

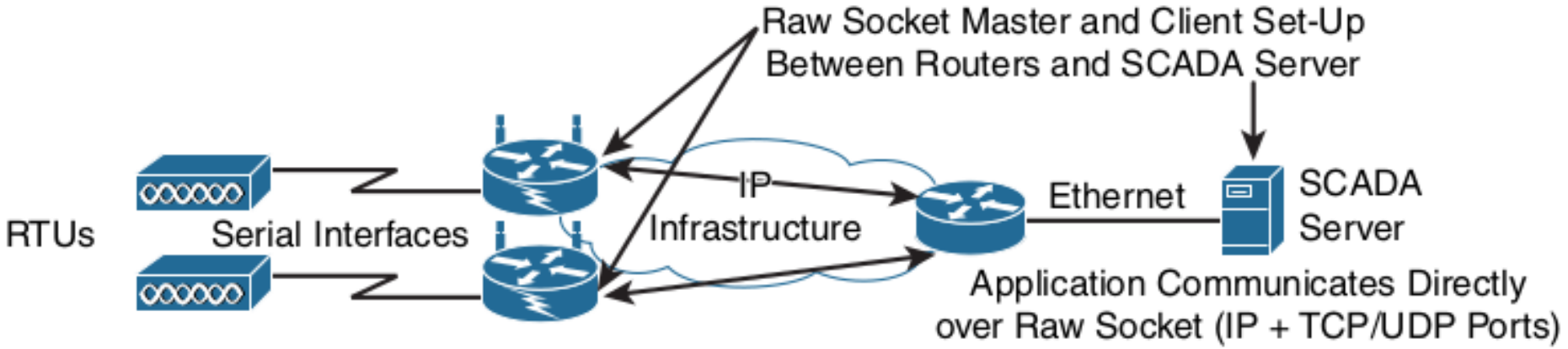


Scenario B: Raw Socket between Router and SCADA Server – no SCADA application change on server but IP/Serial Redirector software and Ethernet interface to be added

In this Scenario;

Software is installed on the server that maps the serial COM ports to IP ports

(IP/serial redirector)



Scenario C: Raw Socket between Router and SCADA Server – SCADA application knows how to directly communicate over a Raw Socket and Ethernet interface

In this scenario;

- No help needed to send serial data

SCADA Protocol Translation

With protocol translation, the legacy serial protocol is translated to a corresponding IP version.

The IoT gateway in this figure below performs a protocol translation function that enables communication between the RTUs and servers, despite the fact that a serial connection is present on one side and an IP connection is used on the other.This can be accomplished by offering computing resources on IoT gateways or routers.

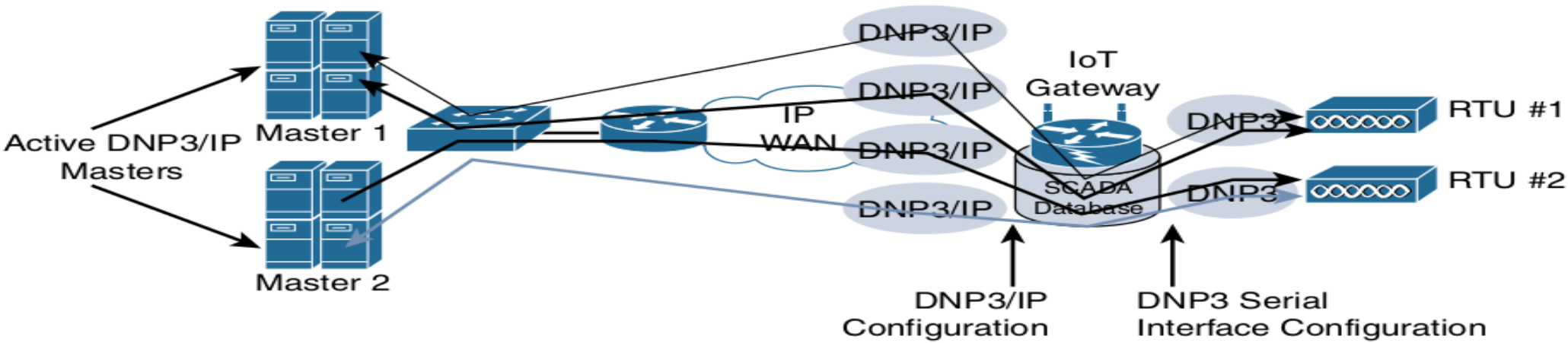


Figure 6-4 *DNP3 Protocol Translation*

SCADA Transport over LLNs with MAP-T

Most of the industrial devices supporting IP today support IPv4 only. When deployed over LLN subnetworks that are IPv6 only, a transition mechanism, such as MAP-T (Mapping of Address and Port using Translation) needs to be implemented.

This allows the deployment to take advantage of native IPv6 transport transparently to the application and devices.

MAP-T makes the appropriate mappings between IPv4 and the IPv6 protocols. This allows legacy IPv4 traffic to be forwarded across IPv6 networks.

In other words, older devices and protocols can continue running IPv4 even though the network is requiring IPv6.

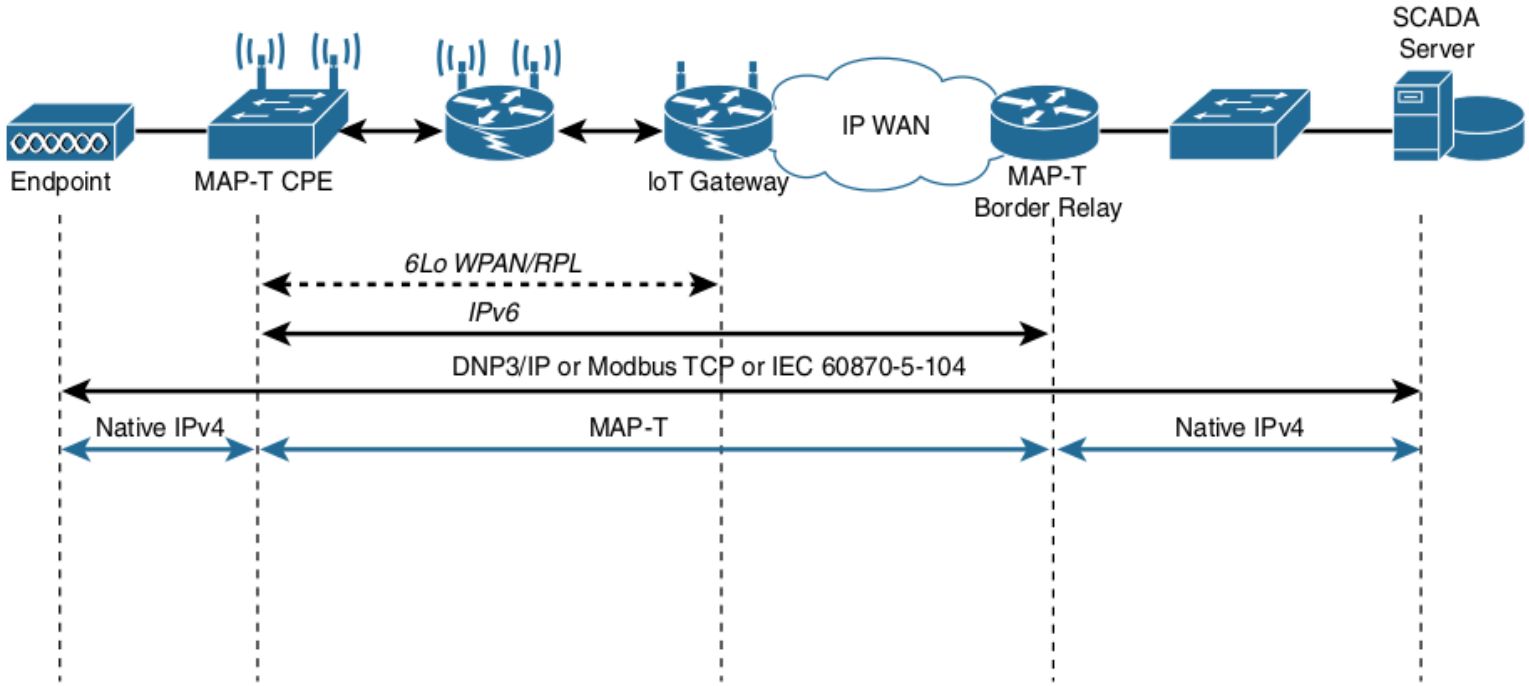


Figure 6-5 *DNP3 Protocol over 6LoWPAN Networks with MAP-T*

Method 3: Generic Web-Based Protocols

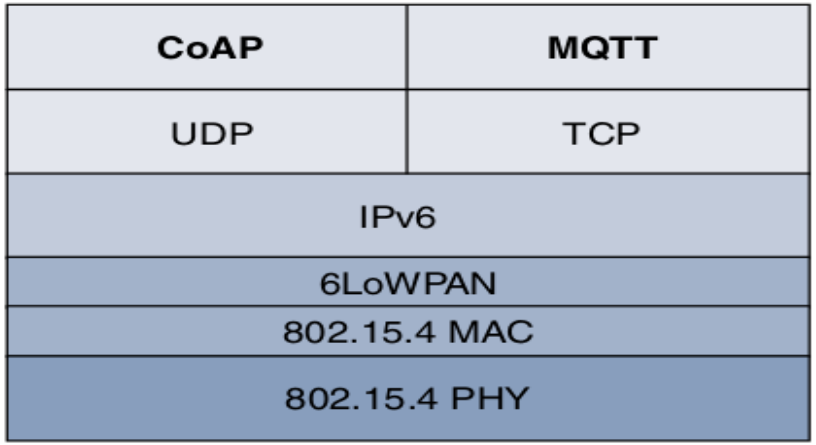
Allows Programmers with basic web programming skills to work on IoT applications

- For Non-constrained networks, normal data formats e.g XML or JSON can be transported over HTTP/HTTPS or WebSocket
- Web service implementation in IoT networks can either be from the client side or the server side.
  - Devices that only push data would be client side implementation (HTTP only sends info and doesn´t receive)
  - Devices that would require many connections and has limited resources (e.g video surveillance) would be server side.

Method 4: IoT application layer protocols

Constrained network/nodes cannot handle verbose web-based and data model protocols

Lightweight protocols are needed



ocol)

y Transport)

Figure 6-6 *Example of a High-Level IoT Protocol Stack for CoAP and MQTT*

CoAP

Known for Simple and flexible ways to manipulate sensors and actuators for data or device management

Constrained Application Protocol is a generic framework for resource-oriented applications targeting constrained nodes and networks.

Primarily designed to manage exchange of messages over UDP between endpoints

The IETF CoRE working group has published multiple standards-track specifications for CoAP, including the following:

- RFC 6690: Constrained RESTful Environments (CoRE) Link Format.
- RFC 7252: The Constrained Application Protocol (CoAP)
- RFC 7641: Observing Resources in the Constrained Application Protocol (CoAP)
- RFC 7959: Block-Wise Transfers in the Constrained Application Protocol (CoAP)
- RFC 8075: Guidelines for Mapping Implementations: HTTP to the Constrained
- Application Protocol (CoAP)

CoAP message format is relatively simple and flexible;

- Low overhead,
- Easy to parse for constrained devices,
- 4bytes header field,
- Can be run over IPv4 and IPv6

4 types of messages in CoAP:

- Confirmable
- Non-Confirmable
- Acknowledgment
- Reset

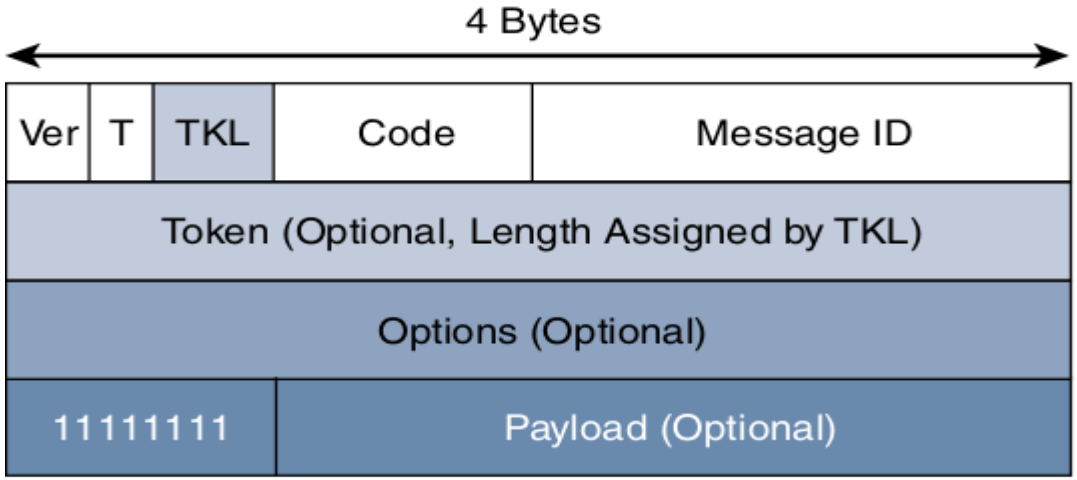


Figure 6-7 *CoAP Message Format*

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.

CoAP communications across an IoT infrastructure can take various paths.

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.

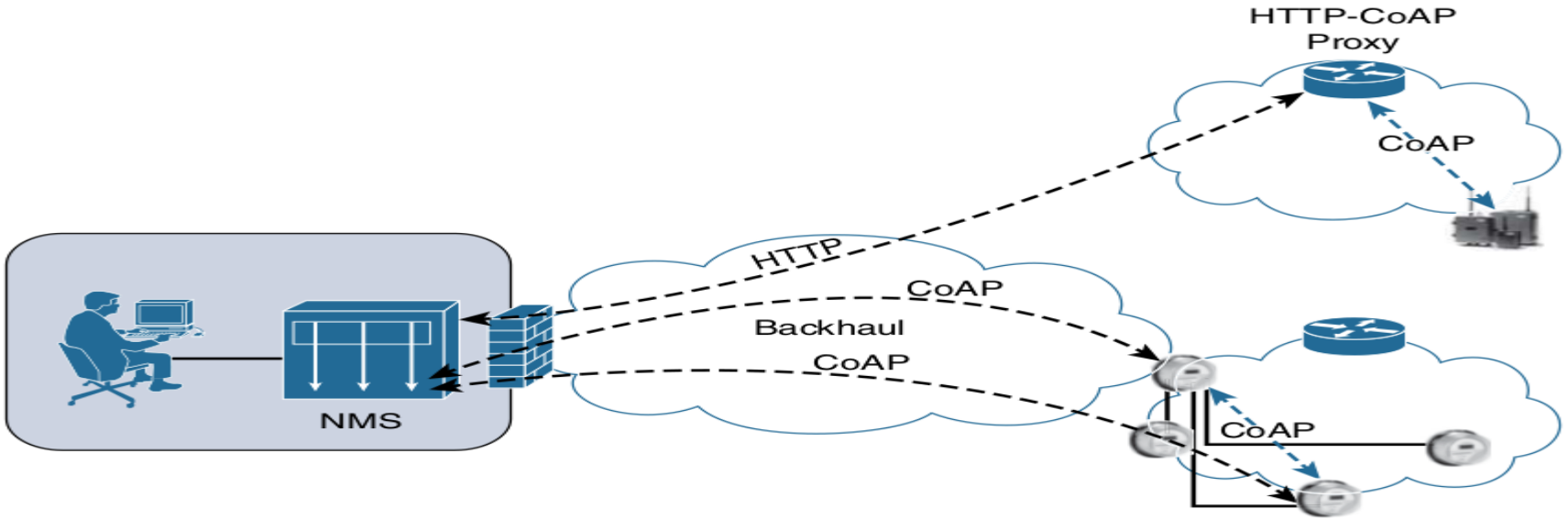


Figure 6-8 *CoAP Communications in IoT Infrastructures*

- Just like HTTP, CoAP is based on the REST architecture, but with a "thing" acting as

  both the client and the server.

- 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.

Example 6-2 *CoAP URI format*

```
coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]
coaps-URI = "coaps:" "//" host [":" port] path-abempty  ["?" query]
```

CoAP defines four types of messages: confirmable, non-confirmable, acknowledgement, and reset in a message response or a request

While running over UDP, CoAP offers a reliable transmission of messages when a CoAP

header is marked as "confirmable."

CoAP supports basic congestion control with a default time-out, simple stop and wait retransmission with exponential back-off mechanism, and detection of duplicate messages through a message ID.

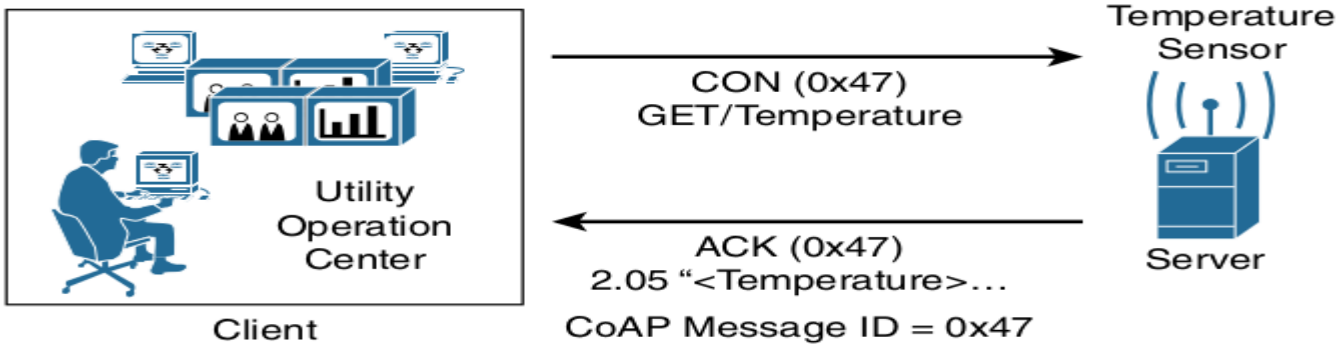


Figure 6-9 *CoAP Reliable Transmission Example*

- Much as with accessing web server resources, CoAP specifications provide a description of the relationships between resources in RFC 6690, "Constrained RESTful Environments (CoRE) Link Format."

  This standard defines the CoRE Link format carried as a payload with an assigned Internet media type.

  To improve the response time and reduce bandwidth consumption, CoAP supports caching capabilities based on the response code.

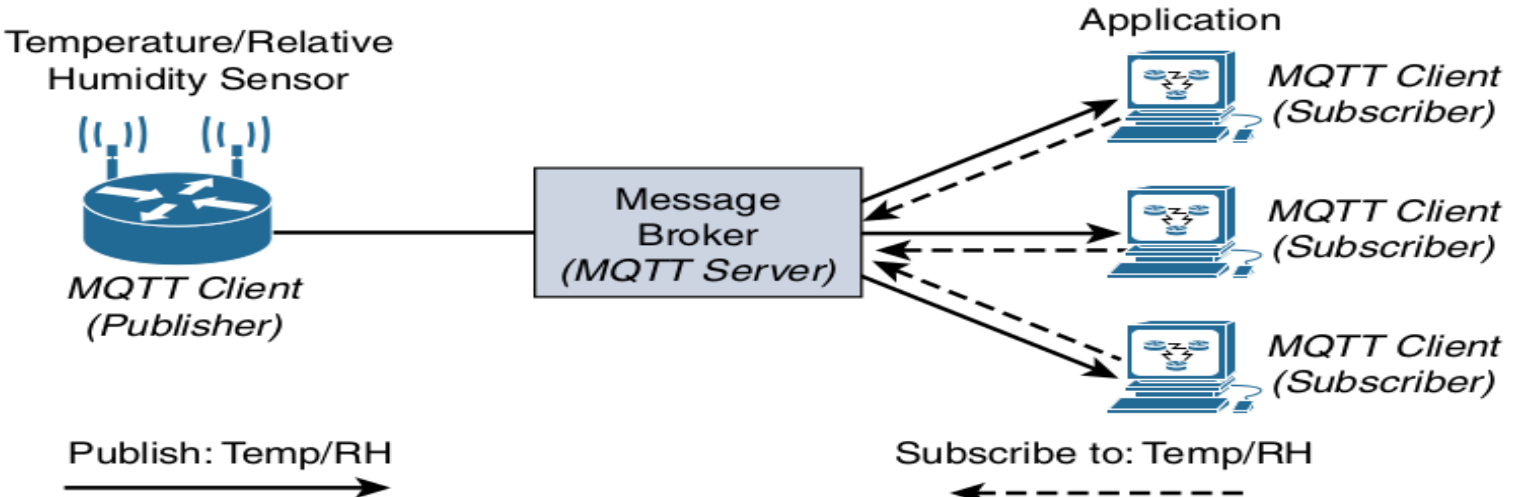
- CoAP is fully optimized for IoT constrained nodes and networks, while leveraging traditional web programming techniques to make it easily understandable by the development community

Message Queuing Telemetry Transport (MQTT)

Is an extremely simple protocol with only a few options designed with considerations for constrained nodes, unreliable WAN backhaul communications, and bandwidth constraints with variable latencies.

- How does it work?

  Sorta acts like YouTube. A channel pushes out content which is only seen by the subscribers of the channel



- Client can act as a publisher to send data to the server
- Server acts as a message broker
- Server accepts the network connection along with application messages from the publishers.
  - Handles subscription/unsubscription process
  - Pushes application data to MQTT clients acting as subscribers
  - Subscribers to the data receives the information, and can subscribe to either all data or specific data
  - Publishers and subscribers do not know each other and don't have to be online at the same time

- An MQTT client can act as a publisher to send data (or resource information) to an MQTT server acting as an MQTT message broker.
- The MQTT server (or message broker) accepts the network connection along with application messages, such as Temp/RH data, from the publishers.
- With MQTT, clients can subscribe to all data (using a wildcard character) or specific data from the information tree of a publisher.
- The presence of a message broker in MQTT decouples the data transmission between clients acting as publishers and subscribers.
- 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.
- MQTT control packets run over a TCP transport using port 1883. TCP ensures an ordered, lossless stream of bytes between the MQTT client and the MQTT server.

- MQTT is a lightweight protocol because 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.

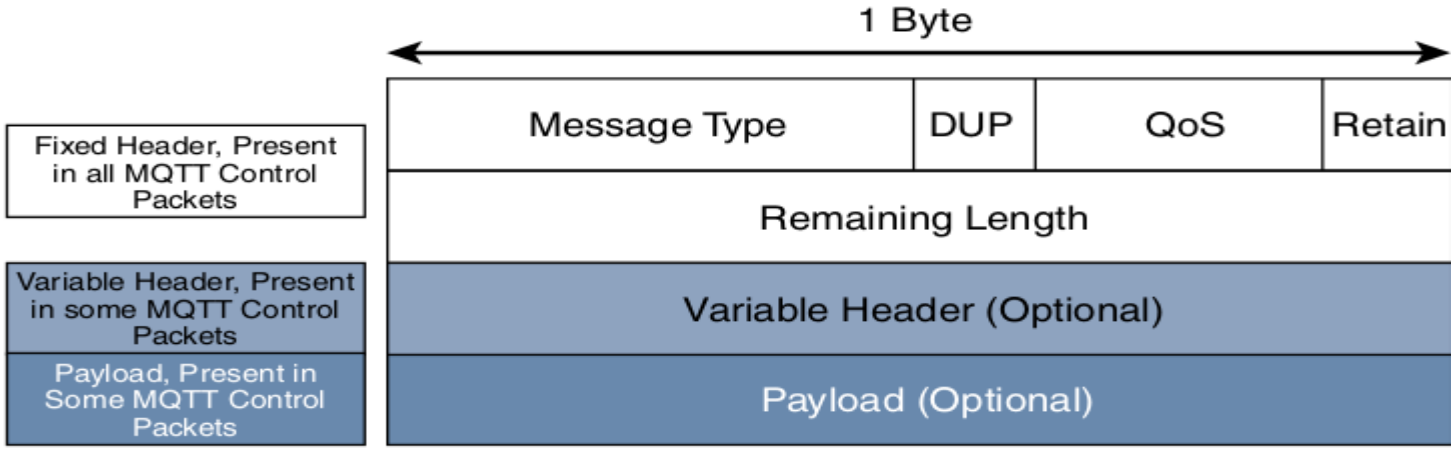


Figure 6-11 *MQTT Message Format*

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

DUP (Duplication Flag);

- Checks for Duplication messages

- QoS3
  - Quality of Service levels:
    - 0 - best effort and highest priority data sent, but message is sent only once
    - 1 - ensures message is sent
    - 2 - highest QoS, makes sure exactly 1 message is sent and received
  - Retain flag
    - Tells server to hold on to message data
  - Remaining Length
    - Number of bytes in the packet

Table 6-3 *Comparison Between CoAP and MQTT*

Factor	CoAP	MQTT
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/subscribe
Effectiveness in LLNs	Excellent	Low/fair (Implementations pairing UDP with MQTT are better for LLNs.)
Security	DTLS	SSL/TLS
Communication model	One-to-one	many-to-many
Strengths	Lightweight and fast, with low overhead, and suitable for constrained networks; uses a RESTful model that is easy to code to; easy to parse and process for constrained devices; support for multicasting; asynchronous and synchronous messages	TCP and multiple QoS options provide robust communications; simple management and scalability using a broker architecture
Weaknesses	Not as reliable as TCP-based MQTT, so the application must ensure reliability.	Higher overhead for constrained devices and networks; TCP connections can drain low-power devices; no multicasting support

MQTT is different from the “one-to-one” CoAP model in its “many-to-many” subscription framework, which can make it a better option for some deployments.

MQTT is TCP-based, and it ensures an ordered and lossless connection. It has a low overhead when optionally paired with UDP and flexible message format, supports TLS for security, and provides for three levels of QoS.