



LABORATORIO DI SISTEMI EMBEDDED E IOT

Modulo 1

Laurea in Tecnologie dei sistemi informatici

Università di Bologna - Sede di Imola

Lorenzo Pellegrini, Ph.D.

l.pellegrini@unibo.it - miatbiolab.csr.unibo.it

Protocolli per l'IoT

Middleware

Middleware

Abbiamo visti diversi protocolli con cui un dispositivo può connettersi a internet. Purtroppo nel mondo IoT è presente una forte frammentazione per cui non tutti i sistemi usano la medesima modalità per ottenere la connettività.

Un modo con cui è possibile standardizzare la connettività tra le varie componenti di un sistema IoT è quello di definire una struttura di riferimento basata su un **Middleware**.

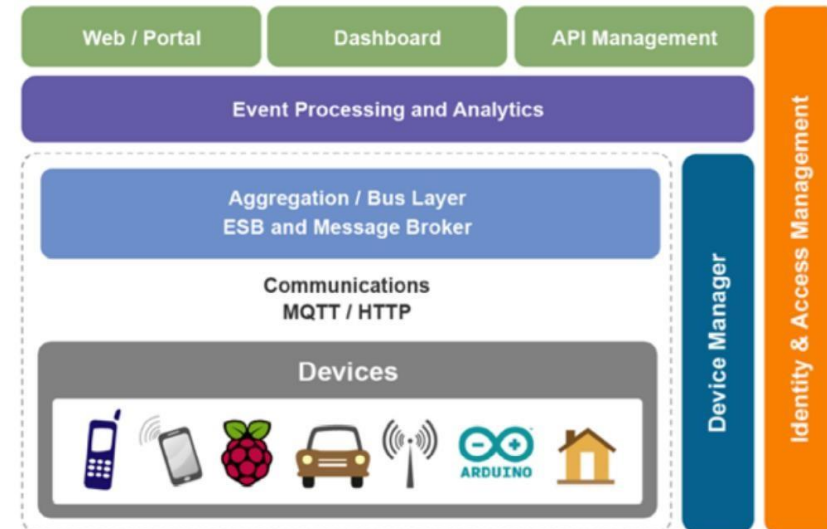
Middleware: caratteristiche

I middleware rappresentano una soluzione in grado di coprire:

- ❑ Connettività e comunicazione: ovvero fornire interoperabilità tramite l'uso di protocolli di alto livello
- ❑ Device management: discovery dinamico, aggiornamento, disconnettere dispositivi problematici
- ❑ Data collection: memorizzazione, analisi, elaborazione
- ❑ Scalabilità
- ❑ Sicurezza

Middleware: layers

- ❑ Web / Dashboard / API
- ❑ Event processing
- ❑ Aggregation: aggrega e gestisce le comunicazione (fa da ponte tra i diversi protocolli)
- ❑ Communication: diversi protocolli, tra cui HTTP, MQTT, CoAP, XMPP
- ❑ Devices: i dispositivi, identificato a livello hardware, che usano diversi protocolli



Modello a scambio di messaggi

Alla base di tutti i sistemi embedded vi è la comunicazione, elemento che richiede modelli e meccanismi ad alto livello oltre ai relativi supporti tecnologici.

Quali tecnologie usare? Quale modello di comunicazione?

Uno dei modelli più utilizzati nell'ambito IoT è quello a **scambio di messaggi**.

Nota: il modello a scambio di messaggi risulta essere un'ottima soluzione anche per diverse applicazioni non-IoT!

Scambio di messaggi

La comunicazione avviene tramite invio e ricezione di messaggi. Questo semplifica la definizione della trasmissione di informazioni in quando la comunicazione si basa semplicemente su operazioni di *send* e *receive*.

Lo scambio di messaggi può avvenire:

- ☐ In modo diretto o indiretto
- ☐ Con trasmissione sincrona o asincrona

Vedremo sistemi per lo scambio dei messaggi più popolari nel mondo IoT, ovvero MQTT, XMPP e CoAP. Tuttavia questi differiscono per il modello alla base: publish/subscribe o request/response.

Scambio di messaggi: le possibili scelte

Diretta

VS

Indiretta

La comunicazione avviene direttamente fra i processi, specificandone un identificativo che ovviamente deve essere univoco.

In questo caso si sfruttano dei canali di comunicazione quindi la comunicazione avviene specificando un determinato canale.

Sincrona

VS

Asincrona

L'invio (send) è andato a buon fine nel momento in cui il messaggio è stato recepito dal destinatario tramite una receive.

L'invio (send) è andato a buon fine nel momento in cui il messaggio viene inviato.

Modello publish-subscribe

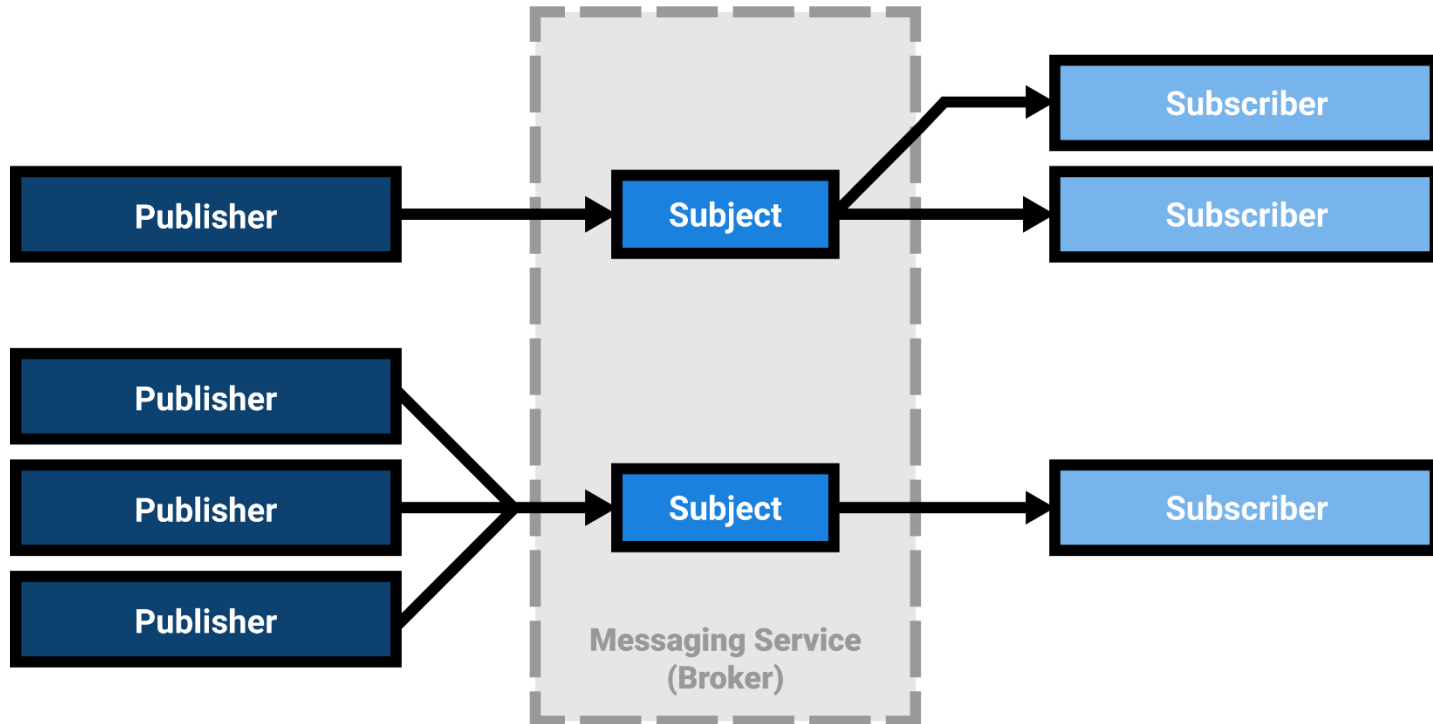
Modello molto utilizzato per le comunicazioni ad eventi.

Possiamo individuare tre elementi principali:

- ❑ **Topics:** ovvero i canali a cui è possibile inviare i messaggi
- ❑ **Publisher:** dispositivi (o in generale entità software) che possono inviare messaggi sui topics
- ❑ **Subscriber:** si registrano ai topics per ricevere i messaggi in arrivo

I sistemi a scambio di messaggi implementando solitamente il topic come coda di messaggi, ovvero un buffer a cui è possibile accodare messaggi e riceverli in ordine di arrivo.

Publish-subscribe



Request-response

Un'alternativa al modello publish-subscribe è dato dal modello request-response. Questo modello è tipico dei protocolli basati su tecnologie web (HTTP).

In questo modello i dispositivi scambiano dati tra di loro (o con un main repository) per mezzo di richieste e relative risposte ad esse. Questo implica l'assenza di una coda di messaggi e la comunicazione è solitamente stateless (come nell'approccio REST).

Vedremo come esempio per il mondo IoT il protocollo CoAP.

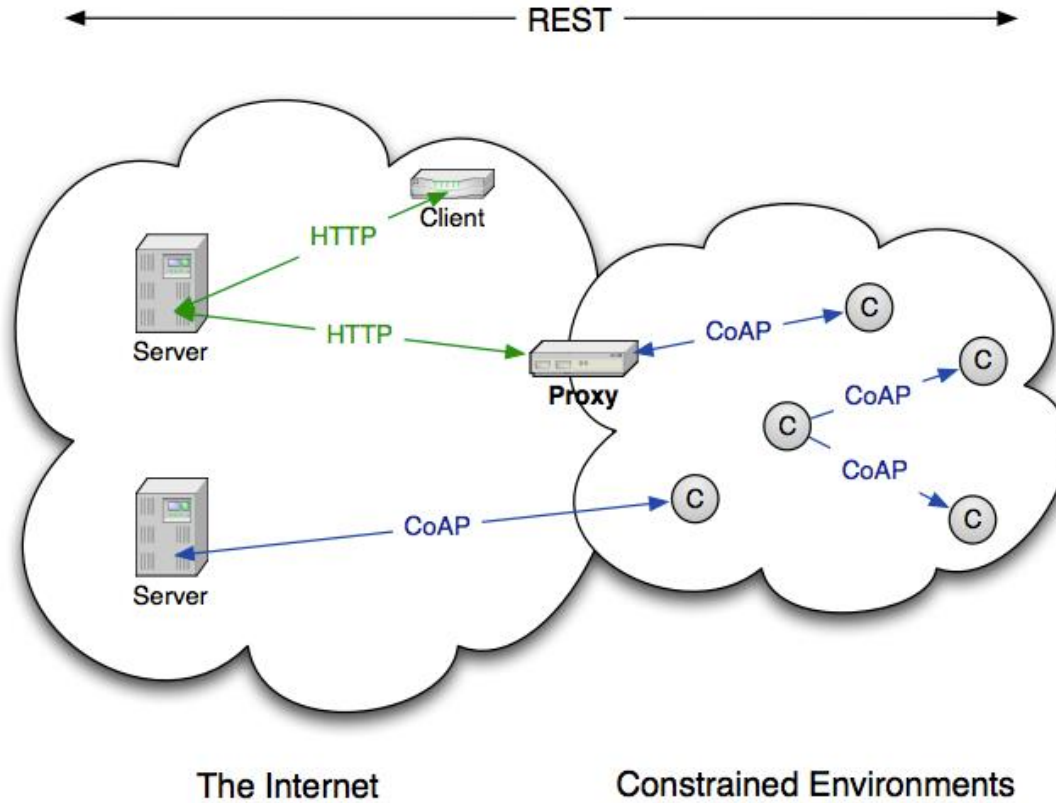
CoAP

Request-response basato su HTTP e REST

CoAP

Constrained Application Protocol (CoAP): protocollo specializzato per lo scambio di messaggi. Particolarmente indicato per la comunicazione in reti con forti vincoli. Pensato per funzionare su sistemi embedded con poca memoria (anche solo 10KB).

- ❑ Meccanismo request-response
- ❑ Concetti principali presi da protocolli web: URI, content-type, ...
- ❑ Si tratta di una versione efficiente dell'HTTP per l'IoT
- ❑ Usa UDP anzichè TCP
- ❑ Interazione esplicitamente REST
- ❑ Layer di observability (da non confondere con il una coda di messaggi!)



RESTful

REpresentational State Transfer (REST): un insieme di principi che un qualsiasi sistema distribuito può adottare.

"REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems"

(Architectural Styles and the Design of Network-based Software Architectures, Roy Fielding)

Sistemi RESTful: principali vincoli

1. **Client/server:** le componenti hanno un chiaro ruolo. Il client ha l'iniziativa: il client invia richieste al server, il server risponde e l'interazione finisce lì (statelessness, terzo punto). Possibilità di intermediari (proxy) per girare le comunicazioni.
2. **API uniforme:** interfaccia uniforme e indipendente dall'applicazione. In pratica il modello di interazione è basato sullo scambio di rappresentazioni e sulla modifica locale dello "stato" di una risorsa. Limitato e ben definito insieme di primitive per ottenere, impostare/aggiornare e cancellare uno stato/risorsa.
3. **Stateless:** le interazioni dovrebbero essere indipendenti dai messaggi precedenti. Ogni messaggio dovrebbe contenere tutte le informazioni necessarie per gestire una data richiesta.

Sistemi RESTful: principali vincoli

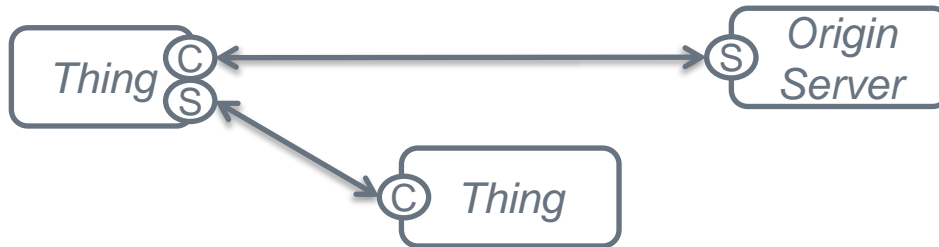
4. **Cacheable:** le risposte dovrebbero avere un sistema di cache-control. Questo permette a eventuali intermediari di memorizzare le risposte per un tempo definito in modo da poter rispondere più velocemente a richieste per la stessa risorsa.
5. **Layered System:** i client non devono aver necessità di "vedere" dietro al server con cui stanno interagendo. Vincolo in qualche modo legato al concetto di encapsulation. Questo rende più semplice la modifica e manutenzione. Ad esempio, al client solitamente non interessa sapere se sta parlando direttamente con il server target o con un intermediario.

Client/server e IoT: nota

Nei sistemi web gli intermediari sono solitamente limitati a fare da proxy, reverse proxy e gateway.



Nell'IoT le varie Thing possono fungere contemporaneamente da intermediario, client e server (il ruolo varia in base alla funzionalità che si sta implementando).



URI

- ❑ Ogni risorsa ha un identificatore univoco indirizzabile utilizzando un meccanismo di referenziazione univoco
- ❑ Solitamente le risorse sono identificate da un Uniform Resource Identifier (URI), ovvero uno schema standard
- ❑ L'URI è una sequenza di caratteri che identifica univocamente e in maniera non ambigua una certa risorsa (astratta o fisica)
- ❑ Gli URI sono utilizzati per indicare una risorsa con cui è possibile interagire, per referenziare una risorsa da un'altra risorsa, per indicizzarla, ...

Sono un meccanismo standard nel mondo web (e non solo).

Addressable resources

Un URI è composto di 5 parti:

1. Scheme: indica un namespace di risorse e definisce come le parti successive identificano la risorsa all'interno del namespace
 - Esempio: http(s), ftp, file, ...
2. Authority: nome di dominio o indirizzo ip (+ eventuale porta) è l'entità che governa quella parte del namespace
 - Esempio: www.google.com
3. Path: il percorso della risorsa
4. Query parameters (opzionale): path+query identificano la risorsa
5. Fragment identifier: permettono di identificare una parte della risorsa

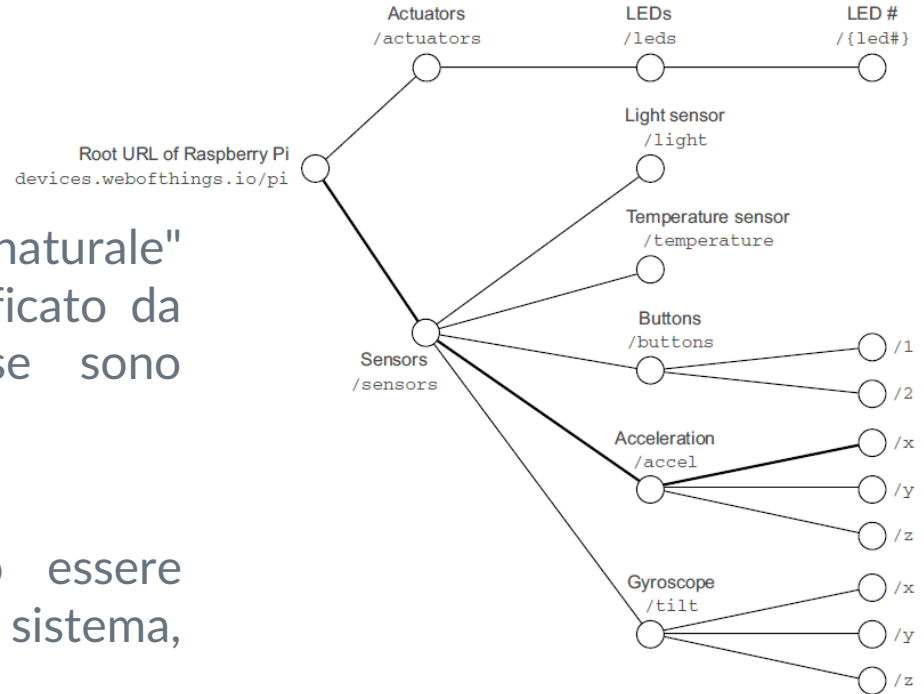
URI: esempio

`http://devices.webofthings.io/pi/sensors/light`

Scheme **Authority** **Path**

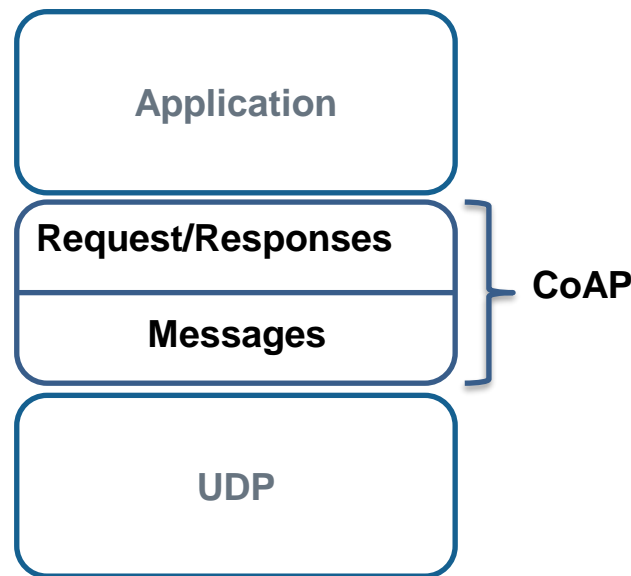
Nel mondo dell'IoT (e in maniera "naturale" in CoAP) ogni dispositivo è identificato da un url root e le varie risorse sono organizzate in maniera gerarchica.

Ad esempio le risorse possono essere sensori, attuatori, proprietà del sistema, elementi della configurazione, ...



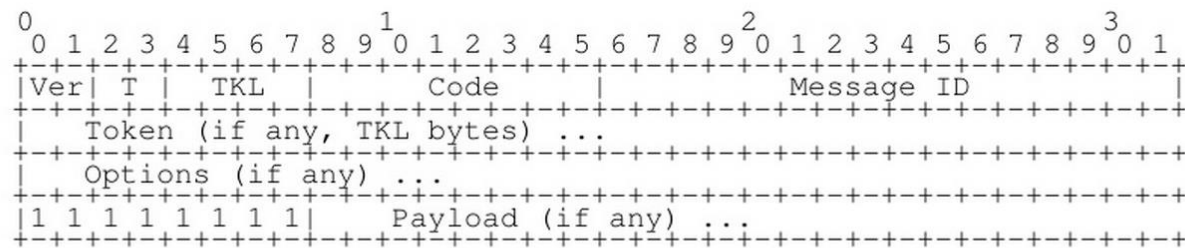
CoAP: interazione

- ❑ Quattro tipi di messaggio
 - Confirmable (CON)
 - Non-confirmable (NON)
 - Acknowledgment (ACK)
 - Reset (RST)
- ❑ Nota: le tipologie di messaggio ortogonali all'interazione request-response:
 - Le richieste possono essere sia in messaggi Confirmable e Non-confirmable
 - Le risposte possono essere trasportate sia in questi tipi di messaggio, ma anche aggiunte sopra (piggyback) nel messaggi di Acknowledgment



CoAP: formato

CoAP è una versione dell'HTTP pensato per l'IoT. Utilizza UDP anziché TCP (quindi connectionless) e il formato dei messaggi è compresso:



Ver - Version (1)

T - Message Type (Confirmable, Non-Confirmable, Acknowledgement, Reset)

TKL - Token Length, if any, the number of Token bytes after this header

Code - Request Method (1-10) or Response Code (40-255)

Message ID - 16-bit identifier for matching responses

Token - Optional response matching token

CoAP: affidabilità

I messaggi CON permettono di ottenere affidabilità nella comunicazione:

- ❑ I messaggi CON vengono ri-trasmessi se non è stato possibile inviarli con successo (fino a che non viene ricevuto l'ACK)
- ❑ Se chi riceve non può gestire il messaggio, viene inviato un RST anziché ACK
- ❑ Ai messaggi NON non viene data risposta ACK, ma i messaggi hanno un id univoco per rilevare duplicati
- ❑ Ai messaggi NON può comunque venire data una risposta RST

CoAP e REST

L'elemento fondamentale di CoAP è il suo allineamento ai metodi di interazione tipici delle tecnologie web. In particolar modo, CoAP è stato progettato per un approccio REST.

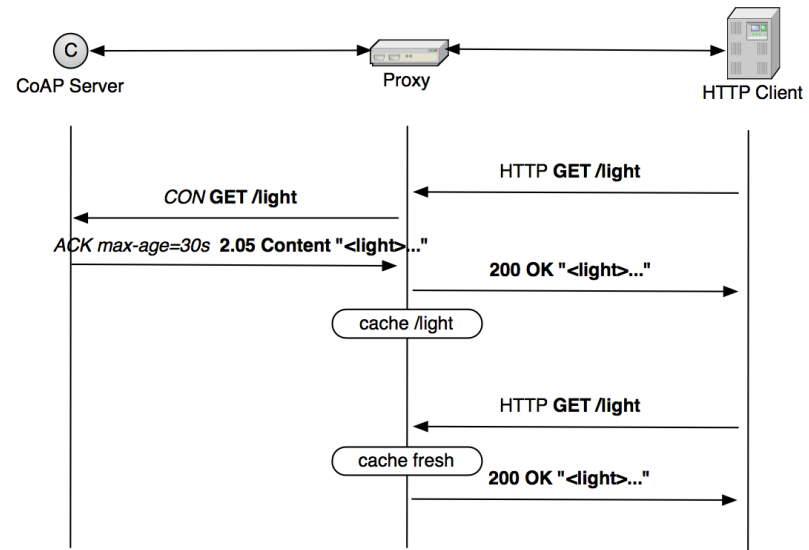
- ❑ Metodi GET, POST, PUT, DELETE
- ❑ In pratica i dispositivi impostano, modificano, cancellano informazioni esattamente come nelle interazioni REST
 - L'informazione su cui avviene la manipolazione è identificata da un URI
- ❑ Ogni risposta consiste in uno status code che identifica il risultato
 - Molto simili agli HTTP status code

La Thing diventa un'entità web (Web of Things) il cui stato è accessibile tramite tecnologie web.

CoAP caching

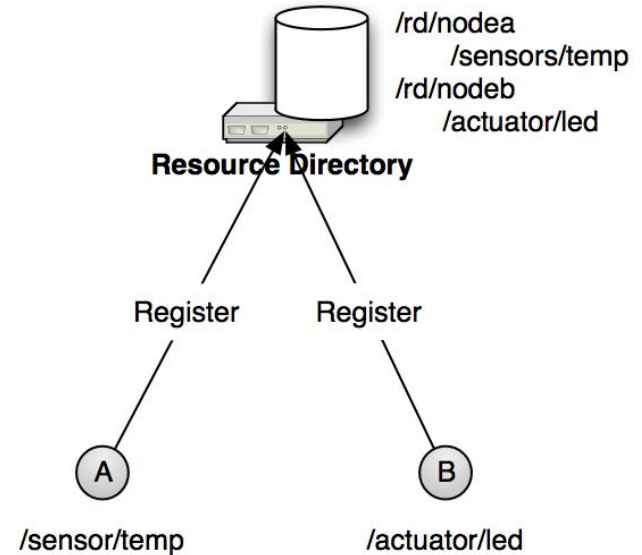
CoAP mette a disposizione anche un sistema di caching, che permette di velocizza l'accesso alle risorse:

- ❑ La possibilità di mettere in cache una risorsa è dato dal response code
- ❑ Max-age: opzione per indicare per quanto tempo tenere in cache una risorsa
- ❑ Solitamente un proxy supporta il caching per rendere più efficiente l'accesso alle risorse per quei dispositivi con risorse limitate



CoAP resource directory

- ❑ In una rete CoAP è comune avere una resource directory che mantiene lo stato delle varie Things
 - Utile anche per il processo di discovery: per trovarsi tra di loro, le Things possono semplicemente chiedere al server una lista di altri dispositivi
- ❑ Ogni dispositivo può registrarsi postando le proprie risorse su `"/.well-known/core"`



Differenze con HTTP

❑ CoAP supporta multicasting

- Comunicazione con tanti dispositivi contemporaneamente
- Discovery dei dispositivi locali
- Possibilità di comunicare attraverso firewall

❑ Estensioni:

- Block transfer algorithm (trasferimento di grandi quantità di dati)
- Subscription and notification (grazie a un layer di Observability)
 - Altri dispositivi possono essere informati se un dispositivo cambia il proprio stato
 - Attenzione: non è una coda di messaggi

❑ Sicurezza: Datagram Transport Layer Security (DTLS)

- Un'implementazione di TLS per UDP
- Impiegato anche in altri settori (esempio: WebRTC)

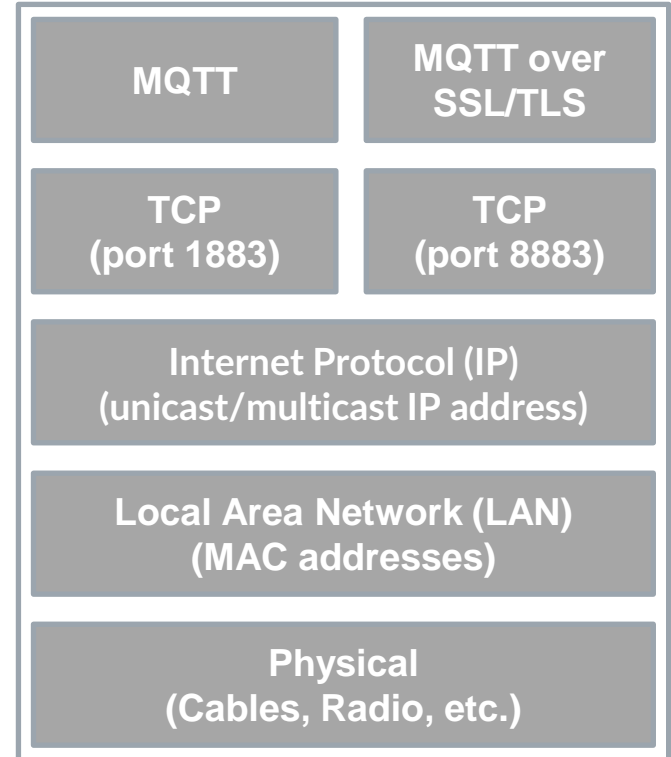
MQTT

Publish-subscribe con broker

MQTT

Message Queue Telemetry Transport (MQTT) è un protocollo publish/subscribe nato per permettere a dispositivi con risorse limitate di comunicare. È stato inventato nel 1999 da Andy Stanford-Clark and Arlen Nipper.

Si basa su TCP/IP. Si basa sulla presenza di un broker dei messaggi per fare da tramite tra i vari client: permette di superare la presenza firewall.



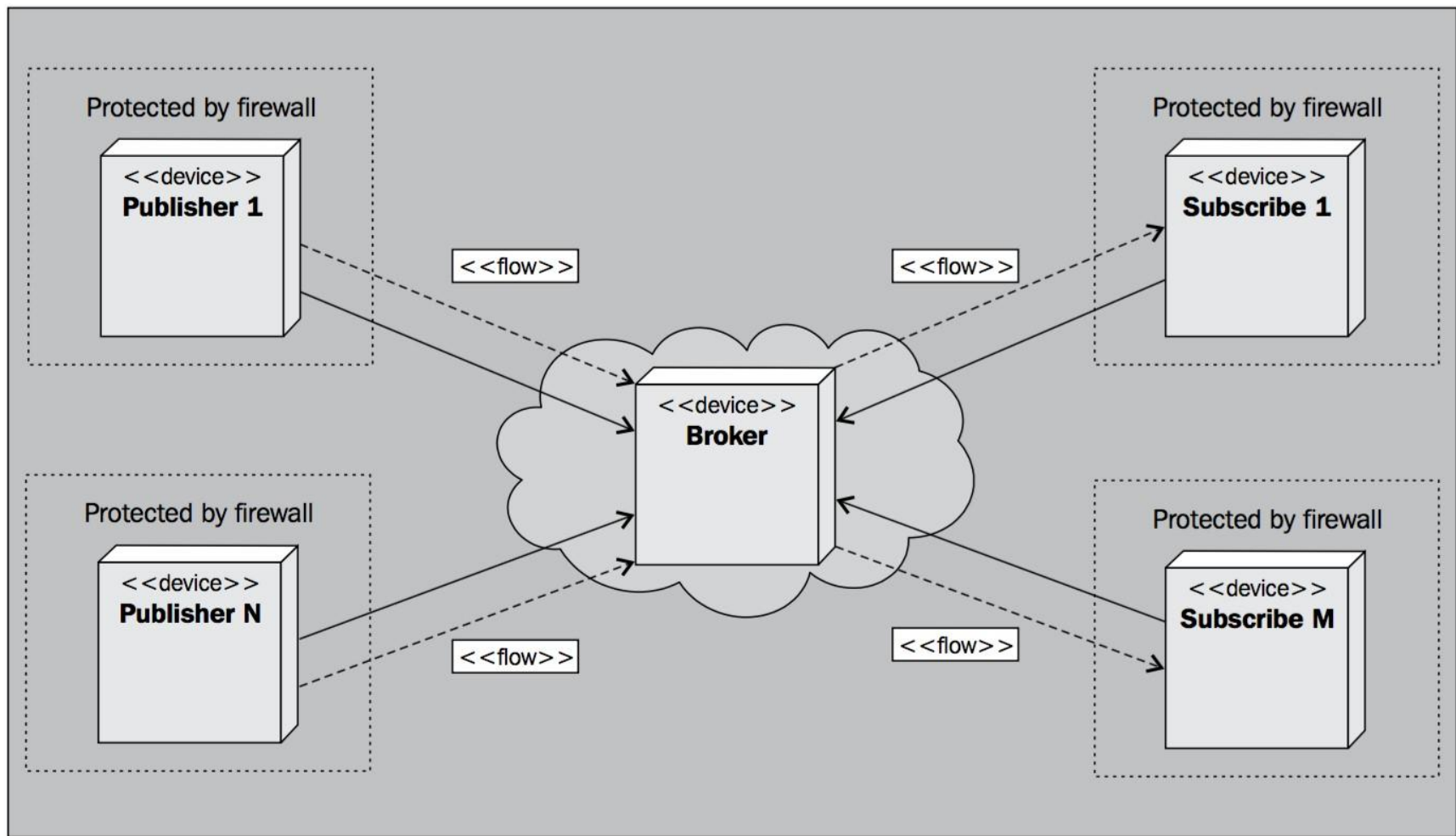
MQTT: caratteristiche

- ❑ Utilizza la porta 1883 TCP, assegnata da IANA
 - Per MQTT-over-SSL si usa la porta 8883
- ❑ Di base non integra un sistema di crittografia
 - Può essere usato SSL, ma questo vanifica la componente "efficienza"
- ❑ Anche i borker possono essere identificati tramite URI:
 - `mqtt[s]://[username][:password]@host.domain[:port]`
 - Username e password solitamente inviati nel pacchetto CONNECT

MQTT: modello

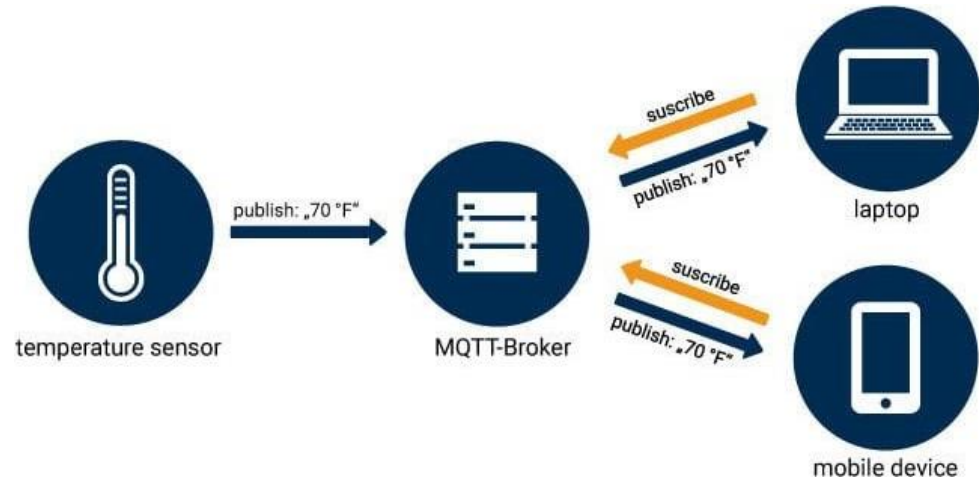
Come abbiamo già accennato, nel modello publish/subscribe la comunicazione ruota intorno a tre elementi: publishers, subscribers, topics. In MQTT i topics sono in gestione a un message broker, che fa da intermediario tra i client e fornisce diverse funzionalità aggiuntive. Quindi:

- ❑ Publishers: si connettono al broker per pubblicare messaggi in uno o più topic. In questo modello i messaggi sono classicamente legati al concetto di evento.
- ❑ Subscribers: si connettono allo stesso broker e sottoscrivono a uno o più topic per ricevere i relativi messaggi.
- ❑ Message broker: gestisce uno o più topics



MQTT e IoT

- ❑ Basso overhead
- ❑ Può funzionare su diversi protocolli (TCP, ZigBee)
- ❑ Funziona in presenza di firewall
- ❑ Possibilità di impostare il QoS



Topics e Topics tree

❑ Il contenuto è identificato dal topic

- Quando viene pubblicato un messaggio, il publisher può scegliere se il contenuto deve essere mantenuto dal server (retained)
- Ogni subscriber riceve, al momento in cui esegue la subscription, l'ultimo messaggio retained nel topic (che quindi è antecedente al momento della subscription). Utile per informare i subscriber dell'ultimo stato postato

❑ I topic sono organizzati in una struttura ad albero

- Slash (/) come delimitatore (come per le cartelle di un PC o sito web)
- I subscriber possono sottoscrivere per ricevere i messaggi di specifici topic o anche di interi alberi
 - Utile per sottoscrivere ai messaggi di tutti (per esempio) sensori di temperatura di una certa stanza

QoS

MQTT mette a disposizione tre livelli di Quality-of-Service (QoS), da scegliere all'atto della pubblicazione di un messaggio viene pubblicato.

- ❑ **Unacknowledged service (0):** messaggio inviato *al più una volta* a ogni subscriber (nessuna conferma di ricezione)
- ❑ **Acknowledged service (1):** ogni subscriber conferma la ricezione del messaggio
 - Il messaggio può essere re-inviato se non arriva una conferma
 - Assicura che l'informazione sia ricevuto *almeno una volta*
- ❑ **Assured service (2):** la ricezione dell'informazione non solo è confermata, ma si adotta un sistema in 2 step: prima è trasmessa e poi consegnata
 - Assicura che l'informazione sia ricevuta *esattamente una volta*

QoS 0 : At most once (fire and forget)



QoS 1 : At least once

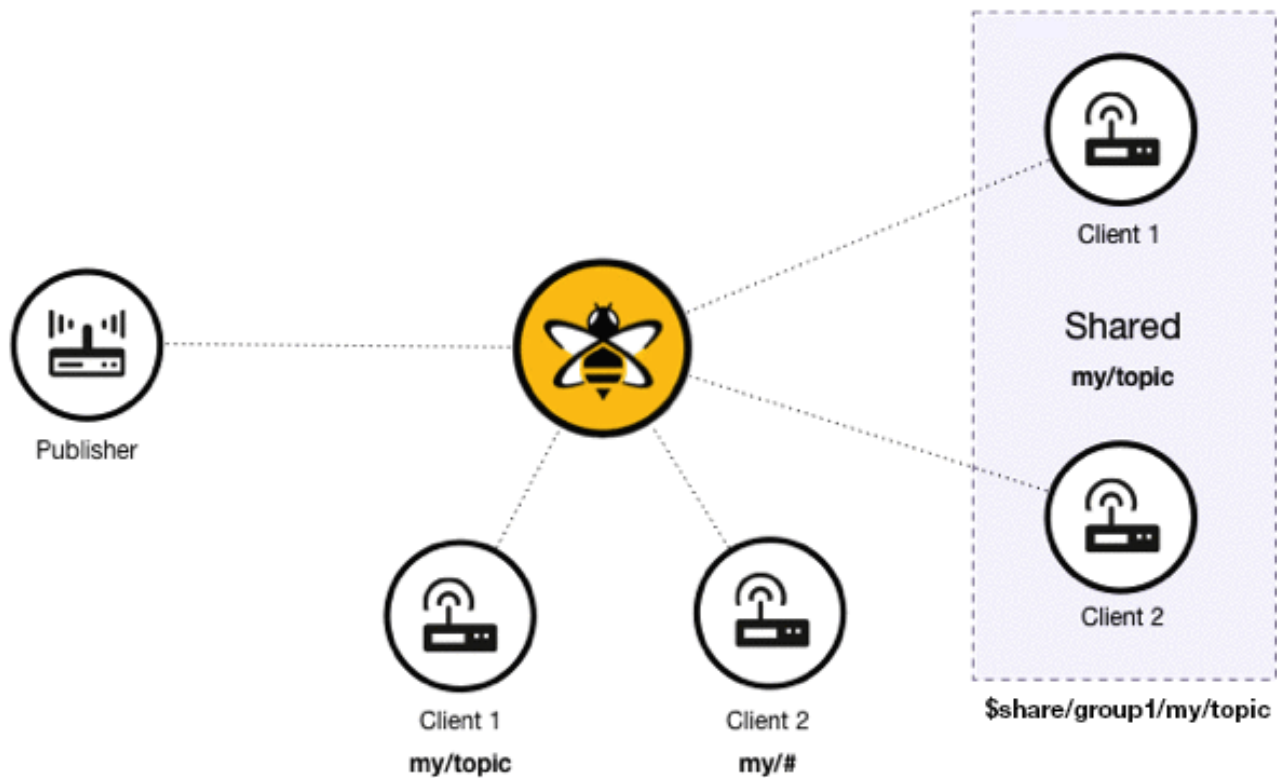


QoS 2 : Exactly once



MQTT: ulteriori funzionalità

- ❑ Oltre alla subscription ad un intero albero (#), un client può anche usare il carattere "+" per indicare un qualsiasi elemento intermedio:
 - Esempio: a/b/+/d/#
- ❑ Clean sessions: i subscriber possono impostare una flag per comunicare se, in caso di disconnessioni, i messaggi di QoS 1 e 2 devono essere trattenuti dal broker per poter essere ricevuti alla riconnessione (realizzando una coda di messaggi)
- ❑ Will (testamento): quando un client si disconnette il broker può inviare un messaggio pre-impostato a un certo topic per informare i partecipanti
- ❑ Shared subscriptions: meccanismo per distribuire i messaggi in un gruppo di subscribers (una sorta di load balancing)
 - Struttura del topic: *\$share/GROUPID/TOPIC*



MQTT: implementazioni e alternative

- ❑ Client per MQTT esistono in tutti i principali linguaggi di programmazione
 - Paho: progetto di Eclipse per diversi linguaggi (C, Java, Python)
 - Client specifici per diversi dispositivi (tra cui Arduino)
 - <https://mqtt.org/software/>
 - Server: Moquitto, RabbitMQ
- ❑ Alternative (meno usate nel mondo IoT)
 - AMQP: più complesso ma anche più versatile, idea più orientata al concetto di coda di messaggi (solitamente implementata in server che supportano anche MQTT: RabbitMQ, Apache ActiveMQ)
 - Soluzioni proprietarie: Amazon Simple Queue Service, Azure Service Bus, Google Cloud Pub/Sub
 - Simili: Apache Kafka (per stream processing)

MQTT: hands-on

Proviamo a vedere come funziona MQTT in pratica. Useremo un broker di messaggi pubblico (quindi non sicuro, va bene solo per prove veloci) per testare le varie feature.

Alcuni broker pubblici comprensivi di client web:

- ❑ <https://www.hivemq.com/demos/websocket-client/>
- ❑ <http://www.emqx.io/online-mqtt-client>

Inoltre esploriamo la possibilità di connettere direttamente un sistema embedded direttamente a MQTT usando la [libreria PubSubClient](#):

<https://wokwi.com/projects/397047501415867393>

XMPP

Messaggistica istantanea per dispositivi IoT

XMPP

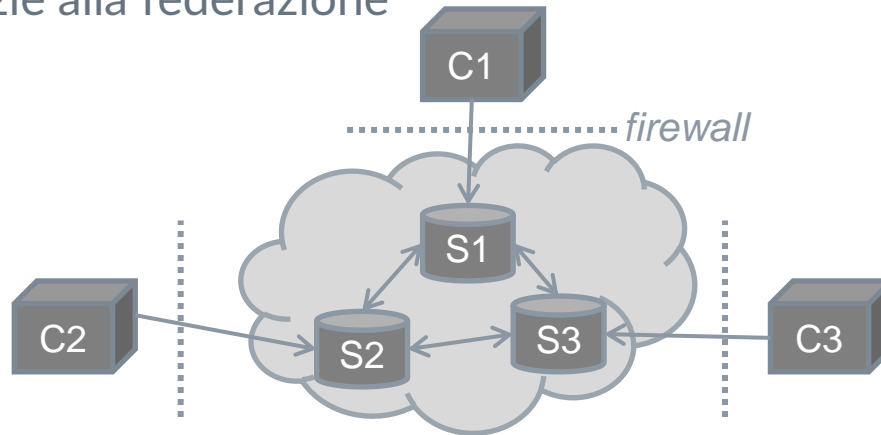
L'extensible Messaging and Presence Protocol (XMPP) è un protocollo per lo scambio di messaggi basato su stream di dati XML. Anch'esso si basa sulla presenza di message brokers.

- ❑ Supporta sia la modalità publish-subscribe che la comunicazione point-to-point, request-response e async messages
- ❑ Inizialmente pensato per applicazioni di messaggistica istantanea
- ❑ Pensato per la comunicazione near-real-time
 - Piccoli messaggi, trasmessi con la minor latenza possibile
- ❑ Possibilità di creare federazioni di server
 - Architettura simile alle tecnologie mail

XMPP: federazione

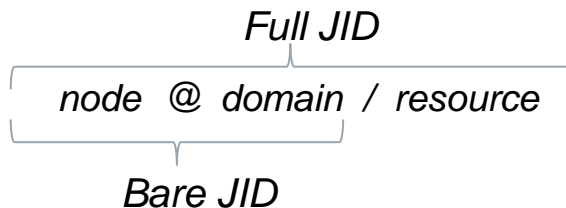
XMPP usa una rete federata di server usati come message brokers per i client che si trovano in diverse reti (separate da firewall)

- ❑ Ogni server controlla il suo dominio (che definisce gli utenti riconosciuti)
- ❑ I client su domini differenti possono comunicare tra loro o tra domini diversi grazie alla federazione

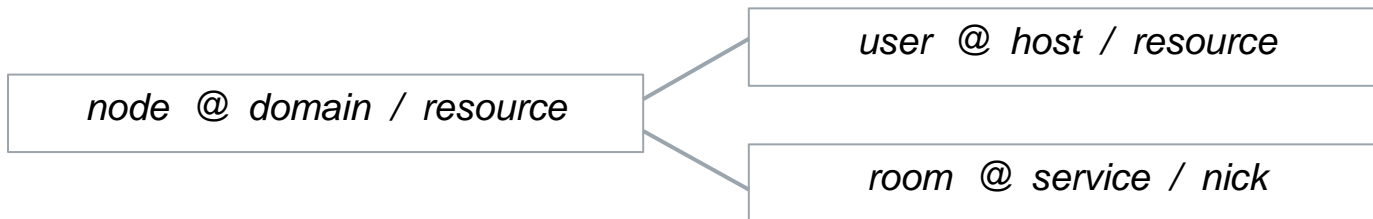


XMPP: identità e autenticazione

- ❑ In XMPP i client devono avere un'identità autenticata
 - Uso del Simple Authentication and Security Layer (SASL)
 - Possibilità di usare TLS
 - Le identità delle diverse entità sono chiamate XMPP address o Jabber Identities (JID)
- ❑ Un JID valido contiene: identificatore di dominio, di nodo e di risorsa



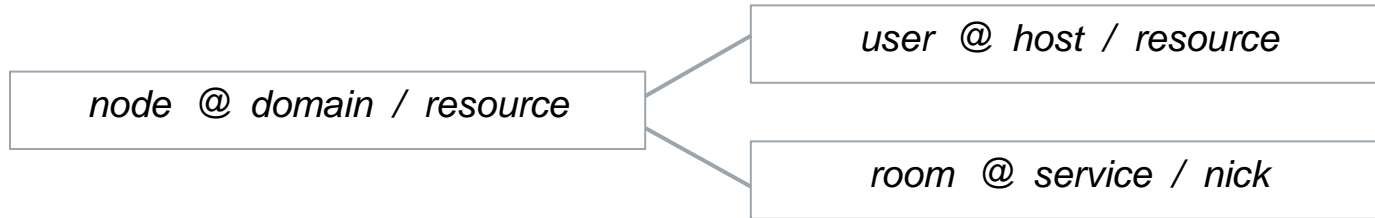
XMPP addressing scheme



❑ Domain identifier: di fatto l'unico elemento richiesto nel JID

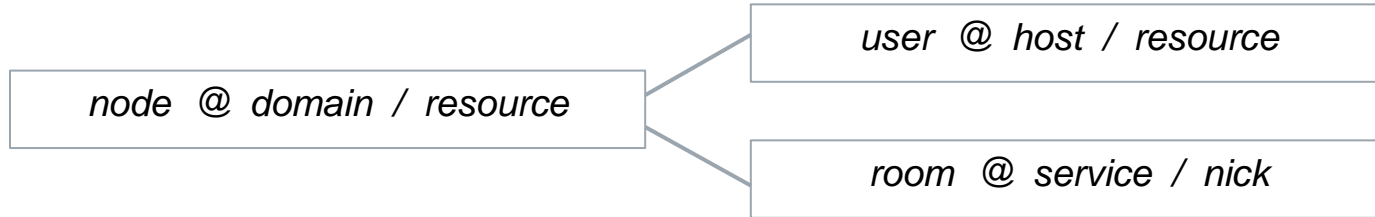
- Rappresenta la gateway o il "primary server" a cui le altre entità si connettono per eseguire il routing e gestione dei dati XML
- Si noti che l'entità identificata dal domain identifier non deve per forza essere un server: potrebbe essere un più astratto servizio che mette a disposizione una data funzionalità
- Il domain identifier può essere anche un indirizzo IP

XMPP addressing scheme



- ❑ **Node identifier:** l'identificatore del client che richiede i servizi del broker
 - Questo identificatore può anche essere utilizzato per identificare altre tipologie di entità (ad esempio una chat room)

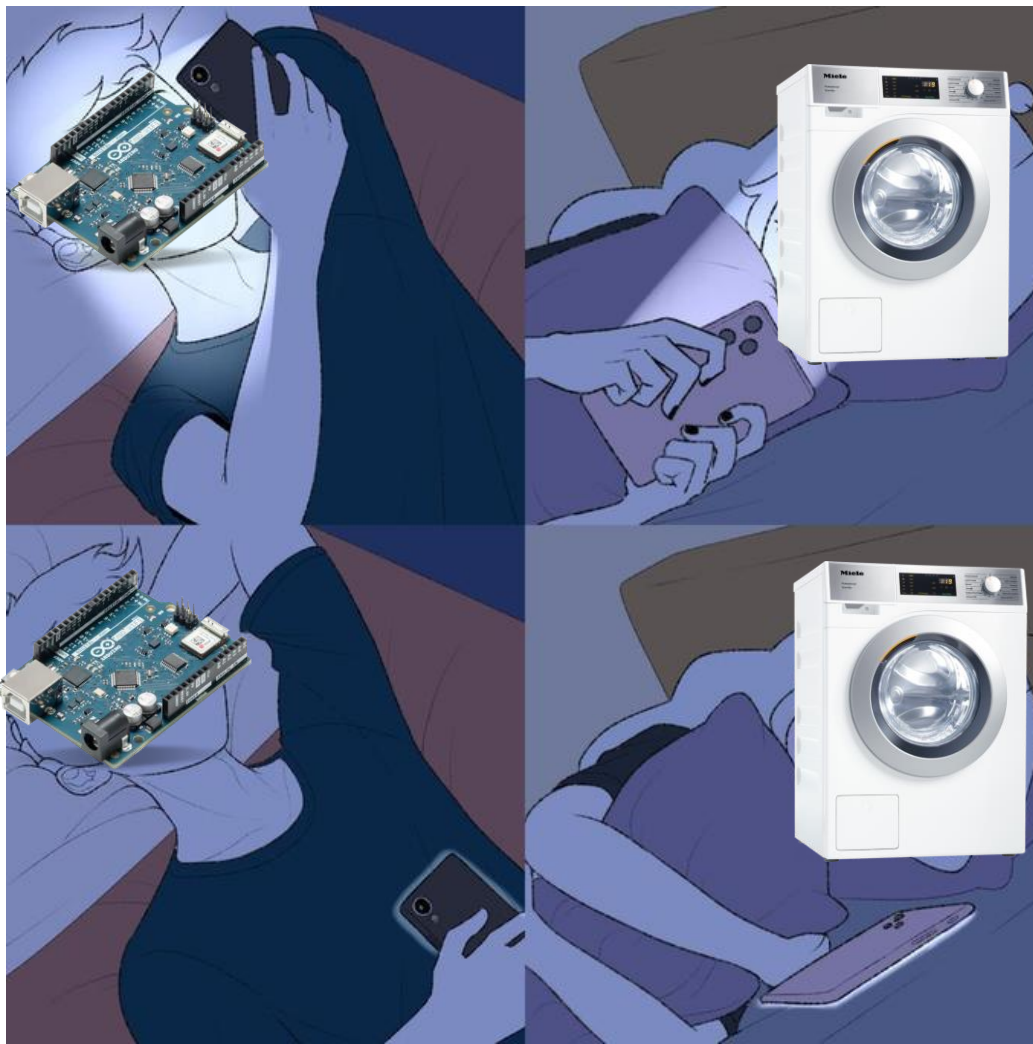
XMPP addressing scheme



- ❑ **Resource identifier:** identificatore generico, può identificare tante cose
 - Una specifica sessione, connessione (un dispositivo o un posto fisico), un oggetto, o in generale qualsiasi cosa che appartiene all'entità rappresentata dal node identifier
 - Ogni entità può avere più risorse connesse simultaneamente, ogniuna con il suo resource identifier

XMPP: presence e rosters

- ❑ Un client può richiedere le risorse disponibili inviando una "presence subscription" al suo JID
- ❑ Se accettata, un "presence message" viene inviato ogni qual volta in cui lo stato del contatto cambia (online, offline, impegnato, ...)
- ❑ Se anche l'altra entità invia un "presence message" che viene accettato, allora le due entità diventano "friends"
- ❑ I server XMPP mantengono una lista dei contatti per ogni entità e le loro subscriptions. Queste liste sono i "roasters"
- ❑ Se ci si connette con lo stesso account da un altro dispositivo, il client potrà chiedere al broker il roaster
- ❑ **Ricordiamoci che XMPP è nato per la messaggistica istantanea**



XML e EXI

Come già detto lo scambio di dati avviene attraverso messaggi XML

- ❑ XML: tanto overhead dato dal formato, ma ci sono implicazioni positive:
 - Il formato del contenuto è fisso e questo semplifica lo scambio dei dati
 - XML è molto conosciuto ed è facile da codificare e decodificare
 - Formato testuale: semplice da debugging
 - Disponibilità di tool di validazione per XML
 - Disponibilità di tool per l'analisi di stream XML
- ❑ Possibilità di utilizzare una versione più efficiente di XML chiamata Efficient XML Interchange (EXI)
 - Compressione dei messaggi XML in binario
 - Messaggi più compatti, gestione più efficiente

XMPP: comunicazione

La comunicazione XMPP è basata su due elementi:

1. XML streams

- Uno stream è un container per lo scambio di elementi XML tra due entità di una rete
- L'inizio e la fine di uno stream sono dati dai tag `<stream>` e `</stream>`
- In uno stream possono venire trasportati un numero illimitato di elementi
- L'"initial stream" è negoziato dall'entità che inizia la comunicazione (un client o un server) con l'entità ricevente (solitamente un server)
- L'"initial stream" permette la comunicazione unidirezionale. Uno stream in senso opposto (response stream) deve essere negoziato
- La sessione di un client con un server di fatto consiste di due documenti XML costruiti incrementalmente tramite accumulo di "XML stanzas" (prossima slide)

XMPP: comunicazione

2. XML stanzas

- Una unità semantica di informazione strutturata inviata da un'entità ad un'altra su uno stream XML
- Ogni stanza è un elemento XML immediatamente sotto il tag `<stream>`
- Una stanza XML è quindi denotata da un tag XML al livello 1 (immediatamente sotto il tag `stream`) e dal suo relativo tag di chiusura
- Ogni stanza può contenere tanti elementi
 - Ma i tag delle stanze sono limitati a Presence, Message e Iq
- Da notare come i dati utilizzati per TLS o SASL non sono considerati stanzas
- La chiusura dello stream (tramite invio del tag `</stream>`) comporta anche la contestuale chiusura della connessione

XMPP: stanze

- ❑ Presence stanza: per inviare informazioni circa l'entità client
- ❑ Message stanza: per inviare messaggi
- ❑ Iq stanza: per ottenere e inviare informazioni con meccanismo request/response

```
<Presence
  from="user1@mydomain.com"
  id="1232312312312"
  to="user2@mydomain.com"
>
  <show>online</show>
</Presence>
```

```
<message
  from="user1@mydomain.com"
  id="1232312312312"
  to="user2@mydomain.com"
  type="chat"
>
  <body>Hi friends</body>
</message>
```

```
<iq
  type="result"
  id="..."
  from="user2@mydomain.com"
  to="user2@mydomain.com/converse.js-132196857"
  xmlns="jabber:client">

  <vCard xmlns="vcard-temp">
    <FN>Nifro</FN>
    <NICKNAME>AkaLana</NICKNAME>
    <URL/>
    <ROLE>Dev</ROLE>
    <EMAIL>
      <INTERNET/>
      <PREF/>
      <USERID>demo@gmail.com</USERID>
    </EMAIL>
    <PHOTO>
      <TYPE>image/png</TYPE>
      <BINVAL>iVBORw0KGgoAAAANSUuEU</BINVAL>
    </PHOTO>
  </vCard>
</iq>
```

XEPs

Nel tempo sono nate diverse estensioni al protocollo XMPP. In particolare, la XMPP Standards Foundation (XSF) pubblica un insieme di estensioni revisionate e discusse pubblicamente.

- ❑ Queste estensioni sono chiamate XMPP Extension Protocols (XEPs)
 - <http://xmpp.org/extensions/>
- ❑ Tre livelli di discussione
 - Experimental stage: si riconosce il valore della proposta, ma viene modificata in maniera significativa per renderla più robusta
 - Draft stage: l'estensione viene discussa, revisionata dal punto di vista tecnica, eventualmente modificata con modifiche minori e retrocompatibili
 - Final stage: l'estensione non viene più modificata

WebSocket

Comunicazione con tecnologie standard web

WebSocket

Si tratta di una tecnologia parte della specifica HTML5


- ❑ Comunicazione bidirezionale: client e server possono scambiarsi messaggi in ogni momento
- ❑ Connessione TCP: utilizzata per creare un canale full-duplex
- ❑ Nessuna gestione della connessione o coordinazione
- ❑ Si basa su una connessione HTTP (di cui viene fatto l'upgrade)

Si tratta di una specifica non pensata per l'IoT, ma presenta il vantaggio di essere standard e poter attraversare i firewall (visto che usa una connessione HTTP).

WebSocket: handshake

Una connessione WebSocket inizia con un handshake

- ❑ La connessione avviene a un server HTTP alla stessa porta su cui il server normalmente gestisce le normali connessioni (solitamente 80 o 443)
- ❑ Di fatto l'handshake è una richiesta di upgrade, eseguita inviando un apposito header:



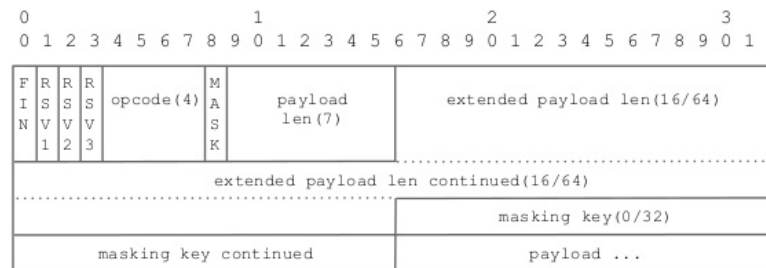
```
GET /chat HTTP/1.1  
Host: server.example.com  
Origin: http://example.com  
Upgrade: websocket  
Connection: Upgrade
```

WebSocket: handshake

- ❑ Il primo passo è quello di inviare una richiesta HTTP al server con un header visto in precedenza per chiedere l'upgrade del protocollo
 - Si tratta, come per tutte le richieste HTTP, di una richiesta fatta specificando l'URL di una risorsa
 - Di fatto Host + Risorsa formano il topic
 - I WebSocket richiedono che nell'header di handshake venga specificato Origin (policy same-origin) e i valori "Upgrade: websocket" e "Connection: upgrade"
- ❑ Essendo una richiesta HTTP, se il server non supporta i WebSocket ma comunque la risorsa esiste, allora è possibile usare il normale HTTP come fallback

WebSocket handshake

- ❑ Se il server supporta WebSockets, allora risponderà con un codice 101 (Switching Protocols)
 - Da questo momento in poi la connessione diventa persistente e full-duplex
 - La comunicazione è gestita attraverso l'invio di appositi frame seguendo un protocollo standard
 - L'overhead per un data frame è molto contenuto: solo 2 byte
- ❑ Sia il client che il server possono chiudere la WebSocket in ogni momento



WebSocket e MQTT

Interessante interazione tra MQTT e WebSocket: WebSocket può essere utilizzata come protocollo di comunicazione per comunicare con il broker al posto del semplice TCP. Nota: vale anche per AMQP, non solo MQTT.

- ☐ Migliore capacità di attraversare i firewall senza dover attivare regole ad-hoc (WebSocket usa le porte standard del web) ...
- ☐ ... ma anche maggiore overhead
- ☐ Meccanismo spesso usato per supportare client browser

Approfondimento:

<https://www.hivemq.com/blog/understanding-the-differences-between-mqtt-and-websockets-for-iot/>



LABORATORIO DI SISTEMI EMBEDDED E IOT

Modulo 1

Laurea in Tecnologie dei sistemi informatici

Università di Bologna - Sede di Imola

Lorenzo Pellegrini, Ph.D.

l.pellegrini@unibo.it - miatbiolab.csr.unibo.it

Protocolli per l'IoT