MQTT
Message Queue Telemetry Transport

http://mqtt.org/
MQTT - Open Connectivity for Mobile, M2M and IoT

A lightweight publish/subscribe protocol with predictable bi-directional message delivery
MQTT

- MQTT is described on the mqtt.org site as a machine-to-machine (M2M) / IoT connectivity protocol.
- MQTT is an Event based IoT middleware (one to many)
  - publish/subscribe messaging transport protocol
  - Over TCP/IP (or MQTT-S over UDP for LAN)
- Its protocol is lightweight
  - it can be supported by some of the smallest measuring and monitoring devices (ex. Arduino)
  - it can transmit data over far reaching networks
  - It can transmit data over sometimes intermittent networks.

Event based IoT Middleware
- Event pattern of communication (one to many)
- Over IP (TCP)
Publish / Subscribe Messaging (One to Many)

- A producer publishes a message (publication) on a topic (subject)
- A consumer subscribes (makes a subscription) for messages on a topic (subject)
- A message server (called BROKER) matches publications to subscriptions
  - If none of them match the message is discarded after modifying the topic
  - If one or more matches the message is delivered to each matching consumer after modifying the topic

Publish / Subscribe has three important characteristics:
  1. It decouples message senders and receivers, allowing for more flexible applications
  2. It can take a single message and distribute it to many consumers
  3. This collection of consumers can change over time, and vary based on the nature of the message.
MQTT Topics & Wildcards

- Topics are hierarchical (like filesystem path):
  - `/wsn/sensor/R1/temperature`
  - `/wsn/sensor/R1/pressure`
  - `/wsn/sensor/R2/temperature`
  - `/wsn/sensor/R2/pressure`

- A Subscriber can use wildcards in topics:
  - `/wsn/sensor/+/temperature`
  - `/wsn/sensor/R1/+`
  - `/wsn/sensor/#`
MQTT Topic : Details

- A topic forms the namespace
  - Is hierarchical with each “sub topic” separated by a /
  - An example topic space
    - A house publishes information about itself on:
      - `<country>/<region>/<town>/<postcode>/<house>/energyConsumption`
      - `<country>/<region>/<town>/<postcode>/<house>/solarEnergy`
      - `<country>/<region>/<town>/<postcode>/<house>/alarmState`
    - And subscribes for control commands:
      - `<country>/<region>/<town>/<postcode>/<house>/thermostat/setTemp`

- A subscriber can subscribe to an absolute topic or can use wildcards:
  - Single-level wildcards “+” can appear anywhere in the topic string
  - Multi-level wildcards “#” must appear at the end of the string
  - Wildcards must be next to a separator
  - Cannot be used wildcards when publishing
  - For example
    - `UK/Hants/Hursley/SO212JN/1/energyConsumption`
      - Energy consumption for 1 house in Hursley
    - `UK/Hants/Hursley/+/+/energyConsumption`
      - Energy consumption for all houses in Hursley
    - `UK/Hants/Hursley/SO212JN/#`
      - Details of energy consumption, solar and alarm for all houses in SO212JN
MQTT publish subscribe architecture

- The MQTT messages are delivered asynchronously ("push") through publish subscribe architecture.
- The MQTT protocol works by exchanging a series of MQTT control packets in a defined way.
- Each control packet has a specific purpose and every bit in the packet is carefully crafted to reduce the data transmitted over the network.
- A MQTT topology has a MQTT server and a MQTT client.
- MQTT client and server communicate through different control packets. Table below briefly describes each of these control packets.

<table>
<thead>
<tr>
<th>Control packet</th>
<th>Direction of flow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT</td>
<td>Client to Server</td>
<td>Client request to connect to Server</td>
</tr>
<tr>
<td>CONNACK</td>
<td>Server to Client</td>
<td>Connect acknowledgment</td>
</tr>
<tr>
<td>PUBLISH</td>
<td>Client to Server</td>
<td>Publish message</td>
</tr>
<tr>
<td></td>
<td>or Server to Client</td>
<td></td>
</tr>
<tr>
<td>PUBACK</td>
<td>Client to Server</td>
<td>Publish acknowledgment</td>
</tr>
<tr>
<td></td>
<td>or Server to Client</td>
<td></td>
</tr>
<tr>
<td>PUBREC</td>
<td>Client to Server</td>
<td>Publish received (assured delivery part 1)</td>
</tr>
<tr>
<td></td>
<td>or Server to Client</td>
<td></td>
</tr>
<tr>
<td>PUBREL</td>
<td>Client to Server</td>
<td>Publish release (assured delivery part 2)</td>
</tr>
<tr>
<td></td>
<td>or Server to Client</td>
<td></td>
</tr>
<tr>
<td>PUBCOMP</td>
<td>Client to Server</td>
<td>Publish complete (assured delivery part 3)</td>
</tr>
<tr>
<td></td>
<td>or Server to Client</td>
<td></td>
</tr>
<tr>
<td>SUBSCRIBE</td>
<td>Client to Server</td>
<td>Client subscribe request</td>
</tr>
<tr>
<td>SUBACK</td>
<td>Server to Client</td>
<td>Subscribe acknowledgment</td>
</tr>
<tr>
<td>UNSUBSCRIBE</td>
<td>Client to Server</td>
<td>Unsubscribe request</td>
</tr>
<tr>
<td>UNSUBACK</td>
<td>Server to Client</td>
<td>Unsubscribe acknowledgment</td>
</tr>
<tr>
<td>PINGREQ</td>
<td>Client to Server</td>
<td>PING request</td>
</tr>
<tr>
<td>PINGRESP</td>
<td>Server to Client</td>
<td>PING response</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>Client to Server</td>
<td>Client is disconnecting</td>
</tr>
</tbody>
</table>
Sample of protocol use
Ideal for constrained networks (low bandwidth, high latency, data limits, and fragile connections)

- MQTT control packet headers are kept as small as possible.
- Each MQTT control packet consist of three parts, a fixed header, variable header and payload.
- Each MQTT control packet has a 2 byte Fixed header. Not all the control packet have the variable headers and payload.
- A variable header contains the packet identifier if used by the control packet.
- A payload up to 256 MB could be attached in the packets.
- Having a small header overhead makes this protocol appropriate for IoT by lowering the amount of data transmitted over constrained networks.
Quality of Service (QoS) for MQTT

- Quality of service (QoS) levels determine how each MQTT message is delivered and must be specified for every message sent through MQTT.
- Three QoS for message delivery could be achieved using MQTT:
  - QoS 0 (At most once) - where messages are delivered according to the best efforts of the operating environment. Message loss can occur.
  - QoS 1 (At least once) - where messages are assured to arrive but duplicates can occur.
  - QoS 2 (Exactly once) - where message are assured to arrive exactly once.
- There is a simple rule when considering performance impact of QoS:
  
  “The higher the QoS, the lower the performance”.

Quality of Service (QoS) for MQTT

QoS 0: At most once (fire and forget)

QoS 1: At least once

QoS 2: Exactly once
MQTT Clients and APIs

- You can develop an MQTT client application by programming directly to the MQTT protocol specification ...... however it is more convenient to use a prebuilt client

- Open Source clients available in Eclipse Paho project
  - C, C++, Java, JavaScript, Lua, Python and Go

- Clients for other languages are available, see mqtt.org/software
  - E.g. Delphi, Erlang, .Net, Objective-C, PERL, PHP, Ruby
    - Not all of the client libraries listed on mqtt.org are current. Some are at an early or experimental stage of development, whilst others are stable and mature.

- Even in shell script like we are seeing in the practical course ...