Appunti IoT Systems

Capitolo 8

La comunicazione tra più smart objects

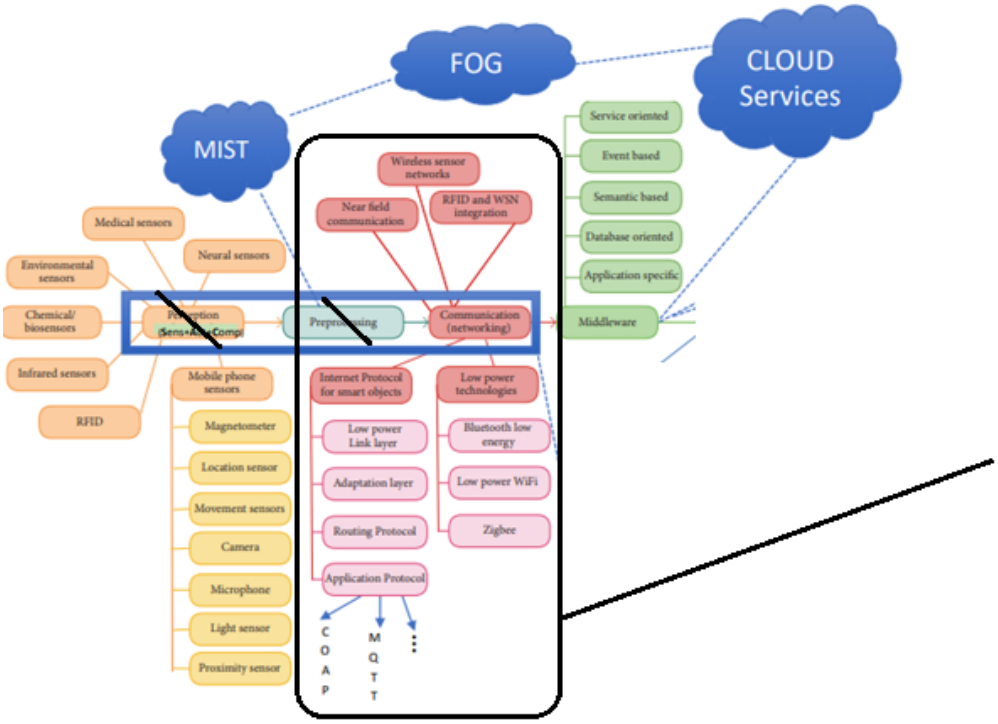
Simone Coco

La comunicazione tra più smart objects

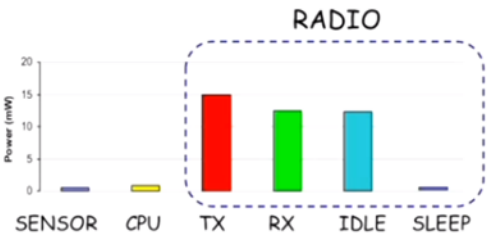
Il Communication layer

Abbiamo sempre detto che due oggetti, per essere definiti “smart” devono avere delle funzionalità aggiuntive (che non rientrano tra le “funzionalità base” dell’oggetto) ma soprattutto **devono essere in grado di comunicare con altri smart objects.**

Abbiamo trattato i vari livelli dell’architettura IoT trattando il perception: parlando dei vari sensori attuatori e commutatori esistenti e di preprocessing, introducendo il preprocessing ed il paradigma fluid computing, abbiamo parlato del middleware;
Tuttavia abbiamo sorvolato sul livello della **communication** per trattarla in dettaglio in un secondo momento.



Come abbiamo già detto, la comunicazione segna il maggior spreco di energia. Quindi è bene far in modo che questa consumi la minor potenza possibile (in modo che, accostando questo comportamento all’enfatizzazione del computing, ovvero del preprocessing che semplifica i dati da trasmettere “filtrando” solo quelli fondamentali, si risparmi potenza e quindi batteria). E’ quindi necessario definire alcuni protocolli di comunicazione.



Gli smart objects sono abilitati grazie alla comunicazione a vedere, sentire, pensare, parlare, etc.. per prendere delle decisioni. Vi sono caratteristiche comuni degli smart objects nell'IoT, tra cui:

1. **Ubiquità:** accessibilità possibile ovunque si trovi lo smart object; è impensabile avere uno smart object come un pc desktop: si hanno oggetti che sono distribuiti ovunque.
2. **Mobilità:** essendo ovunque, gli smart objects sono anche mobili.

Queste due principali caratteristiche sono quelle che ci guidano nello sviluppo di un corretto sistema di comunicazione.

Una prima affermazione, seppur banale, che possiamo fare è che questi objects non possono essere wired: devono, chiaramente, essere **wireless** per favorire, appunto, la mobilità. Quindi i protocolli che intendiamo introdurre sono dei **protocolli che si basano su connessioni wireless.**

Visto i grandi sprechi di potenza per la comunicazione necessitiamo di una connessione che non sia solo wireless ma che, inoltre, si occupi di una comunicazione che avvenga a bassi consumi. Parleremo quindi di **connessione wireless low power**

Le comunicazioni wireless "soffrono" intrinsecamente di molti problemi: esse sono solitamente poco affidabili per via di eventuali interferenze.

Quindi i protocolli che servono per stabilire queste connessioni sono orientati alla comunicazione wireless, ai bassi consumi e alle basse perdite di informazioni. Parliamo, infatti di protocolli che sono **Low-Power an Lossy Networks (LLNs).**
Le tecnologie wireless (fondamentali per ubiquità e mobilità) pongono delle sfide importanti:

- **Low power Reliable Wireless Network:** sono praticamente **aspetti legati alla comunicazione in senso stretto,** si devono stabilire protocolli wireless che vincano problemi quali il fading (ovvero l'attenuazione dei segnali), le interferenze (che possono essere di varia natura: i percorsi, le altre comunicazioni, gli ostacoli, rumori…), problemi energetici (di consumi)

- **Light protocols:** sono **aspetti legati ai protocolli per la comunicazione**: il fatto che molti nodi hanno bassa capacità di computazione e che non deve essere sprecata troppa energia per la comunicazione ci porta a dover orientare dei protocolli opportuni. Se un nodo ha risorse limitate (poco storage e scarso computing), se volessi connettere questo nodo ad internet dovrei utilizzare i protocolli per la connessione internet; tuttavia questi protocolli sono troppo pesanti per uno smart object: sono adatti a PC e telefoni che sono godono di risorse computazionali e di storage notevoli rispetto agli smart objects, si necessita quindi di utilizzare dei **light protocols:** protocolli light nati dalla rivisitazione dei protocolli TCP/IP già esistenti per il collegamento internet.
  I protocolli light si possono ben connettere ai protocolli classici della pila TCP/IP, parleremo dunque di **smart objects IP Compliant**: che si possono connettere ad internet.
  Un'altra alternativa sono **smart objects Non IP Compliant:** si parla di smart objects che utilizzano dei protocolli light ma quest'ultimi non si sposano bene con la suite TCP/IP, quindi non possono connettersi ad internet; questi smart objects sono i primi nati nel mondo dell'IoT, essi possono connettersi tra loro in determinati ambienti (come una rete di sensori) ma non ad internet; un esempio è ZigBee che, al di sopra del livello fisico, ha uno stack che non è assolutamente comparabile a quello TCP/IP, quindi non è IP Compliant. Quindi un sistema che utilizza come protocollo ZigBee è difficilmente integrabile in una smart home che si connette praticamente con tecnologia IP: l'unico metodo per integrare questo sistema nella smart home sarebbe quello di utilizzare un gateway che faccia da tramite tra ZigBee ed IP.

Il Gateway però non è la migliore soluzione perché per mettere in comunicazione il mondo IP Compliant e Non IP Compliant deve potersi interfacciarsi con entrambi, se uno di essi si aggiornasse, non sarebbe banale effettuare un simil aggiornamento sul gateway. Un esempio di tale gateway può essere lo smartphone che può interfacciare il Bluetooth (protocollo Non IP Compliant) con il WiFi (protocollo 802.1 che è IP Compliant).
Esistono molti protocolli Non IP Compliant (come il Bluetooth stesso) e questo segna una forte eterogeneità nei protocolli che a volte condividono un livello in comune (ad esempio quello fisico) ma non gli altri livelli superiori: non hanno uno stesso stack.
L'idea dunque è quella di far convivere più protocolli differenti e più tecnologie differenti nel mondo IoT.

Per una soluzione con protocolli IP Copliant si devono rivisitare quelli che sono i protocolli della pila TCP/IP: i protocolli TCP e UDP non possono essere implementati poiché sono troppo pesanti per gli smart objects, si necessita di un protocollo simile ma più leggero. A livello fisico non si possono utilizzare protocolli come Ethernet o l'802.3 per il WiFi perché consumano troppa energia, si necessita di una soluzione simile ma a basso consumo.

I protocolli si possono classificare come proprietari o standard:

- proprietari: WirelessHART (molto diffuso nell'ambito IoT come nell'automation e in applicazioni consumer)
- standard: WiFi, ZigBee

Chiaramente l'essere standard prescinde dall'essere IP Compliant o meno. Questo significa che un protocollo può essere reso standard con uno sviluppo standard di questo ma non per forza IP Compliant.

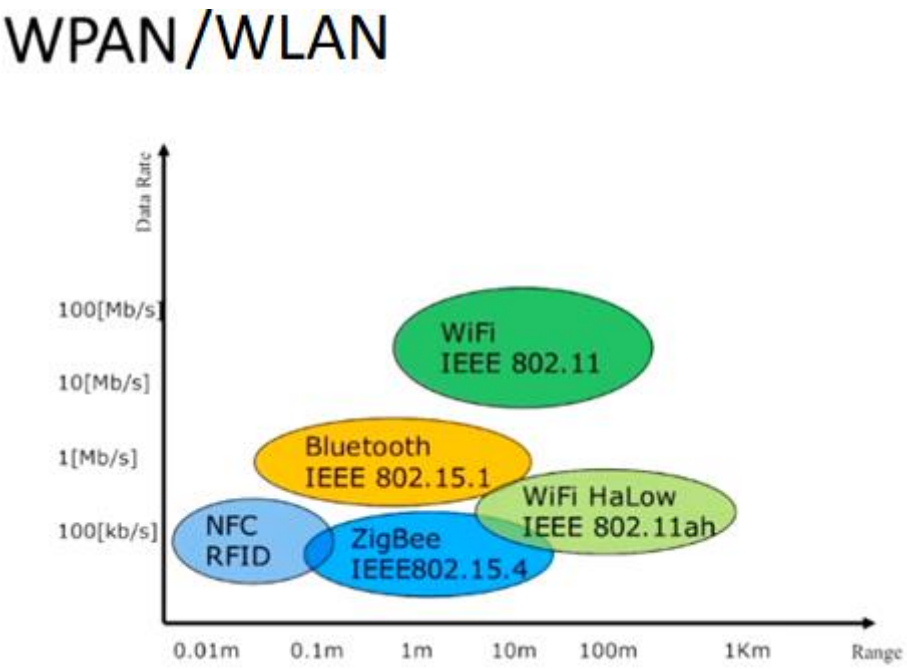
Si possono classificare in:

- Application specific: orientati verso una specifica applicazione (ZWAVE per home automation, HART per applicazioni industriali)
- Application agnostic: possono essere di largo uso in diversi scenari (6LowPAN/Thread per tutto)

Anche in questo caso non si parla di IP Compliant o meno

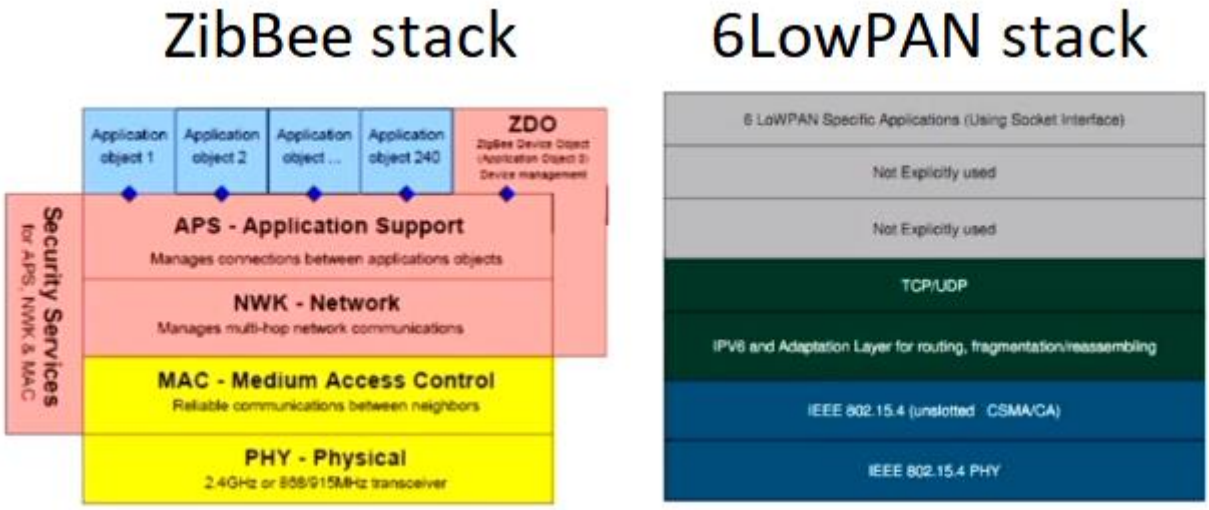
Un'altra classificazione vede, chiaramente, suddivisi i protocolli in **IP Compliant** e **Non IP Compliant.**
Lo standard IEEE cui faremo riferimento è l'**802.15 Wireless Personal Area Network (WPAN)** che ha definito il protocollo 802.15.4 di cui parleremo a breve.

In base alla distanza possiamo utilizzare diverse tecnologie di comunicazione:



Confronto tra IP Compliant e Non IP Compliant: Architetture ZigBee e 6LowPAN

6LowPAN è un protocollo IP Compliant, notiamo le differenze principali che ci sono tra la sua architettura e quella del protocollo ZigBee che è un protocollo Non IP Compliant, analizzando quindi tra queste due classi



I primi due livelli dello stack ZigBee sono identici a quelli dello stack di 6LowPAN: troviamo al livello più basso il livello fisico e sopra di esso il livello MAC (Medium Access Control), corrispondenti ai livelli stabiliti dalla IEEE nello standard 802.15.4 (implementato dal 6LowPAN). Oltre i primi due livelli non si evidenziano le differenze tra i due: 6LowPAN ha una pila strettamente fedele a quella TCP/IP (con tutte le eventuali restrizioni del caso sui consumi e sulle risorse allocate), mentre di tutt'altro aspetto è il protocollo ZigBee che implementa il livello Network per il collegamento su internet e successivamente il livello di Application Support dove entrano in gioco tutte le API per il supporto delle applicazioni (rientra il livello middleware, in cui vengono applicate le API per avere un livello più "standard" su cui lavorare).
Chiaramente, rifacendoci allo stack IP Compliant non necessiteremo di gateway addetti al

collegamento tra due “nature” differenti: quella IP compliant e quella Non IP compliant.

ZigBee è un protocollo molto utilizzato, quindi ne definiamo alcune caratteristiche:

- E’ formato da **un hardware ed un software low cost**
- Ha un limite di **range di trasmissione nell’ordine dei 10m**
- Supporta una **bassa latenza**
- Ha un’alta **efficienza energetica** (poca energia per il funzionamento)

Viene utilizzato per l’automation, per il fitness monitoring, in ambito industriale, in TV e lettori DVD, in mouse joystick e tastiere, in giocattoli, in ambiti di sicurezza, in controlli da remoto di sistemi di raffreddamento/riscaldamento.

L’unico **difetto** del ZigBee è il **non essere IP Compliant** poiché questo di traduce nel **non potersi collegare direttamente ad internet.**

Dall’altro lato possiamo osservare 6LowPAN che implementa lo standard 802.15.4 della IEEE. Questo è uno standard per protocolli che vengono detti **low-rate wireless private area net-works (LR-WPAN)**; esso può supportare fino a 65 mila nodi e per questo motivo è già un buon candidato per essere un protocollo per l’IoT visto l’elevato numero di connessioni possibili.

Lo standard 802.15.4 - Essentials

Le entità (i dispositivi) previste per questo protocollo possono essere di due tipi:

1. **Full Function Device (FFD):** sono dispositivi che hanno un insieme di capacità che consente più funzionalità.
   Un FFD **può:**
   - √ Trasmettere beacon frames (dei frame di controllo)
   - √ Comunicare con altri FFDs
   - √ Comunicare con gli RFD
   - √ Può fare il routing di frame
   - √ Può agire da PAN Coordinator
   - √ Può essere alimentato dalla rete elettrica

2. **Reduced Function Device (RFD):** dispositivi che hanno delle capacità ridotte: sono device con dei constraints maggiori.
   Un RFD **non può:**
   - × Fare il routing di frames
   - × Comunicare con altri RFDs
   - × Comunicare con FFD
   - × Spesso viene alimento da batteria

3. **PAN Coordinator:** un dispositivo FFD può diventare un PAN coordinator, ovvero il “responsabile” della Personal Area Networking, il responsabile della piccola rete che si forma con i nodi smart dell’utente. Chiaramente, vi può essere solo un PAN Coordinator.

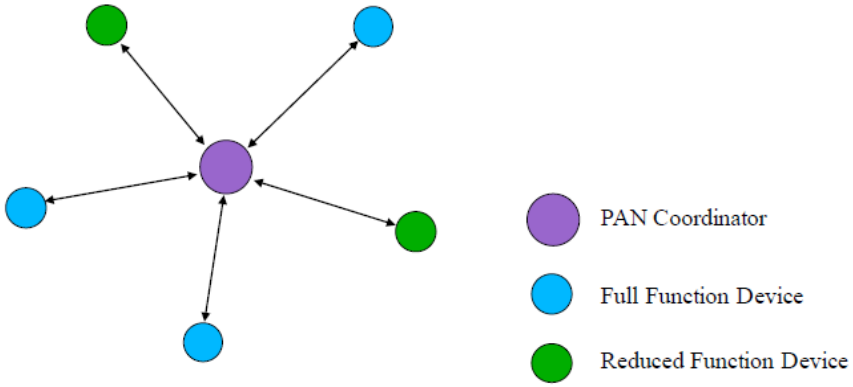
[Un PAN Coordinator potrebbe essere, ad esempio, l’Echo Dot di Amazon che controlla gli altri smart object della piccola rete PAN domestica e può gestire il traffico da e verso la smart home (la

PAN). Un dispositivo che potrebbe essere FFD, oltre chiaramente all'Echo Dot, è lo smartphone che può eseguire più compiti. Un RFD potrebbe essere qualsiasi strumento "passivo" comandabile da Alexa: una ciabatta intelligente, un relè intelligente, una lampada intelligente, etc.. che sono tutti strumenti adoperati per eseguire un unico scopo ben preciso]

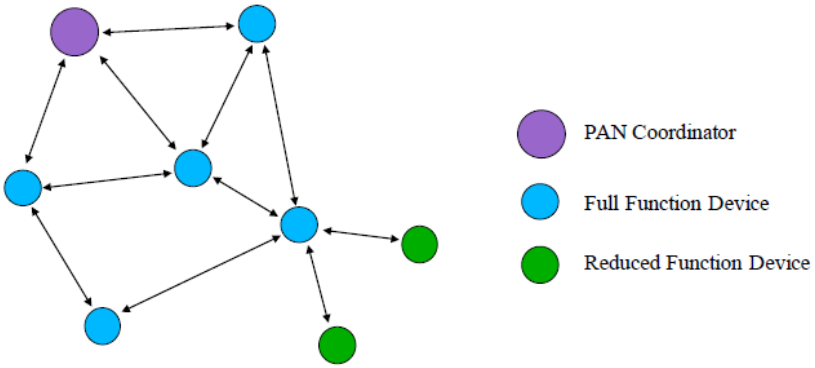
Topologie per lo standard 802.14.5

Vi sono diverse topologie che descrivono la connessione attraverso lo standard 802.14.3:

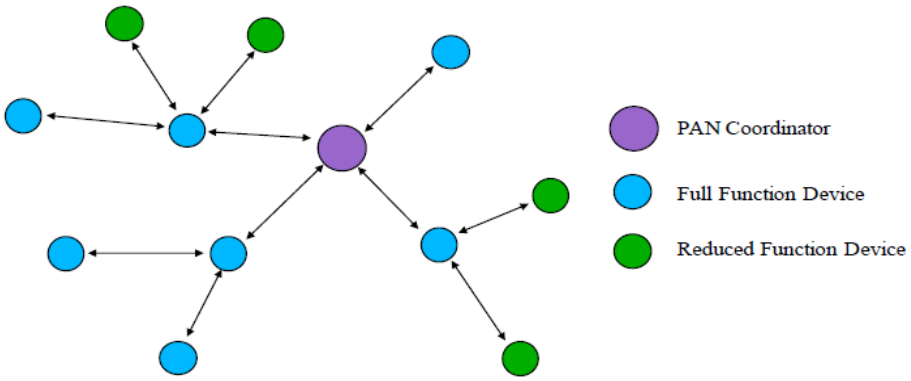
1. **Topologia a stella:** al centro della stella vi è il PAN Coordinator e connessi ad esso vi sono sia i Full Function Device. Qui i nodi non possono comunicare direttamente tra loro ma bisogna passare attraverso il PAN Coordinator.



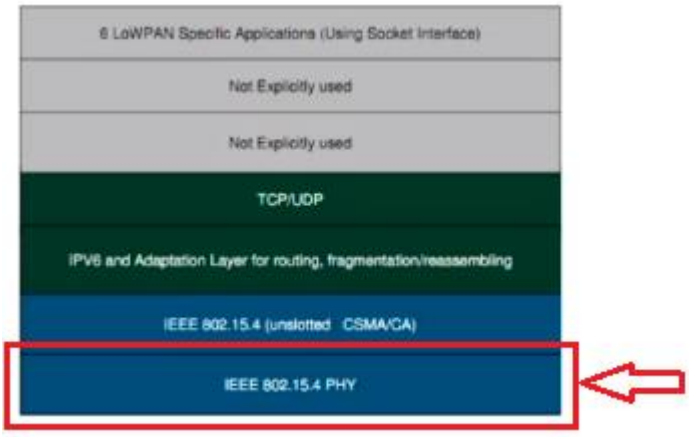
2. **Topologia MESH:** i nodi possono comunicare tra loro (purché siano FFD) e questi sono collegati al PAN Coordinator. Questa topologia è molto affermata nelle wireless sensors network: si parla di un numero consistente di sensori, ogni sensore può ricoprire una distanza limitata: utilizzando una topologia a stella non si riuscirebbe a ricoprire una vasta area; utilizzando una topologia mesh, situando i sensori a meno di 10m l'uno dall'altro (poiché 10m è la portata massima con questo protocollo" si può ricoprire un'area più vasta. Se si vogliono trasferire i dati per il PAN Coordinator, questi devono passare per vari FFD quindi è fondamentale il routing e poiché gli FFD hanno limiti di risorse e la trasmissione impatta sulla potenza e sulla batteria di questi (poiché la maggior parte si essi è alimentata a batteria, come i sensori) allora è importante seguire un protocollo di routing efficiente.



3. **Cluster tree:** consiste in alberi che tra loro si connettono, chiaramente vi sarà sempre solo un PAN Coordinator

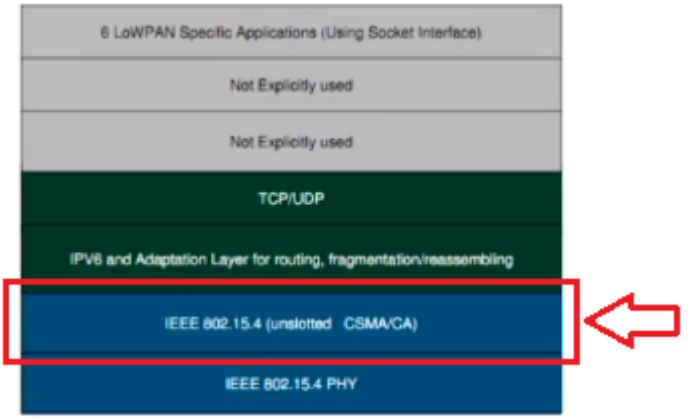


Standard IEEE 802.14.5: livello fisico



Essendo il canale di comunicazione wireless, si deve avere un sistema di energy detection (ED) all'interno del canale per capire se vi è passaggio di dati. Un Link Quality Indicator per indicare la qualità dei pacchetti ricevuti. Un altro elemento fondamentale è il Clear Channel Assesment (CCA) per il protocollo di accesso MAC.
Il livello fisico di occupa del **trasporto dei dati** e della **scelta della frequenza più opportuna**

Standard IEEE 802.14.5 livello MAC



Il livello MAC del IEEE gestisce delle **frame di crontrollo** chiamate **beacon**. Gestisce **l'accesso al canale** attraverso la tecnologia wireless, il **GTS** (Slot temporali per garantire determinate funzioni), etc…
Il protocollo a livello Mac ha **due modi di operare:**

1. **Beacon Enabled (Modalità Beacon):** normalmente viene utilizzata per la topologia star. In questa modalità il PAM Cooperator invia frames di controllo per la sincronizzazione che vengono chiamati beacons. Per l'accesso al canale condiviso viene utilizzata la tecnica **CSMA/CA slotted** (CA: Collision Avoidance, evita le collisioni, al contrario del CD). La tecnica di CSMA non sono a banda garantita perché in caso di collisioni vi potrebbero essere delle ritrasmissioni; inoltre è una tecnica molto efficiente quando vi è basso traffico perché si abbassa la probabilità di collisione e si ha un troughput migliore. Aumentando la quantità di nodi aumenta la probabilità di collisione e quindi le performance peggiorano.
2. **Beacon Disabled:** funziona anche questa modalità con tecnica CSMA/CA slotted ma non vi sono beacon frames.

L'accesso ad un canale condiviso soffre di alcune problematiche tra cui l'accesso a quest'ultimo.

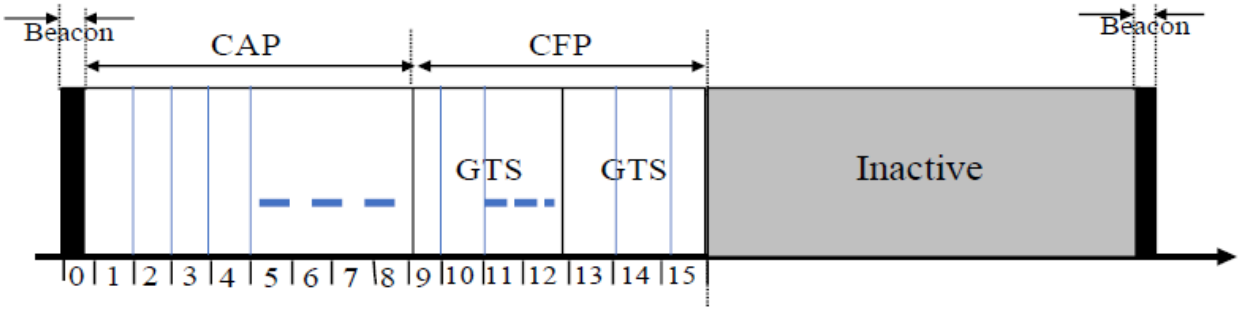
Per affrontare questo problema vi sono sostanzialmente due soluzioni:

1. Accesso Schedulato: può trasmettere solo chi ha il token, si basa principalmente su schemi polling e ha uno schema di scheduling centralizzato ("ti dico quando puoi parlare"). Alcuni protocolli famosi che adoperano questa soluzione: **Bluetooth, GSM**
   Vantaggi: performance "garantite"
   Svantaggi: Coordinazione richiesta (nodo centrale, sincronizzazione..)
2. Accesso Randomico: il nodo trasmette ma deve essere pronto alla possibilità che avvenga una collisione e che quindi debba poter ritrasmettere i dati ("Decidi quando parlare ma devi essere pronto a recuperare qualora vi fosse una collisione). Questa soluzione è quella adottata dal meccanismo **CSMA/CA: una volta che viene captata la collisione, il nodo dovrà aspettare un tempo casuale per trasmettere**, si dice che questo meccanismo è basato su **random retrasmission delay** (ritardo casuale di ritrasmissione). Così facendo, se il traffico non è elevato, si diminuisce la probabilità che vi siano delle collisioni perché i tempi di attesa sono casuali.
   Alcuni protocolli famosi che adoperano questa soluzione: **WiFi DCF, Ethernet.**
   Vantaggi: Facile da implementare.
   Accesso opportunistico alle risorse.
   Svantaggi: Performance solo "statisticamente garantite".
   Scarse performance all'aumentare del traffico.

La risorsa critica da condividere è il tempo: quando si ha una trasmissione sullo stesso slot temporale di ha una collisione. Nel caso del protocollo 802.15.4 la soluzione è un mix tra accesso schedulato e accesso casuale: **l'accesso schedulato viene implementato attraverso il PAN Coordinator in beacon mode** e **l'accesso casuale spetta agli RFDs e gli FFDs mediante CSMA/CA.**

Beacon enabled mode (Tree Topology): [Topologia a stella]

Il PAN Cooperator in bacon mode genera una **super frame (beacon)** così formata:



La super frame è slotted: suddivisa in slot temporali, essa è delimitata da due slot: due **beacon**, un particolare frame di controllo;
la super frame è suddivisa in due macro gruppi: CAP (contention accessed period) formato da 16 slot temporali, l'accesso a questi slot temporali (appartenenti a CAP) avviene con **CSMA/CA,** quindi vi possono essere collisioni per contendersi gli slot.
Vi è poi il gruppo CFP: qui vengono generati dei blocchi contenenti slot, dei "pacchetti di slot" chiamati **GTS (Guranteed Time Slot)** in cui vengono trasportati dati che non devono subire alcuna collisione e che possono attendere il proprio turno affinché possano essere trasmessi il **PAM Coordinator assegna ai nodi che devono trasmettere un'informazione senza alcuna collisione uno slot del blocco GTS.**

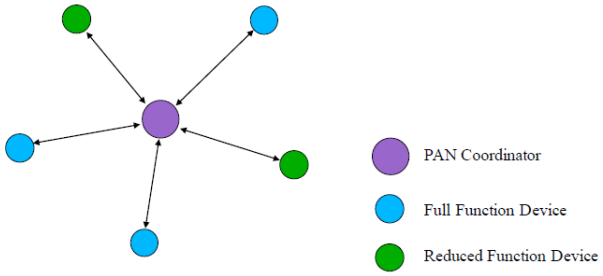
Concentriamo l'attenzione sulla contesa per gli slot CAP; iniziamo descrivendo tre parametri per la gestione della contesa:

- **NB: numero di richiesta di backoff**, quando si ha una collisione parte un periodo di backoff randomico; questo parametro conta il numero di volte che il nodo è andato in backoff.
- **CW: contenion window,** trascorso il tempo di backoff il nodo deve aspettare un numero di time slotprima di trasmettere. **CW ci indica il numero di time slot da attendere prima che un nodo possa ritrasmettere.**
- **BE:** parametro utilizzato per rendere randomico il periodo di backoff, il tempo da aspettare è pari a $2^{BE\ -1}$

Chiariti i parametri, parliamo **funzionamento del protocollo:**
Quando un nodo vuole trasmettere, non va a verificare direttamente che il CCA sia nello stato di Idle (che indica che il canale è libero) ma fa partire un **periodo di backoff** (come se ci fosse stata una collisione), avviando quindi il **timer di backoff** con un periodo random scelto nell'intervallo [0 - 2^{BE-1}] [NB: il numero è casuale ma è sincronizzato con il time slot, si parla infatti di numero di time slot che trascorrono]. **Scaduto il timer il nodo si accerta se può trasmettere: fa passare un numero di slot pari a CW**, che solitamente è inizializzato pari a due: il nodo fa passare due slot temporali e al terzo slot trasmette.
Se non si trova uno slot libero si fa ripartire il timer di backoff. Se invece lo slot è libero e qualcuno lo occupa prima del nodo in questione allora quest'ultimo si rimetterà in attesa avviando il timer di backoff. Quindi: **il nodo fa partire inizialmente il timer di backoff, che avrà un valore temporale randomico tra 0 e 2^{BE-1} e ne attende la terminazione, dopo di che attende CW slot temporali. Se trova degli slot temporali liberi trasmette, altrimenti si rimette in attesa avviando nuovamente il timer di backoff. Se per un numero elevato di volte non si trovano slot liberi, si rinuncia e non si trasmette**. Per sapere se si è provato troppe volte a trasmettere vi è la variabile NB che viene incrementata fino ad un valore massimo, raggiunto il valore massimo si rinuncia a trasmettere.

- **Trasferimento dati tra Device e PAN Coordinator: Modalità Beacon Enabled**

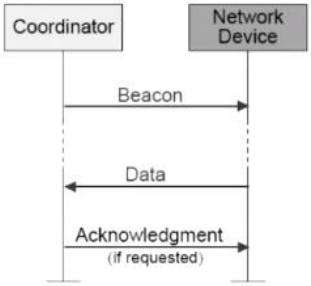
In una topologia a stella i device possono trasmettere solo al PAN Coordinator. Vediamo allora come avviene la trasmissione:



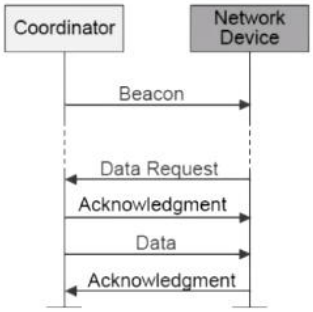
- **Trasmissione dal device al coordinator:** il coordinator può confermare il fatto che la trasmissione è avvenuta con successo attraverso un ACK nello stesso time slot: può capitare che due dispositivi inviano **contemporaneamente** i dati, dando vita ad una collisione. Il device chiede allora al il coordinator di mandare un **ACK di corretta ricezione**, l'assenza di ACK significa che la trasmissione non è andata a buon fine
- **Trasmissione dal coordinator al device:** il coordinator può inviare dati al device ma non lo fa attraverso gli slot addetti ai device (quelli del gruppo CPA) ma lo fa attraverso il **frame beacon: il coordinator utilizza il beacon frame per notificare al device l'intenzione di trasmettere qualcosa.** Quando un device riceve la "notifica" di trasmissione da parte del coordinator attraverso il beacon frame, prova a collegarsi ad uno slot libero e, una volta che ci riesce, manda una **data request** in questo slot per comunicate al coordinator che è pronto a ricevere (questa richiesta viene effettuata con il CSMA/CA). Dopo di che il coordinator sarà pronto a comunicare, attraverso uno slot libero, i dati al device. Una volta che riceve i dati il device può inviare un ACK di conferma.

Per il corretto funzionamento, **gli ACK devono essere trasmessi all'interno del CAP** (gruppo di slot per CSMA/CA)

Il backoff, come già accennato, deve essere proporzionale alla durata di un time slot di un super frame e deve essere sincronizzato ad esso.
Il CSMA/CA (Accesso in contesa) non viene utilizzato dal PAN Coordinator per mandare il beacon e gli ACK; gli ACK devono essere inviati all'interno dello stesso gruppo di slot CAP, quindi **la durata dello slot temporale è pari alla durata per inviare il dato più un "extra time" per ricevere l'ACK.** In caso di collisione l'ACK non viene ricevuto poiché i dati sono corrotti, quindi il nodo deve ritrasmettere.



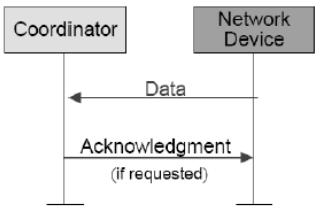
From device



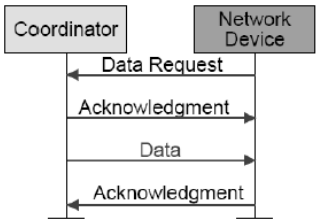
From Coordinator

❖ **Trasferimento dati tra Device e PAN Coordinator: Modalità Beacon Disabled**

Quando non vi è la beacon frame, il protocollo si semplifica: utilizzando il protocollo CSMA/CA, il nodo accede e fa partire il backoff. Attende il tempo randomico e quando scade il timer e si verifica se il canale è libero [CCA è Idle] e si trasmette, senza attendere che passano CW slot temporali. Altrimenti di attende, incrementando NB.
[Nella modalità che non è slotted non si ha il CW]



From device



From coordinator

Possiamo osservare i format del Frame del livello MAC nello standard 802.15.4:

MAC Sublayer

Frame Format

Octets: 2	1	0/2	0/2/8	0/2	0/2/8	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address	Auxiliary Security Header	Frame Payload	FCS
		Addressing fields						
MHR							MAC Payload	MFR

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame Type	Security Enabled	Frame Pending	Ack. Request	PAN ID Compression	Reserved	Dest. Addressing Mode	Frame Version	Source Addressing Mode

Beacon Frame format

Octets: 2	1	4/10	0/5/6/10/14	2	variable	variable	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS fields (Figure 45)	Pending address fields (Figure 46)	Beacon Payload	FCS
MHR				MAC Payload				MFR

Octets: 1	variable
Pending Address Specification	Address List

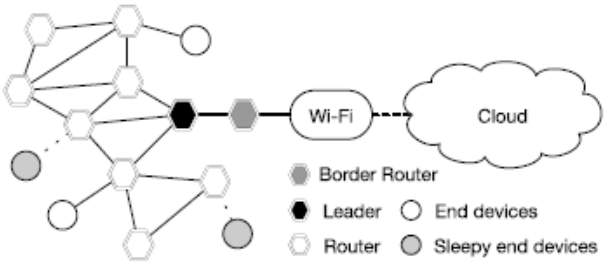
Octets: 1	0/1	variable
GTS Specification	GTS Directions	GTS List

Dell'IEEE 802.15.4 sono nate varie estensioni, ognuna per migliorare qualche aspetto del protocollo originale o per aumentare alcuni parametri a seconda dello scenario di applicazione.
Un esempio è l'IEE 802.15.4 su cui si basa la **Wireless Smart Utility Network (Wi-SUN)**, inizializzata dal Giappone per diverse funzioni smart come la misurazione del gas.

Protocollo Thread (IP Compliant Stack)

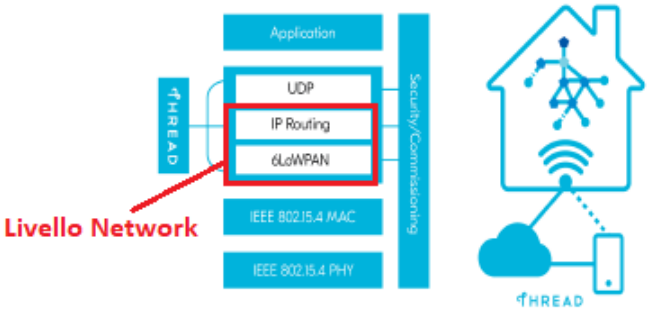
Il protocollo thread è un protocollo IP Compliant, significa che si può sposare bene con il funzionamento dello standard TCP/IP e quindi permette la comunicazione attraverso la rete internet.

Questo protocollo è costruito sulla base dei due livelli dello standard IEEE 802.15.4: il livello fisico ed il livello MAC ed opera alla frequenza di 2.4 GHz. Si basa su una tecnologia finalizzata all'ottimizzazione per la bassa latenza (meno di 100ms), le applicazioni networking come la gestione di energia, di controllo degli accessi etc…, per la smart home o più in generale per le smart buildings.



La topologia proposta da Thread è una topologia Mesh: il border router, equipaggiato con altri end devices può comunicare con il WiFi che comunica con il cloud.

Nel livello network esso utilizza l'insieme dei protocolli di IP Routing e 6LowPan che è praticamente il corrispondente protocollo IP per l'IoT. Utilizza come protocollo di trasporto UDP poiché esso è più leggero del protocollo di trasporto TCP dato che esso è connectionless (non deve stabilire una connessione prima di trasmettere i dati, andando però incontro alla perdita dei pacchetti)

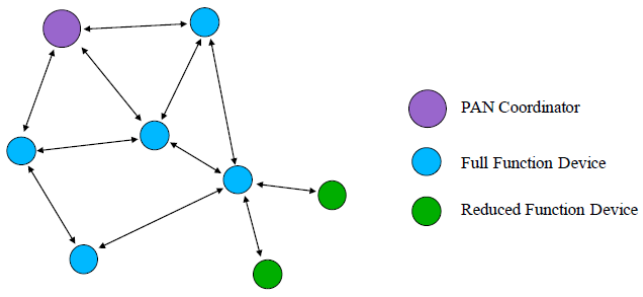


A livello applicativo si possono utilizzare i protocolli **COAP: Constrained Application Protocol** ed **MQTT: Message Queue Telemetry Transport** che funzionano bene con l'architettura (semplificata) TCP/IP [E non abbiamo quindi bisogno di utilizzare un gateway per l'adattamento]

Protocollo 802.11AH Low Power WiFi

Il protocollo 802.11 è il protocollo WiFi, nel 2013 è stato sviluppato, prendendo in considerazione quest'ultimo il protocollo Low Power Wifi: il protocollo 802.11AH. Questo protocollo è quindi utile nell'IoT per i bassi consumi. Si noti inoltre che il protocollo consente di connettere i dispositivi non più in una PAN (Personal Area Networking) ma in una rete più grande: una WLAN a tutti gli effetti (dato che il protocollo è comunque una rivisitazione dell'802.11 per le WLAN).

Lavorando, ad esempio, con il protocollo Thread si utilizza una topologia mesh: la mesh serve per far sì che si copri una vasta area con più nodi (dato che ogni nodo può stare al massimo a 10m di distanza da un altro), infatti visto il piccolo range a disposizione per la connessione, se si deve coprire una distanza maggiore il protocollo IEEE 802.15, che sia esso nella versione Thread o in qualsiasi altra versione (come ZigBee), entra in crisi; ecco perché è fondamentale l'implementazione di una topologia mesh.



Questo problema viene ovviato dal protocollo 802.11AH perché questo protocollo copre un'area geografica maggiore dato che è un'innovazione del protocollo 802.11 (quello per la WLAN).

Si noti che, a differenza del protocollo 802.15, le frequenze stanno **sotto 1 GHz** questa è stata una

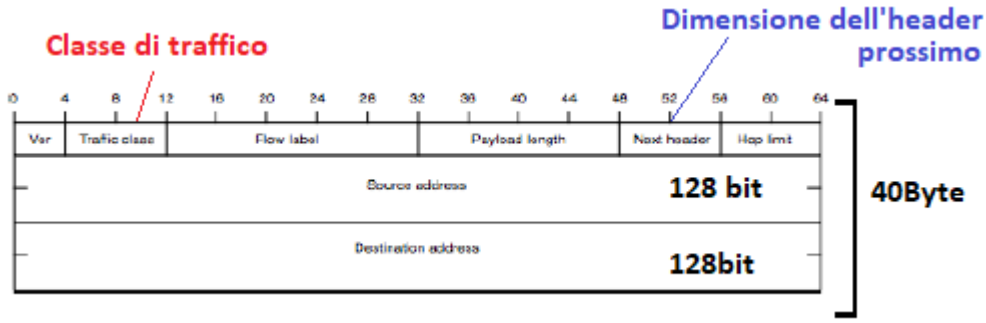
scelta strategica: siccome le tecnologie wireless utilizzano perlopiù le bande da 2.4 GHz, quest'ultima sono occupate dai segnali circolanti in queste reti wireless, utilizzando delle frequenze che stanno al di sotto si evitano delle possibili interferenze con questi.
Inoltre, basse frequenze sono sinonimo di maggior penetrazione, quindi possono essere superati gli ostacoli più facilmente.
Si pone come un ottimo candidato per smart homes, smart cities, industrial automation, etc…
[Grazie ad un 802.11AH uno smart watch potrebbe connettersi direttamente su internet senza dover connettersi necessariamente allo smartphone che fa da tramite e questo è possibile grazie al fatto che il protocollo è low power].
Il protocollo permette l'istallazione di **6000 nodi.**

Vien spontaneo chiedersi: ma quando si parla di Internet of Things, come si fa ad integrare gli smart objects su internet?
Nei nodi con scarse risorse potrebbe essere complicato integrare il funzionamento di una suite di un protocollo come quello TCP/IP. Da questo problema nasce il protocollo **6LoWPAN** e un protocollo di routing che prende il nome di **ROLL: Routing Over Low-Power and Lossy**

Protocollo 6LoWPAN

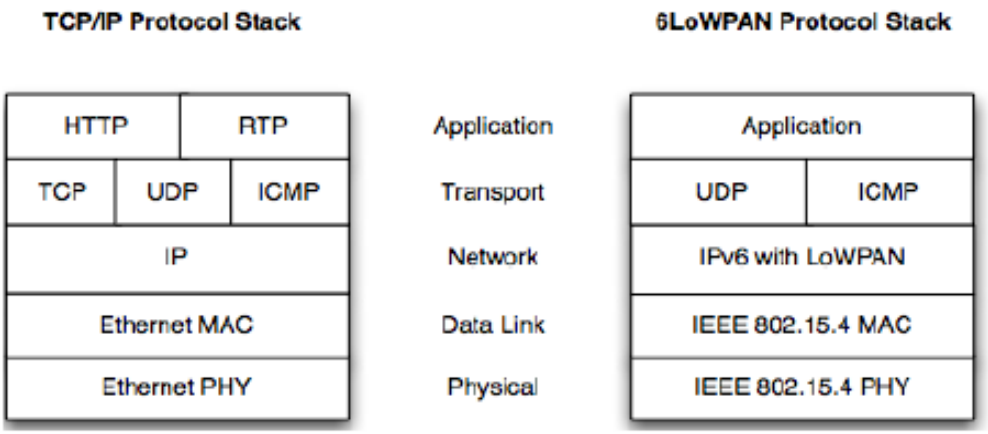
Il protocollo 6LoWPAN è un "livello di adattamento" che permette di utilizzare i frame di livello 3 che fa riferimento a IPv6 (non IPv4 perché il notevole incremento dei dispositivi connessi sui internet sta portando all'esaurimento degli indirizzi IPv4) nei nodi. In poche parole**, il protocollo 6LoWPAN è un livello di adattamento che permette di unire la "pesantezza" di IPv6 con le scarse risorse dei nodi IoT, unendo il protocollo 802.15.4 (il MAC Layer) con il protocollo IPv6.**

Il modello di un indirizzo IPv6 è il seguente:



Esso è troppo grande per un nodo smart che ha delle risorse limitate, quindi questo deve essere adattato dal 6LoWPAN.

Osserviamo le differenze tra lo stack del protocollo TCP/IP e quello del protocollo 6LoWPAN (che è praticamente una sua versione semplicificata):



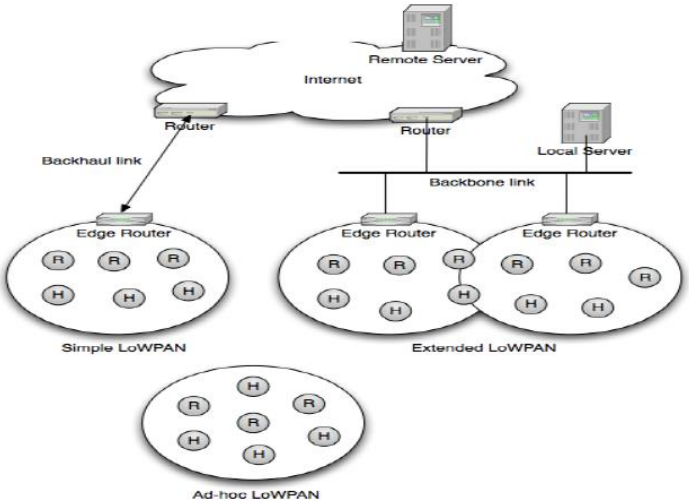
I livelli Ethernet fisico e MAC sono stati sostituiti con i protocolli IEEE 802.15.4, rispettivamente Fisico e MAC.
Il protocollo IP è stato rivisitato, mantenendo le proprie caratteristiche ma semplificandone le dimensioni con IPv6 LoWPAN (i livelli superiori vedranno come se esso fosse IPv6)
Vengono mantenuti UDP ed ICMP e al di sopra di essi vi è il generico livello di applicazione, qui vi sono protocolli che dipendono dall'applicazione che viene utilizzata.

Il protocollo 6LoWPAN permette di connettere un nodo wireless a bassa potenza con IPv6, permettendo quello che viene definito **Wireless Embedded Internet**: un oggetto con scarse risorse di computazione e memorizzazione può connettersi ad internet.ù

Il 6LoWPAN:
- √ È di facile implementazione poiché si conosce già la pila TCP/IP
- √ Ha un'integrazione su internet trasparente dato che utilizza uno stack semplificato del TCP/IP
- √ È scalabile
- √ Permette l'utilizzo di socket API
- √ Minimizza l'utilizzo di codice e di memoria
- √ È integra il dispositivo direttamente su internet.

Topologia di rete per protocollo 6LoWPAN

Notiamo la topologia di rete per il protocollo 6LoWPAN:



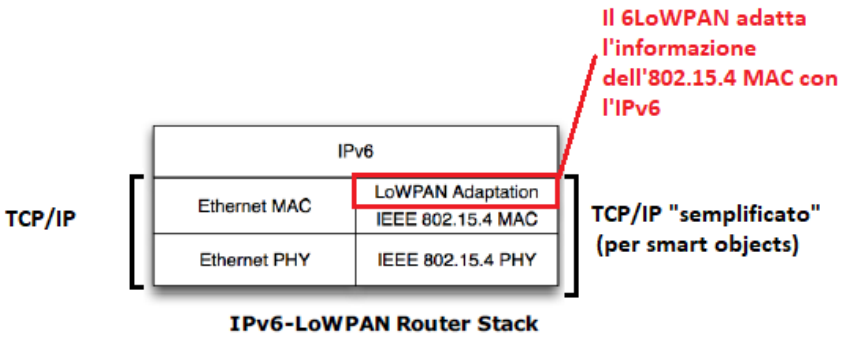
Viene chiamata **stub network**, è caratterizzata dal fatto che ogni nodo appartenente ad una PAN (Personal Area Networking) viene collegato ad un Edge Router che sta al confine della PAN stessa. Questo router sarà poi quello che si connetterà ad internet. Si possono avere:
3. **Simple LoWPAN**: con un singolo Edge Router
4. **Extended LoWPAN**: con più Edge Router
5. **Ad-hoc LoWPAN**: senza alcun Edge Router

Adattamento del protocollo 802.15.4 MAC con IPv6: utilizzo di 6LoWPAN

Vi sono alcuni problemi quando si prova ad integrare uno smart object su internet con IPv6:

- × Massima unità di trasmissione
- × Protocolli di applicazione
- × Interconnettività con IPv4
- × Firewall e NAT
- × Sicurezza

Abbiamo detto che l'obiettivo è quello di far in modo che gli smart objects possano utilizzare IPv6 per comunicare su internet.
Utilizzando 6LoWPAN è possibile **adattare l'informazione proveniente dal livello MAC del protocollo 802.15.4 al protocollo standard per internet IPv6.**



Per adattare il protocollo IPv6 per smart objects si **deve comprimere l'header di IPv6, portandolo da 40byte a 2byte,** si noti però che questo non è sempre possibile**.** La massima unità di trasmissione (MTU) di un frame di Ipv6 è di 1280 byte, mentre l'MTU per protocollo 802.15.4 è di 127 byte, **utilizzando un frame IPv6 si utilizzerebbero 40byte che sarebbe una grande porzione della dimensione massima per 802.15.4** quindi la compressione è fondamentale**.**

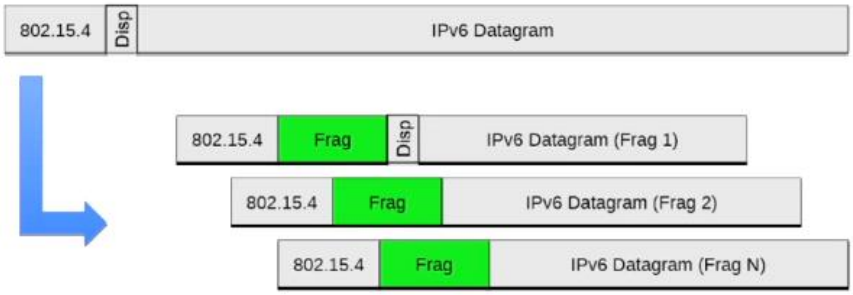
Nel MAC frame 802.15.4 è a 64-bit, tuttavia quando la rete è semplice e si ha un PAN Coordinator all'interno, di può assegnare ad esso un indirizzamento a 16bit che vale solo all'interno della PAN per alleggerire le informazioni trasmesse e quindi risparmiare potenza.

Vi sono dei "trucchetti" per la compressione dell'header di IPv6:

- ➢ Si utilizza un byte che indica la versione Ipv6, dato che utilizziamo sempre e solo Ipv6 per collegare smart objects, questo campo diventa superfluo e si può omettere.
- ➢ Il campo flow e traffic class non vengono spesso utilizzato quindi vengono "eliminati", settando queste label a zero.
- ➢ A volte non vengono sfruttati tutti i bit a disposizione, quindi questi possono essere compressi e si possono utilizzare solo quelli necessari.
- ➢ Il campo payload length si può ereditare dal livello 2 della pila: il datalink (collegamento) perché sono correlati quindi si omette perché sta già nel livello inferiore.
- ➢ Source address e Destination address sono formati da 128 bit, di questi 64 bit sono utilizzati come prefisso per identificare nodi appartenenti alla stessa rete. Ma se i nodi che appartengono alla stessa PAN, è inutile mantenere questi 64bit di prefisso poiché i nodi si potranno interfacciare solo nella piccola area di networking a disposizione (come la rete domestica). Altri 64 bit vengono utilizzati per l'interface ID, questo sarebbe l'indirizzo estratto dal livello MAC che ha cablato il costruttore, che è unico (sarebbe l'indirizzo di fabbrica statico e unico per i PC, assegnato dalla fabbrica). Questo indirizzo è contenuto nell'indirizzo MAC dell'802.15.4 e quindi, per questo motivo, non viene ritrasmesso.

Eseguendo queste semplificazioni si ottiene un livello di compressione efficiente.

Vi è inoltre il payload: quando un indirizzo IP, seppur frammentato, mantiene delle dimensioni elevate (ricordiamo che la Massima dimensione del datagram IP per IPv6 è di 1280) e supera la MTU (Maximum Transfer Unit) per l'802.15.4, pari a 127 bytes, si parla di **frammentazione in trasmissione e di assemblaggio in ricezione.**



Ma tutte le volte che si lavora con IoT, sopra al livello IP vi è il livello di applicazione, qui le applicazioni fanno in modo che si generi un che rientri nei 127 bytes del MAC 802.15.4, così da evitare la frammentazione poiché questa impatta nelle prestazioni.

Protocolli Non IP Compliant: Bluetooth Low Energy (BLE)

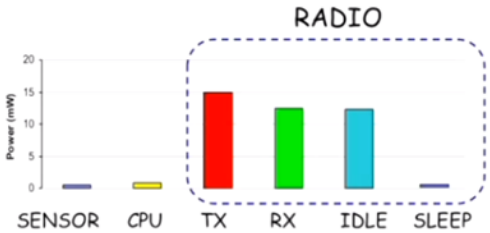
Come abbiamo già detto, alcuni protocolli possono non essere "compatibili" con IP Compliant, quindi necessità di una connettività mediante gateway specifici.

Bluetooth Low Energy (BLE)

Il bluetooth nasce nel 1999 e viene rilasciato nel 2001; nel 2020 gli viene assegnato lo standard IEEE 802.151. Questo è il Bluetooth Basic Rate and Enhanced Data Rate: **Bluetooth BR/EDR**, che è la prima versione di Bluetooth.
Successivamente ad esso è nato il Bluetooth Low Energy: BLE, il Bluetooth per connettere dispositivi con bassi consumi di energia: in poche parole, il Bluetooth per l'IoT. Essendo low energy non può trasmettere un grande flusso di dati come il BR/EDR. Questi due lavorano bene in coppia quindi nei dispositivi si può spesso trovare entrambe le versioni di Bluetooth: questo accade, ad esempio, negli smartphone.

La prima versione del **Bluetooth BR/EDR** condivideva flussi di dati in maniera asincrona. La versione successiva implementò la versione sincrona.
Il Bluetooth BR/EDR soffre di alcuni **difetti:**

- **Consumi energetici:** il nodo master effettua un continuo **polling** per vedere se vi sono nodi slave: esegue sempre un controllo per verificare che vi siano o meno nodi slave da connettere. Questo porta ad un alto consumo di energia poiché non vi è praticamente uno stato di Idle in cui viene consumata minor energia.



- **Scalabilità:** il numero di nodi che si possono interconnettere è poco elevato; permette la connessione di pochi dispositivi slave con il dispositivo master: permette una connessione con topologia a stella e che interconnette pochi dispositivi.

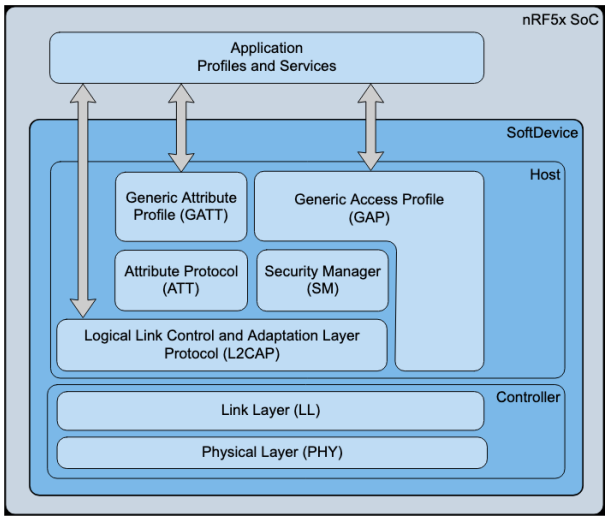
Il **Bluetooth BLE** è nato nel 2011 ed è caratterizzato da **basso throughput** e questo premia la dissipazione di potenza. Elenchiamone i **Vantaggi:**

- √ Utilizza uno **short range radio**,
- √ è caratterizzato da **consumi molto bassi**: con il solo Bluetooth BLE la batteria di un dispositivo potrebbe durare anni.
- √ Il **range che può ricoprire è di circa 100m**, circa 10 volte maggiore del BR/EDR
- √ è caratterizzato da una **latenza pari a 15 volte meno quella del BR/EDR.**
- √ La **potenza di trasmissione** può variare **da 10µW a 10mW.**

Queste caratteristiche portano il BLE ad essere un buon candidato come protocollo di comunicazione per l'IoT.

Le grandi distanze di copertura e i bassi consumi sono utili per scenari di comunicazione tra veicoli o tra reti di sensori.

Rispetto, ad esempio, a ZigBee i consumi energetici sono minori ed ha una trasmissione più efficiente.
Dalla struttura dello stack per protocollo BLE possiamo osservare che esso non compatibile con IP, che è quindi Non IP Compliant:



Possiamo vedere:

❖ Il **livello GATT** che fornisce dei servizi per l'adattamento dell'applicazione come le API

❖ L' **L2CAP** che si occupa di frammentare e di assemblare i pacchetti dati troppo grandi

❖ Il **livello Link** serve per lo stabilimento della connessione, il controllo del flusso dati, il controllo di errori, etc…

Come si può notare, nessuno di questi livelli è compatibile con lo stack della pila TCP/IP (semplificata per smart objects)

Quindi per far comunicare BLE con internet, si necessita di un gateway apposito (come lo smartphone)

Inizialmente, il BLE aveva un payload di 27 bytes, che si è incrementato nella versione 4.2 fino ad arrivare a 251 bytes.

Il **Bluetooth 5**, introdotto nel 2016, ha migliorato alcuni aspetti:

- ➢ **incremento del range di copertura:** ampliando le comunicazioni indoor, come quella building-to-building.**,**
- ➢ **dimensione dell'advertisement payload,**
- ➢ **miglioramento del throughput.**

Altri miglioramenti sono stati apportati, l'ultimo di essi è quello riguardante la **data rate (**la **"velocità" di trasmissione dei dati),** portando alla versione chiamata **BLE high-speed mode.**

Bluetooth Beacon

L'idea del Bluetooth Beacon è nata da Apple che ha creato la iBeacon, che propone di sfruttare i vantaggi del Bluetooth BLE che porta ad una trasmissione a bassi consumi.
Sfruttando i bassi consumi che consentono di avere un'alimentazione a batteria che dura anche anni, si sono creati dei dispositivi IoT (smart objects), chiamati appunto beacon, che possono essere attaccati a parete e che possono essere utilizzati, ad esempio, ad esempio per localizzazione interna (all'interno dei musei) o che possono essere integrati in altri scenari di vario tipo. Questi funzionano in modalità **periodical broadcast**, trasmettendo **piccoli parti di informazioni di massimo 31bytes,** quindi possono essere utilizzati per informazioni di piccola dimensione: una posizione, un dato proveniente da un sensore, etc…

Bluetooth 5: BLE Codec e FEC

Uno dei problemi principali dei dispositivi IoT è la **limitazione del range di connessione**, per questi problemi si può sfruttare una potenza in trasmissione maggiore (ma non è la soluzione ottima visto i problemi dei consumi elevati per la comunicazione), una soluzione valida che a volta si è l'utilizzo di una **topologia mesh:** collegando più dispositivi entro il range massimo di collegamento, mappando così la zona.

Per evitare l'utilizzo di troppi dispositivi, una tecnologia che ovvia a questo problema, che **aumenta il range di copertura senza aumentare la potenza di trasmissione** è la **FEC.** La FEC consente di coperture nel range di **1km/1.5km** (distanze che quasi avvicinano la rete ad essere una WLAN).

Quando vengono collegati due dispositivi con BLE si ha un'area che prende il nome di **connected region**, all'interno di questa area, nella trasmissione dei bit, si può avere un **errore sul bit (Bit Error Rate - BER) nell'ordine di 10^{-3} bits**, quest'ultima è accettabile: sarebbe auspicabile ottenere una BER minore (nell'ordine di 10^{-6}) ma nelle trasmissioni wireless consideriamo quest'ordine di grandezza sufficientemente basso.

Al di fuori di quest'area le onde continuano comunque a propagarsi, in cui questa BER però aumenta d'intensità (passando ad ordini di grandezza più elevati).
L'idea della tecnologia FEC è quella di **sacrificare il throughput per coprire più distanze:** viene codificato un singolo bit con più bit (ad esempio 2 o 8 bit), diminuendo la BER, creando una "ridondanza" di informazioni; così facendo, utilizzando 8 bits il throughput diventa $\frac{1}{8}$, con 2 bits si dimezza.

Con BLE 5 si è anche aumentata la velocità fino a 2MBps. Inoltre, nel 2017, è stata standardizzata e implementata una **versione di BLE chiamata mesh, che implementa una topologia BLE.** Una volta che viene supportata la topologia Mesh si può utilizzare questa risoluzione, evitando di utilizzare la tecnologia FEC, evitando di sacrificare quindi il throughput.

Bluetooth 5: Advertising

Il Bluetooth low energy supporta 40 canali fisici con banda nella frequenza di 2.4GHz, ognuna separata da 2MHz. La tecnologia Bluetooth definisce **due tipi di trasmissioni: trasmissione di dati e di advertising** con ben 3 canali dedicati all'advertising.
Qualsiasi dispositivo che implementi BLE (sia esso uno smart watch che trasmette informazioni più grandi e che mantiene la connessione Bluetooth o un beacon che trasmette informazioni più semplici), stabilisce una connessione, almeno inizialmente, sfruttando l'**advertising mode.**
L'advertising consente ai dispositivi di trasmettere delle informazioni in broadcast per comunicare le intenzioni dei dispositivi.

Il motivo dell'utilizzo di advertising è legato principalmente a due motivazioni:

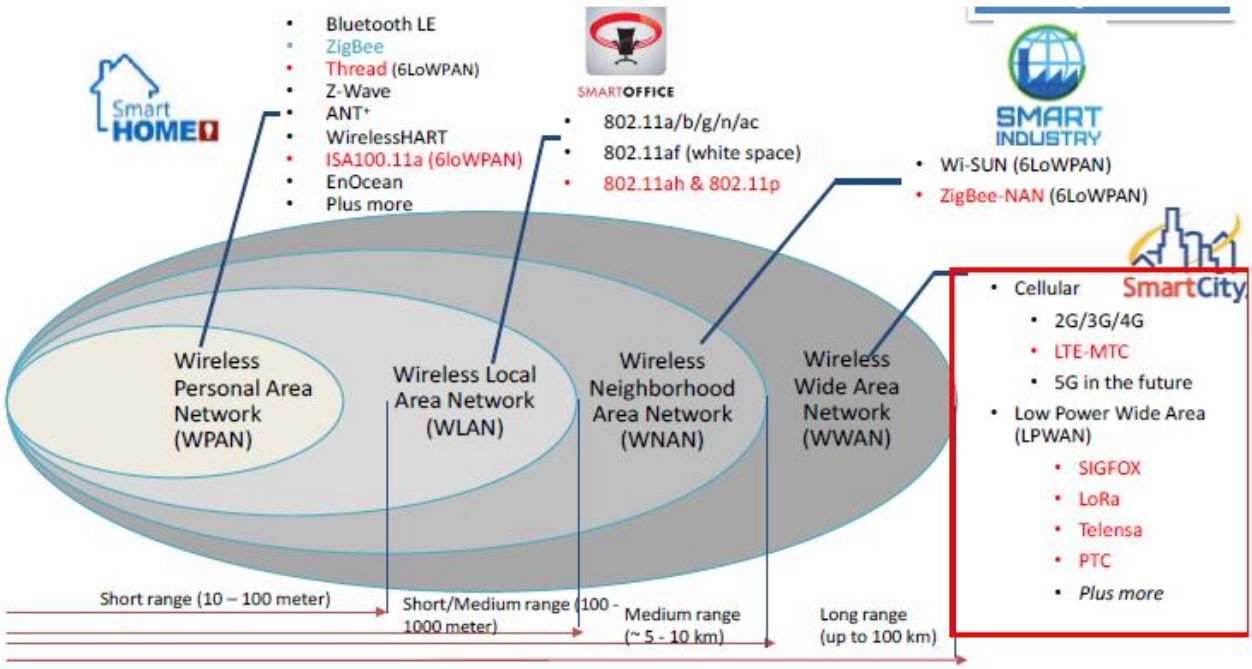
1. Stabilire una connessione bidirezionale con un altro dispositivo (ad esempio lo smart watch che si connette allo smartphone)
2. Iniziare un broadcast di informazioni senza stabilire altre connessioni con altri devices (ad esempio un beacon).

Possono essere inviati **4 tipi di advertising**:

1. **ADV_IND (Advertising Indications)**: un dispositivo periferico richiede la connessione ad un qualsiasi dispositivo centrale (esempio: uno smart watch che richiede connessione ad un qualsiasi dispositivo centrale)
2. **ADV_DIRECT_IND (Advertising Direct Indications)**: un dispositivo richiede la connessione ad un preciso dispositivo centrale
3. **ADV_NONCONN_IND (Advertising Non connectable Indications):** un dispositivo richiede una connessione unica, non permettendo altre connessioni Bluetooth con altri dispositivi. (esempio: un beacon richiede la connessione ad un dispositivo centrale)
4. **ADV_SCAN_IND (Advertising Scan Indications):** è simile all'ADV_NONCONN_IND ma con l'opzione aggiuntiva che il dispositivo centrale può chiedere informazioni in più al dispositivo periferico (beacon device).

[Le cuffie Bluetooth non possono utilizzare il BLE perché questo trasmette le informazioni in maniera asincrona, in questo modo avremmo un audio non sincronizzato. Per questo motivo si preferisce il Bluetooth BR/EDR]

Riportiamo la seguente panoramica per avere un'idea di come avvengono le connessioni a seconda della distanza:



Low Power Wide Area Networks (LPWAN)

Gli scenari in IoT possono essere vari: smart room, smart home, smart building, smart city… Gli esempi che abbiamo riportato sono, in scala, implementate in aree che sono sempre più grandi. Se provassimo a connettere un'intera città con le connessioni che sfruttiamo nelle smart home, avremmo che le tecnologie studiate finora entrerebbero in crisi poiché non riuscirebbero a ricoprire un'area così vasta.
Abbiamo visto che spesso una soluzione a questo problema è la topologia mesh, tuttavia questa tipologia sfrutta la trasmissione tra nodi e questo potrebbe essere problematico per la generazione di overhead, quindi questa soluzione è valida ma fino a certe distanze.
Esiste allora quelle che vengono chiamate Low Power Wide Area Network LPWAN che riescono a coprire fino a 100km;
queste reti sono tipicamente implementate con **topologia a stella.**
I dispositivi che vengono connessi con questi protocolli devono avere dei requisiti fondamentali:

- Piccole quantità di dati trasmessi: più si trasmette e più potenza si spreca **(Low data rates)**
- Batterie durature **(Long battery life)**
- Tipicamente non trasmettono e se lo fanno allora trasmettono un po' alla volta (2/3 volte al giorno)

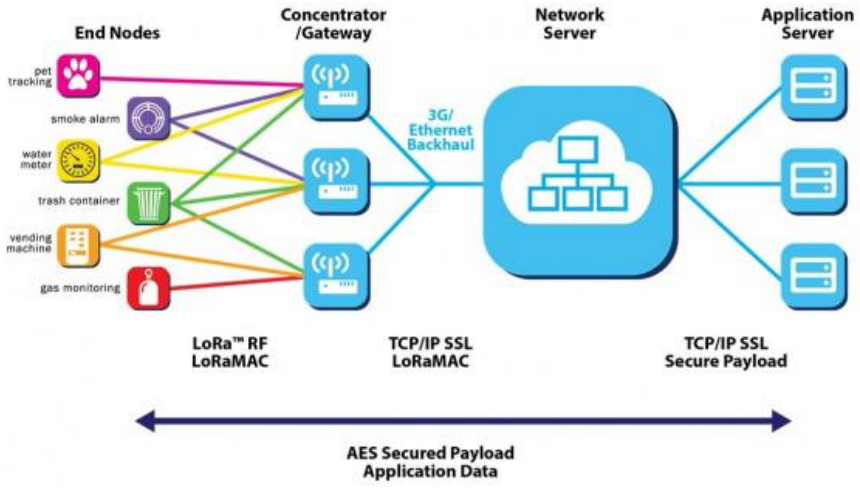
In questo ambito trovano applicazione:

- Street lighting (illuminazione stradale)
- Parking: sensori che segnalano se uno spazio è occupato o meno
- Pet Tracking
- Garbage collection (raccolta dei rifiuti)
- Sensori nell'ambito dell'agricoltura
- Forest fire detection: sistema per tracciare eventuali incendi nelle foreste

I protocolli che sono nati per queste tipologie di rete sono **LoRa** e **SIGFOX**

Protocollo LoRa:

LoRa (Long Range) è un protocollo per il collegamento Machine to Machine (M2M) che permette la connessione a grandi distanze, nell'ordine di **15/20 Km** con uno **sfruttamento della batteria garantito per 10 anni**.
È una protocollo low cost che è stato creato dalla LoRa Alliance che include membri importanti come IBM, Cisco, Semtech, etc…
Lavora con una banda di frequenze al disotto dei GHz: la banda va da 7.8 kHz a 500 KHz.



Un possibile scenario per l'utilizzo di LoRa prevede una serie di end nodes che trametton a distanza piccoli pezzi di dati a dei concentratori (o gateway) e questi possono trasmettere i dati verso la rete internet senza alcun problema

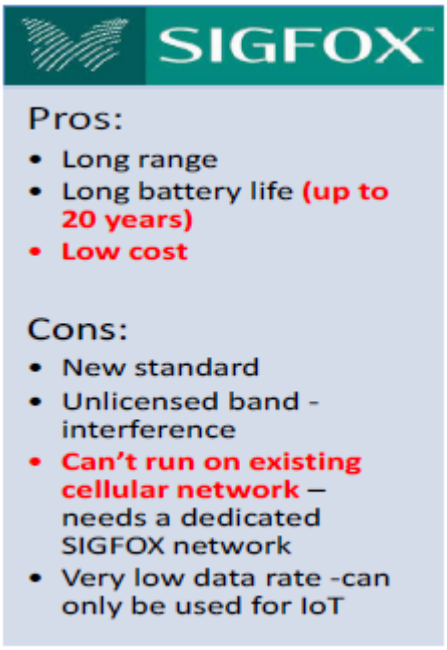
SIGFOX

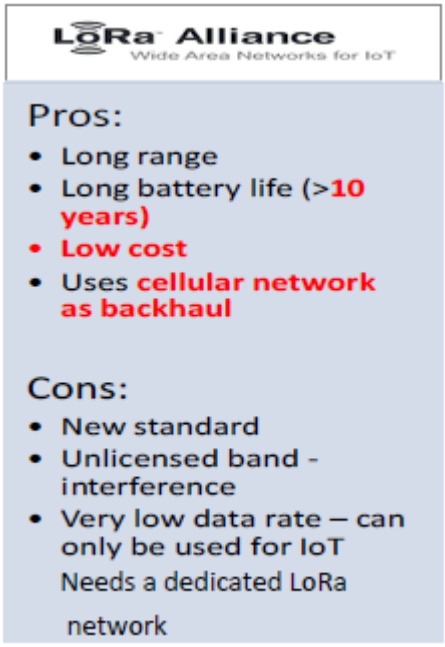
In contrapposizione a LoRa esiste SIGFOX, nato da una startup francese;
E' caratterizzato di un **ultra low throughput** per minimizzare i consumi: si parla di circa **100 bps** (un dispositivo può trasmettere da 0 a 140 messaggi al giorno ed ogni messaggio è di 12bytes).

La **durata della batteria** è preservata per **20 anni** e prevede un **range di comunicazione pari a 30 miglia per area rurale e 2-6 miglia per area urbana** (nelle aree urbane vi sono più interferenze).

I dispositivi richiedono un SIGFOX modem per connettersi alla rete SIFOX. Le applicazioni a cui SIGFOX è adatto potrebbero essere inerenti al pet tracking, smoke detector, etc…

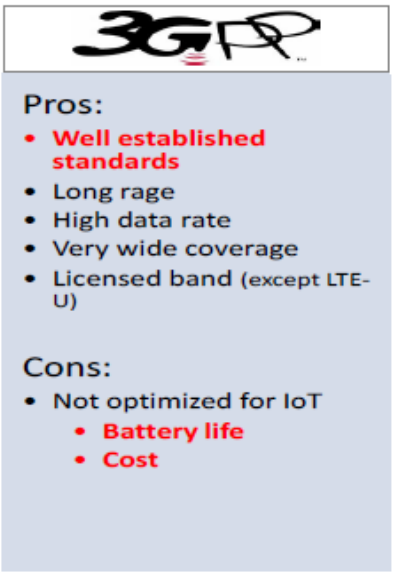
Mettiamo a confronto LoRA e SIGFOX:





[Un aspetto svantaggioso per SIGFOX è legato al fatto che la rete è proprietaria e quindi lo sviluppo è ancora un po' lento.]

Idealmente, si potrebbe anche utilizzare la rete cellulare, tuttavia questo implicherebbe l'adozione di una SIM per ogni dispositivo che dovrebbe connettersi; inoltre questa non è ottimizzata per l'IoT.



Standardizzazione per supportare l'IOT

Dal seguente grafico possiamo avere un'idea globale della standardizzazione della pila TCP/IP semplificata (suite protocollare) per favorire la comunicazione nel mondo IoT tra smart objects:

<table>
<tr><td colspan="2">Application Protocol</td><td>DDS</td><td>CoAP</td><td>AMQP</td><td>MQTT</td><td>MQTT-SN</td><td>XMPP</td><td>HTTP REST</td></tr>
<tr><td colspan="2">Service Discovery</td><td colspan="4">mDNS</td><td colspan="3">DNS-SD</td></tr>
<tr><td rowspan="4">Infrastructure Protocols</td><td>Routing Protocol</td><td colspan="7">RPL</td></tr>
<tr><td>Network Layer</td><td colspan="5">6LoWPAN</td><td colspan="2">IPv4/IPv6</td></tr>
<tr><td>Link Layer</td><td colspan="7">IEEE 802.15.4</td></tr>
<tr><td>Physical/ Device Layer</td><td colspan="2">LTE-A</td><td colspan="2">EPCglobal</td><td>IEEE 802.15.4</td><td colspan="2">Z-Wave</td></tr>
<tr><td colspan="2">Influential Protocols</td><td colspan="5">IEEE 1888.3, IPSec</td><td colspan="2">IEEE 1905.1</td></tr>
</table>

Come si può notare:

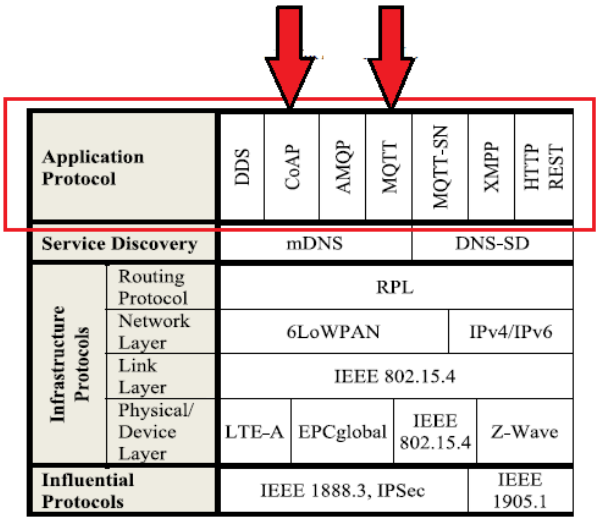
- A **livello fisico** e a **livello di collegamento (MAC)** possiamo notare il **protocollo IEE 802.15.4.**
- Al di sopra si poggia il **protocollo 6LoWPAN**. Il livello 6LoWPAN viene definito "livello di adattamento" e si colloca nel **livello di collegamento**; viene definito livello di adattamento perché adatta il protocollo sottostante: 802.15.2 definito per l'IoT con IPv6 della pila TCP/IP, per adattare quest'ultimo a dei dispositivi con risorse limitate, quali sono i nodi IoT.
- Al di sopra di 6LoWPAN vi è il **protocollo RPL**, che è un **protocollo di routing** per loss network e low power network, principi fondamentali per il mondo IoT.
- Sopra ad esso vi sono i livelli successivi, tra cui **l'application protocol**: l'insieme di tutti i protocolli applicativi utilizzati dalle varie applicazioni.

L'Application layer

Quando parliamo di livello applicativo, probabilmente il primo protocollo che ci viene in mente è l'HTTP. L'dea darebbe vincente poiché **si potrebbe connettere il nodo sul web, consentendo l'utilizzo di tutti gli standard ad esso associati**, tuttavia, questo protocollo è piuttosto complesso per uno smart node poiché **utilizza risorse troppo potenti.**

Per questo motivo sono nati una serie di protocolli: **DDS, CoAP, AMQP, MQTT, MQTT-SN, XMPP, HTTP REST…**

I protocolli per l'Application Layer che affronteremo sono CoAP e MQTT (contrassegnati in figura)

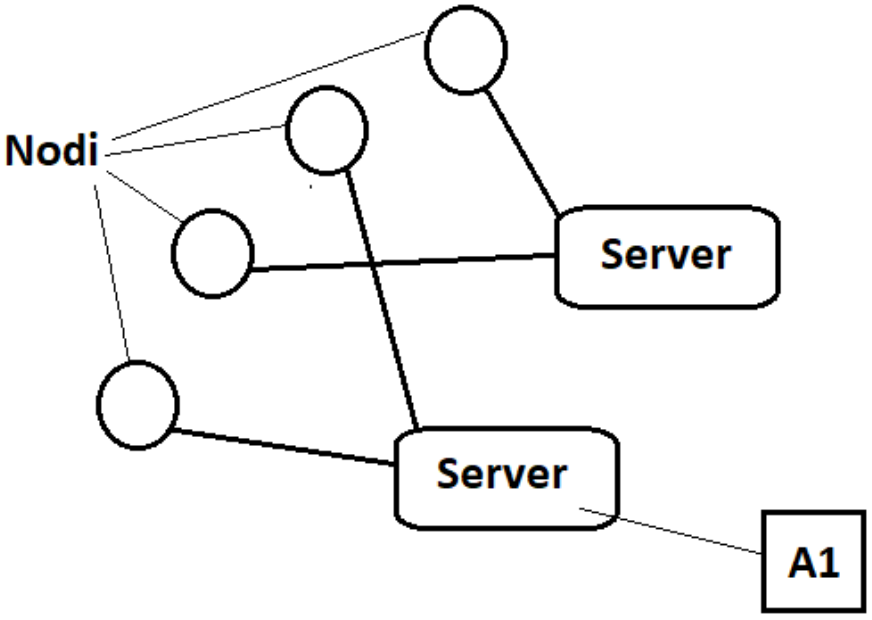


Il protocollo **COAP (Constrained Application)** ed il protocollo **MQTT (Message Queuing Telemetry Transport)** nascono proprio per applicazione constrained ovvero per **applicazioni che devono girare in device con risorse limitate**, quindi per nodi IoT.

Paradigmi di interazione

Questi due protocolli si possono basare sostanzialmente su due paradigmi di interazione:
Supponiamo vi siano dei nodi Iot e dei server connessi tra loro (ad esempio con la suite in cui vi è la 6LoWPAN). Ogni nodo ha dei sensori e sui server girano le applicazioni.
Le applicazioni, per svolgere il loro compito, hanno bisogno di acquisire i dati che vengono captati dai sensori nei vari nodi.
Supponendo che in un server vi sia l'applicazione A1, essa per acquisire i dati (che ricordiamo possono anche essere stati filtrati nella fase di preprocessing); può agire sostanzialmente in due modi:

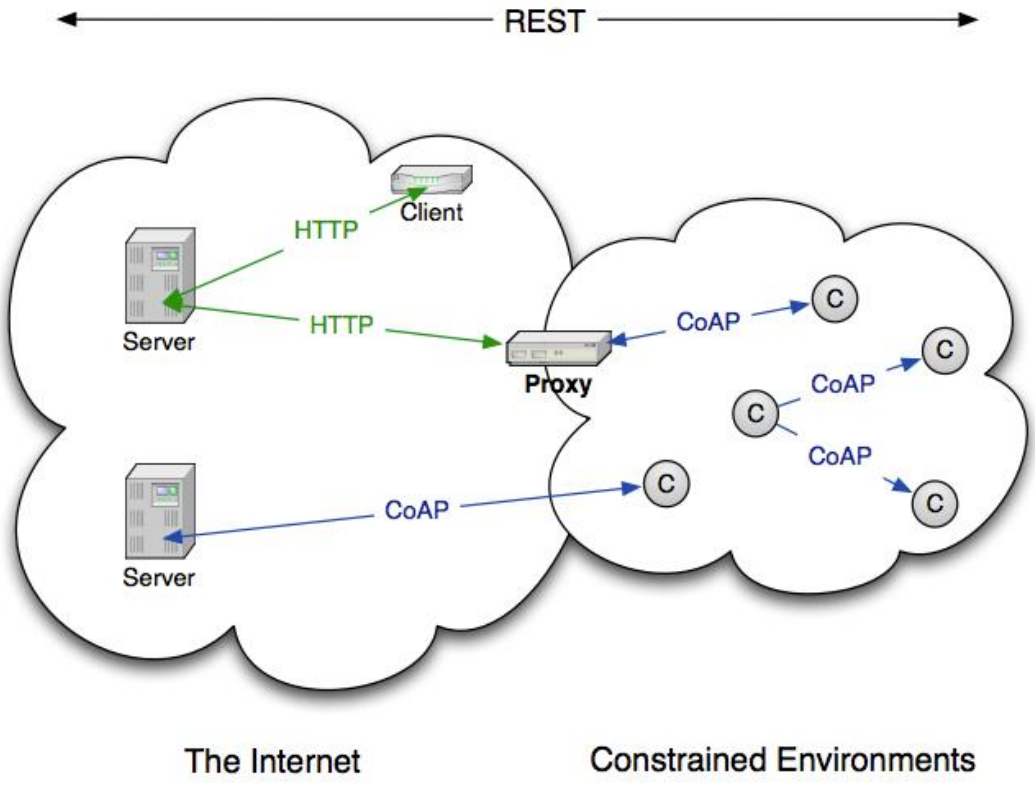
- **Polling:** Può **interrogare tutti i nodi**, ottenendo i dati dalle varie applicazioni;
- **Publisher/Subscriber:** In alternativa, potrebbe essere **il nodo che, attraverso il software che ne sta a capo, invia i dati all'applicazione ogni tot di tempo**;



Protocollo COAP

Il protocollo COAP si basa sulla tecnologia WEB (sulla tecnologia HTTP), è adatto ai nodi che si connettono tra loro con risorse limitate: microcontrollori a 8 bit e memoria limitata con collegamenti a bassa potenza.
Il problema basilare è che HTTP è troppo oneroso per i singoli nodi (non è un high protocol) poiché ha un header molto complesso e consuma molta banda, quindi COAP nasce basandosi su HTTP rendendo quest'ultimo adatto ai device con scarse risorse.
Le caratteristiche del COAP sono le seguenti:

- È un **Web protocol** (o web based): è adatto per il mondo IoT e per l'approccio Machine to Machine (e quest'ultimo chiaramente è compreso nell'IoT).
- Utilizza come **protocollo di trasporto UDP**, questo è più leggero del TCP essendo connectionless
- È un protocollo Client Server che implementa uno **scambio di messaggi asincrono** questp lo fa differire da http che è un protocollo che prevede uno scambio di messaggi sincrono (in cui ad una request corrisponde una response)
- Ha un **header leggero di 4byte** a differenza dell'HTTP che ha un header più pesante
- Le risorse vengono identificate attraverso l'utilizzo di **URI** (indirizzi addetti)
- Supporta **Proxy e Caching** che sono fondamentali nelle architetture Web.



COAP nasce per essere un protocollo **web based** e quindi deve essere in grado di comunicare con il protocollo HTTP.
Nell'immagine possiamo notare il Proxy che fa si che una richiesta attraverso un client HTTP possa arrivare ad server COAP oppure una richiesta COAP può essere inoltrata ad un server HTTP, in modo da rendere la trasmissione del tutto trasparente.
Vista la similitudine tra i metodi utilizzati da COAP e quelli utilizzati da HTTP (GET, POST, PUT, DELETE) non vi è una grande difficoltà per i Server Proxy lavorare con i due protocolli che comunicano tra loro.

Un messaggio COAP ha un header di soli 4 byte che vengono trasmessi sia nella response che nella request (il che lo rende più leggero di un messaggio HTTP), il messaggio viene identificato unicamente il messaggio da un **message ID**: quindi ad un unico messaggio di request corrisponde un unico messaggio di response.

Dato che la connessione è wireless si possono avere molte perdite dei dati per la caduta di connessione o per le varie interferenze. Dunque, come fare a rendere affidabile la connessione al livello di applicazione? Questo è un aspetto centrale per i protocolli COAP ed MQTT
In COAP è stato stabilito un funzionamento di **message reliability** ovvero un sistema di scambio di messaggi affidabile (**Reliable** exchange) attraverso un tipo di messaggi che prendono il nome di **confirmable messages:** la ricezione del messaggio deve sempre essere confermato da un ACK. Quindi dopo che il client fa una richiesta deve attendere l'ACK del server.
NB: teoricamente si potrebbe capire se la richiesta è stata ricevuta dal server dalla corrispondenza univoca che vi è tra request e response (mediante il message ID), tuttavia questo non viene fatto perché in alcuni scenari potrebbe capitare che la response non è temporaneamente disponibile e questo deve essere notificabile (attraverso un ACK). Quindi è preferibile utilizzare un ACK.

Si noti che con l'utilizzo degli ACK potrebbe nascere il problema di eventuali duplicati: gli ACK infatti potrebbero andare perduti, proprio come i dati; quando viene inviato un dato viene avviato un timer entro il quale di attende un ACK di corretta ricezione, se questo non arriva entro il tempo stabilito il dato viene nuovamente trasmesso. Qualora il dato venisse correttamente ricevuto ma l'ACK di risposta di perdesse, il mittente invierebbe nuovamente il dato ed il destinatario riceverebbe così un duplicato.

Chiaramente il meccanismo introdotto dai messaggi confermabili potrebbe appesantire la comunicazione, per questo motivo sono stati anche instaurati lo scambio non affidabile dei messaggi: Unreliable exchange, quindi con uno scambio di messaggi che sono **Non-Confirmable Message**, questo è ad esempio il caso in cui si ha un sensore di temperatura che invia continuamente dati al server: se viene perso un dato non è una perdita critica poiché la temperatura non varia in maniera brusca. Con la mancanza di conferma si alleggerisce il traffico.

La presenza del **message ID** permette di verificare se vi sono **duplicati** di entrambe le tipologie di messaggio: confirmable o non-confirmable che siano: può capitare che il server riceva lo stesso messaggio più volte, in tal caso capirà che vi è stata una duplicazione grazie all'ID e quindi non risponde due volte.

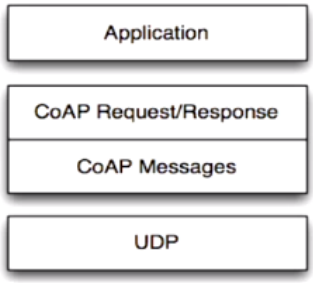
Semantica dei messaggi COAP

Il messaggio COAP si può suddividere in due parti: il COAP Message e il COAP Request/Response. Il sottolivello COAP message contiene le richieste del client o le risposte del server.

Una **request COAP** (simile ala request HTTP, con gli stessi metodi) è inviata dal client che richiede un'azione (metodo)su una determinata risorsa che viene identificato da un URI.

Il server risponde con una **response COAP** in cui torna ciò che ha richiesto il client (a seconda del metodo) o un errori qualora non potesse tornare quello che viene richiesta.

Rispetto ad HTTP, **COAP utilizza i messaggi COAP che possono essere affidabili o meno a seconda dell'implementazione**. Questi messaggi incapsulano le request e le response.
A differenza di HTTP, **COAP prevede uno scambio di messaggi asincrono**: il client invia una richiesta con opportuno metodo al server, il server una volta che ha ricevuto la richiesta deve mandare un ACK e dopo di che invierà la risposta.



Esempio: Utilizzando HTTP, se il client fa una GET per una temperatura, il server manda la temperatura (o segnala un eventuale errore).
Utilizzando COAP il client richiedere la temperatura con GET, il server risponde con un ACK se ha ricevuto correttamente la richiesta ma potrebbe capitare che non abbia la risorsa a disposizione quindi risponderà con il solo ACK. Questo segnala che il contenuto richiesto non è ancora a disposizione.
Il server potrebbe successivamente mandare un altro messaggio al client contenente la response con il valore della temperatura che era stato precedentemente richiesto dal client.
Questo metodo è asincrono poiché non vi è il meccanismo "istantaneo" richiesta-risposta (instaurato in HTTP), poiché la risorsa potrebbe essere mandata in un secondo momento

Tipologia di messaggi in COAP

COAP definisce praticamente 4 tipologie di messaggi, quest'ultimi potranno incapsulare le request e le response:

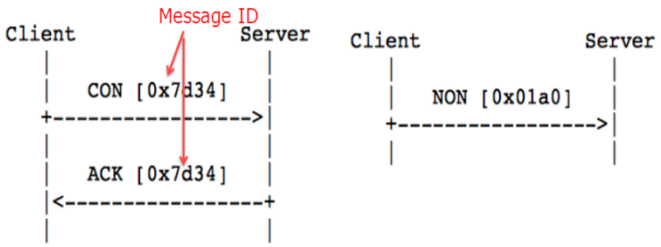
1. **Confirmable:** messaggi che richiedono un ACK quando vengono ricevuti dall'altra parte
2. **Non-confirmable:** messaggi che non prevedono la ricezione di un ACK
3. **Acknowledgement (ACK):** messaggio di conferma
4. **Reset:** messaggio particolare, quando il server non può fornire la risorsa richiesta risponde con un reset.

I codici dei metodi e delle response sono incapsulati all'interno del messaggio.

Scambio di messaggi in COAP

Scambio di messaggi confirmable:

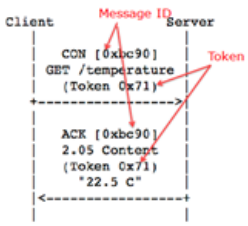
Possiamo osservare nell'immagine lo scambio di messaggi confirmable. Viene generato un messaggio dal Client avente un determinato Message ID. Il server risponde con un ACK avente lo stesso Message ID (qualora il messaggio fosse non-confirmable non viene inviato l'ACK) e successivamente viene inviata una risposta a seconda del metodo richiesto dal client.



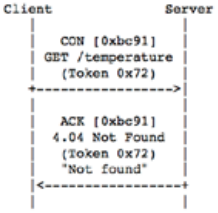
Scambio di messaggi con piggybacked response:

Il client manda un **messaggio confirmable** al server, identificando il messaggio con un Message ID ed incapsulando un determinato metodo (nell'esempio GET/"risorsa") e viene poi incapsulato anche un **token che identifica la richiesta**.
Esempio: siccome lo scambio di messaggi è asincrono, se il client invia una richiesta GET ed il server non ha il contenuto richiesto, quest'ultimo risponde con un ACK senza contenuto per segnalare l'assenza dell'informazione richiesta. Se dopo un certo tempo il server ha il valore richiesto dalla request che aveva ricevuto dal client, deve occuparsi di inviarla. Supponendo che il client aveva richiesto una temperatura, il server, per capire che deve inviare la temperatura fa riferimento al token. In questo modo, identificando la richiesta con un Token si riesce a risalire ad essa in maniera asincrona.

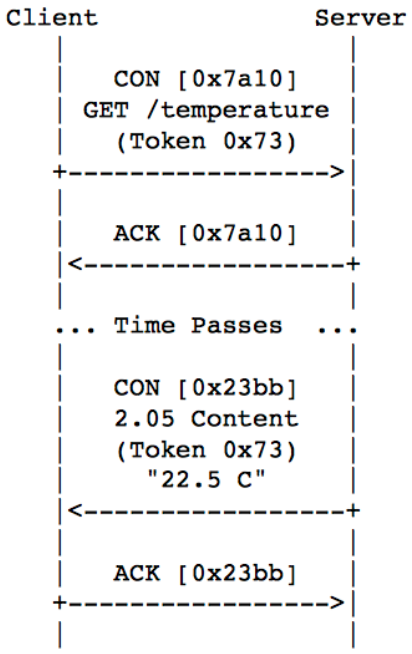
Caso 1



Caso 2



Esempio di trasmissione dei messaggi con piggybacked response:



In questo esempio possiamo vedere che il client effettua una richiesta di tipo GET riferendosi al valore della temperatura. Il messaggio ha un ID: 0x7a10.
Il server riceve correttamente la richiesta e quindi invia un messaggio di ACK, incapsulando in esso l'ID del messaggio, in questo modo si specifica che l'ACK si riferisce all'ok del server nell'avviare la comunicazione.
In questo caso il server non possiede la risorsa e quindi passa del tempo fin quando non la ottiene. Non appena ottiene la risposta, quest'ultimo la invia, incapsulando all'interno del messaggio response (di tipo confirmable) la risorsa e inserendo il token originariamente inviato al client, per sottolineare il fatto che la risposta contenga la risorsa richiesta inizialmente.
Dopo di che il client manda un ACK di corretta ricezione al server, incapsulando in esso l'ID Message della response.

Da questo esempio è evidente che **la conversazione tra client e server è asincrona** dato che avviene in momenti differenti. Questo è utile in ambienti IoT perché:

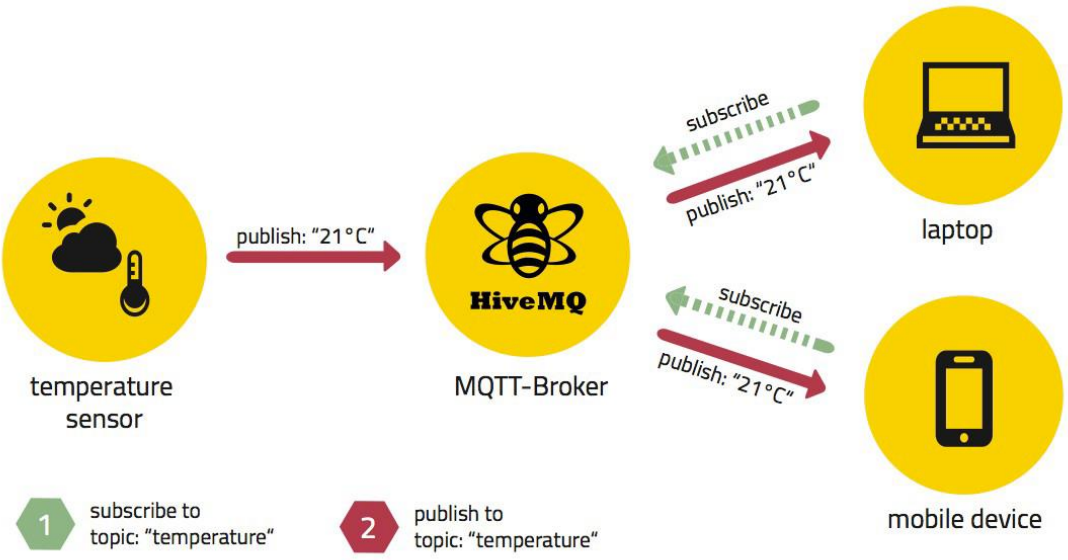
- √ **Aumenta l'affidabilità**: poiché vengono inviati degli ACK di conferma e questo risponde alle necessità che vien fuori dall'utilizzo di connessioni wireless (soggette a cadute di connessione o ad interferenze).
- √ **Aumenta la flessibilità**: si ha una maggiore flessibilità sulla sincronizzazione tra client e server poiché quest'ultimo invia il messaggio quando è possibile.

Protocollo MQTT

Il protocollo MQTT utilizza il paradigma Publisher/Subscriber, esso è duale a COAP: quando si utilizza il polling viene chiesto in continuazione una determinata risorsa ma se questa non è disponibile si perde tempo.

E' nato alla fine degli anni 90' ed è stato pubblicato come MQTT nel 2010, nel settembre 2014 è diventato uno standard OASIS.
MQTT è nato per le applicazioni Machine-to-Machine, parlare di questo scenario applicativo significa poter estendere questo concetto al mondo IoT.
Questo protocollo è nato per le macchine che trasmettevano i dati ad un server, anche in condizioni avverse quali le interferenze e le cadute di linea nel caso venisse utilizzato un canale wireless. Ad oggi è in uso in molte applicazioni come Facebook.

Come abbiamo detto, è un protocollo Publisher/Subscriber: vi sono dei client ed un server che viene chiamato **MQTT-Broker;** la comunicazione con i client viene mediata dal server.
Supponendo vi sia un client addetto alla misurazione di temperatura, quando questo misura una determinata temperatura la **pubblica (publish)**, inviandola al broker.
Il broker può poi inoltrare (inviare) questa informazione (la temperatura) ad altri client interessati a riceverla, questi client "ricevitori" si chiamano **subscriber** (sottoscritti), essi vengono sottoscritti per la ricezione di un **topic**: il dato che intendono ricevere (nel nostro caso la temperatura).



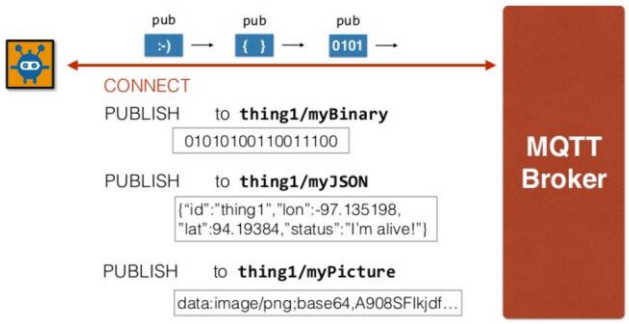
Si può notare che nei nodi subscriber vi è una freccia che va dal server al client che "trasporta" il topic inoltrato dal client publisher, e poi vi è una freccia opposta, dal client subscriber al server MQTT-Broker, questa indicano la sottoscrizione del client, ovvero la volontà del client di ricevere i topic pubblicati da un certo publisher.
Tutti i restanti client che non sono sottoscritti non riceveranno il topic in questione.

In questo paradigma il client non richiedere continuamente al server una risorsa, come veniva nel polling, ma è proprio quest'ultimo che può inviarla comportandosi da publisher o riceverla comportandosi da subscriber, il server Broker farà solo l'inoltro della risorsa (topic).
La concezione di client e server è differente rispetto al paradigma polling: nel publisher/subscriber i client possono "fungere da server", pubblicando il topic, in questo caso di chiamano **client publisher** o altrimenti possono comportarsi come i classici client che richiedono una risorsa; si attenzioni però che la risorsa non viene direttamente richiesta come avveniva nel paradigma polling ma essa viene ricevuta solo dopo che il client si sia sottoscritto e abbia quindi dato il consenso alla ricezione di un determinato topic, chiameremo quest'ultimo **client subscriber.** Il ruolo del **Broker server** in questo ambito è diverso dal server nel paradigma polling: in questo scenario, infatti, il server ha il solo ruolo di **inoltrare le informazioni**: inoltra i topic dei publisher ai subscriber interessati.

I client non fanno richieste tra loro, essi si sottoscrivono ma solo grazie al server: non vi è mai una interazione diretta tra essi ma tra loro ci sta il Broker server, che si occupa dell'inoltro. Quindi il subscriber si sottoscrive per ricevere un determinato topic riferendosi al server e il publisher pubblica verso il server; i due client non si conoscono e non vengono mai a conoscenza della provenienza dell'informazione. Questo porta ad un **disaccoppiamento totale.**

Caratteristiche principali del paradigma publisher/subscriber:

- Si ha un disaccoppiamento totale tra publisher e subscriber
- I Client sono sempre connessi con un broker ed ogni client per comunicare con esso ha bisogno di stabilire una connessione.
- Sia publisher che subscriber sono client
- MQTT utilizza i topic per determinare quale messaggio deve essere inoltrato ad un client, sulla base del topic il broker può "accoppiare" il publisher con il subscriber.
- Un topic è una stringa contentente una struttura gerarchica che segue la sintassi dei pathname (separando con il carattere "\")
- Il messaggio non prevede una specifica del tipo del dato trasportato: è agnostico al contenuto, questo significa che il riconoscimento del formato è demandato all'applicazione che sta sul client subscriber (ad esempio, deve sapere in che formato arriva la temperatura: binario, json, etc..)



I topic sono quindi parte fondamentale del funzionamento di questo paradigma. Nel protocollo MQTT il topic è una sorta di **namespace**, viene utilizzato uno slash separatore per definire la gerarchia. Nel pattern esistono le **widecards** che servono per "accomunare" delle parti di pathname, che evitano di scrivere lunghi pathname, le widecards non sono altro che dei "caratteri speciali", essi sono "**+**" e "**#**".

Facciamo un esempio pratico per capirne il funzionamento:
Supponiamo di voler sottoscrivere un subscriber ad una serie di client, posti in diverse stanze di uno specifico piano di una edificio. Indichiamo con la seguente nomenclatura:

- E : Edificio **- P : Piano** **- S : Stanza** **- N : Nodo**

Per identificare ogni singolo nodo avremmo bisogno di un pathname del tipo: **E/P/S/N.**

Vediamo allora come utilizzare le wildcards "**+**":
+/P/S/N: utilizzando in questa posizione il "**+**" significa che ci vogliamo riferire a tutti gli edifici
E/P/+/N: stavolta il "**+** " sta a significare che ci riferiamo a tutte le stanza dell'edificio E del piano P

Vediamo adesso come utilizzare le wildcard "**#**":
#: utilizzando un unico hashtag sto comunicando di sottoscrivermi a tutti i nodi delle stanze di tutti i piani di tutti gli edifici (in poche parole, a tutti i publisher)
E/#: in questo caso stiamo esplicitando di sottoscriverci ai topic di tutti nodi in tutte le stanze di tutti i piani dell'edificio E
E/P/#: in quest'altro caso ci si sottoscrive a tutti i nodi in tutte le stanze del piano P dell'edificio E

L'hashtag sostituisce un ramo, il più raggruppa un insieme omogeneo: tutti gli edifici o tutti i nodi..
I caratteri speciali per identificare i topic d'interesse hanno senso per i subscriber poiché altrimenti avrebbe dovuto inviare tante richieste (una per ogni ramo se non si usa # o tutte le possibili entità se non si usa +). **Un publisher non può utilizzare i caratteri speciali poiché può usare il topic identificato dal proprio path** (il nodo 2 della stanza S nel piano P dell'edificio E dovrà per forza utilizzare il topic: E/P/S/2).

Ricapitolando:

Simbolo	Descrizione
/	Separatore tra topic
+	Wildcard singolo livello, unisce un livello di topic
#	Wildcart multilivelo, unisce multipli livelli di topic

Sia COAP che MQTT nascono per essere implementati su comunciazioni wireless, entrambi devono avere dei meccanismi che rendono affidabile la comunicazione, per COAP si hanno i messaggi confirmable, per MQTT viene implementato la **Quality of Service (QoS).**

Quality of Service (QoS)

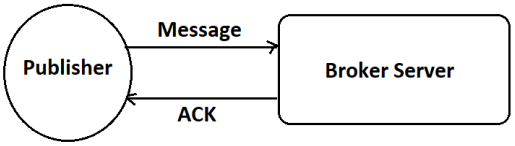
Per garantire l'affidabilità del servizio wireless nel protocollo MQTT si può specificare il **Quality of Service:** la qualità di servizio, parleremo di qualità del servizio riferendoci alla sicurezza garantita, parleremo dunque di **QoS per publisher client** intendendo il livello di sicurezza che garantisce che i topic inviati dal publisher raggiungano correttamente il broker server e di **QoS per broker server** intendendo la qualità di servizio che il server instaura con il subscriber.
Allora **si noti che i livelli di qualità di servizio che staremo per definire si riferiscono, in genere, ad un mittente e ad un destinatario, con questi ruoli che possono essere definiti da publisher e broker server quando il client publisher vuole pubblicare qualcosa o tra broker e subscriber quando l'informazione viene inoltrata al client subscriber dal broker**. Diremo allora che:

0. **At most once**: la qualità del servizio è molto bassa; il publisher non richiede alcuna garanzia che ciò che ha pubblicato venga ricevuto dal broker server.
1. **At least once:** si ha la garanzia che il messaggio sia ricevuto dal destinatario almeno una volta: il publisher invia un dato che viene ricevuto dal broker server ma quest'ultimo, al momento di inoltrarlo, potrebbe inviarlo più di una volta allo stesso subscriber
2. **Exactly once:** è il massimo livello di sicurezza, si garantisce che il messaggio venga ricevuto dal destinatario e che viene ricevuto esattamente una sola volta

Chiaramente, aumentando il livello di sicurezza aumenta il costo: sia in comunicazione che in computazione.
Esempio: prendiamo in considerazione la comunicazione tra un publisher ed un broker ma la situazione è analoga alla comunicazione tra broker e subscriber.

- Nel caso di livello 0 il publisher "spera" che il messaggio venga ricevuto dal server;

- nel caso di livello 1 si stabilisce una comunicazione "bidirezionale": il publisher invia il messaggio al server ed il server risponde con un PUBACK (Publisher Acknowledgement) di corretta ricezione. In questo caso potrebbe esserci una duplicazione di messaggi perché se si perde il PUBACK il publisher invia nuovamente il messaggio [stesso principio del COAP].



- Nel livello 2 lo scambio di messaggi è più articolato: il publisher invia un dato, il broker invia un ACK di conferma, il publisher invia un ACK di conferma di aver ricevuto l'ACK di corretta ricezione ed il broker conferma nuovamente con un ultimo ACK (vi sono 4 messaggi scambiati).