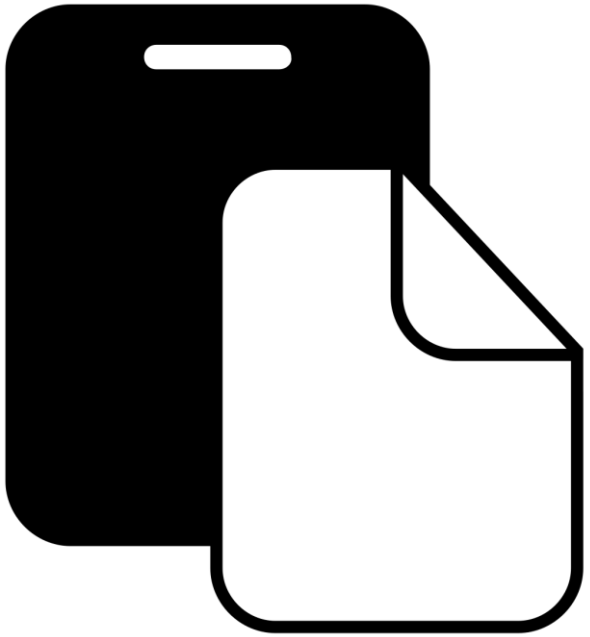
Indirect Messaging

Frank Walsh

Agenda

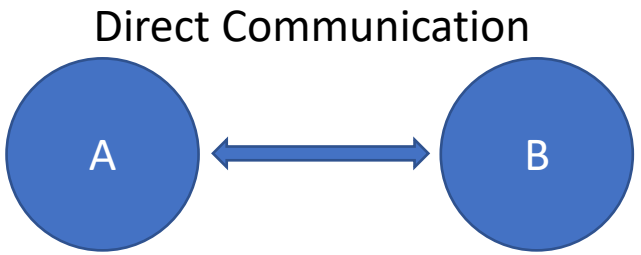


- Indirect Messaging
  - MQTT
  - Firebase Realtime DB
- Example: RealTime DB for Environment Sensor

Indirect Messaging

Publish Subscribe

Using the “Middleman”



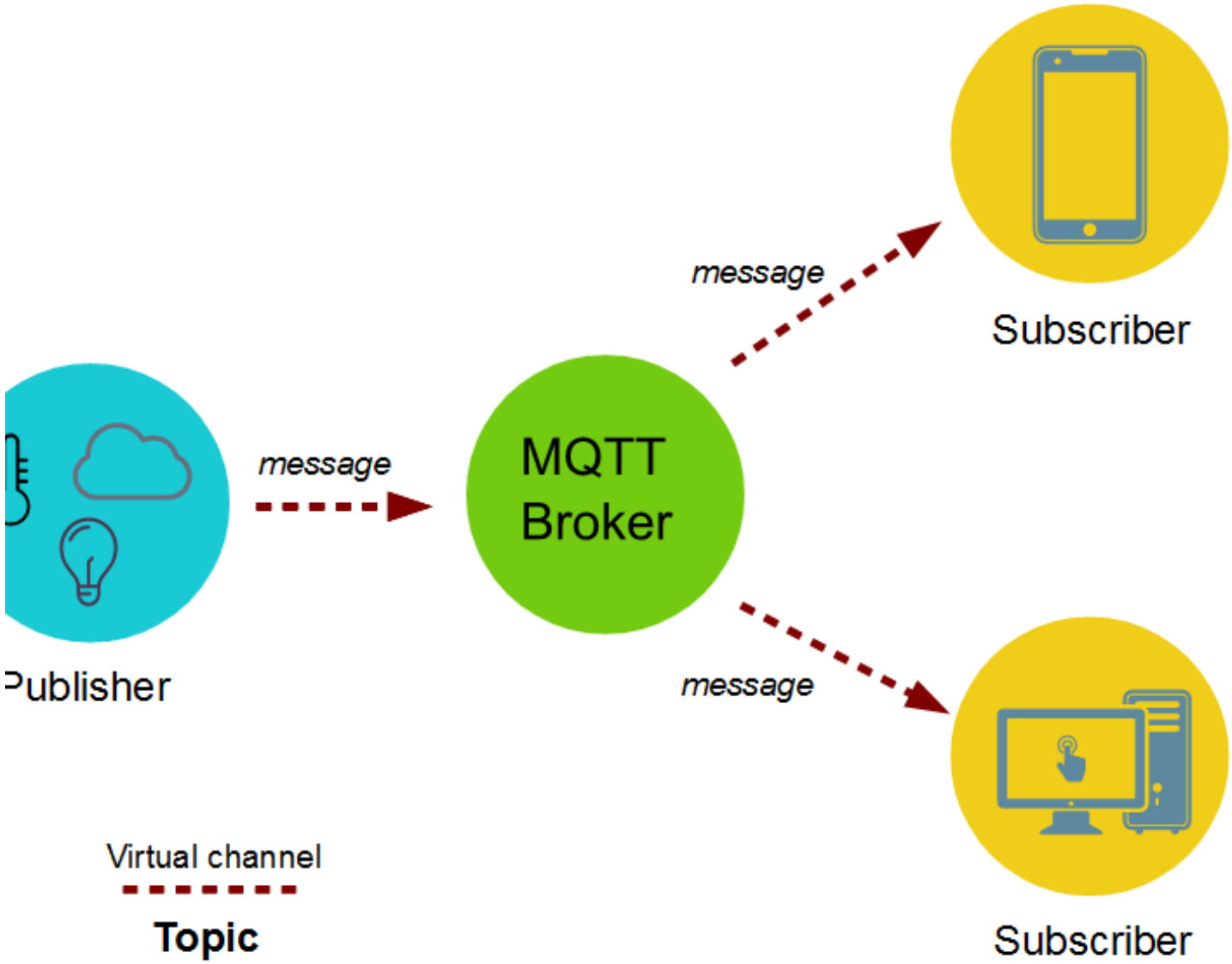
- Communication between processes using an intermediary
  - Sender → “The middle-man” → Receiver
  - No direct coupling
- Up to now, only considered Direct Coupling
  - Introduces a degree of rigidity
- Consider…
  - What happens if a device fails during communication in Direct Coupling?
  - What if you'd like to add
- Two important properties of intermediary in communication
  - ***Space uncoupling*** *(devices don't need to "know" about each other)*
  - ***Time uncoupling*** *(devices don't need to be "available" at the same time)*

MQTT

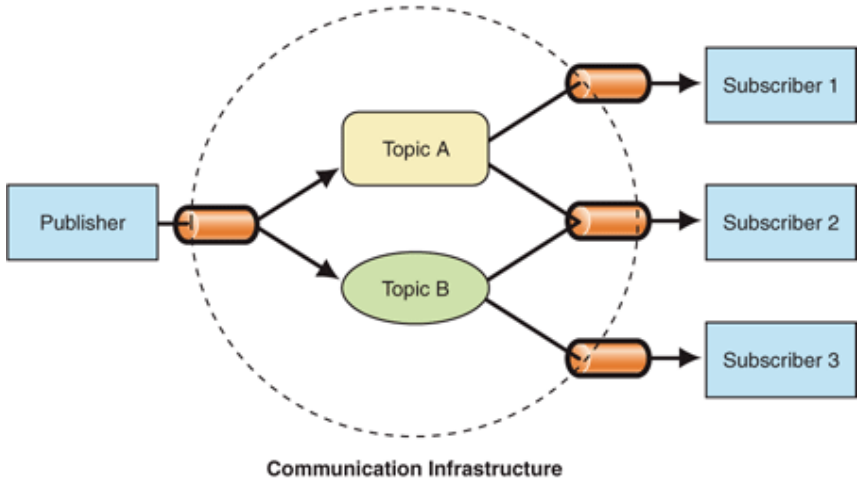
- MQ Telemetry Transport (MQTT)
- Telemetry
  - Remote measurements
- Created by IBM
  - from message queueing (MQ) architecture used by IBM for service oriented networks.
  - Telemetry data goes from devices to a server or broker.
  - Uses a **publish/subscribe** mechanism.
- Lightweight both in bandwidth and code footprint

MQTT – publish subscribe

- **Topics/Subscriptions:** Messages are published to topics.
  - Clients can subscribe to a topic or a set of related topics
- **Publish/Subscribe**: Clients can subscribe to topics or publish to topics.



Publish Subscribe Process



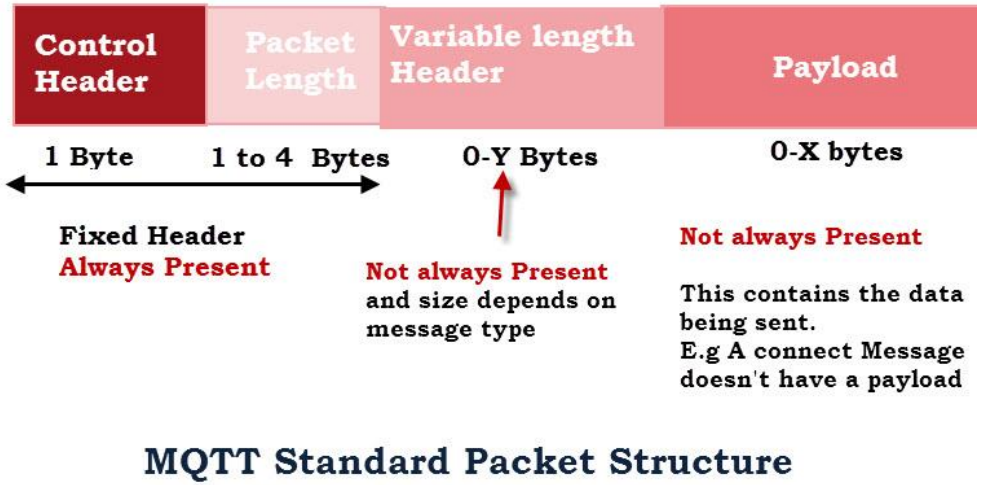
- A message is published once by a publisher.
- Many things can receive the message.
- The messaging service, or “broker”, provides decoupling between the producer and consumer(s)
- A producer sends (publishes) a message (publication) on a topic (subject)
- A consumer subscribes (makes a subscription) for messages on a topic (subject)
- A message server / broker matches publications to subscriptions
  - If no matches the message is discarded
  - If one or more matches the message is delivered to each matching subscriber/consumer

Publish-Subscribe Characteristics

- A published messages may be **retained**
  - A publisher can mark a message as “retained”
  - The broker / server remembers the last known good message of a retained topic
  - The broker / server gives the last known good message to new subscribers
- A Subscription can be **durable or non-durable**
  - Durable: messages forwarded to subscriber immediately, if subscriber not connected, message is stored and forwarded when connected
  - Non-Durable: subscription only active when subscriber is connected to the server / broker

MQTT Characteristics

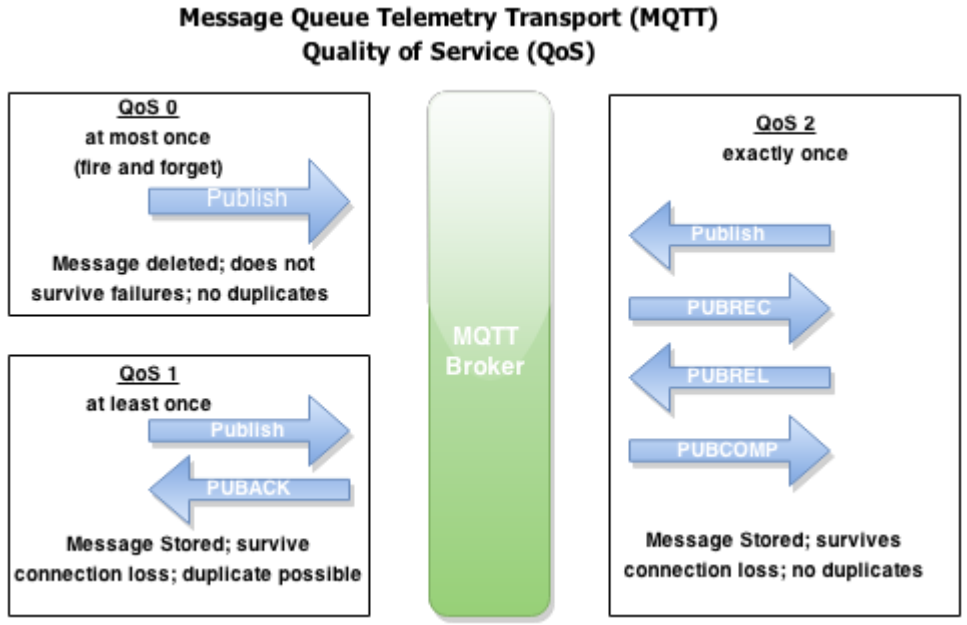
- MQTT protocol compresses to small number of bytes
  - Smallest packet size 2 bytes
  - Supports always-connected and sometimes connected
  - Provides Session awareness
  - “Last will and testament” enable applications to know when a client goes offline abnormally
  - Typically utilises TCP-based networks.



MQTT Standard Packet Structure

MQTT Characteristics

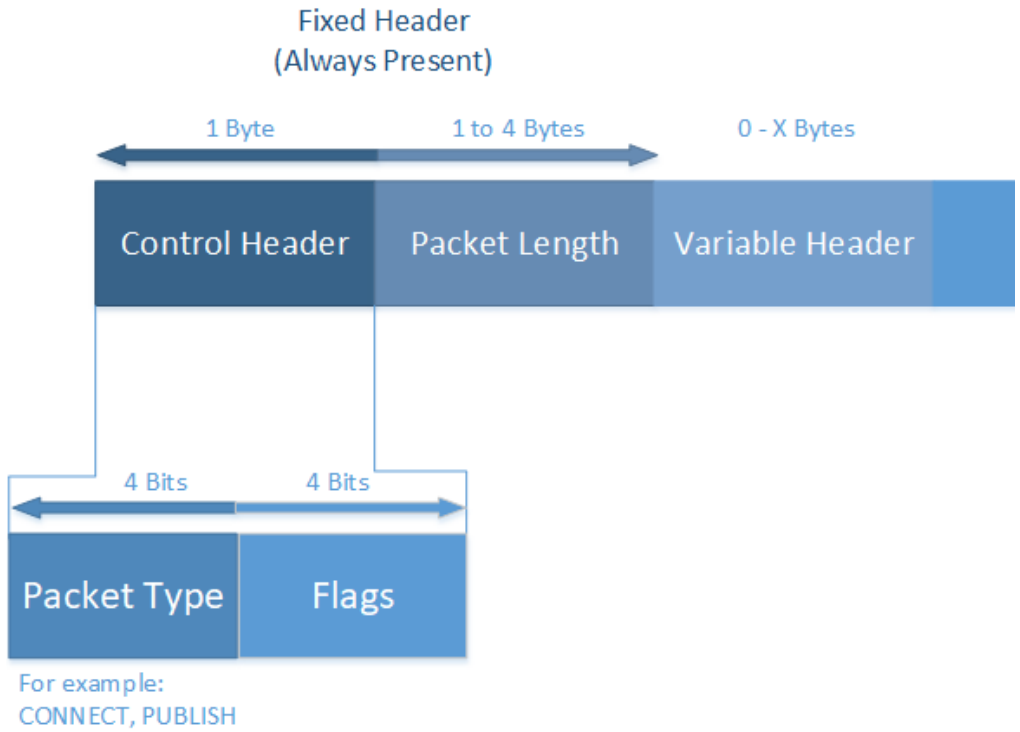
- Three quality of service levels:
  - 0 = At most once (Best effort, No Ack),
  - 1 = At least once (Acked, retransmitted if ack not received),
  - 2 = Exactly once [Request to send (Publish), Clear-to-send(Pubrec), message (Pubrel), ack (Pubcomp)]



https://dzone.com/articles/internet-things-mqtt-quality

MQTT Control Packets

- The MQTT protocol uses MQTT Control Packets using the Control Header.
- Control Packet consists of three parts:
  - *fixed header*(present in all packets),
  - *variable header*(optional)
  - *Payload*(optional)



https://docs.devicewise.com/Content/Products/IoT_Portal_API_Reference_Guide/MQTT_Interface/MQTT-Control-Packets.htm

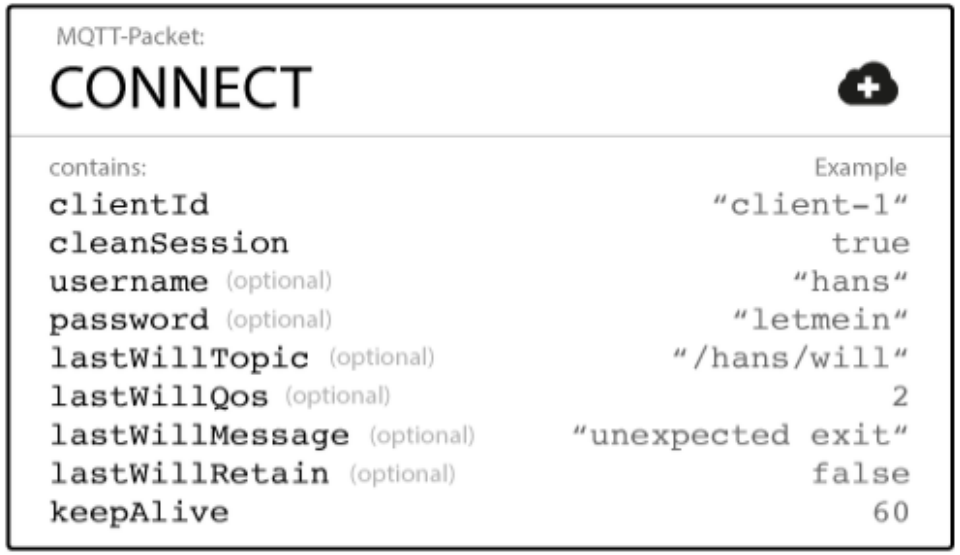
MQTT Control Packets

- MQTT Control Packet Types

Packet Type	Description	Value	Direction of flow
CONNECT	Client requests a connection to a Server	1	Client to Server
CONNACK	Acknowledge connection request	2	Server to Client
PUBLISH	Publish message	3	Client to Server or Server to Client
PUBACK	Publish acknowledgment	4	Client to Server or Server to Client
SUBSCRIBE	Subscribe to topics	8	Client to Server
SUBACK	Subscribe acknowledgment	9	Server to Client
UNSUBSCRIBE	Unsubscribe from topics	10	Client to Server
UNSUBACK	Unsubscribe acknowledgment	11	Server to Client
PINGREQ	Ping request	12	Client to Server
PINGRESP	Ping response	13	Server to Client
DISCONNECT	Disconnect notification	14	Client to Server

MQTT Last Will and Testament

- Notify other clients about an ungraceful disconnected client.
- Client can specify its last will message when it connects to a broker.
  - normal MQTT message with a topic, retained message flag, QoS, and payload
  - When an “ungraceful disconnect” occurs, the broker sends the last-will message to all subscribed clients of the last-will message topic.



https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/

MQTT is Open Source

- Lots of implementations:
    - Mosquitto
    - Micro broker
    - Really small message broker (RSMB): C
    - Cloud broker services

MQTT vs HTTP

- Push delivery of messages / data / events
  - MQTT – low latency push delivery of messages from client to server and **server to client.** Helps bring an event oriented architecture to the web
  - HTTP – push from client to server but poll from server to client
- Reliable delivery over fragile network
  - MQTT will deliver message to QOS even **across connection breaks**
  - Decoupling and publish subscribe – **one to many delivery**

http://stephendnicholas.com/archives/1217

MQTT vs HTTP

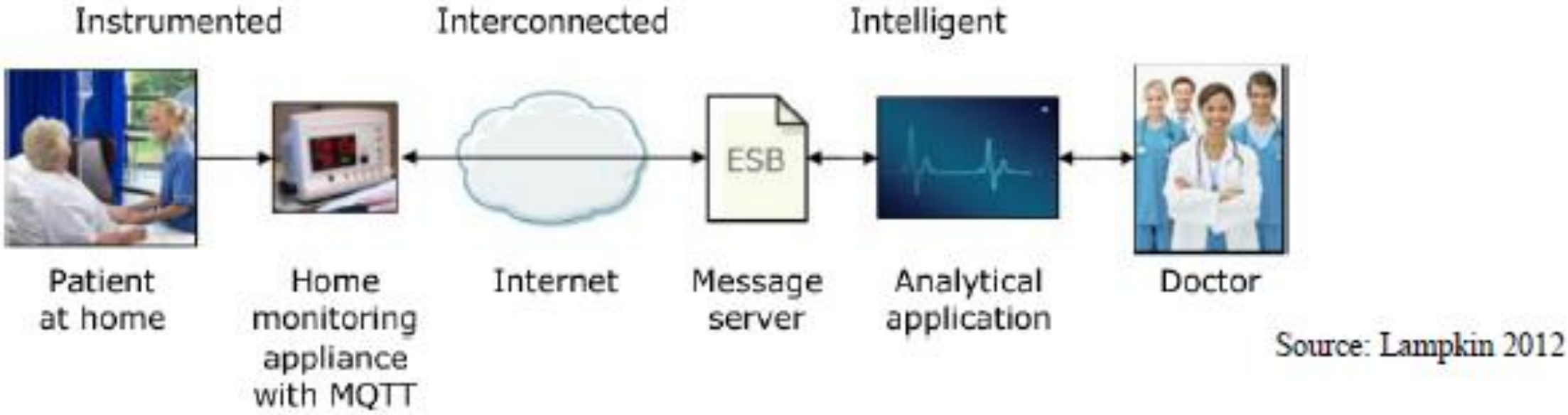
	MQTT	HTTP
Design orientation	Data centric	Document centric
Pattern	Publish/subscribe	Request/response
Complexity	Simple	More complex
Message size	Small, with a compact binary header just two bytes in size	Larger, partly because status detail is text-based
Service levels	Three quality of service settings	All messages get the same level of service
Extra libraries	Libraries for C (30 KB) and Java (100 KB)	Depends on the application (JSON, XML), but typically not small
Data distribution	Supports 1 to zero, 1 to 1, and 1 to *n*	1 to 1 only

Characteristics		3G		WiFi	
		HTTPS	***MQTT***	***HTTPS***	***MQTT***
Receive Messages	**Messages / Hour**	1,708	160,278	3,628	263,314
	Percent Battery / Hour	18.43%	16.13%	3.45%	4.23%
	Percent Battery / Message	0.01709	0.00010	0.00095	0.00002
	Messages Received (Note the losses)	240 / 1024	1024 / 1024	524 / 1024	1024 / 1024
Send Messages	**Messages / Hour**	1,926	21,685	5,229	23,184
	Percent Battery / Hour	18.79%	17.80%	5.44%	3.66%
	Percent Battery / Message	0.00975	0.00082	0.00104	0.00016

*sending and receiving 1024 messages of 1 byte each.(source: https://www.ibm.com/developerworks)

Application Example

- Home care monitoring solution
  - Home and patient instrumented with sensors.
    - E.g. door motion, blood pressure, pacemaker/defib.
  - Collected by monitoring service (broker) using MQTT
  - Subscribed by a health care service in the hospital
  - Alerts relations/health care profs. if anything is out-of-order



Source: Lampkin 2012

Code Example: Python

```python
import paho.mqtt.client as mqtt

# Define event callbacks
def on_connect(client, userdata, flags, rc):
    print("Connection Result: " + str(rc))

def on_publish(client, obj, mid):
    print("Message ID: " + str(mid))

mqttc = mqtt.Client()

# Assign event callbacks
mqttc.on_connect = on_connect
mqttc.on_publish = on_publish

# parse mqtt url for connection details
url_str = sys.argv[1]
print(url_str)
url = urlparse(url_str)
base_topic = url.path[1:]

# Connect
if (url.username):
    mqttc.username_pw_set(url.username, url.password)

mqttc.connect(url.hostname, url.port)
mqttc.loop_start()
```

```python
# Publish a message to temp every 15 seconds
while True:
    temp=round(sense.get_temperature(),2)
    temp_json=json.dumps({"temperature":temp, "timestamp":time.time()})
    mqttc.publish(base_topic+"/temperature", temp_json)
    time.sleep(15)
```