HTTP e IoT

Protocollo CoAP - Constrained Application Protocol

E' un protocollo che nasce con l'obbiettivo di operare in ambienti vincolati dal punto di vista delle risorse disponibili usando il paradigma REST per l'interazione tra entità comunicanti, dove si fa riferimento a dispositivi con risorse limitate.

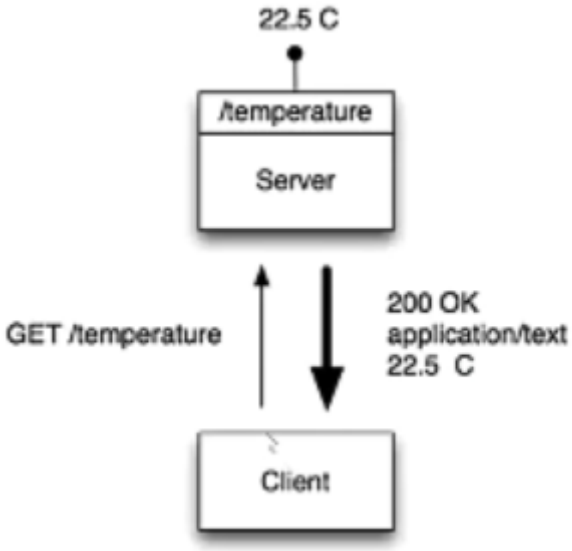
Si tratta di una versione binaria del protocollo HTTP. Per questo protocollo viene riservato il numero di porta **5683** in uno schema non HTTP ma coap; di conseguenza **useremo l'url: coap://.../:5683**.

La differenza con HTTP è il **protocollo usato a livello di trasporto**, che non è più TCP ma UDP: infatti questo protocollo è stato pensato per trasferire messaggi **di piccole dimensioni**: quando si usa un paradigma nel quale le interaizoni sono basate su scambio di messaggi, **è facilmente implementabile un meccanismo di affidabilità** a livello applicativo.

In HTTP si potrebbe facilmente usare UDP per lo strato di trasporto, ma viene usato TCP perchè nato con l'obbiettivo di trasferire contenuti multimediali, e quindi di grandi dimensioni; per questo tipo di contenuti è più comodo operare su un modello a stream che a pacchetti.

Grazie al fatto che a livello di trasporto viene usato UDP la comunicazione può essere sia **unicast** (1 utente - 1 destinatario) che **multicast** (1 mittente - insieme di destinatari).

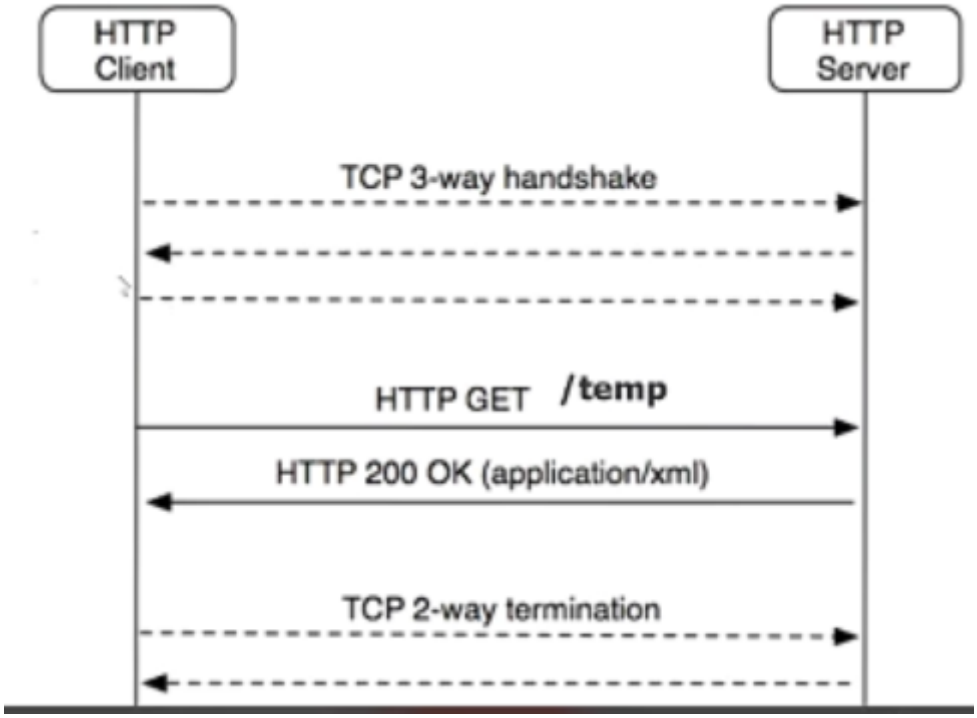
Vediamo un esempio



In questo caso un client vuole interagire con un server per recuperare il valore di un attributo del parametro temperatura, e vediamo la differenza in termini di scambio di pacchetti che è necessario per dare vita a quell'interazione:

HTTP

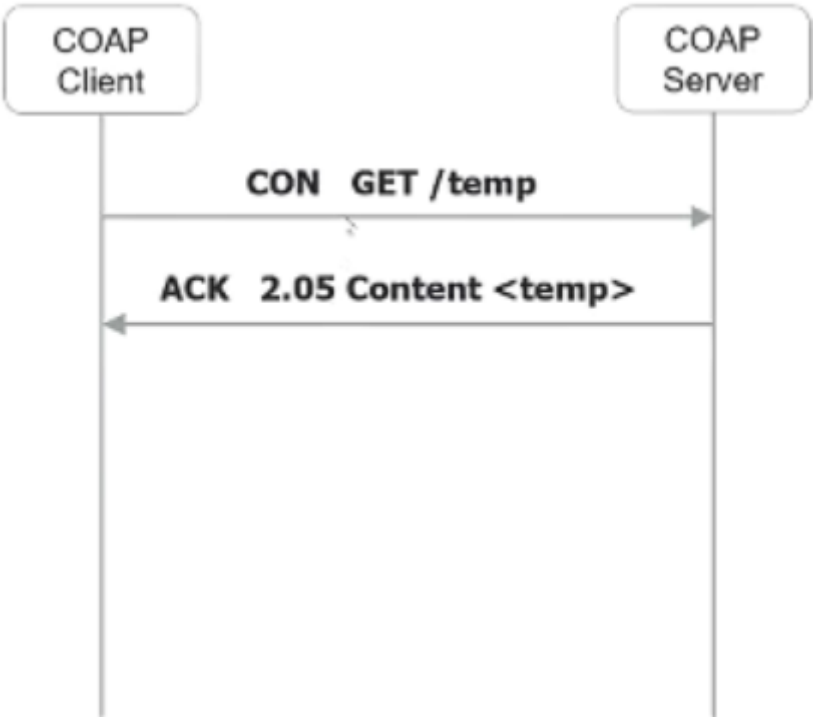
Dobbiamo attivare una connessione, quindi 3-way handshake:



Lo scambio di messaggi HTTP che possono essere veicolati con uno o più segmenti TCP, ed infine abbiamo la chiusura con un 4-way handshake.

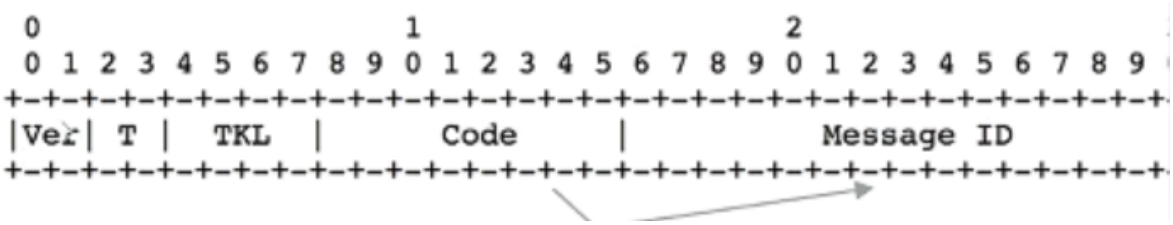
COAP

In questo caso lo scambio è di gran lunga più leggero:



Messaging CoAP

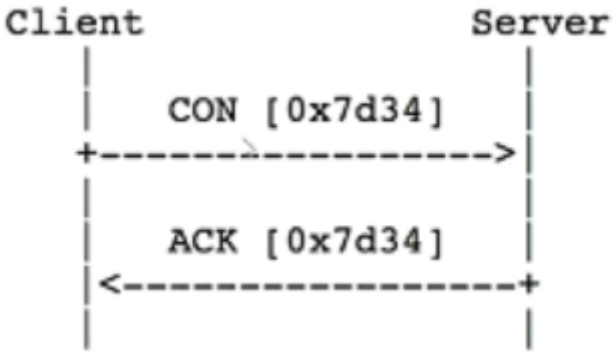
Il protocollo CoAP è basato su un'intestazione abbastanza leggera composta da 4 byte:



In questa intestazione abbiamo **2 bit per la versione del protocollo**, **2 bit per il tipo di pacchetto scambiato** (datagram UDP), **4 bit per un token**, **8 bit per un codice**, ed il resto dei bit destinati al **message ID**.

Messaggi CON - Affidabili

La comunicazione tra client e server può essere realizzata sia in modalità affidabile che non affidabile; i messaggi scambiati per un'interazione affidabile prevedono l'uso del **tipo di messaggio CON**, che è un codice identificativo usato nel campo **T** (vedi immagine). Quando viene realizzato questo tipo di messaggio si chiede al server di inviare un riscontro dell'avvenuta ricezione.



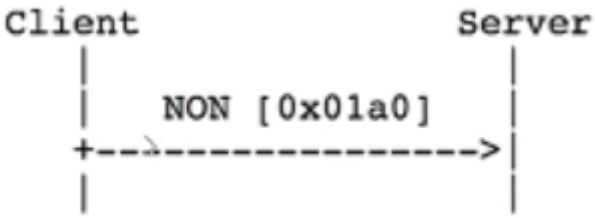
> Esempio di interazione CoAP con CON.

Messaggio ACK

Anche il messaggio di risposta del server ACK è categorizzato come messaggio.

Messaggi NON - non affidabili

Nel caso in cui non si voglia l'affidabilità, il messaggio è detto NON ed ha un codice diverso nel campo **T**. In questo caso non c'è nessun riscontro e la comunicazione è estremamente leggera.



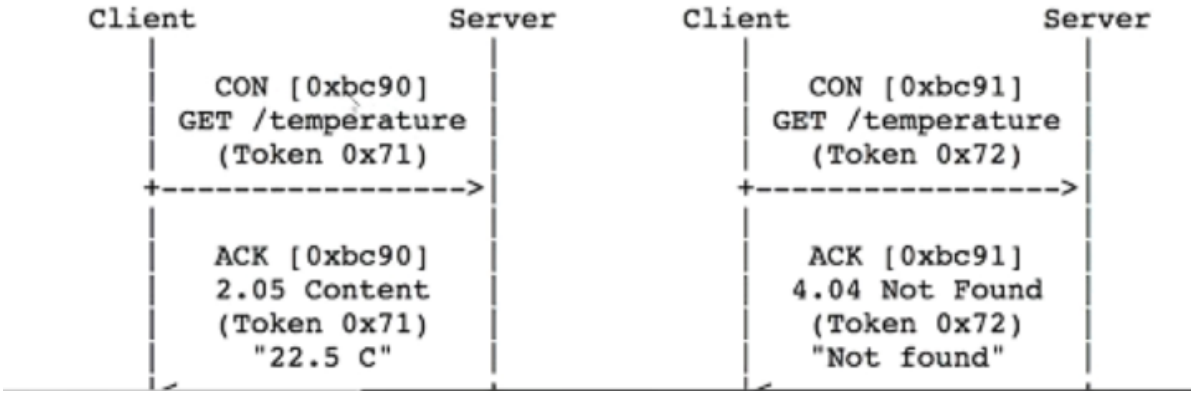
Messaggio RESET

Possiamo rappresentare questo messaggio con 2 bit.

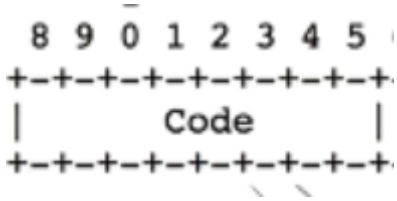
Comunicazione affidabile - Piggybacking

In riferimento alla comunicazione affidabile un messaggio di tipo CON inviato dal client può essere usato per veicolare anche una richiesta; da un lato abbiamo un messaggio applicativo inviato dal client verso il server, dall'altro vogliamo che questo messaggio contenga una richiesta per un contenuto informativo.

La richiesta nello specifico è di recuperare il valore della temperatura di un sensore, ed il message id è il valore associato al messaggio CON.



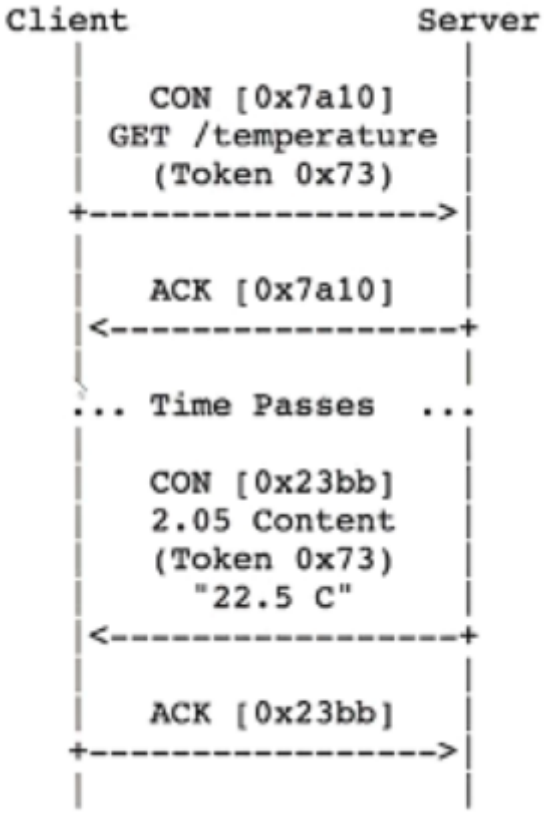
Il server risponde con un messaggio contenente l'informazione e con il valore della temperatura richiesto dal client; questo messaggio è anche un riscontro (ACK); quindi usiamo la tecnica di **piggybacking** dove sovrapponiamo il contenuto informativo alle informazioni di controllo.
Il codice che viene usato è **2.05** che è l'equivalente del codice **200 HTTP**. I codici infatti, sono strutturati nel seguente modo:



Gli 8 bit usati per rappresentare un codice sono suddivisi in due parti: i primi 3 bit (più significativi) sono usati per specificare **la classe del codice** (uguale a quelli HTTP) e gli altri 5 bit rappresentano il vero e proprio valore della classe. Ad esempio rappresentiamo `2.01` dove **2 è rappresentato con 3 bit** e **01 è rappresentato con 5 bit**.

Il token

Per capire il token possiamo guardare questo esempio:



CoAP a differenza di HTTP consente di recuperare una risorsa in modo differito rispetto al momento in cui la richiesta arriva al server; nel senso che è possibile che **arrivi una richiesta per una risorsa che non è ancora stata prodotta**, e la soluzione HTTP sarebbe quella di inviare un codice 404, ma in questo caso il server sa che si tratta di una risorsa che **deve ancora essere prodotta** (Ad esempio da un sensore).

Inizialmente abbiamo un'interazione con un messaggio di tipo CON, che prevede l'invio al server la richiesta della risorsa; questo messaggio viene riscontrato con un ACK che a differenza dei casi precedenti non usa la tecnica del **piggybacking** e non veicola contenuti (è un ACK puro).

La richiesta inviata dal client **presenta un token** (in questo caso 73); quando il sample sarà disponibile, il server potrà inviare al client un sample richiesto in precedenza con uno scambio di messaggi di tipo CON. Ci sarà un messaggio CON per **veicolare il contenuto** ed un riscontro del messaggio CON.

Il token viene quindi usato per **accoppiare una risposta con una richiesta**, visto che la risposta potrebbe non avvenire immediatamente dopo la richiesta.

Protocollo MQTT - Message Queuing Telemetry Transport

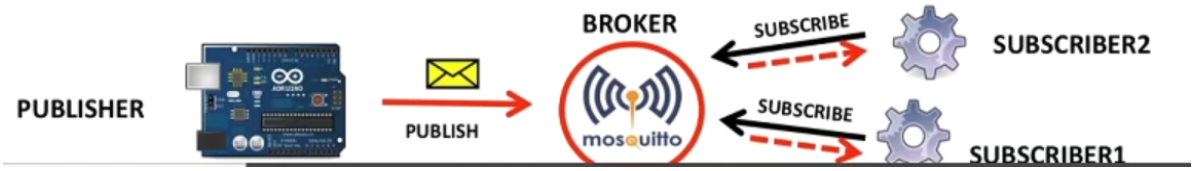
Si tratta di un protocollo completamente diverso rispetto ad HTTP e CoAP, è definito **publish/subscribe**. Nasce per consentire l'interconnessione dei dispositivi a basso consumo energetico che hanno poche energie e devono prevedere un basso contenuto energetico. Come nel caso di CoAP il protocollo MQTT consente la comunicazione 1-1 o 1-n.

E' pensato per trasferire messaggi di piccole dimensioni anche se i messaggi (i segmenti!) possono avere una dimensione fino a **256MB**.

A differenza di CoAP impiega a livello di trasporto il protocollo TCP, anche se è prevista una variante **MQTT-SN** dove viene suato UDP.

Architettura MQTT

Abbiamo parlato di **publish/subscribe**: si prevede una particolare architettura dove sono previste delle componenti con dei ruoli ben precisi.



In questa architettura alcuni componenti (produttrici) vengono chiamate **publisher** ed useranno come operazione di pubblicazione l'operazione **publish**, mentre altre componenti interessate a ricevere le informazioni sono detti **subscriber** ed utilizzeranno come operazione per effettuare la sottoscrizione finalizzata alla ricezione dei dati di interesse, utilizzeranno **subscribe**.

Tra chi produce i dati e chi li consuma, vi è un componente intermedio chiamato **broker**, che ha il compito di disaccoppiare lungo il canale di comunicazione i publisher dai subscribers. Un esempio di broker è **mosquitto**.

Il compito del broker è di ricevere dei dati pubblicati dai publisher (ad esempio dispositivi aventi sensori) e questi dati verranno **temporaneamente memorizzati sul broker** utilizzando un supporto di memoria, e i dati, una volta che il broker sa chi è interessato a riceverli, saranno propagati verso i sottoscrittori.

La sottoscrizione non è finalizzata alla ricezione dei dati, ma è un'operazione finalizzata ad informare il broker dell'esistenza di una componente interessata a consumare un certo tipo di dato perchè quando il dato sarà disponibile, il broker invii il dato verso la componente interessata.

Come si comunica?

Sia publish che subscribe richiedono l'utilizzo di una connessione. Perchè il subscriber possa ricevere dei dati **deve innanzitutto realizzare una connessione con il broker TCP**, deve poi inviare un comando che annuncia al broker il suo interesse a ricevere messaggi.

Il publisher deve anch'esso realizzare una connessione con il broker e poi invierà uno specifico tipo di messaggio MQTT per inviare il messaggio al broker.

Modello di sottoscrizione MQTT

La comunicazione avviene grazie alla sottoscrizione su un topic, ovvero su di un canale specifico. Il broker può gestire diversi canali di comunicazione che vengono definiti topic. Quando viene realizzata la sottoscrizione il subscriber specifica il topic di interesse.

🏁 00:30

I topic sono caratterizzati da **nomi** per poterli caratterizzare e possono essere organizzati in modo gerarchico. Quando sono organizzati in modo gerarchico si usa lo slash per separare le diverse etichette che caratterizzano la gerarchia del nome.

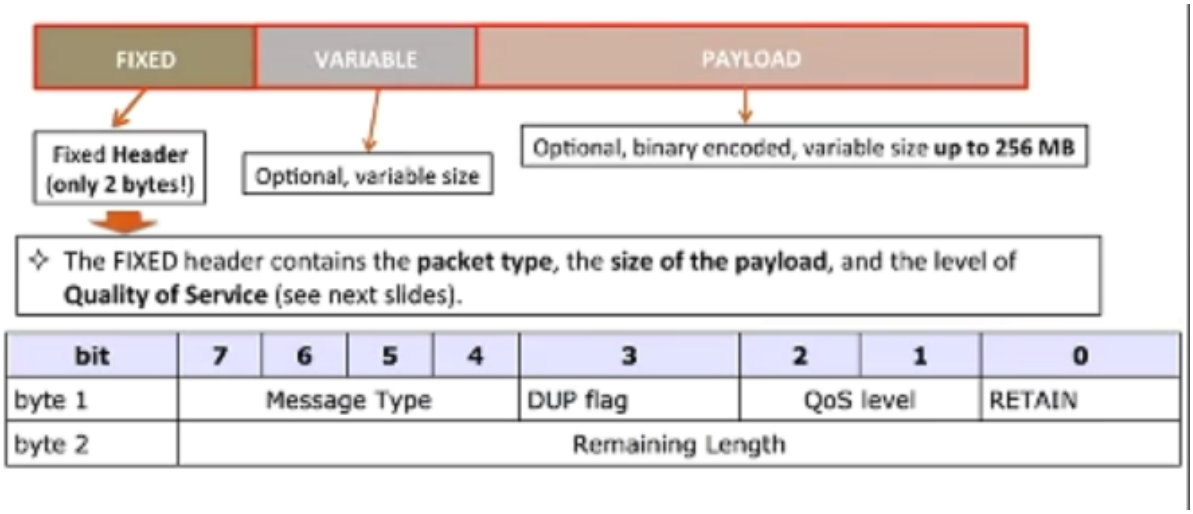
Questo consente di effettuare sottoscrizioni per un certo numero di topic, perchè in fase di sottoscrizione è possibile indicare l'interesse a ricevere dati che sono pubblicati su topic affini.

Utilizzo delle wildcard

Possiamo usare il simbolo "+" per indicare una qualsiasi etichetta in un percorso che **identifica un topic**: `sensor/+/temp` per indicare che il consumatore è intenzionato a ricevere i valori di temperatura associati ad **un qualsiasi sensore**. Al posto del "+" verrà sostituito l'id di un qualsiasi sensore.

Possiamo usare il simbolo "#" per indicare **un sottoalbero**: `sensor/#` chi si sottoscrive è interessato a ricevere dati da tutti i sensori, e non solo quelli di temperatura come nel caso precedente.

Struttura dei messaggi - formato



bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Anche in questo caso l'intestazione dei messaggi è molto leggera, addirittura piàleggera di CoAP.

Nella parte inferiore vediamo l'espansione dell'header ed in questi due byte è presente il **message type**(7-4); Un bit viene usato per indicare se il messaggio è **duplicato**(3); due bit sono usati per specificare il **livello di qualità del servizio** ed un bit per indicare se vogliamo che il messaggio venga mantenuto nel browser.

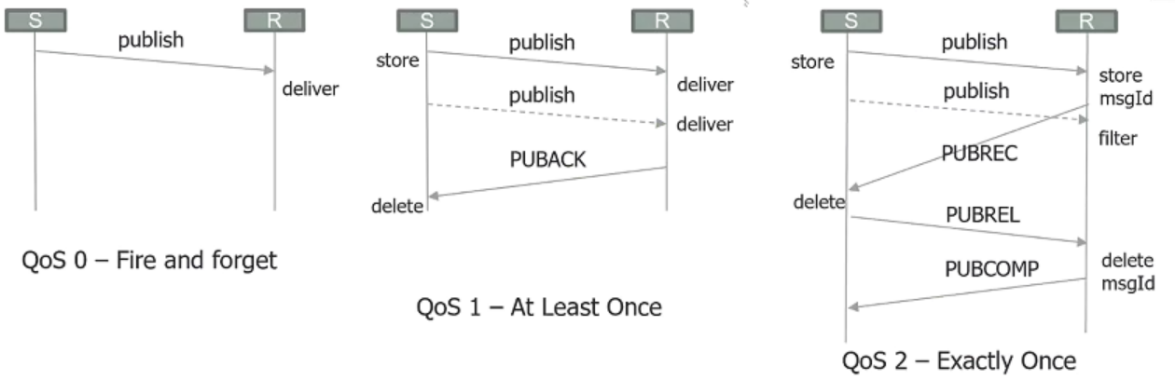
Qualità del servizio - QoS

Anche se usiamo il TCP i messaggi potrebbero ancora essere persi per strada. QoS viene usato per definire l'affidabilità del servizio. Perchè introduciamo dei meccanismi a livello applicativo se è già presente la protezione del TCP?

Vogliamo garantire l'affidabilità end to end tra publisher e subscriber ma vogliamo anche tenere in considerazione che le entità comunicante potrebbero non essere tutte attive al momento della pubblicazione di un dato.

Livelli di QoS

- **0**: **fire & forget** non prevede nessun meccanismo a livello applicativo ma si basa su quella messa a disposizione dal protocollo sottostante
- **1 deliver at least once** è possibile che in caso di malfunzionamento una delle entità comunicanti riceva il messaggio HTTP più di una volta (**è possibile si presentino duplicati**)
- **2 deliver exactly once** i duplicati non sono presenti



Affidabilità di MQTT

Abbiamo la possibilit di gestire i **RETAIN message**, mettendo ad 1 il bit **RETAIN** nell'header di un messaggio. In questo caso si dice al broker di mantenre il messaggio pubblicato su quel topic per spedirlo al sottoscrittore ogni volta che questo si connetterà al broker.

In questo modo, il sottoscrittore che ha perso la connesisone con il broker può continuare a ricevere i messaggi precedenti.

Sicurezza di MQTT

MQTT prevede diversi meccanismi, anche se si affida molto a protocolli esterni per quanto riguarda la sicurezza. Infatti si affida, per la cifratura del canale, totalmente a soluzioni esterne.

Per quanto riguarda **l'autenticazione** si basa o sull'utilizzo **di un id che caratterizza in maniera univoca il client**, oppure usando **username e password**. E' possibile anche prevedere un **certificato**.

Esercitazione

Installiamo **mosquitto** su MacOs con `brew install mosquitto`; non sono riuscito a farlo funzionare su windows (come al solito).

Brew ci restituisce il seguente messaggio dopo l'installazione di mosquitto:

```
To restart mosquitto after an upgrade:
  brew services restart mosquitto
Or, if you don't want/need a background service you can just run:
  /usr/local/opt/mosquitto/sbin/mosquitto -c
/usr/local/etc/mosquitto/mosquitto.conf
```