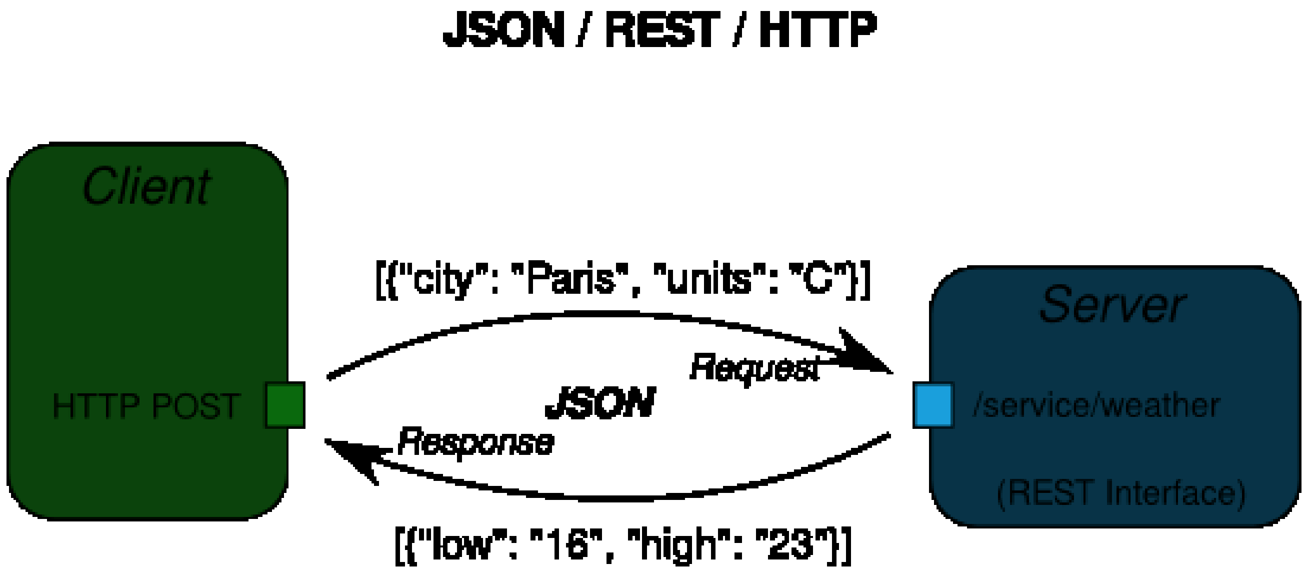
REST and MQTT

An IoT oriented comparison

Credits: Prof.Pietro Manzoni, University of Valencia, Spain

REST

- Basic concepts
- Basic programming



Data Representation

{“value": 237} vs. <value>237</value>

- Data-interchange format. Should be easy for humans to read and write. Should be easy for machines to parse and generate
- Two main formats:

JavaScript Object Notation (JSON) [http://www.json.org/]	XML
See JSON example below	See XML example below

JavaScript Object Notation (JSON) [http://www.json.org/]

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
      "menuitem": [
          {"value": "New", "onclick": ”NewDoc()"},
          {"value": "Open", "onclick": "OpenDoc()"},
          {"value": "Close", "onclick": "CloseDoc()"}
      ]
    }
  }
}
```

XML

```
<menu>
  <id>file</id>
  <value>File</value>
  <popup>
    <menuitem>
      <value>New</value>
      <onclick>NewDoc()</onclick>
    </menuitem>
    <menuitem>
      <value>Open</value>
      <onclick>OpenDoc()</onclick>
    </menuitem>
    <menuitem>
      <value>Close</value>
      <onclick>CloseDoc()</onclick>
    </menuitem>
  </popup>
</menu>
```

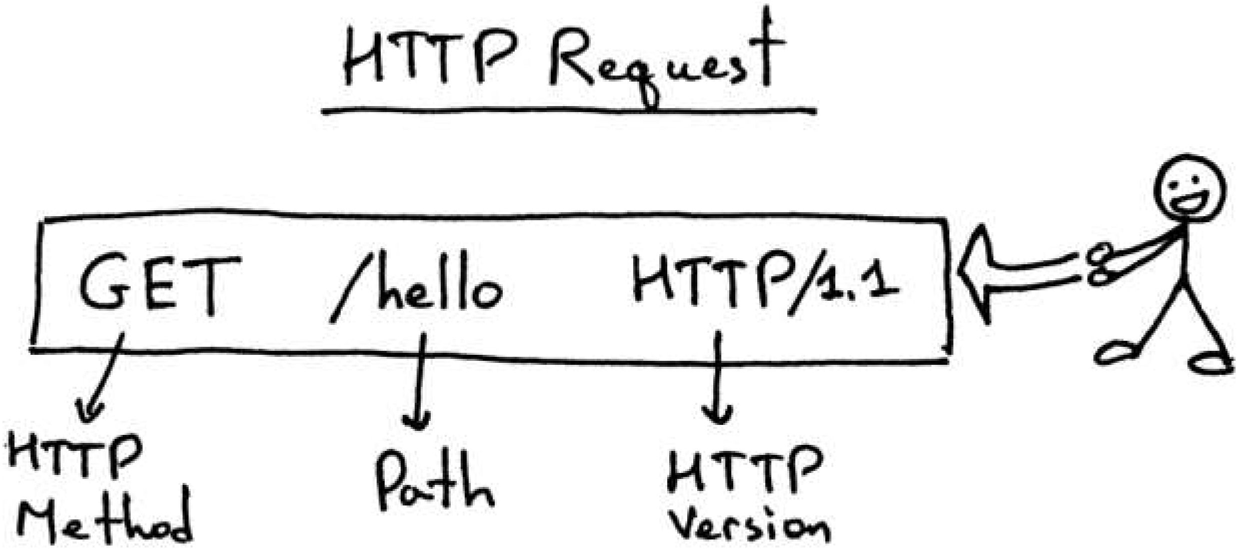
REST and HTTP

- REST stands for Representational State Transfer.
- It basically leverages the HTTP protocol and its related frameworks to provide data services.
- The motivation for REST was to capture the characteristics of the Web which made the Web successful.
- Make a Request – Receive Response – Display Response
- REST is not a standard… but it uses several standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/etc (Resource Representations)
  - text/xml, text/html, image/gif, image/jpeg, etc (Resource Types, MIME Types)

Verbs

- Represent the actions to be performed on resources

- HTTP GET
- HTTP POST
- HTTP PUT
- HTTP DELETE



HTTP GET

- How clients ask for the information they seek.
- Issuing a GET request transfers the data from the server to the client in some representation (JSON, XML, …)

```
GET http://www.library.edu/books
```

- Retrieve all books

```
GET http://www.library.edu/books/ISBN-0011021
```

- Retrieve book identified with ISBN-0011021

```
GET http://www.library.edu/books/ISBN-0011021/authors
```

- Retrieve authors for book identified with ISBN-0011021

HTTP PUT, HTTP POST

- HTTP POST creates a resource
- HTTP PUT updates a resource

- POST http://www.library.edu/books/

  `Content: {title, authors[], …}`

  - Creates a new book with given properties

- PUT http://www.library.edu/books/isbn-111

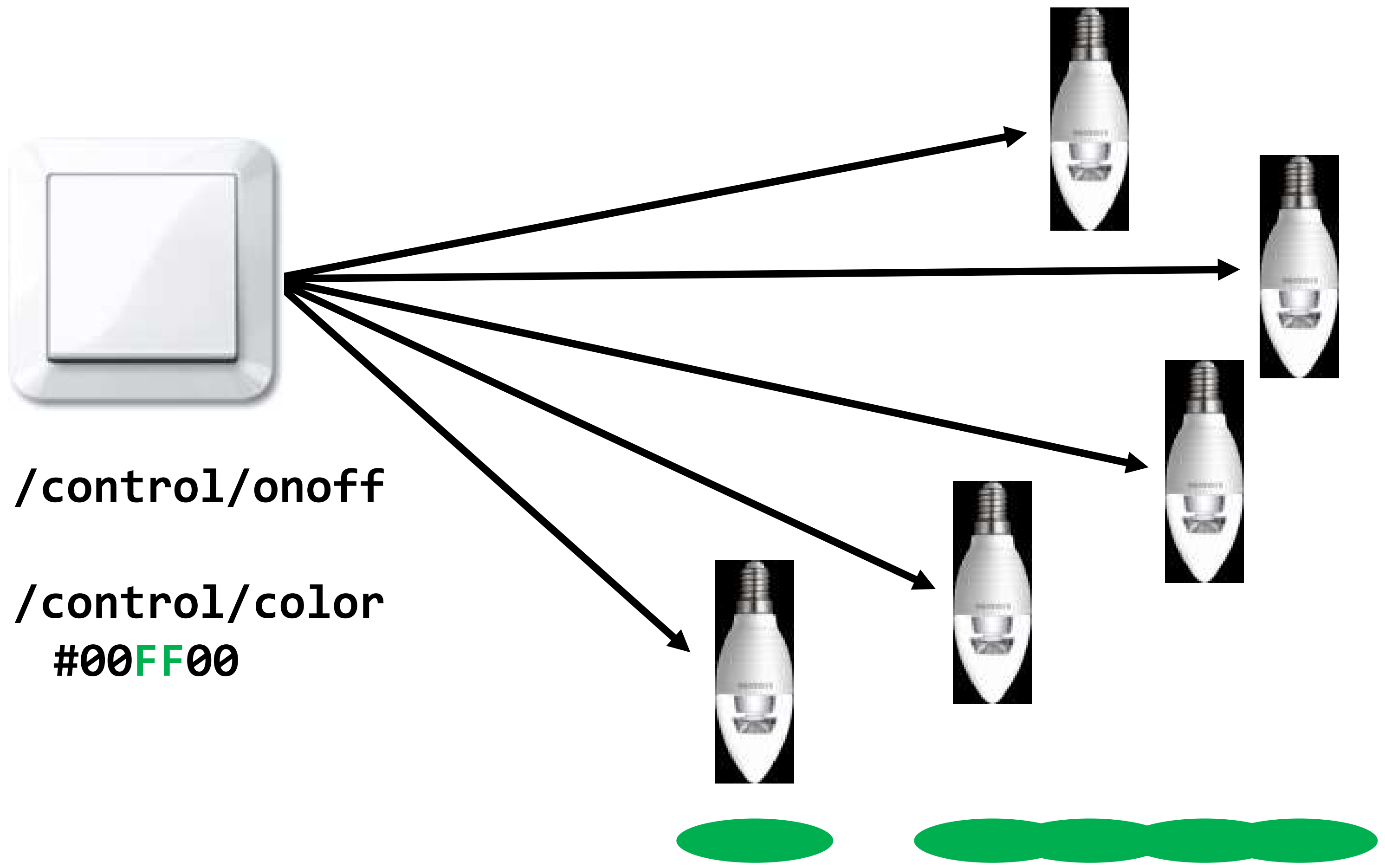
  `Content: {isbn, title, authors[], …}`

  - Updates book identified by isbn-111 with submitted properties

REST for devices control communication

```
GET /status/power
```

```
all-lights.floor-d.example.com
```



```
PUT /control/onoff

PUT /control/color
           #00FF00
```

HTTP DELETE

- Removes the resource identified by the URI

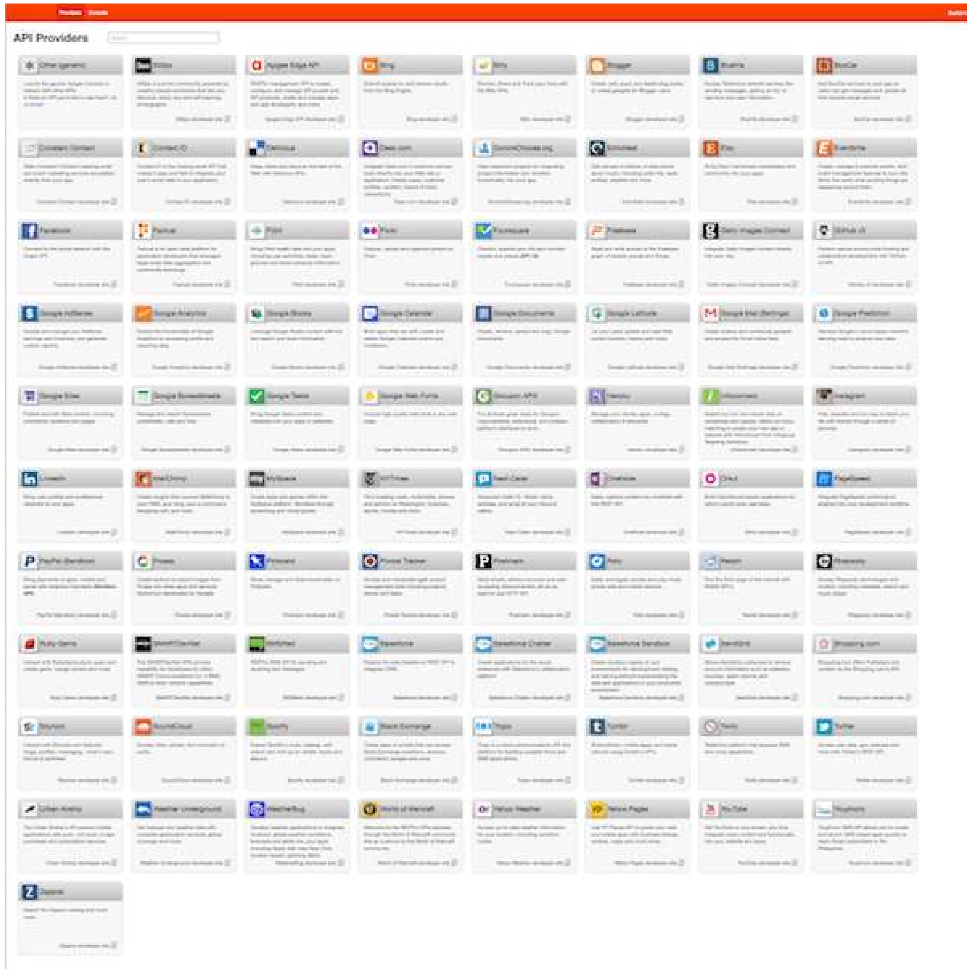
- DELETE `http://www.library.edu/books/ISBN-0011`
  - Delete book identified by ISBN-0011

REST is widely used

- Google Maps:
  - https://developers.google.com/maps/web-services/
  - Try: http://maps.googleapis.com/maps/api/geocode/json?address=bangkok
- Twitter:
  - https://dev.twitter.com/rest/public
- Facebook:
  - https://developers.facebook.com/docs/atlas-apis
- Amazon offers several REST services, e.g., for their S3 storage solution
  - http://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html
- Also:
  - The Google Glass API, known as "Mirror API", is a pure REST API.
    - Here is (https://youtu.be/JpWmGX55a40) a video talk about this API. (The actual API discussion starts after 16 minutes or so.)
  - Tesla Model S uses an (undocumented) REST API between the car systems and its Android/iOS apps.
    - http://docs.timdorr.apiary.io/#reference/vehicles/state-and-settings

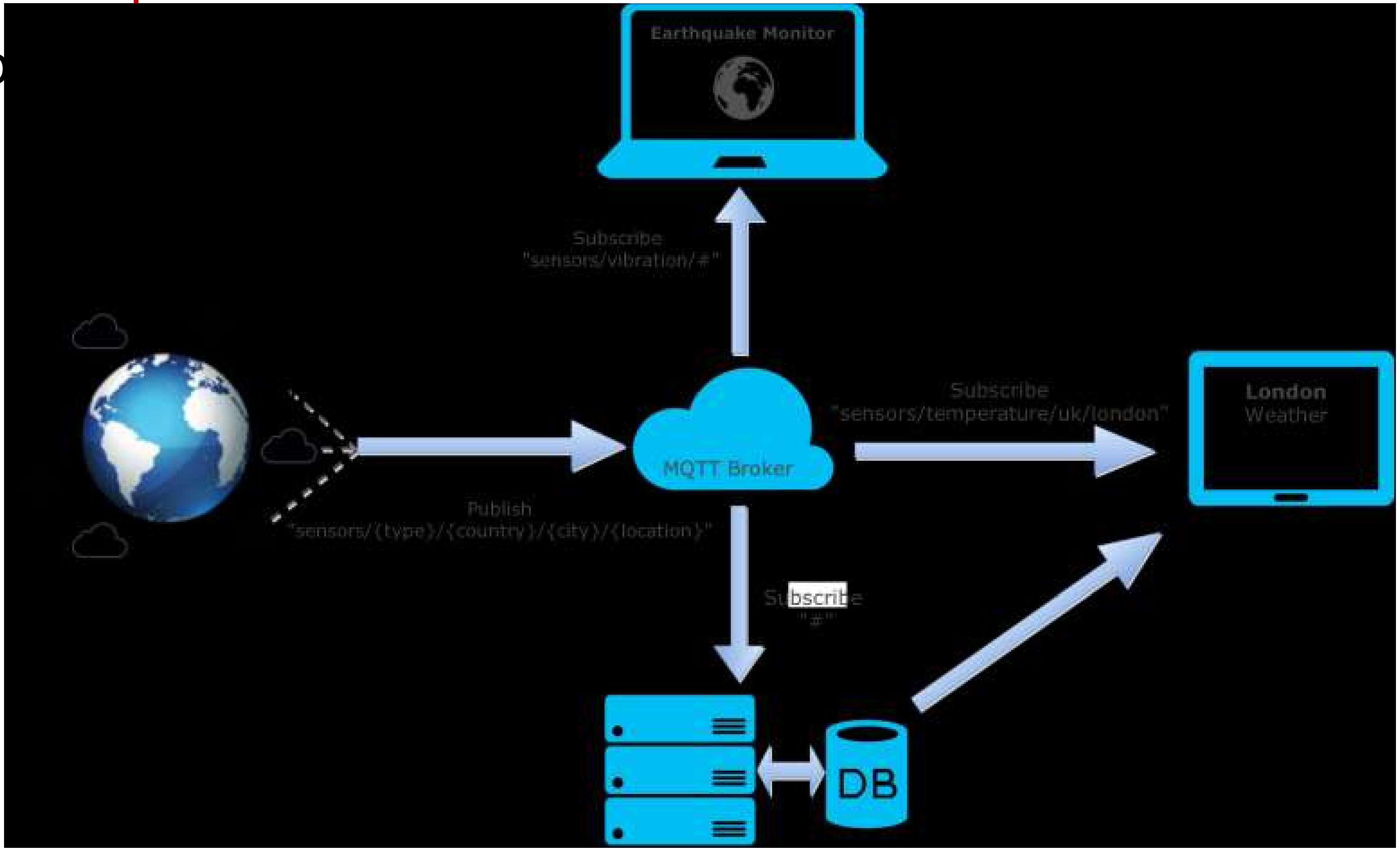
Very widely…

- https://apigee.com/providers
  - https://apigee.com/resources/instagram


API Providers

MQTT

- Basic concepts
- Basic p

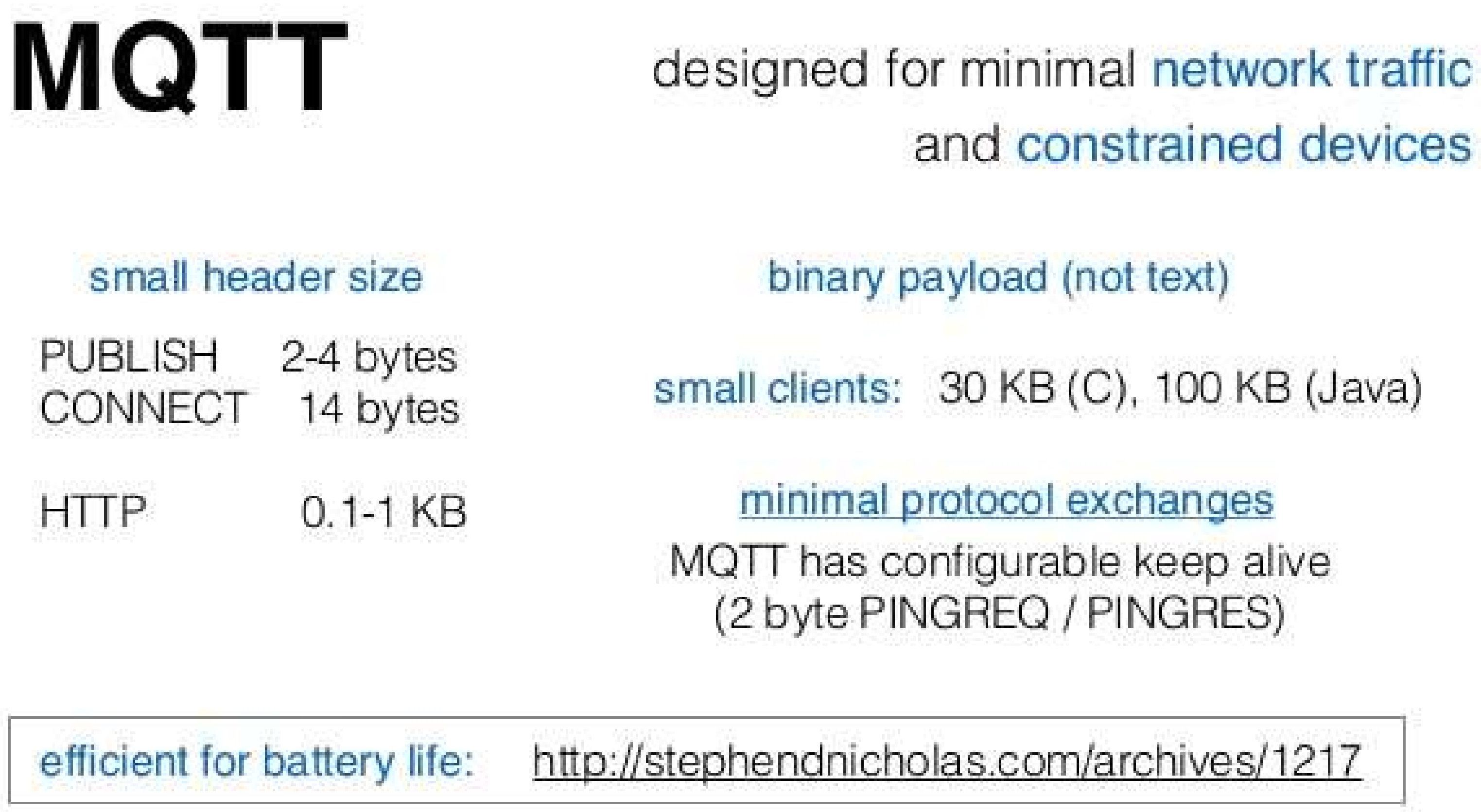


Source: https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot

Message Queuing Telemetry Transport

- A lightweight publish-subscribe protocol that runs on embedded devices and mobile platforms designed to connect the physical world devices with applications and middleware.
  - http://mqtt.org/
  - the MQTT community wiki: https://github.com/mqtt/mqtt.github.io/wiki
  - http://www.hivemq.com/mqtt-essentials/
- Designed to provide a low latency two-way communication channel and efficient distribution to one or many receivers.
- **Minimizes the amount of bytes flowing over the wire**
- **Low power usage.**

Message Queuing Telemetry Transport



MQTT

designed for minimal network traffic
and constrained devices

small header size

PUBLISH	2-4 bytes
CONNECT	14 bytes
HTTP	0.1-1 KB

binary payload (not text)

small clients: 30 KB (C), 100 KB (Java)

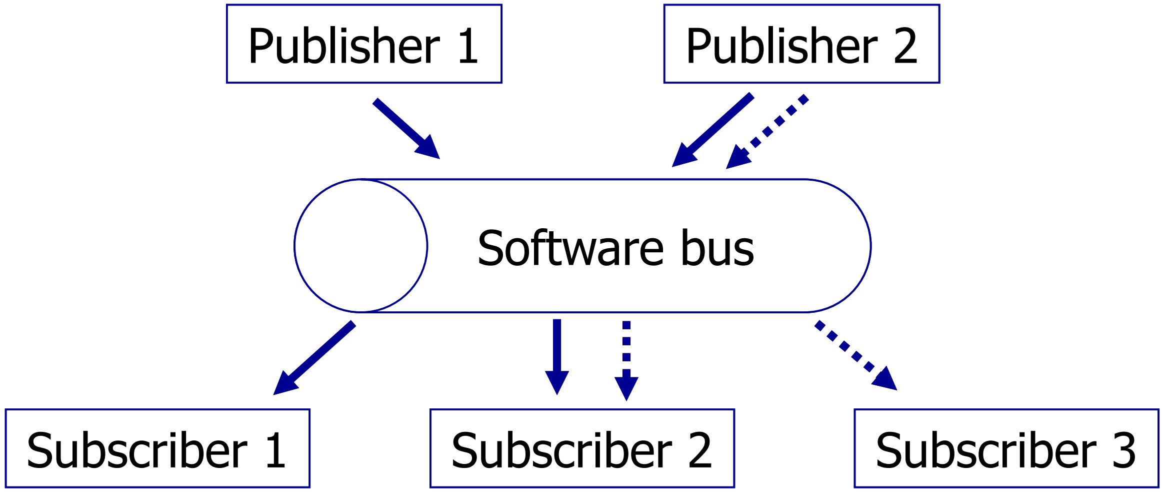
minimal protocol exchanges

MQTT has configurable keep alive
(2 byte PINGREQ / PINGRES)

efficient for battery life: http://stephendnicholas.com/archives/1217

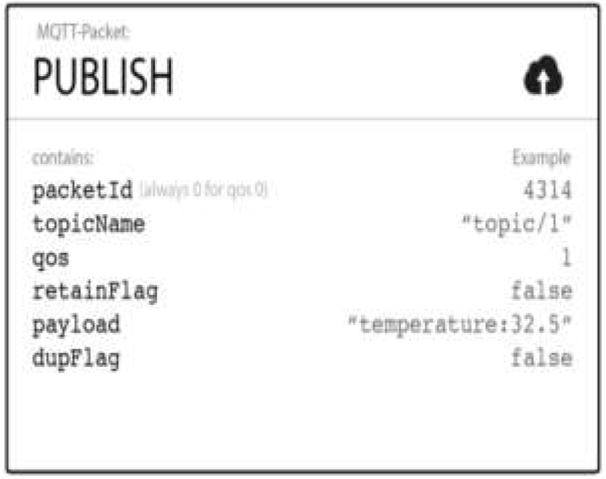
Publish/subscribe pattern

- Pub/Sub decouples a client, who is sending a particular message (called publisher) from another client (or more clients), who is receiving the message (called subscriber). This means that the publisher and subscriber don’t know about the existence of one another.
- There is a third component, called broker, which is known by both the publisher and subscriber, which filters all incoming messages and distributes them accordingly.



Publish

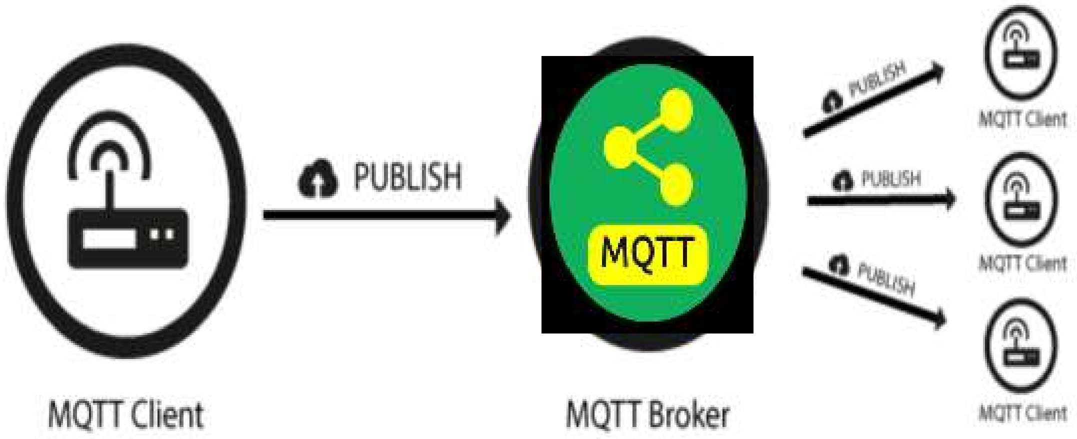
- MQTT is data-agnostic and it totally depends on the use case how the payload is structured. It's completely up to the sender if it wants to send binary data, textual data or even full-fledged XML or JSON.



MQTT-Packet:

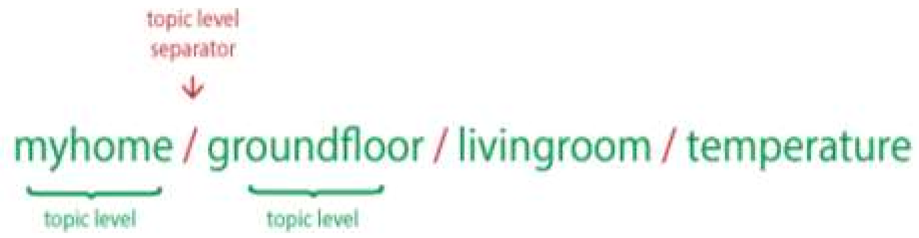
PUBLISH

contains:	Example
packetId (always 0 for qos 0)	4314
topicName	"topic/1"
qos	1
retainFlag	false
payload	"temperature:32.5"
dupFlag	false



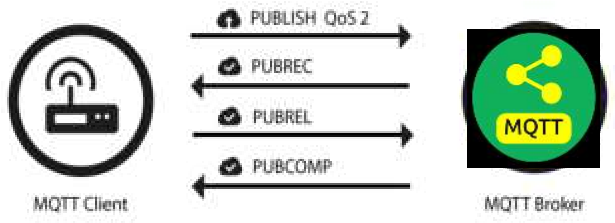
Topics

- Topic subscriptions can have wildcards
  - '+' matches anything at a given tree level so the topic "sensor/+/temp" would match "sensor/dev1/temp", "sensor/dev2/temp", etc.
  - '#' matches a whole sub-tree, so "sensor/#" would match all topics under "sensor/".
- These enable nodes to subscribe to groups of topics that don't exist yet, allowing greater flexibility in the network's messaging structure.



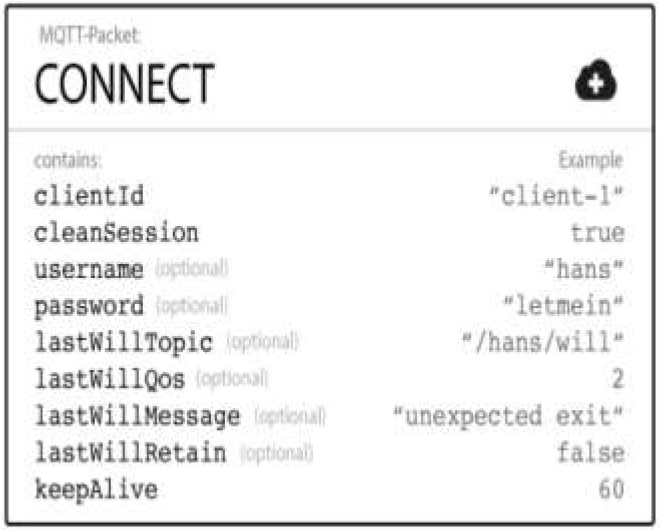
Quality of Service (QoS)

- Messages are published with a Quality of Service (QoS) level, which specifies delivery requirements.
- A QoS-0 ("at most once") message is fire-and-forget.
  - For example, a notification from a doorbell may only matter when immediately delivered.
- With QoS-1 ("at least once"), the broker stores messages on disk and retries until clients have acknowledged their delivery.
  - (Possibly with duplicates.) It's usually worth ensuring error messages are delivered, even with a delay.
- QoS-2 ("exactly once") messages have a second acknowledgement round-trip, to ensure that non-idempotent messages can be delivered exactly once.



“Will” message

- When clients connect, they can specify an optional “will” message, to be delivered if they are unexpectedly disconnected from the network.
  - (In the absence of other activity, a 2-byte ping message is sent to clients at a configurable interval.)
- This “last will and testament” can be used to notify other parts of the system that a node has gone down.



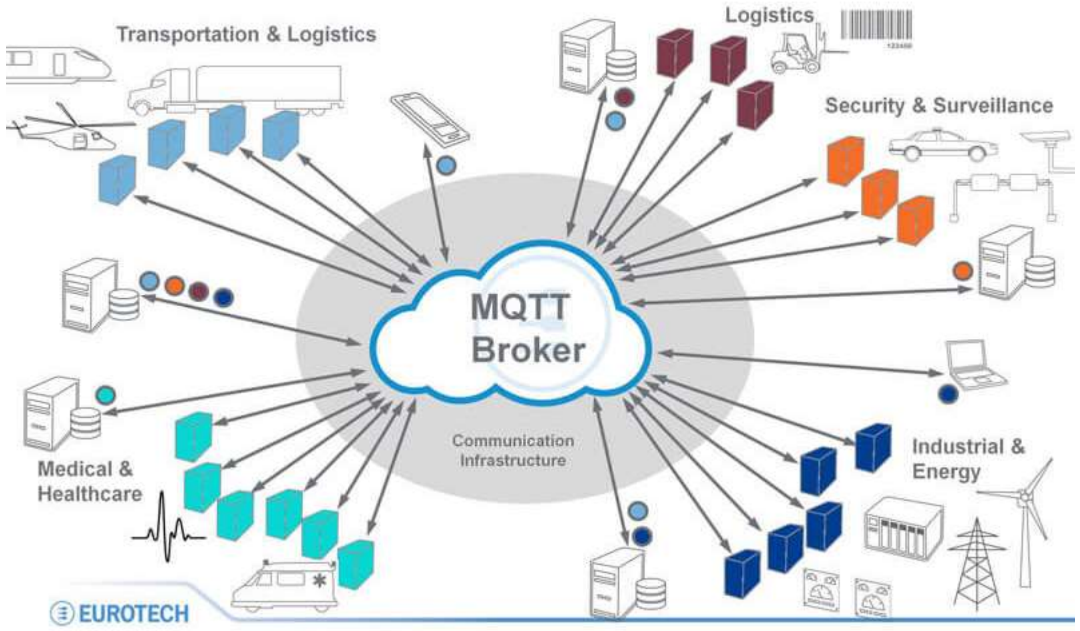
MQTT-Packet:

CONNECT

contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

MQTT

- Basic concepts
- Basic programming



Software available

- Brokers (https://github.com/mqtt/mqtt.github.io/wiki/servers):
  - **http://mosquitto.org/**
  - http://www.hivemq.com/
  - https://www.rabbitmq.com/mqtt.html
  - http://activemq.apache.org/mqtt.html
  - https://github.com/mcollina/mosca
- Clients
  - **http://www.eclipse.org/paho/**
    - Source: https://github.com/eclipse/paho.mqtt.python
  - http://www.hivemq.com/demos/websocket-client/
- Tools
  - https://github.com/mqtt/mqtt.github.io/wiki/tools

The suggested working environment

- Installing the Python client

```
sudo pip install paho-mqtt
```

  - or

```
git clone https://github.com/eclipse/paho.mqtt.python.git
cd org.eclipse.paho.mqtt.python.git
python setup.py install
```

- Installing the whole Mosquitto environment

```
sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
cd /etc/apt/sources.list.d/

sudo wget http://repo.mosquitto.org/debian/mosquitto-jessie.list
sudo apt-get update sudo apt-get install mosquitto

sudo apt-get install mosquitto mosquitto-clients python-mosquitto
```

Controlling the broker

- Starting / stopping the broker

```
sudo /etc/init.d/mosquitto stop
sudo /etc/init.d/mosquitto start
```

- To avoid that it restarts automatically

```
sudo stop mosquitto
```

- Running verbose mode:

```
sudo mosquitto -v
```

- To check whether is running:

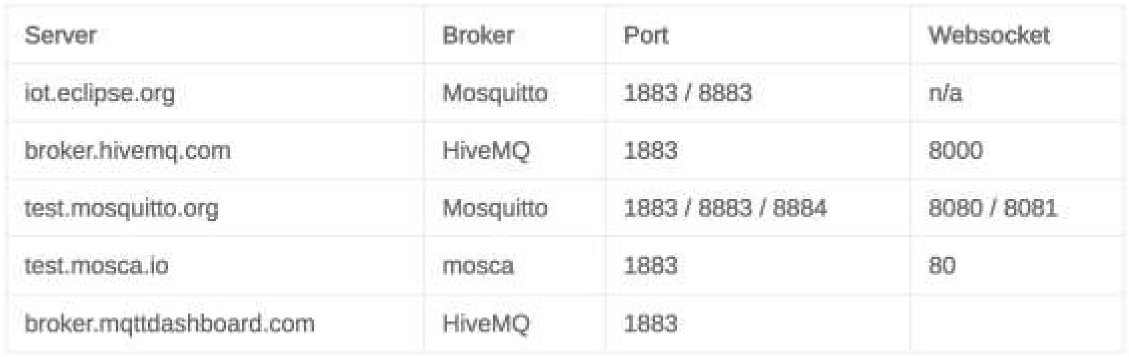
```
sudo netstat -tanlp | grep 1883
ps -ef | grep mosquitto
```

- man page:

https://mosquitto.org/man/mosquitto-8.html

Sandboxes

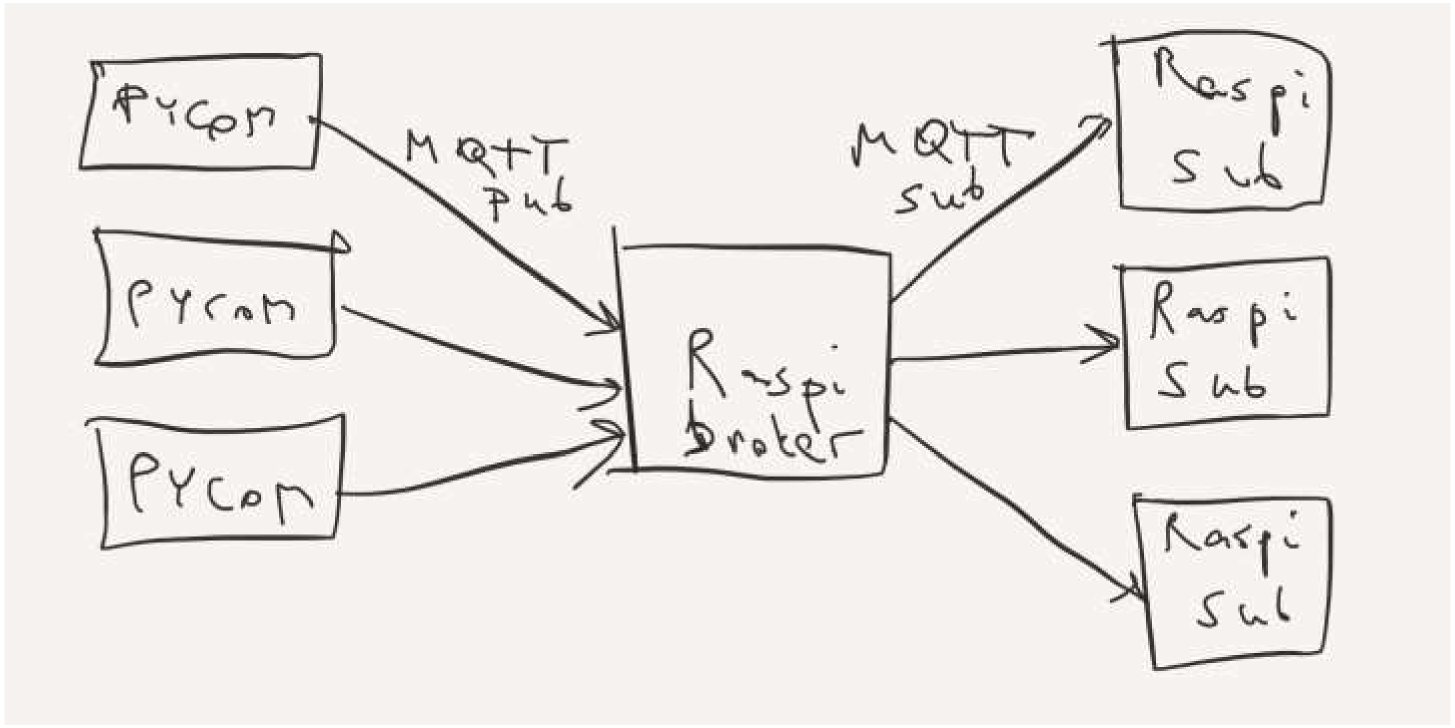
- https://iot.eclipse.org/getting-started#sandboxes
  - hostname iot.eclipse.org and port 1883.
  - encrypted port 8883
  - http://test.mosquitto.org/
  - http://www.hivemq.com/try-out/
  - http://www.mqtt-dashboard.com/



Server	Broker	Port	Websocket
iot.eclipse.org	Mosquitto	1883 / 8883	n/a
broker.hivemq.com	HiveMQ	1883	8000
test.mosquitto.org	Mosquitto	1883 / 8883 / 8884	8080 / 8081
test.mosca.io	mosca	1883	80
broker.mqttdashboard.com	HiveMQ	1883	

- Google CLOUD PUB/SUB A global service for real-time and reliable messaging and streaming data
  - https://cloud.google.com/pubsub/

Excercise


Pycom
Pycom
Pycom
MQTT pub
MQTT sub
Raspi broker
Raspi Sub
Raspi Sub
Raspi Sub

Excercise

- Broker information:

- WiFi SSID: mqtt
- WiFi password: brokerpi
- Broker IP address: 172.24.1.1

Excercise

1. Start with the MQTT example. It will send a text message via MQTT to the broker. Does it show up?

2. Next develop a MQTT + Pysense example. It will read sensor values from Pysense and send them to the broker. Can you read the values in the broker?

3. Install a MQTT client on your PC/smartphone and subscribe to the topics of interest.