EVOLUTION OF THE NET

CLOSED

WAVE ONE:
RESEARCH/ ACADEMICS
Restricted access for military and academic use

OPEN

WAVE TWO:
INDIVIDUAL/ SOCIAL
General, commercially available

ALL

WAVE THREE:
INTERNET OF ALL
Confluence of commodity Internet, the Internet of Things and advanced network applications

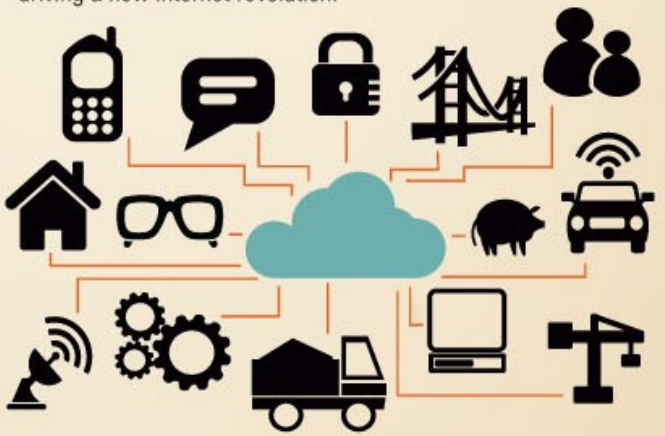
1968-1981 Mainframe/Minicomputer

1982-2012 Personal/Mobile/Cloud

2013 Wearable/Embedded

SMARTER STUFF
INTERNET OF THINGS

It originally linked computers to computers, now billions of devices are connected to the Web: smartphones, automobiles, light bulbs, utility meters, remote sensors and more. In the coming decade, the quantity and types of devices linked to the Web and the apps that will run on this expanding "Internet of Things" will explode, driving a new Internet revolution.



CONNECTED DEVICES

	2003	2010	2015	2020
WORLD POPULATION	6.3 BILLION	6.8 BILLION	7.2 BILLION	7.6 BILLION
CONNECTED DEVICES	500 MILLION	12.5 BILLION	25 BILLION	50 BILLION

CONNECTED DEVICES PER PERSON

Cisco Systems Inc.

99% of physical objects that may one day join the network are still unconnected. — Cisco Systems Inc.

OPPORTUNITY $

Corporations see big opportunity in helping government and business navigate the convergence of the digital with the broader physical world. A few of their initiatives:

Smarter Planet IBM

Industrial Internet General Electric

Planetary Skin Cisco

Central Nervous System for the Earth HP

Powerful Answers Verizon

Smart Community Toshiba

Problem definition

- Distributed network of devices communicating with a central location or to each other.
- Devices that run on batteries or with limited power.
- Information flows over unreliable networks (cellular, satellite, any wireless technology in general).
- No need to write an application protocol from scratch on top of TCP/IP.

Agenda

- Introducing MQTT
- Publish / Subscribe
- Client, Broker and Connection Establishment
- MQTT Publish, Subscribe and Unsubscribe
- MQTT Topics
- MQTT Quality of Service Levels
- Persistent Session and Queuing Messages
- Retained Messages
- Last Will and Testament
- Keep Alive and Client Take-Over
- Demo

What is MQTT ?

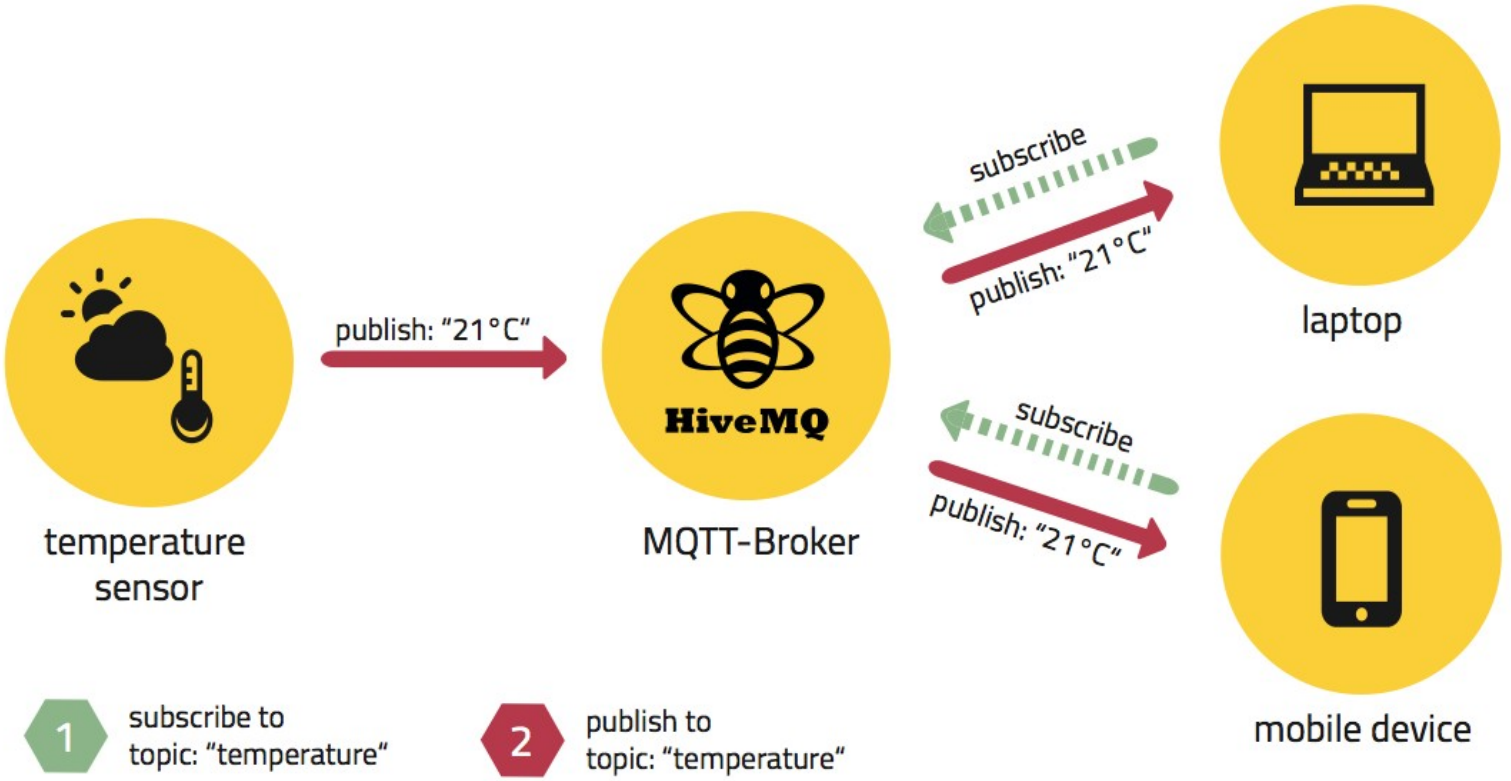
MQTT is a lightweight publish/subscribe protocol with reliable bi-directional message delivery.

Invented in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper. The original problem was to send sensor data from oil pipelines through a satellite link.

Key aspects

- Designed to handle high volumes of data in low bandwidth networks.
- Small code footprint.
- Runs on top of TCP/IP. Both client and broker need a TCP/IP stack. Some variations like MQTT-SN run over non-TCP/IP networks.
- Avoids polling.
- Event oriented.
- Recovery, store and forward, and publish/subscribe are part of the implementation (no need to implement in the application logic).

Publish / Subscribe



Publish / Subscribe

- It decouples the publisher from the subscriber.
- Clients are always connected with a broker.
- Both publishers and subscribers are “clients”.
- A client sends a message (publisher).
- One or more clients receive the message (subscribers).
- MQTT uses “topics” to determine which message is routed to which client.
- A topic is a “hierarchical structured string”.

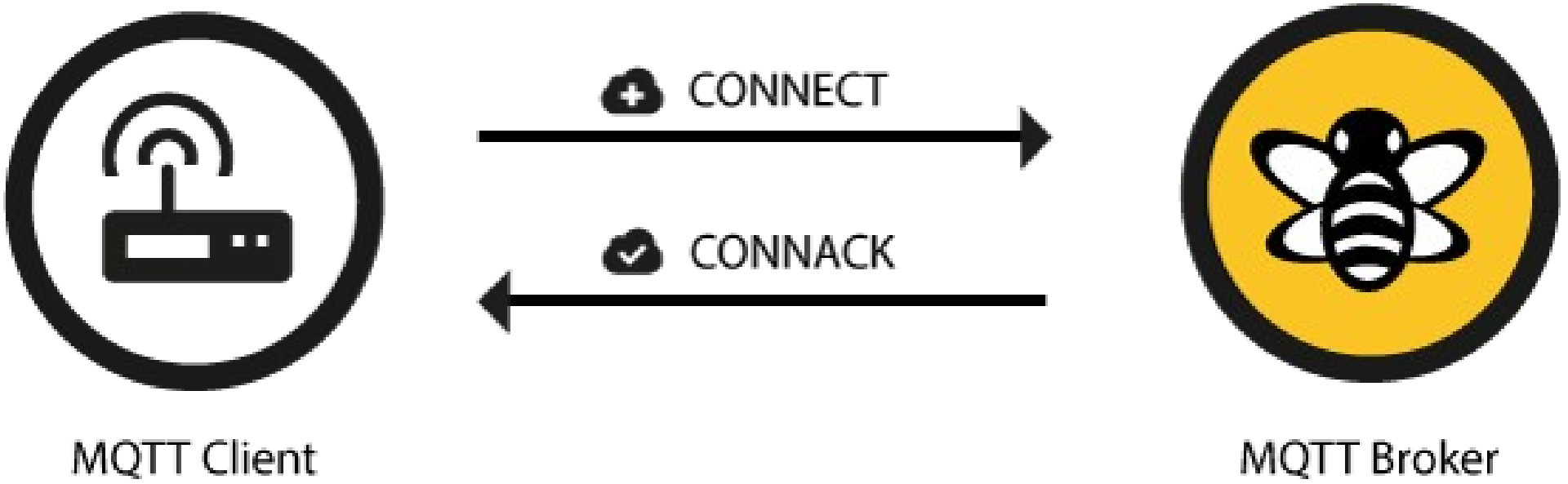
Client

- Can be a microcontroller up to a server.
- Implementation is straight-forward.
- Some numbers: C=30kb, Java=100kb.
- MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET, etc.

Broker

- Core of any publish/subscribe protocol.
- Receives all messages from clients, filters them, and then sends messages to all clients interested in a particular topic.
- Handles authentication/authorization of clients.
- Highly scalable, easy to integrate into backend systems, failure-resistant.

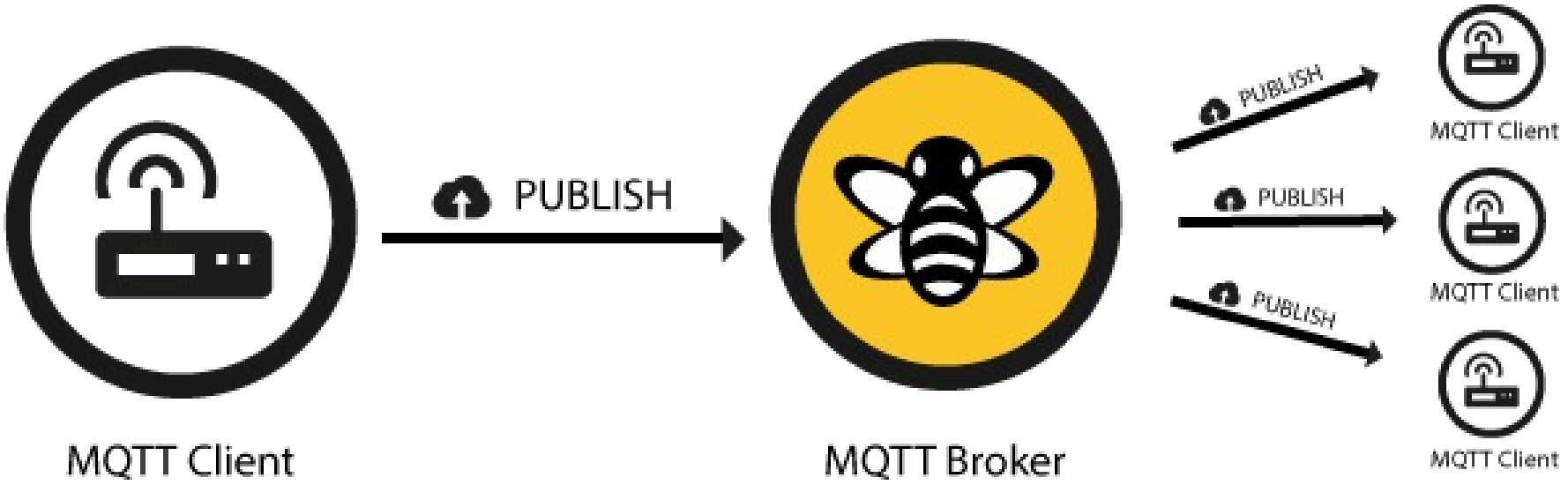
Connection, always initiaited by a client.



Connect packet

- ClientId: Unique ID per broker.
- Clean Session: Flag to indicate if the session must be persistent or not. A persistent session (CleanSession = false) means, that the broker will store all subscriptions for the client and all missed messages (with Quality of Service (QoS) 1 or 2). If the flag is set to true, the broker won’t store anything for the client and will purge all information from a previous persistent session.
- Username/Password: Sent in plaintext. Application must encrypt it.
- Will Message: Its purpose is to notify other clients when a client disconnects ungracefully. The broker sends this message on behalf of the client.
- KeepAlive: Period in secs the client is committed to send a PING to the broker, so each other know if the other end is alive and reachable.

Publish, subscribe and unsubscribe



Publish packet

- Topic Name: A string, hierarchically structured with forward slashes as delimiters, i.e “building/room_number/humidity”
- QoS: Quality of Service Level. Possible values are (0,1,2).
- Retain-Flag: Specifies if the broker saves the latest message for the specified topic as last known good value. New clients that subscribe to that topic will receive the last retained message on that topic instantly after subscribing.
- Payload: In binary form.
- Packet Identifier: Unique identifier between client and broker to identify a message in a message, but only relevant for QoS 1 and 2. The client or the broker must set by the client or the broker.
- DUP flag: Indicates that this message is a duplicate and is resent because no ACK was received. Only relevant for QoS greater than 0. Retries must be handled as an implementation detail by the client or the broker.

Subscribe packet

- Packet Identifier: Only needed for QoS > 0.
- List of Subscriptions: A SUBSCRIBE message can contain an arbitrary number of subscriptions for a client An arbitrary number of subscriptions are valid for a SUBSCRIBE message. Each subscription consists of a topic and QoS level.

Unsubscribe packet

- Packet Identifier: The ACK for an UNSUBSCRIBE packet will have the same packet id.
- List of Topics: The list of topics to unsubscribe from. No QoS is specified, just the topic.

Topics

- Case sensitive
- UTF-8
- Wildcards
  - Single level: building/+/humidity
    - building/room_4/humidity
    - building/room_67/humidity
    - building/room_78/humidity
  - Multiple level (only at the end): building/room_number/#
    - building/room_4/wall/temperature
    - building/room_4/wall/humidity
    - building/room_4/ceiling/temperature
    - building/room_4/ceiling/humidity

Quality of Service (QoS)

- Establishes the guarantees of delivering a message:
  - 0) At most once: No acks from the receiver, or stored and redelivered by the sender.
  - 1) At least once: As its name implies, msg delivered once, but can also be delivered more than once.
  - 2) Exactly once: Highest level QoS, but the slowest.
- QoS is set by the client. The broker will honor the QoS set by clients on each end. Therefore, QoS can get downgraded.

Persistent Session and Queue Management

- The session is identified by the clientId.
- The following is stored in the session:
  - Existence of a session, even when there are no subscriptions.
  - All subscriptions.
  - All messages with a Quality of Service (QoS) 1 or 2, which are not confirmed by the client.
  - All new QoS 1 or 2 messages, which the client missed while offline.
  - All received QoS 2 messages, which are not yet confirmed to the client.

Retained Messages

- A retained message on a topic is the last known good value, that is, the last message with the “retained flag” set to true.
- To delete a retained message, a zero payload message on a topic can be sent with the flag set to false. Since the broker deletes the retained message, new subscribers will not get notified on this topic upon subscription.

Last Will and Testament

- Only sent to subscribers when a client disconnects ungracefully (network error, no PINGS within specified “Keep Alive” period).
- A last will consists of a topic, retained flag, QoS and payload.
- The LWT can be specified on the CONNECT message.
- It will not be sent if the client sends the DISCONNECT message (graceful disconnect).

Keep Alive and Client Take-Over

- Needed because TCP/IP stacks “not always” notify when a socket breaks. Many times the connection seems open, but all writes are not reaching the other end.
- Max keep alive is 18h 12min 15 sec.
- Take-over is when a broker has a half-open connection (connection seems open) but the client reconnects. Then the broker will close the previous connection (based on the client id) and reopen a new one.

MQTT vs HTTP

- Push delivery of data / events:
    - MQTT low latency push from client to server and from server to client.
    - HTTP: Push from client to server but poll from server to client.
- Efficient use of network:
    - MQTT requires around 5 times less bytes than HTTP.
- Reliable delivery: keeps QoS even across connection breaks.

MQTT is being used in:

- POS.
- Slot machines.
- Automotive / Telematics.
- Medical.
- Home Automation.
- Railway.
- Asset tracking / management.
- Fire & Gas testing.

Well known companies using it

- Facebook for their messenger protocol:
  - When we joined Facebook and started to build Messenger, our first technical challenge was learning the entire infrastructure stack for Facebook Messages. It was great to be building on a scalable platform that had already launched to hundreds of millions of users, but the system contained certain assumptions and design decisions that didn't always quite mesh with the product we wanted to build. Luckily, our new colleagues were also excited about the vision for Messenger and joined the effort to make sure the system could do what we needed.

    One of the problems we experienced was long latency when sending a message. The method we were using to send was reliable but slow, and there were limitations on how much we could improve it. With just a few weeks until launch, we ended up building a new mechanism that maintains a persistent connection to our servers. To do this without killing battery life, we used a protocol called MQTT that we had experimented with in Beluga. MQTT is specifically designed for applications like sending telemetry data to and from space probes, so it is designed to use bandwidth and batteries sparingly. By maintaining an MQTT connection and routing messages through our chat pipeline, we were able to often achieve phone-to-phone delivery in the hundreds of milliseconds, rather than multiple seconds.

  - Source: https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920/

Well-known companies using it

- Amazon for their AWS IoT
- AWS IoT is a managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices.
- Currently supports HTTP, MQTT, and WebSockets.

Eclipse Paho Project

https://www.eclipse.org/paho/clients/c/embedded/

- In particular, for environments with limited resources, the Embedded MQTT C/C++ Client Libraries are available, contributed by Ian Craggs.
- Use very limited resources - pick and choose the components needed.
- Do not rely on any particular libraries for networking, threading or memory management.
- ANSI standard C for maximum portability, at the lowest level.
- If needed, higher layer(s) in C and/or C++ are available.

Current libraries

- MQTTPacket

  This is the lowest level library, the simplest and smallest, but hardest to use. It deals with serialization and deserialization of MQTT packets. Serialization means taking application data and converting it to a form ready for sending across the network. Deserialization means taking the data read from the network and extracting the data.

- MQTTClient

  This is a C++ library first written for mbed, but now ported to other platforms. Although it uses C++, it still avoids dynamic memory allocations, and has replaceable classes for OS and network dependent functions. Use of the STL is also avoided. It is based on, and requires, MQTTPacket.

- MQTTClient-C

  A C version of MQTTClient, for environments where C++ is not the norm, such as FreeRTOS. Also built on top of MQTTPacket.

MQTTPacket

```
MQTTPacket_connectData data = MQTTPacket_connectData_initializer;
int rc = 0;
char buf[200];
MQTTString topicString = MQTTString_initializer;
char* payload = "mypayload";
int payloadlen = strlen(payload);int buflen = sizeof(buf);

data.clientID.cstring = "me";
data.keepAliveInterval = 20;
data.cleansession = 1;
len = MQTTSerialize_connect(buf, buflen, &data); /* 1 */

topicString.cstring = "mytopic";
len += MQTTSerialize_publish(buf + len, buflen - len, 0, 0, 0, 0, topicString, payload, payloadlen); /* 2 */

len += MQTTSerialize_disconnect(buf + len, buflen - len); /* 3 */
/* From here, it is up to you to send/receive the produced payload over the network */
rc = Socket_new("127.0.0.1", 1883, &mysock);
rc = write(mysock, buf, len);
rc = close(mysock);
```

Quick cl demo

- On Ubuntu, install mosquitto, mosquitto_sub and mosquitto_pub
- Broker is running as a daemon
- Subscribe to topic "ELC":

  mosquitto_sub -t ELC -d

- - Post a file to the topic ELC, so subscribers get a notification:

  mosquitto_pub -t ELC -f publish.txt -d

Standard ports

- TCP/IP port 1883 is reserved with IANA (Internet Assigned Numbers Authority) for use with MQTT. TCP/IP port 8883 is also registered, for using MQTT over SSL.

Useful links

MQTT information
- http://mqtt.org

MQTT 3.1 Specification
- http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.htm

MQTT 3.1.1 Specification
- http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.pdf

Eclipse M2M Industry Working Group
- http://wiki.eclipse.org/Machine-to-Machine

OASIS MQTT Technical Committee
- https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt

HiveMQ
- http://www.hivemq.com

Eclipse Paho Project
- http://www.eclipse.org/paho/