**Differenze protocolli per l’IoT, Internet protocol suite**

Il classico stack di Internet è pensato per dispositivi “pieni di risorse”.

Invece per le reti IoT ci sono delle esigenze specifiche, le principali:

* Scalabilità (numero di disp.i in una rete) e ridondanza (legata al trade off tra affidabilità e basso costo, ambiente esterno imprevedibile).
* Sicurezza che “tenga conto” dell’eterogeneità dei disp.i.
* Avere protocolli semplici e con poco overhead, a causa delle risorse limitate in ambito memoria/computazione/connessione.
* Occhio di riguardo al risparmio energetico (ridurre duty cycle).

**MQTT, peculiarità in tal senso**

* Non molto vantaggioso dal punto di vista della sicurezza: si può usare sulla porta 8883 sopra SSL, ma in questo modo si introduce un overhead notevole e spesso i dispositivi non possono gestire tale overhead.
* Soddisfa molto bene i requisiti legati alle risorse limitate (memoria, computazione, connessione), anche se è costruito sopra TCP.
* Migliore scalabilità rispetto ad HTTP (se si parallelizzano le operazioni compiute dal broker).

**Perché soddisfa molto bene i requisiti legati alle risorse limitate?**

* Footprint del codice piccolo.
* Overhead dei pacchetti minimale.
* Piccola larghezza di banda della rete.

**Che tipo di protocollo è, su quale protocollo del livello di trasporto è costruito, architettura interna e paradigma che implementa**

Tipo applicativo, TCP, client/server, publish/subscribe

**Cosa fa sì che il footprint del codice nei disp.i IoT sia minimo?**

Lato server complesso, lato client semplice.

**“Attori” paradigma publish/subscribe**

* Publishers (clients)
* Subscribers (clients)
* Event service AKA broker (server)

**Stabilire la connessione**

I clients devono stabilire preliminarmente una connessione con il broker mandando un messaggio di CONNECT, che contiene il Client ID e altri campi opzionali (CleanSession, Will flags, KeepAlive).

Il broker invia quindi un CONNECTACK, in cui comunica l’esito della connessione e se è già presente una sessione interrotta.

**Publishers**

Producono eventi (dati potenzialmente di qualsiasi genere).

**Subscribers**

Esprimono interesse per un evento o per gli eventi che presentano specifiche caratteristiche (vedere filtri del broker). Ricevono una notifica dal broker quando l’evento di interesse è generato da un publisher.

Esprimere interesse = effettuare una richiesta di subscription

Togliere interesse = effettuare una richiesta di unsubscription

**Broker e relativi filtri**

Può essere solo uno strato software accessibile da p. e s., ma tipicamente è un agente indipendente anche dal punto di vista HW (es. server installato su un computer a parte).

Per sapere a quali broker inoltrare gli eventi, applica filtri basati (a seconda delle subscriptions dei sub.) su:

* Subject topic: stringa organizzata in una tassonomia (es. home/firstfloor/+/presence, home/firstfloor/#, $SYS/broker/clients/connected)
* Contenuto: query
* Tipo: riguardano contenuto e struttura nel senso dell’O.O.

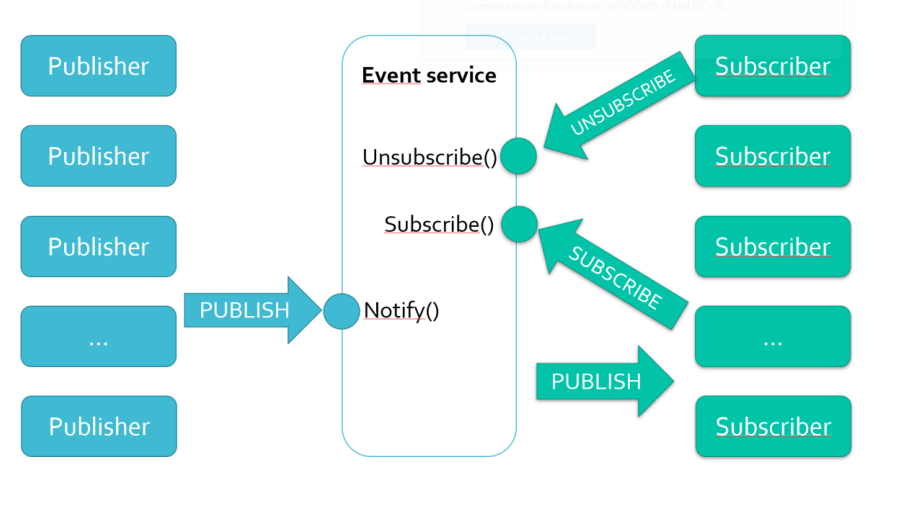
**Tipo di relazione tra publishers e subscribers e in che ambiti si configura questa relazione**

Essi non si conoscono a vicenda e non c’è nessuna connessione diretta tra i due. Le uniche cose che hanno in comune sono il fatto di comunicare con lo stesso broker e di utilizzare gli stessi topic e relativa tassonomia.

Si dice che sono completamente disaccoppiati (fully decoupled):

* Nello spazio (nessuna risorsa scambiata direttamente).
* Nel tempo (non devono essere accesi necessariamente nello stesso momento).
* Nella sincronizzazione (no sincron. nello scambio di informazioni)

**DOMANDA 39**

****

Si assume che tutti i client siano già connessi all’event service. I publishers pubblicano dati sull’event service (broker) accedendo all’interfaccia Notify() dello stesso. Allo stesso tempo i subscribers inviano messaggi di subscribe e unsubscribe. Basandosi su questa informazione l’event service filtra i messaggi pubblicati e li inoltra (pubblica di nuovo) ai subscribers “interessati”.

**Quality of Service (QoS)**

https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

Indica il grado di affidabilità nella consegna dei messaggi. Può essere:

* 0: stesse garanzie del TCP (finché la connessione c’è, c’è garanzia). Nessun PUBACK; il messaggio è immediatamente inviato e non viene memorizzato dal broker.
* 1: (messaggio consegnato almeno una volta): PUBACK dal broker al publisher o dal subscriber al broker. Ogni messaggio è memorizzato nel broker e consegnato almeno una volta a tutti i subscribers che hanno effettuato una subscribe a quel determinato topic con QoS 1. Se il mittente non riceve il PUBACK entro un certo tempo, invia di nuovo il messaggio.
* 2: (messaggio consegnato esattamente una volta) Usa un doppio two-way handshake: PUBLISH, PUBREC, PUBREL, PUBCOMP.

Più il QoS è alto, più gli invii sono lenti.

**In quali situazioni conviene utilizzare ognuno dei possibili livelli di QoS?**

* 0 se la connessione è molto stabile oppure tipicamente se non interessa perdere qualche messaggio occasionalmente
* 1 se è importante che ogni messaggio sia consegnato e i subscribers sono in grado di gestire i duplicati
* 2: come l’1, ma se i subscribers non sono in grado di gestire i duplicati

**A cosa servono PUBREC, PUBREL e PUBCOMP?**

PUBREC: Il destinatario comunica che ha ricevuto il messaggio

PUBREL: Il mittente comunica che ha saputo che il messaggio è arrivato e non invierà più duplicati. Il destinatario può quindi cancellare lo stato relativo al messaggio “in sospeso” e l’id può essere riutilizzato.

PUBREC: Il destinatario comunica che ha ricevuto la PUBREC. Il mittente può quindi smettere di inviare PUBREL relative a quel determinato messaggio.

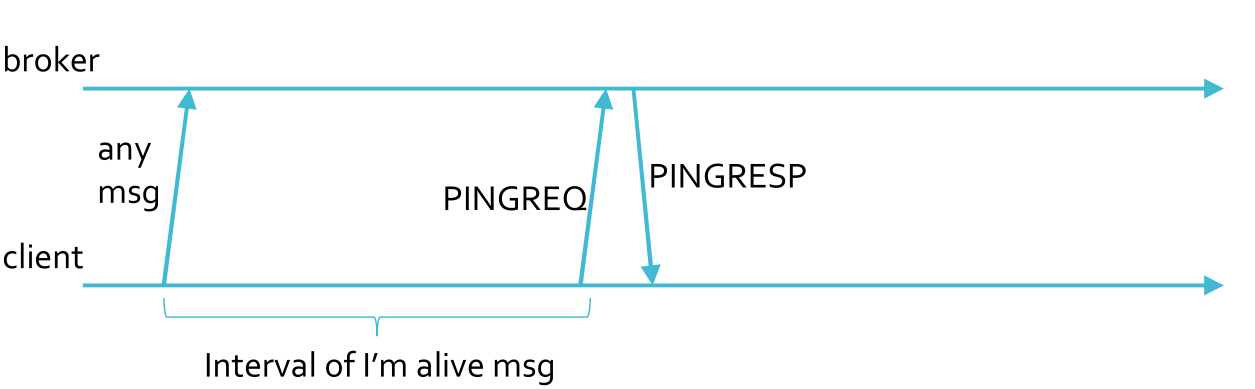
**Il broker può memorizzare messaggi per i subscribers off-line?**

Sì se:

* La subscription al topic è già stata effettuata.
* Subscribers connessi in maniera persistente (flag CleanSession).

**Campi del Last Will e KeepAlive (campi del messaggio CONNECT)**

Tipicamente utilizzati da un publisher per rendere possibile informare i subscribers iscritti ai suoi topic in caso di disconnessione imprevista.



**Principali broker MQTT**

* Mosquitto
* Mosca
* HiveMQ

**Quali sono i principali limiti di MQTT?**

* Non c’è comunicazione point to point
* Broker punto cruciale in caso di guasti
* TCP: alto costo in termini di risorse rispetto a UDP

**Come risolve tali problemi CoAP?**

* Possibile comunicazione point to point e connettersi direttamente al cloud utilizzando un protocollo Internet standard
* Basato su una “classica” architettura client-server. Di solito i client sono le applicazioni e i server i sensori e attuatori
* Costruito su UDP

Per fare ciò, adotta un modello di comunicazione simile a quello di http, ma ha molti vantaggi relativamente alla gestione di risorse limitate.

A livello di sicurezza è molto valido, equivalente all’RSA a 3072 bit.