

Parte I (IoT)

IoT \Rightarrow internet of things \Rightarrow c'è una rete di oggetti che tramite identificazione dei dispositivi, intelligenza e capacità di rilevamento collega persone e cose attraverso internet

sensori e attuatori

raccolgono i dati \rightarrow possono essere elaborati:

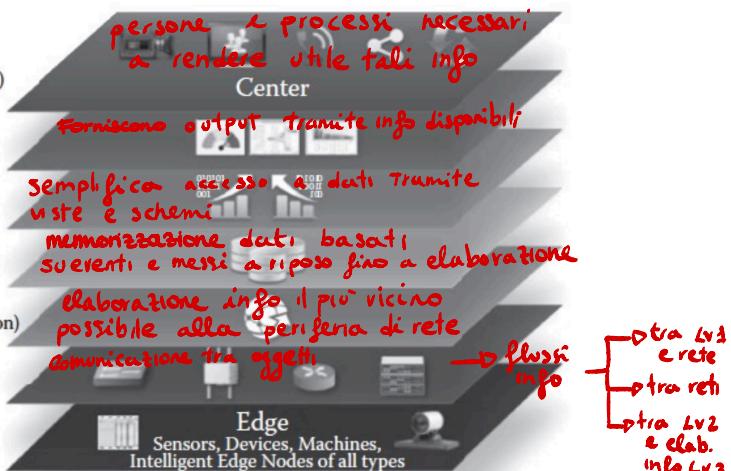
\sim localmente su uno smart device \Rightarrow Edge Computing

\sim inviati ad intermittenza su un gateway per elaborazione a \Rightarrow Fog Computing zone

\sim inviata a postazione cloud \Rightarrow Cloud Computing che archivia ed elabora

Levels

- 7 Collaboration & Processes (Involving People & Business Processes)
- 6 Application (Reporting, Analytics, Control)
- 5 Data Abstraction (Aggregation & Access)
- 4 Data Accumulation (Storage)
- 3 Edge Computing (Data Element Analysis & Transformation)
- 2 Connectivity (Communication & Processing Units)
- 1 Physical Devices & Controllers (The "Things" in IoT)

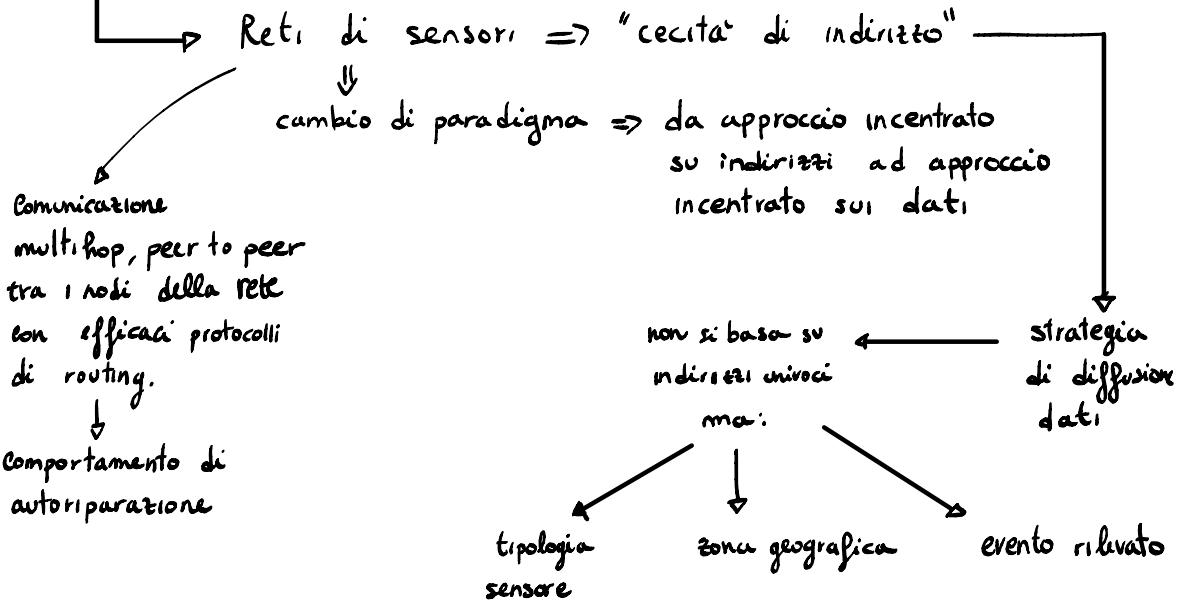


Annotations in red text for Level 1:

- \rightarrow inviano/ricevono info
- \rightarrow controllati tramite internet \rightarrow generare dati contestuali o connettere
- \rightarrow mandati ad intermittenza a piccole unità
- \rightarrow segnali analogici in digitali

Reti ad hoc: sono reti generali, prive di infrastrutture, basate su cooperazioni opportunistiche tipicamente personalizzate su scenari e applicazioni specifiche.

↳ dispositivi comunicano tra di loro



ARCHITETTURA:

↳ IoT basato su Client - Server \Rightarrow pratiche derivate dal web

accoppiare
liberamente
server con APP

supportare
evoluzione
a lungo
di sistemi

robustezza

paradigma
REST

LIVELLO 1/2: (FISICO / COLLEGAMENTO)

- ~ IEEE 802.15.4 + zigBee
- ~ Bluetooth 4.0 → BLE (Bluetooth a bassa energia)
 - radio a corto raggio ($\sim 100\text{ m}$)
 - topologia a stella master/slave
- ~ IEEE 802.11 ah → WiFi HaLow o a basso consumo
 - ampio raggio (1 Km)
 - num dispositivi elevato ($\approx 10^5$)
 - servizi di risparmio energetico

LIVELLO 3: (LIVELLO DI RETE)

- ~ IPv4 + NAT (traduzione indirizzi IP tra pubblico e privato) → raggiungibilità con una limitata scalabilità
- ~ IPv6
 - \Rightarrow 128 bit
 - \Rightarrow link-local (derivato dal MAC)
 - \Rightarrow global (forniti dal router della rete)
 - \Rightarrow IPsec

N 6 LOW PAN

LIVELLO 4: (LIVELLO DI TRASPORTO)

- ~ Comunicazione tramite UDP

LIVELLO 5: (LIVELLO DI APPLICATIONE)

- ~ COAP => protocollo di trasferimento web per reti a bassa potenza e con perdite.
(smart home) Viene usato nelle reti con risorse limitate che operano su reti instabili o a bassa potenza.
- ~ MQTT => scenari in cui è necessaria comunicazione affidabile asincrona e scalabile tra dispositivi IoT.
↓
multi dispositivi (smart city) Funziona con un modello publisher / subscriber in cui un broker gestisce comunicazione tra server.
- ~ CoSIP => variante ottimizzata di SIP, progettata per dispositivi IoT con risorse limitate.
↓
(smart security) SESSIONE → comunicazione diretta e interattiva tra dispositivi

REST => insieme di regole e principi che consentono alle applicazioni web di:

~ scalare (num clienti che interagiscono)

~ robustezza (evoluzione a lungo termine)

→ si divide in due parti → ROA (Resource-oriented Architectures)
le risorse vengono chiamate per nome.

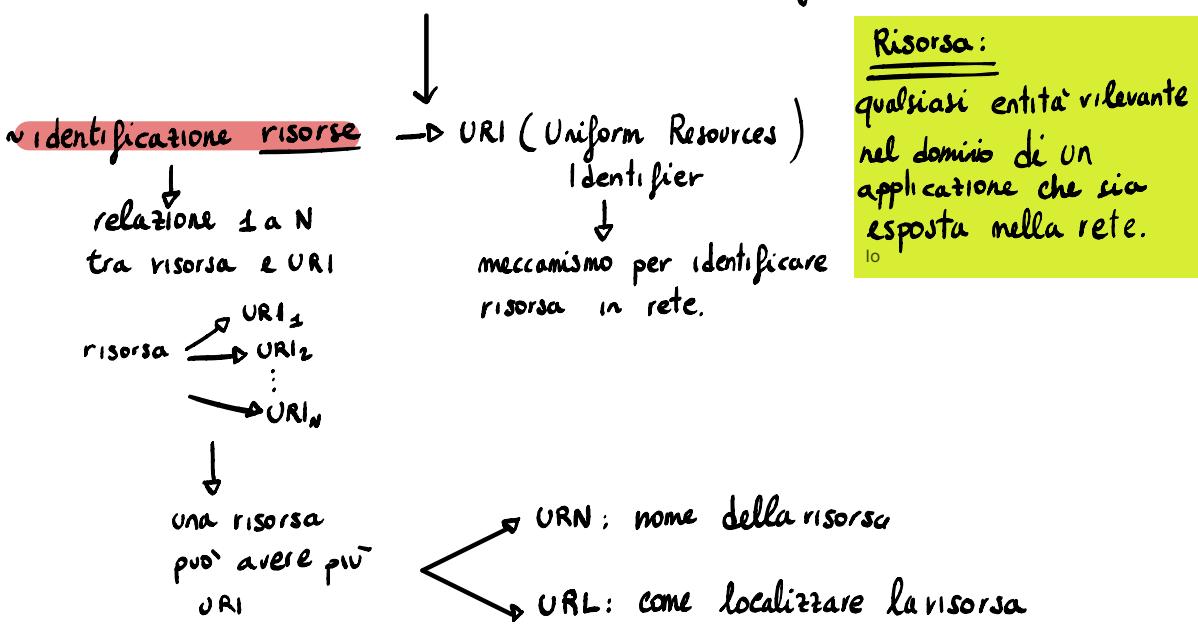
→ SOA (Service-oriented Architectures)
due endpoint comunicano attraverso un insieme di servizi offerti dal server

chiamata metodo ↙ utente manda messaggio
su server

L'architettura è fatta da:

- client: si interfaccia con app e fa iterazioni con l'obiettivo di recuperare informazioni.
- server: ospita risorse e serve le richieste del client
- principio della separazione delle competenze: modularità / riutilizzabilità degli elementi del sistema
- Intermediari: Proxy ⇒ agiscono come Client e Server a seconda della situazione
 - Forward: punti di uscita richiesta
 - Reverse: server di origine per cliente ma che inoltrano richieste.

Una app che segue i principi di REST viene definita **RESTful**



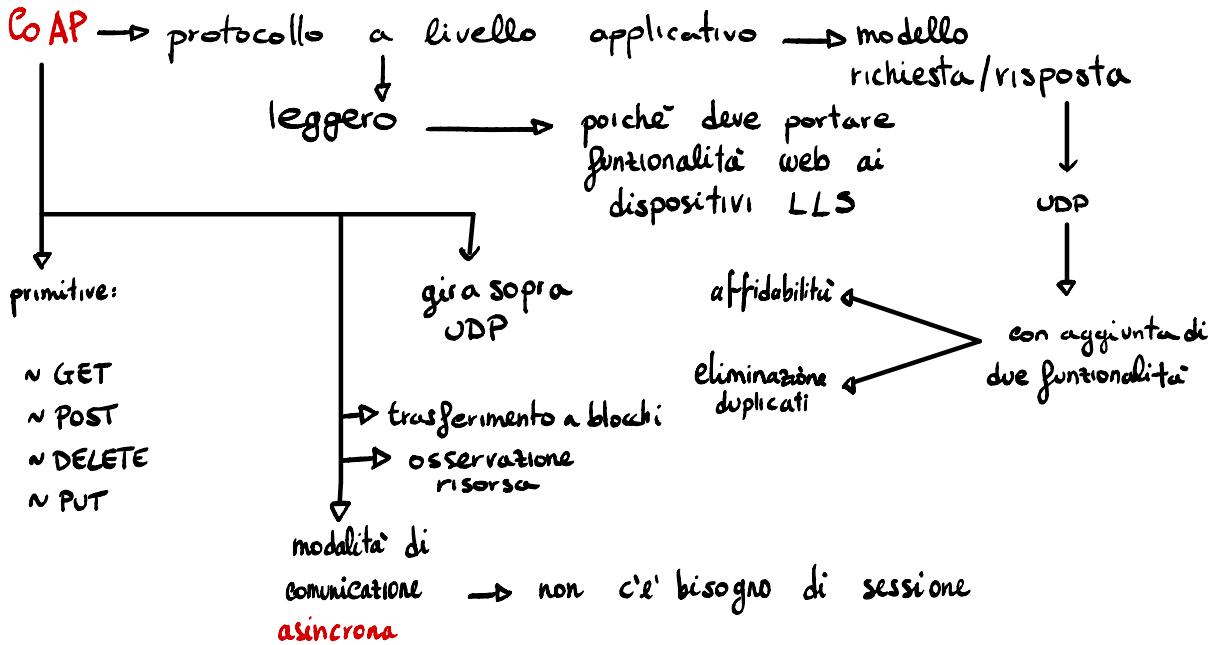
v manipolazioni risorse attraverso rappresentazioni → risorsa non viene mai scambiata ma viene scambiata una vista, una rappresentazione della risorsa in quell'istante
↓
XHTML, Atom, XML, JSON, MP3, JPEG

v messaggi autodescrittivi: la richiesta deve contenere tutte le informazioni in modo che i server possano elaborarla senza contesto

↑
stateless → nessuna sessione

v permette di essere motore degli stati applicativi.

↓
server non tiene traccia dello stato.



Affidabilità del CoAP → back off esponenziale := ogni volta si ha una trasmissione timeout raddoppia

4 tipi di messaggio:

- ~ CON(confirmable): messaggi in modo affidabile devono avere riscontro.
- ~ NON(NO-confirmable): non è necessaria affidabilità ⇒ la risposta sarà con NON
- ~ ACK: conferma ricezione messaggio CON
- ~ RST: annullare trasmissione affidabile

Nelle risposte CON si puo' avere due situazioni:

⇒ **piggy banking** = il server risponde immediatamente e invia un messaggio ACK con dentro la risposta inclusa.

⇒ **separata** = il server non puo' rispondere in quel momento e invia solo un ACK per confermare la richiesta. Quando i dati sono disponibili, server invia CON con la risposta e il client risponde con un ACK.

Lo AP message format:

4 bytes	V	T	TKL	Code	Message ID
TKL bytes				Token	
variable				Options	
variable	0x00			Payload	

header + optional payload

⇒ V := versione protocollo (2 bit)

⇒ T := tipo di messaggio
 3 := RST
 0 := CON
 1 := NON
 2 := ACK

⇒ TKL := lunghezza Token

⇒ Code := formato da 8 bit → 3 per classe
 0 classe
 1 per significato
 messaggio di richiesta
 ↳ info relative a risposta

⇒ Message ID: identificativo messaggi per trovare duplicati e per affidabilità

⇒ Token: serve a correlare una richiesta con una risposta.

⇒ CoAP options: questo campo contiene info obbligatorie e facoltative per tutti i messaggi (URI/tipo media payload)

Option number	Option name	Format	Length (bits)
1	If-Match	opaque	0-8
3	Uri-Host	string	1-255
4	ETag	opaque	1-8
5	If-None-Match	empty	0
7	Uri-Port	uint	0-2
8	Location-Path	string	0-255
11	Uri-Path	string	0-255
12	Content-Format	uint	0-2
14	Max-Age	uint	0-4
15	Uri-Query	string	0-255
17	Accept	uint	0-2
20	Location-Query	string	0-255
35	Proxy-Uri	string	1-1034
39	Proxy-Scheme	string	1-255
60	Size1	uint	0-4

- specifica nome host a cui è destinata richiesta
- numero di porta a livello trasporto
- un segmento del percorso assoluto risorsa
- formato rappresentazione nel payload
- compatto usando:
 - option number
 - lunghezza option
 - valore option

I clienti possono ottenere lo stato della risorsa (che cambia spesso) in due modi:

⇒ polling: si esegue periodicamente richieste GET.
 Ma le risorse possono non cambiare stato tra due GET consecutivi (quindi inutile ~ consumo batterie).
 Se cambio stato più volte tra richieste successive ⇒ qual'è stato più aggiornato?

⇒ observe:

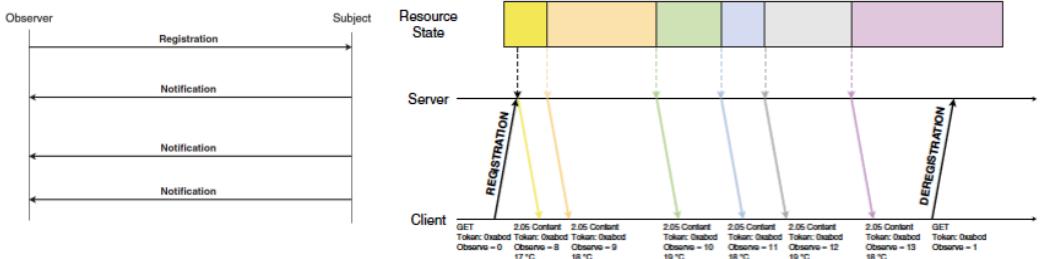
Design Pattern Observer

O := registrazione ←
 I := deregistration

⇒ quando arriva una richiesta GET con l'opzione 6 **Observer**, il server sa che deve mandare notifiche al client quando lo stato cambia.

Observer ←
 è un direzionale
 "il server invia dati,
 ma i client non
 possono pubblicare a loro
 volta"

↓
 simile a pub/sub
 solo che la comunicazione
 è ancora basata su
 richieste e risposte
 ↓
 per comunicazione tra
 client e server

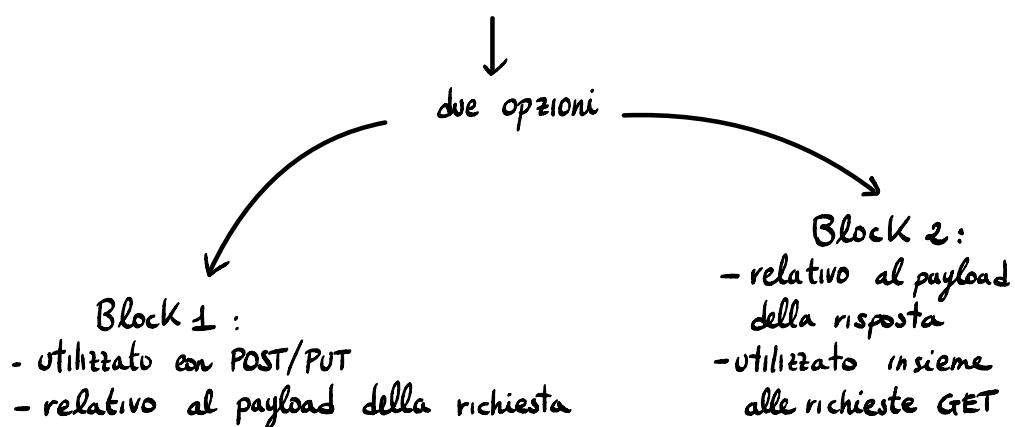


⇒ lista di clienti per ogni risorsa.

↳ può anche essere NON osservabile

utilizzando attributo
obs

Block Transfer: divide i dati da mandare in più blocchi di una certa dimensione.



NUM: numero relativo del blocco all'interno di una sequenza di blocchi con determinata dimensione

H (1 bit)
seguono altri blocchi o no

SZx: 3 bit
 $0 \leq SZx \leq 6$
 2^{4+SZx}

Comunicazione Multicast → più endpoint per una singola richiesta

per ottenere URI
delle risorse ospitate
da un CoAP di servizio
endpoint

Esempio: una singola richiesta
POST per un gruppo
multicast di indirizzi.

→ cliente richiede al server il percorso:
/well-known/core

modi in modalità
riposo
o gestione
traffico multicast
inefficiente.

alcune volte non
è fattibile

si usa directory di
risorse (RD)

↓
endpoint può
modificare/aggiungere
eliminare

Il cliente riceve una lista
di risorse formattate
CoRE Link Format

ogni link contiene lista
di attributi che descrivono
la risorsa

obs := osservabile o no

rt := tipo di risorsa

ct := tipo di contenuto

if := insieme di metodi

Proxy HTTP/CoAP

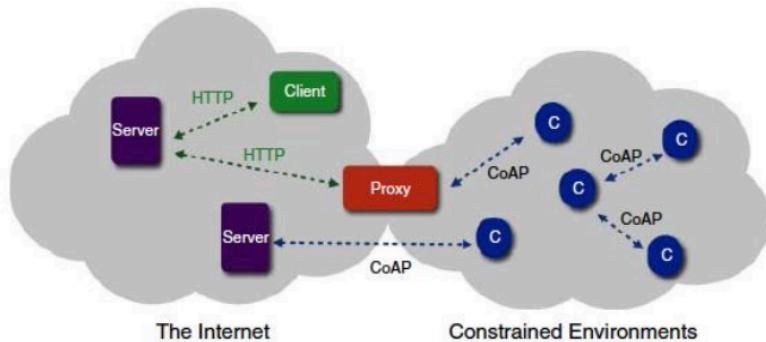
→ si utilizzano perché clienti potrebbero
non supportare CoAP

supportare i dati
che potrebbero non essere
adatti per rete vincolata

proteggere rete
da attacchi
DDOS

utilizzare
cache

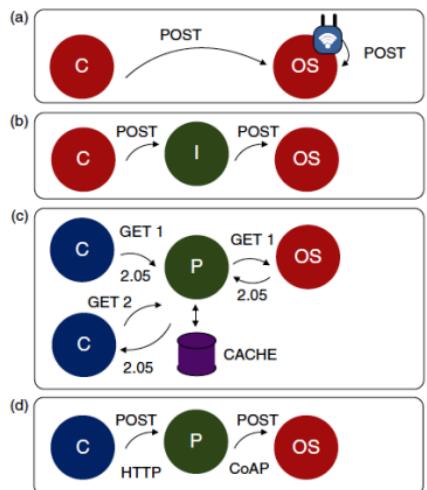
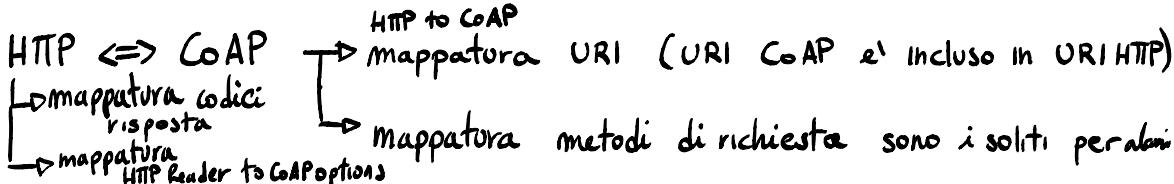
ridurre
carico rete



Terminologia:

- **Server di origine (a):**
server CoAP su cui risiede o deve essere creata una risorsa
- **Intermediario (b):**
endpoint CoAP che agisce sia come Server che come client verso un server di origine
- **proxy:**
intermediario che inoltra le richieste e ritrasmette le risposte

- ~ traduce le richieste da HTTP a CoAP
- ~ CoAP-to-CoAP: mappa una richiesta CoAP con un'altra
- ~ **per forma una traduzione di protocollo da CoAP a HTTP e viceversa (Cross - Proxy)**



CoSIP → protocollo che unisce caratteristiche del **CoAP** e del **SIP**. Per abilitare comunicazioni in tempo reale tra dispositivi IoT con risorse limitate.

↓
usa CoAP → sta più in basso del CoSIP
invece di HTTP/TCP

↓
sessioni

↓
serve per scoprire nodi, registrare servizi, indirizzare richieste fra nodi CoAP

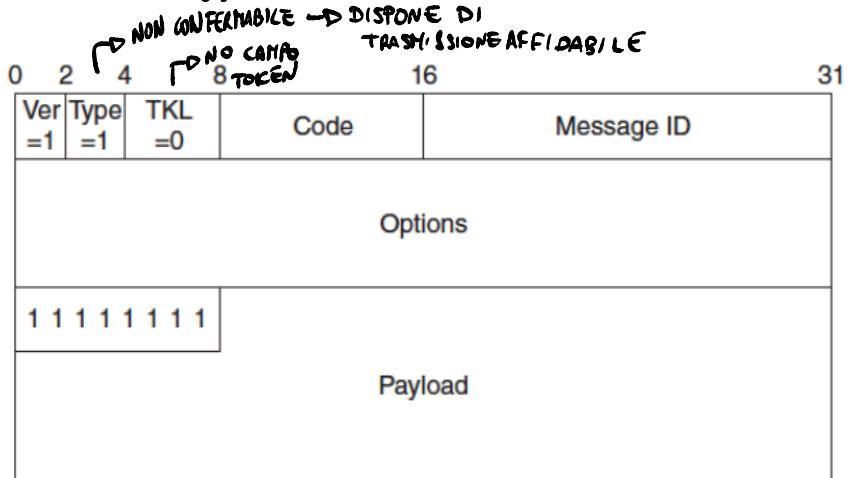
↓
constrained SIP
viene utilizzato per LLN
(Low Power and Lossy Networks)

↓
fasi:

1. **Discovery**: dispositivi CoAP vengono scoperti nella rete.
 - richiesta discovery ⇒ invia una richiesta discovery (multicast) per scoprire specifiche
 - risposta discovery ⇒ rispondono con indirizzo e tipo risorsa.
2. **Registrazione**: dopo la risposta ⇒ server si salva info sul dispositivo vincolato (indirizzo, tipo risorsa)
3. **Initializzazione Sessione**: messaggio invito per sessione con dispositivo vincolato. Negotiazione.
4. **Gestione della Sessione**: scambio dati (lettura periodica o observer). Il dispositivo può mandare aggiorn.
5. **Redirect**: risorsa spostata in altro posto → indirizzo di quel posto.
6. **Chiusura**: Bye (messaggio)

↓
registro delle risorse → **REGISTRAR**

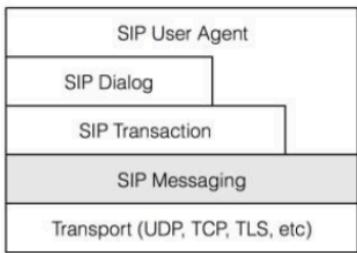
Formati dei messaggi CoSIP \Rightarrow ricicla quello del CoAP



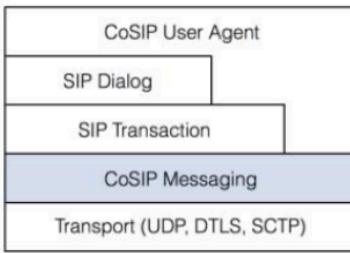
Campo Code \Rightarrow 8 bit

codifica:

- ~ metodi di richiesta
- ~ codici di risposta



(a) SIP layered architecture



(b) CoSIP layered architecture

Scenari applicazione CoSIP :

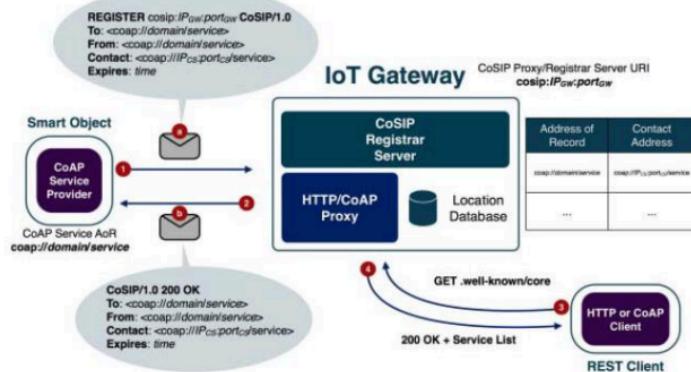
1.) Individuazione del servizio CoAP

→ CoSIP registrar Server funge da Resource Directory

trovare ←
altre risorse

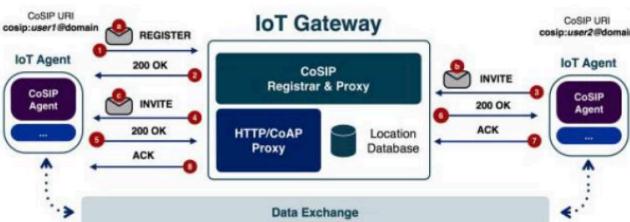
tiene conto delle
risorse

aggiornare
o cancellare
risorse

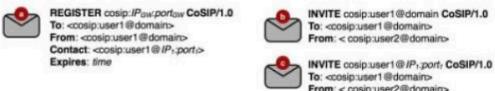


2) Stabilire una sessione:

- ci sono due Agenti IoT, registrati nel gateway IoT

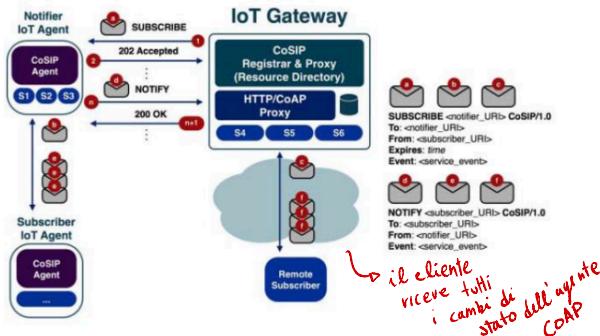


Proxy → parametri della
sessione sono
incapsulati
nel Payload



3) subscribe/ notify application

- notificatore IoT Agent (Registrato nel CoSIP registrar Server)
- iscritto IoT Agent
- IoT Gateway (Proxy)
- remote subscriber



Sistemi di messaggistica \Rightarrow offrono vantaggi in scenari dove REST può avere limitazioni

quelli che richiedono

i dati e quelli che non si mandano
li mandano non
hanno relazione

messaggi
diretti

gli oggetti sono
piccoli server ma
con capacità di
elaborazione limitata
e
problemi di connettività

middleware
orientato a messaggi

middleware:

un software che
fa da tramite a
due parti

lo

i suoi approcci sono tramite:

1. coda di messaggi

il mittente manda il messaggio al server che lo mette nella coda. Lo tiene in coda finché il richiedente lo riceve

Indipendenza
tra
i due

\rightarrow comunicazione
asincrona

2. publisher / subscriber

↓
manda un
messaggio su
un "topic" del
server

\hookrightarrow iscritto al topic di
un server per ricevere
una copia di tutti i
messaggi che sono stati
pubblicati

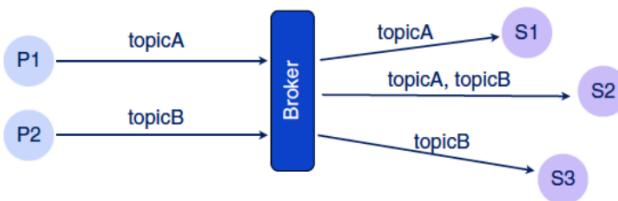
\rightarrow un messaggio può
essere consumato
da più richiedenti

Qui c'è un intermediario \Rightarrow Broker \rightarrow separa Publisher da Subscriber

↓
funtione:

↓

quando riceve un messaggio
dal publisher, esso invia
i messaggi in accordo
alle loro iscrizioni



PRO

- ~ i publisher non sanno (e non gliene importa) dei subscriber.
- In confronto al paradigma Client Server
- ~ Scalabilità \Rightarrow si aumenta i broker possono essere facilmente replicabili
- ~ il carico maggiore è supportato dai Broker

CONTRO

- ~ non c'è interazione (NO NEGOZIAZIONE)
- ~ nessun supporto per sicurezza e/o
- ~ Broker devono esseri scalati per non avere congestione

MQTT → per dispositivi e reti vincolate.

- basato su TCP pub/sub protocol.
- dim e header sono minimizzati.

MQTT-SN ⇒ reti di

sensori

- per dispositivi a batteria bassa e low cost

Message
Queue
Telemetry
Transport

Quando viene usato MQTT non viene usato il CoAP ⇒ server e il broker

I messaggi vengono pubblicati in uno spazio topic condiviso sul Broker
è gerarchico ← → è un filtro
topic/subtopic/sub-subtopic in un flusso di messaggi
Consumer → Broker

I messaggi vengono mandati a tutti i clienti che si sono iscritti a quel topic (1:N)

L'iscrizione è un'operazione non duratura e in tempo reale.

Se si vuole iscriversi ad uno specifico topic ⇒ messaggi con quel topic

↓
jolly → + viene fatto per un singolo livello di gerarchia
→ # è usato per indicare tutti i livelli rimanenti di una gerarchia

+ can be used as a wildcard for a single level of hierarchy

sensors/+ /temperature/+

can be used as a wildcard for all remaining levels of hierarchy

sensors/COMPUTER_NAME/#

- Utilizzo del simbolo +:

 - sensors/+/temperature/+

 - * Questo pattern potrebbe corrispondere a:

 - sensors/livingroom/temperature/celsius

 - sensors/bedroom/temperature/fahrenheit

 - * Il + sostituisce un singolo livello nella gerarchia

- Utilizzo del simbolo #:

 - sensors/COMPUTER_NAME/#

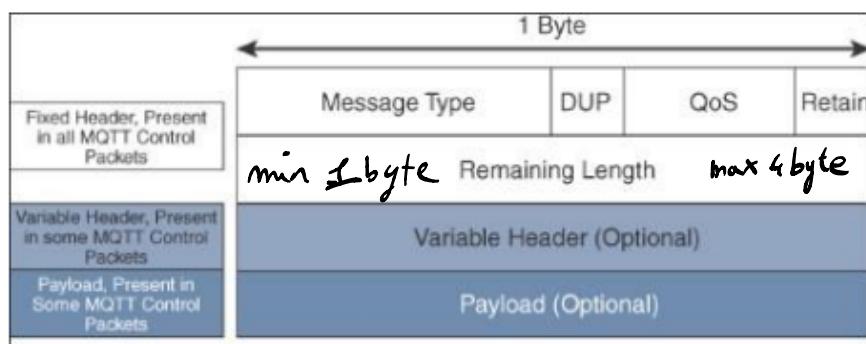
 - * Questo pattern potrebbe corrispondere a:

 - sensors/COMPUTER_NAME/cpu/temperature

 - sensors/COMPUTER_NAME/gpu/fan_speed/rpm

 - sensors/COMPUTER_NAME/memory/usage/percentage

 - * Il # sostituisce tutti i livelli rimanenti nella gerarchia dopo COMPUTER_NAME



tipo messaggio \Rightarrow CONNACK (HEADER FISSO)

\downarrow
Control header + Length

\Rightarrow PUBACK (HEADER FISSO + VARIABILE)

\Rightarrow CONNECT (HEADER FISSO + VARIABILE + PAYLOAD)

Control field \rightarrow 4 bit \rightarrow Message Type

\rightarrow 4 bit \rightarrow flag di controllo

mess. stato \leftarrow DUP QoS Retain Remaining Length

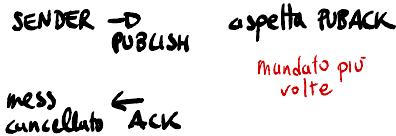
mandato ma

NO ACK

\hookrightarrow num byte nel resto del MQTT packet

livelli:

- 1: almeno una volta
(mess. mandato almeno
una volta)



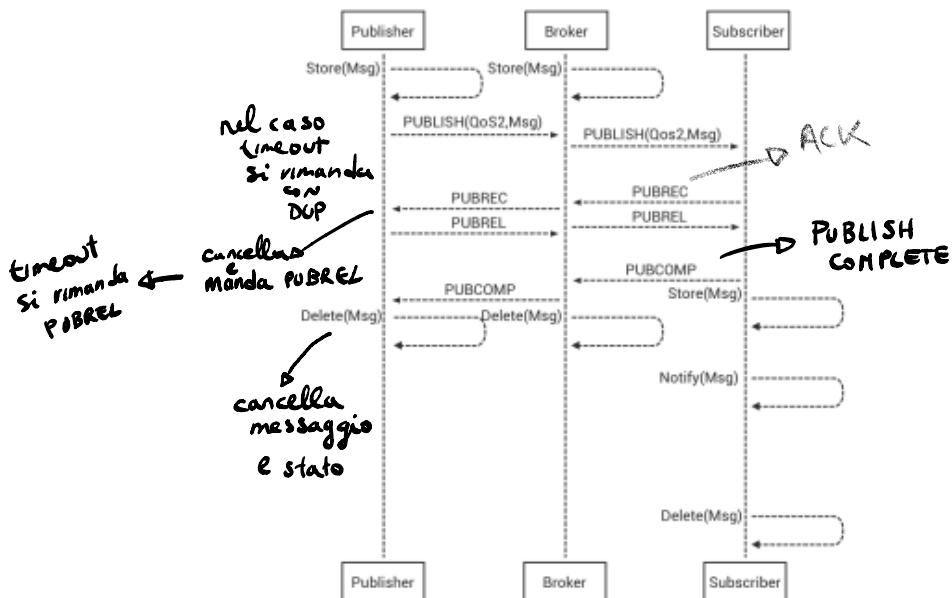
il server deve salvare
questo messaggio e
il QoS per inviarlo
ai prox subscriber

il quale iscrizione
matcha con topic

TIMEOUT → rimandato con
DUP

0. al max una volta ⇒ una volta mandato, viene
cancellato (NIENTE ACK)

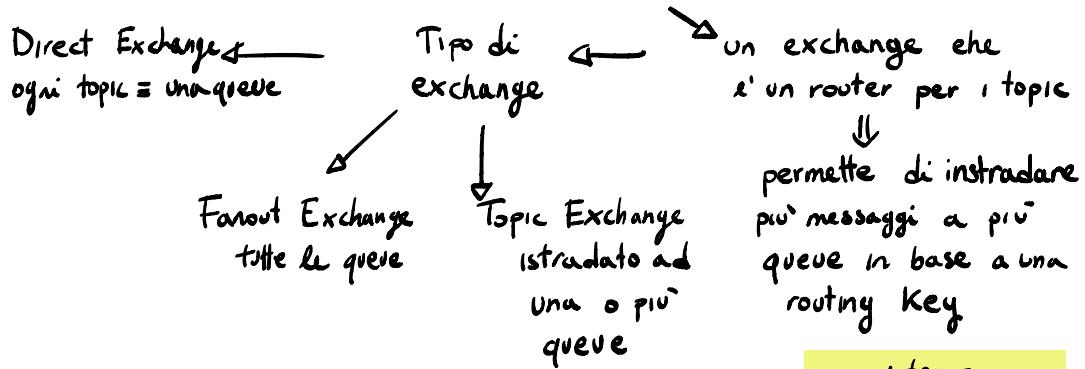
2. ESATTAMENTE UNA VOLTA senza duplicati



AHQP (ADVANCED Message Queue Protocol)

- non pub/sub
- MOM (message oriented middleware)
 - ↳ interfaccia software facilita scambio di messaggi.
 - Responsabile gestione delle code e della consegna tra mittente e consumer

Un AHQP Publisher manda direttamente → coda



persistence:
se viene resettato il broker perde messaggi altrimenti no.

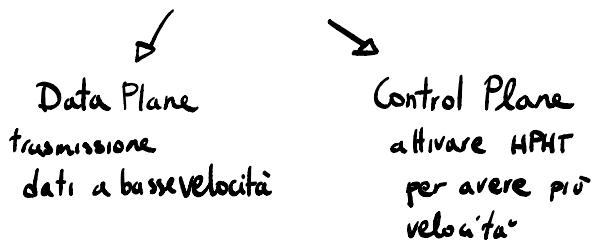
Io

Dual-Network Management Protocol (DNMP)

↳ gestisce due tipi di interfaccia di rete:

1. Low-Power Low-Throughput (Basso consumo/Velocità)
2. High-Power High-Throughput

LPLT \Rightarrow sempre attiva e ha due funzioni



Interfaccia tramite IP \Rightarrow basato su UDP per stabilire rotte E2E tra nodi multi hop

DN HP \rightarrow componenti principali \rightarrow CP (Control Plane) \rightarrow gestisce rotte HPHT quando serve velocità

RP^{low} \rightarrow protocollo per trasmissione a bassa velocità

RP^{high} \Rightarrow protocollo gestione interfaccie HPHT

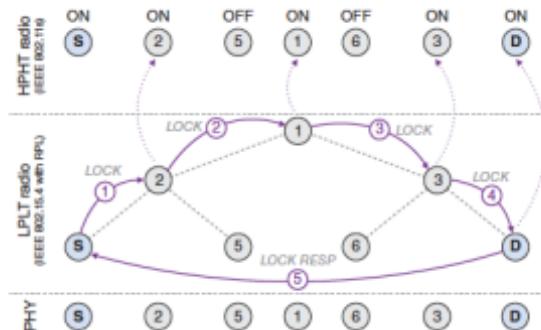
C^{high} \Rightarrow numero di percorsi RP^{high} per rendere HPHT attiva

\hookrightarrow se a 0 vuol dire che HPHT è spenta

LOCK
viene mandato lungo il percorso migliore LPLT (determ. dinamicamente)

\hookrightarrow quando i nodi ricevono il LOCK \rightarrow HPHT attivato \rightarrow C^{high} + 1

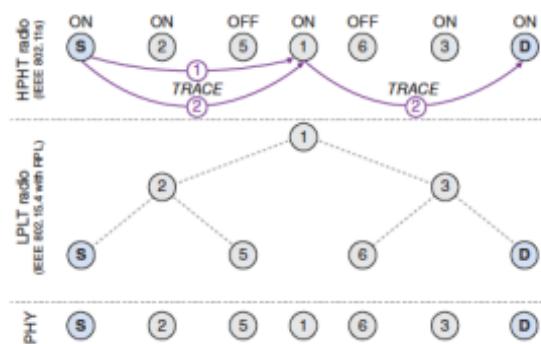
1 → Locking phase



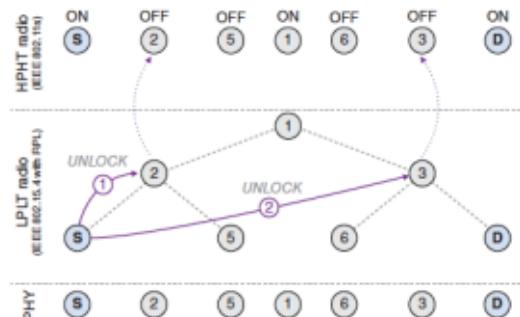
2 → Identification phase

TRACE → per verificare
il percorso

↓
feedback



3 → Unlocking phase +
Route Tear down



CoAP - CoSIP - MQTT - AMQT - DNMP