

MOBILE AND CYBER-PHYSICAL SYSTEMS
ACADEMIC YEAR 2019-2020
UNIVERSITY OF PISA
Luca Corbucci

Contents

1	Wireless Network	10
1.1	Introduzione	10
1.2	Wireless Network	11
1.2.1	Protocolli Mac	12
1.2.2	MACA Protocol	14
1.3	IEEE 802.11	15
1.3.1	DCF	16
1.3.2	PCF	19
1.4	Mobility in Cellular Networks	22
1.4.1	Definizioni	23
1.4.2	Come funziona la comunicazione?	23
1.4.3	Managing Mobility in Cellular Networks	25
2	Mobility in Ad Hoc Networks	30
2.1	Routing nelle Ad Hoc Network	30
2.1.1	DSR	31
2.1.2	Route Discovery in DSR	32
2.1.3	Route Maintenance	34
2.1.4	Altre feature della Route Discovery	35
2.2	AODV	36
2.2.1	Route Discovery	36
2.2.2	Multicast con AODV	39

2.3	Confronto tra DSR e AODV	42
2.4	DYMO	42
3	Wireless Sensor Networks: Design Aspects	45
3.1	Energy Efficiency	46
3.2	Multi-Hop Communication	53
3.3	Application Issues	58
4	WSN: Procolli	61
4.1	Protocolli MAC	61
4.1.1	S-MAC	61
4.1.2	B-MAC	64
4.1.3	X-MAC	69
4.1.4	Box-Mac	69
4.2	Polling	70
4.3	Network Protocol	71
4.4	Directed Diffusion	74
4.4.1	Funzionamento	76
4.4.2	Analisi di Direct Diffusion	79
4.4.3	Possibili problemi per Direct Diffusion	83
4.4.4	Applicazioni più complesse	85
4.5	GPSR: Greedy Perimeter Stateless Routing	85
4.5.1	Funzionamento di GPSR	86
4.5.2	Limiti e problemi di GPSR	98
4.5.3	Configurazioni che possono causare intersezioni	103
4.5.4	Considerazioni su GPSR	105
4.5.5	Altri protocolli Geo-Routing	106
4.6	Data-Centric Storage and Geographic Hash Tables	106
4.6.1	Storage vs Transmission	107
4.6.2	Data Centric Storage e GHT	110
4.6.3	Possibili problemi con DCS e GHT	116
4.7	Physical and Virtual Coordinates	118

4.7.1	Pro e Contro	123
4.7.2	Full Virtual Coordinates	123
4.8	Conclusione	126
4.9	Clustering nelle Wireless Sensor Network	127
4.9.1	Design Aspects	129
4.9.2	K-Neighbor Clustering	131
4.9.3	Assegnare i ruoli ai nodi	133
4.9.4	Dominating Set	134
4.10	Algoritmi di Clustering: DMAC	135
4.10.1	Funzionamento del protocollo	136
4.11	Conclusione	141
5	Embedding Programming - Case Studies	142
5.1	Arduino	145
5.2	TinyOS	147
5.3	Caso di studio: Arduino	149
5.4	Case Study: TinyOS	152
5.4.1	Communication Stack	157
6	Signal Theory - Introduction to Theory of signals	159
6.1	Trasmissione	160
6.1.1	Trasmissione delle informazioni analogiche	161
6.1.2	Trasmissione digitale	162
6.2	Teoria di Shannon	164
6.3	Classificare i segnali	165
6.3.1	Segnali Discreti vs Segnali Continui	168
6.4	Energia di un segnale	171
6.4.1	Impulso	172
6.5	Potenza di un segnale	173
6.6	Aspetti fisici di potenza ed energia	177
6.7	Sampling di segnali discreti	179

7 Signal Theory - Serie di Fourier	182
7.1 Serie di Fourier	183
7.1.1 Definizione	184
7.1.2 Condizioni	185
7.1.3 Esempi	186
7.1.4 Segnale pari o dispari	188
7.1.5 Segnali di Fourier con periodi arbitrari	193
8 Signal Theory - Fourier Transform	198
8.1 Introduzione	198
8.1.1 Numeri complessi	198
8.1.2 Esponente di Eulero	199
8.1.3 Fasore	200
8.2 Serie di Fourier con base esponenziale	201
8.2.1 Interpretazione geometrica delle serie di Fourier .	203
8.3 Serie di Fourier e Teorema di Dirichlet	204
8.4 Serie di Fourier con segnali reali	205
8.5 Teorema di Parseval	206
8.6 Spettro del segnale	207
8.7 Trasformata di Fourier	208
8.8 Trasformata di Fourier per segnali non periodici	211
8.9 Energia di un segnale continuo	214
9 Signal Theory - Analog to Digital Conversion	217
9.1 Sampling	218
9.1.1 Ideal Sampling	221
9.1.2 Teorema del sampling	222
9.1.3 Problemi con Nyquist	227
9.2 Aliasing	228
9.2.1 Da dove arriva l'aliasing	229
9.2.2 Sampling	230
9.3 Possibili errori con Nyquist	231

9.4 Quantizzazione	232
10 Thingspeak	237
10.1 ThingSpeak	239
10.1.1 REST	239
10.1.2 Channel	240
10.1.3 Thingspeak come Relay Point	241
10.2 ThingSpeak integration	241
10.2.1 Usare un database esterno	243
11 IEEE 802.15.4 e Zigbee	245
11.1 IEEE 802.15.4	247
11.1.1 Livello Fisico	247
11.1.2 Livello Mac	249
11.1.3 Frames	254
11.1.4 Rete senza superframe	254
11.1.5 Data transfer	254
11.2 Mac Layer Services	257
11.2.1 Associate Protocol per il MAC Layer	259
11.2.2 MAC Layer Security	262
11.3 Zigbee	263
11.3.1 Network Layer	264
11.3.2 Join through association	267
11.3.3 Application Layer di Zigbee	273
11.3.4 Zigbee Device Object	281
11.3.5 Zigbee Cluster Library	284
12 Bluetooth	292
12.1 Bluetooth Basic Rate/Enhanced Data Rate	295
12.2 Bluetooth Low Energy	296
12.3 Topologia della rete	298
12.3.1 Esempio di topologia della rete nel caso BR/EDR .	299

12.3.2 Piconet con BLE	299
12.4 Livelli di funzionamento di Bluetooth	301
12.4.1 Livello Fisico	301
12.4.2 Baseband Layer	303
12.4.3 Stati del Bluetooth BR/EDR	306
12.4.4 Baseband Layer BR/EDR: Connection State	308
12.4.5 Architettura di Low Energy	310
12.4.6 Baseband Layer states nel caso del BT Low Energy	314
12.4.7 Link Manager Layer	316
12.4.8 Livelli superiori	316
12.4.9 Gatt Services	317
12.4.10 Attribute protocol (ATT)	318
12.4.11 Sicurezza di BT	319
13 Signal Sampling con Arduino	325
13.0.1 Cosa si intende con sampling?	325
13.0.2 Codice Arduino	326
13.1 Sampling	332
13.1.1 Come risolviamo?	335
13.1.2 Timer	336
13.1.3 Risposta ad una interruzione	339
13.2 Spectrum Analysis	349
14 MQTT	352
14.1 MQTT	354
14.1.1 PUB SUB Paradigm	356
14.1.2 Persistent Sections	370
14.1.3 Retained Messages	371
14.1.4 Last Will	372
14.1.5 Keep Alive	372
14.1.6 Struttura dei pacchetti di MQTT	373
14.2 MQTT e Arduino	374

14.3 Alternative a MQTT	375
14.3.1 Limiti di MQTT	375
14.3.2 Competitor	376
15 Synchronization in Wireless Sensor Network	379
15.1 Synchronization Algorithms	380
15.1.1 Clock Model	382
15.1.2 Period between synchronization operations	389
15.1.3 Esempio	391
15.1.4 Clock Aging	391
15.1.5 Esercizi	393
15.1.6 Clock Accuracy	394
15.2 Clock Distance	395
15.3 Problema della sincronizzazione nelle WSN	396
15.3.1 Metodi per la sincronizzazione	397
15.3.2 Direct vs Multi-Hop synchronization	401
15.4 Synchronization Protocol	402
15.4.1 Symmetric vs asymmetric synchronization	404
15.4.2 Problemi	404
15.5 Reference Broadcast synchronization	405
16 Indoor Localization	407
16.1 Introduzione	407
16.2 Outdoor Localization	408
16.2.1 Computation Phase	410
16.3 Indoor Localization	411
16.3.1 Topologies	411
16.3.2 Tipi di segnali e tecnologie	414
16.3.3 Time Difference of arrival	420
16.3.4 Round Trip Time of Flight	422
16.3.5 Angle of Arrival	423
16.3.6 TOA, TDOA, AOA	424

16.3.7 Received Signal Strength	424
16.4 Processing Methods	428
16.4.1 Range Free: Centroid	430
16.4.2 Range Free: DV-HOP	431
16.4.3 APIT	432
16.4.4 Range-Based: Multilateration	432
16.4.5 Range-Based: Min-Max	433
16.4.6 Range-Based: Maximum Likelihood	436
16.5 Scene Analysis	438
16.6 Device-Free Localization	441
16.6.1 CSI Based	441
16.6.2 Proximity	441
16.7 Performance Metrics	442
16.8 Futuro	445

Chapter 1

Wireless Network

1.1 Introduzione

Con mobile ad hoc networks si intende un sistema di host mobili connessi tramite una connessione wireless, i nodi sono indipendenti ed autonomi, non esiste una infrastruttura fissata per la rete. All'interno di queste reti la comunicazione tra i nodi non adiacenti deve essere supportata da altri dispositivi.

Quali sono i problemi principali di queste reti?

- Il range per la trasmissione è limitato, la potenza del segnale diminuisce a mano a mano che ci allontaniamo del nodo.
- Il protocollo è basato sulla conoscenza dei vicini.
- I nodi sono router, questo vuol dire che abbiamo bisogno di un routing multi hop, è un tipo di comunicazione in una rete in cui la copertura del segnale del singolo nodo è minore rispetto al raggio della rete intera.
- Gli indirizzi non riflettono la posizione dei nodi presenti all'interno della rete.

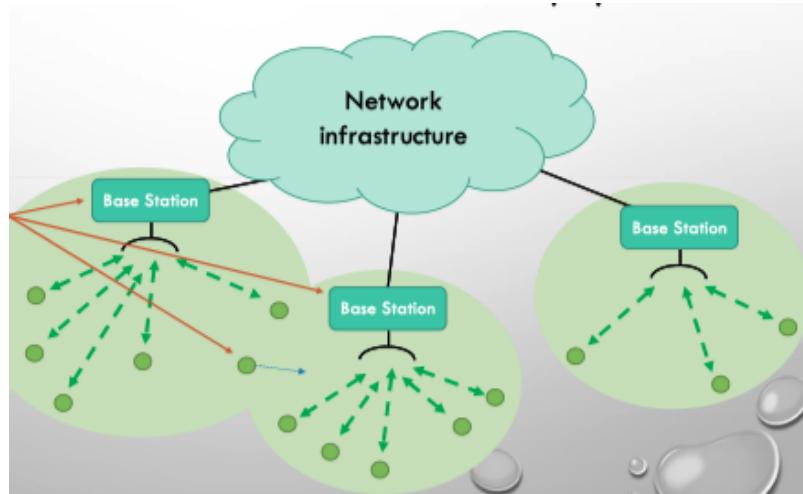
Definiamo ora la Wireless Sensor Network (WSN).

Una Wireless Sensor network è una rete formata da sensori intelligenti che sono autonomi, wireless e possono formare una rete. Ogni sensore ha una potenza bassa, costa poco ed è piccolo. Un sensore può produrre uno stream di dati e non abbiamo bisogno di cavi per trasferire questi dati. Rispetto alle Ad-Hoc network abbiamo che il numero di sensori è più alto rispetto al numero di nodi all'interno di una ad-hoc network, i sensori possono fallire e possono essere mobili, la topologia della rete può cambiare.

1.2 Wireless Network

Cominciamo a vedere quali sono le principali caratteristiche delle Wireless Network:

- L'infrastruttura della nostra rete è la seguente:



Abbiamo una serie di nodi che sono connessi ad una base station. Questa base station connette i vari nodi alla rete e si occupa di inoltrare pacchetti agli altri nodi e di riceverli.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- All'interno di una rete di questo genere è necessario un protocollo che permetta l'accesso al canale wireless disponibile per la comunicazione (un protocollo come CSMA/CD non può essere utilizzato ad esempio).
- È necessario un protocollo per il routing, questo in particolare mi serve perchè devo trovare un percorso dalla fonte alla destinazione di un messaggio all'interno di una rete multi hop.
- Devo anche poter gestire lo spostamento di un nodo da una base station all'altra.

1.2.1 Protocolli Mac

Per quanto riguarda il protocollo Mac che viene utilizzato in reti di questo genere dobbiamo considerare che è disponibile un singolo canale, tutte le stazioni possono trasmettere su questo canale e ricevere da questo canale. Due stazioni che trasmettono sullo stesso canale allo stesso momento causano una collisione e tutte le stazioni sono in grado di capire che c'è stata una collisione. Ci serve quindi un protocollo per evitare le collisioni, questo protocollo è il protocollo Mac.

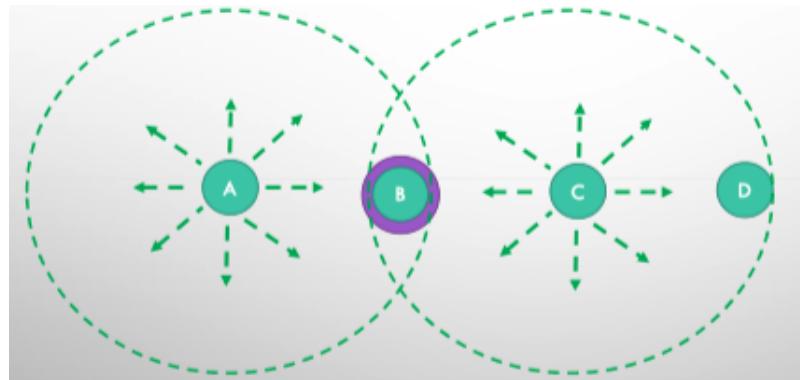
Un possibile protocollo Mac è il CSMA/CD:

- Una stazione deve controllare il canale per vedere se qualcuno sta già trasmettendo.
- Se il canale è occupato allora la stazione deve attendere.
- Se il canale è vuoto allora la stazione può trasmettere il messaggio
- Se avviene una collisione allora la stazione si accorge della collisione e blocca la comunicazione. Quando una comunicazione fallisce e il nodo che si accorge della collisione blocca l'invio, non prova subito a inviare di nuovo il pacchetto, il tempo dopo la collisione viene

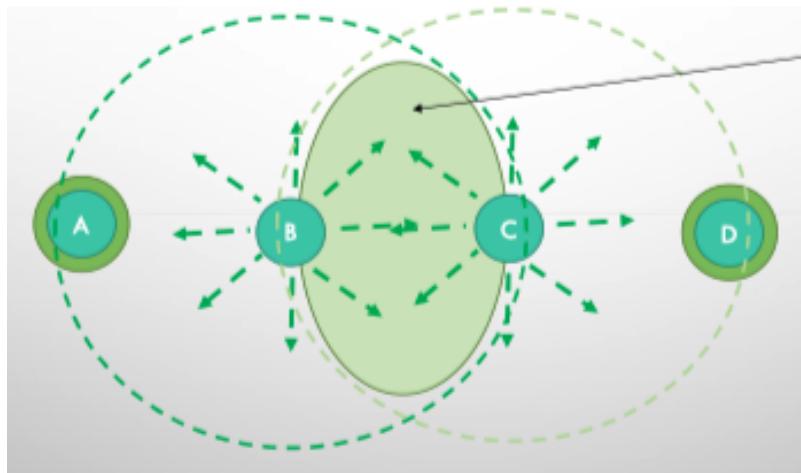
infatti suddiviso in contention slots. La lunghezza di un contention slot è uguale al caso pessimo del tempo di propagazione. Per decidere dopo quanto tempo inviare di nuovo il pacchetto si utilizza un backoff esponenziale, in particolare abbiamo che dopo la prima connessione attendiamo 0 o 1 contention slots, dopo la seconda invece attendiamo un numero compreso tra 0 e $2^2 - 1$. In generale la formula dice che alla collisione i -esima devo attendere un numero di contention slots compreso tra 0 e $2^i - 1$.

Con CSMA/CD però si verificano le seguenti problematiche:

- Abbiamo il problema del terminale nascosto, qua ad esempio abbiamo che A invia a B e B intanto sta ricevendo da C, quindi in B avremo una collisione perchè A non si rende conto prima della comunicazione in corso. Qui il problema è che la collisione la vediamo nel ricevente, non nel mittente che invia il pacchetto.



- Exposed Terminal Problem: qua succede che B sta trasmettendo ad A e C nel frattempo vorrebbe trasmettere qualcosa a D ma non può perchè vede che il canale è utilizzato da B.

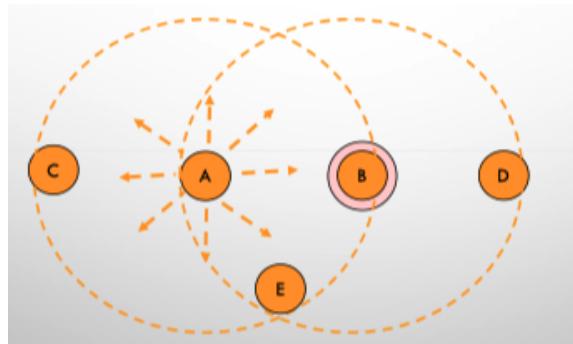


Con CSMA/CD ci accorgiamo della collisione nel nodo mittente e non nel nodo di destinazione, questo può andare bene per le reti non Wireless. Dato che nelle Wireless Network abbiamo bisogno di capire nel ricevente se abbiamo una collisione o meno abbiamo bisogno di un protocollo differente.

1.2.2 MACA Protocol

L'idea basilare di questo protocollo è la seguente:

- In questo protocollo il mittente trasmette un piccolo frame inizialmente. Questo frame iniziale che viene trasmesso è chiamato RTS.
- Quando il nodo ricevente riceve l'RTS allora invia un CTS.
- Quando il nodo che vuole spedire il pacchetto riceve il CTS allora spedisce il pacchetto



Quindi abbiamo

- C non riceve il CTS e quindi è libero di trasmettere
- Il nodo D riceve solamente il CTS quindi non dovrebbe essere un problema e rimarrà in silenzio
- Il nodo E invece riceve RTS e CTS quindi anche questo non sarà un problema.

L'unico problema è che C e B inviano l'RTS a A contemporaneamente e quindi si verifica una collisione, non viene generato il CTS, possono usare il Binary Exponential Backoff per provare di nuovo.

Nel caso delle wireless network abbiamo che la soluzione migliore da utilizzare non è MACA ma è MACAW, CSMA/CA si basa su MACAW. MACAW aggiunge un meccanismo di carrier sensing per evitare che due nodi inviano contemporaneamente l'RTS e introduce anche un ack da inviare quando riceviamo correttamente un data frame.

1.3 IEEE 802.11

IEEE 802.11 è lo standard per le wireless network, abbiamo un gruppo di stazioni che operano sotto una coordination function data. Possiamo avere un access point e in questo caso la comunicazione funziona grazie a

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

questo access point che mi fornisce comunicazione con altri Access point e con altri gruppi di stazioni. Se abbiamo l'access point le varie stazioni possono comunicare tramite l'access point. IEEE 802.11 supporta anche le ad hoc networks ovvero un gruppo di stazioni in cui viene definita una single coordination function ma in cui non abbiamo una rete strutturata. Questo fa sì che possiamo avere una comunicazione diretta tra le stazioni senza dover far passare il traffico in un Access Point.

Con IEEE 802.11 abbiamo il livello Mac che ha due tipi differenti di operazioni:

- DCF: questo è un meccanismo completamente decentralizzato che prova (best effort) a controllare tutte le attività. Questo meccanismo deve essere implementato da ogni stazione ed è completamente decentralizzato.
- PCF: In questo caso utilizziamo una base station per controllare tutte le attività nella cella, l'access point esegue il poll delle stazioni per chiedere il permesso della trasmissione.

Consideriamo più nel dettaglio l'architettura del Livello MAC dell'IEEE 802.11:

1.3.1 DCF

Come abbiamo detto il DCF è un metodo per il carrier sensing che deve essere implementato da ogni stazione ed è completamente decentralizzato. Il metodo ha due livelli di carrier sensing:

- Fisico: viene analizzata la presenza di altri utenti della rete wireless tramite il controllo dei pacchetti che vengono scambiati.
- Virtuale: si effettua inviando informazioni relative alla durata della trasmissione all'interno dell'RTS, CTS e del data frame. In questo

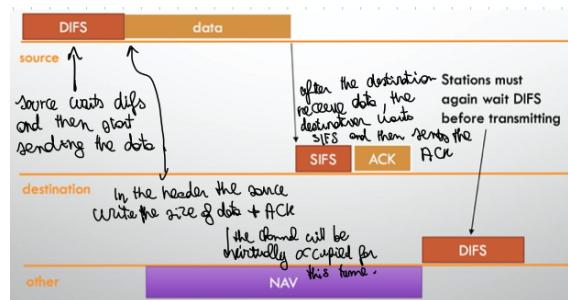
modo si mantiene il canale virtualmente occupato fino alla fine della trasmissione.

L'accesso al mezzo è controllato utilizzando l'IFS time interval ovvero un periodo di tempo in cui il mezzo di comunicazione è libero. Esistono tre tipi di IFS:

- SIFS: è il tempo di attesa più corto.
- PIFS: Point Coordination function.
- DIFS: Distributed Coordination function, è il tempo più lungo.

Come funziona la comunicazione in questo caso:

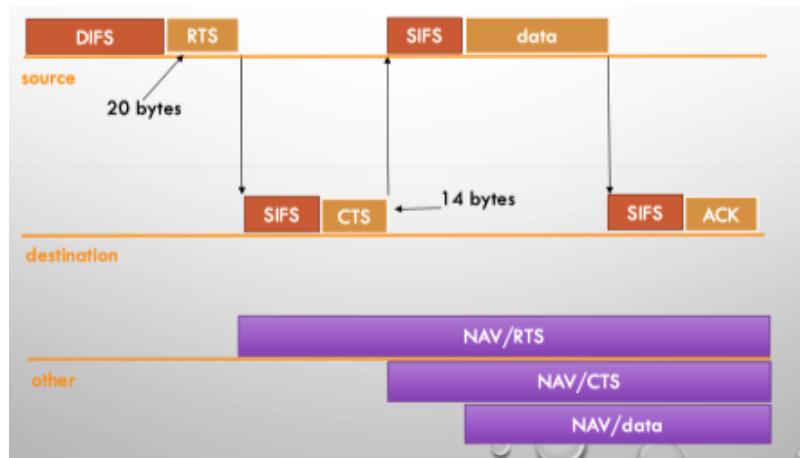
- Per prima cosa si controlla se il canale è libero per un tempo DIFS.
- Se il canale è libero per un tempo DIFS allora possiamo inviare i nostri dati, nell'header scriviamo la dimensione di dati e ACK in modo che gli altri possano attendere quel tempo. Il canale sarà virtualmente occupato per questo periodo di tempo.
- Quando termino l'invio dei dati, dopo che la destinazione riceve i dati, la destinazione attende un tempo SIFS e poi invia un ACK.
- Prima di trasmettere di nuovo si attende un altro DIFS.



È anche possibile evitare collisioni usando DCF, in particolare il controllo del mezzo viene effettuato tramite i seguenti pacchetti:

- RTS: Ready To Send
- CTS: Clear To Send

Questi pacchetti possiamo decidere di utilizzarli quando inviamo messaggi più lunghi del solito oppure sempre oppure mai, il funzionamento è il seguente e ricorda il funzionamento del protocollo MACA.



DCF Fragmentation

Se frammentiamo i nostri dati, possiamo migliorare l'affidabilità della trasmissione. Solitamente questo si fa quando abbiamo che la dimensione del pacchetto che dobbiamo spedire è maggiore rispetto ad un certo threshold.

Se il mittente non riceve un ack allora competerà di nuovo per avere accesso al canale e poi invierà nuovamente le informazioni partendo dall'ultimo frammento che era stato inviato.

Random backoff

Il funzionamento del Random Backoff:

- Abbiamo slot di una certa lunghezza.
- Dopo una collisione una stazione controlla il canale e se questo è occupato allora aspetta. Se invece lo trova libero deve attendere che il canale sia libero per un tempo di lunghezza difs.
- Calcoliamo un tempo random di backoff compreso tra 0 e $2^2 + i$ dove la i è il numero volte che abbiamo avuto la collisione.
- Aspettiamo per un numero di slot che è deciso con il tempo di backoff.
- Se abbiamo due stazioni che attendono lo stesso numero di slot x allora avremo una nuova collisione. Se invece il canale diventa occupato dopo un numero Y di slots e io ne dovevo attendere X allora attendo che il canale sia di nuovo libero per un tempo difs e poi attendo che passino $X-Y$ slots prima di poter trasmettere.

1.3.2 PCF

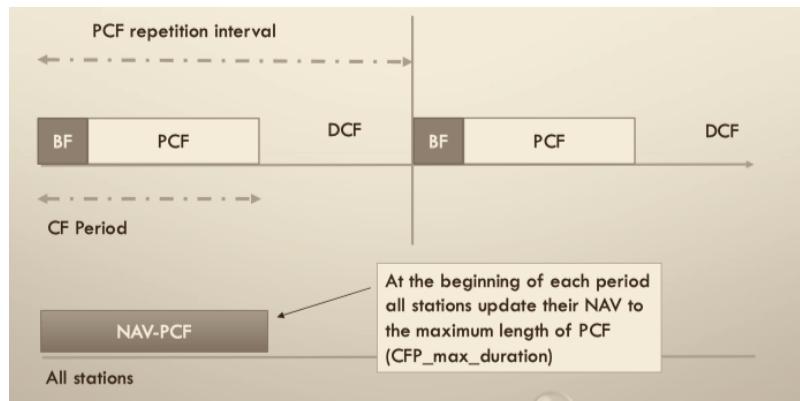
Questo è un protocollo "Connection-Oriented" che utilizza un coordinatore che esegue il polling e abilita le stazioni a trasmettere senza contendersi il canale. Quali sono le differenze con il DCF:

- Entrambi sono protocolli di accesso al mezzo.
- DCF controlla se il canale è libero prima di trasmettere qualcosa.
- PCF usa un coordinatore per controllare che il mezzo sia utilizzabile.

Il coordinatore di PCF solitamente è l'Access Point. PCF funziona utilizzando delle finestre temporali composte da:

- Un Content Free period in cui il coordinatore PC invia un beacon frame BF in modo tale che le stazioni possano sincronizzarsi.
- Contention Based Traffic: in questo periodo la stazione accederà il canale utilizzando il protocollo DCF.

Durante ciascuno di questi intervalli che vengono ripetuti abbiamo un certo tempo destinato al contention free e un certo intervallo destinato invece al contention based traffic.



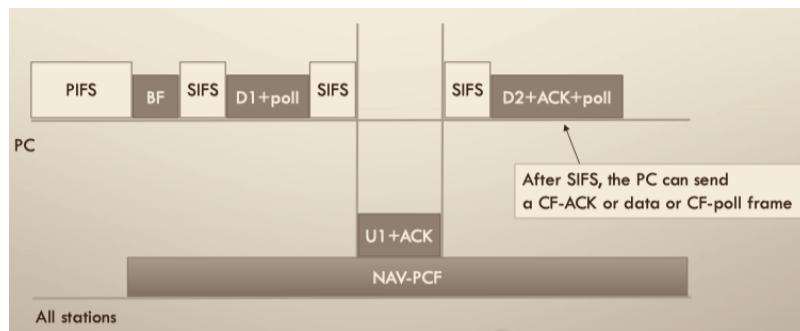
Il beacon frame serve alle varie stazioni per sincronizzarsi, nell'intervallo che viene ripetuto la prima parte funziona con PCF mentre la seconda con DCF. La prima fase contention free è sempre chiusa da un pacchetto Contention Free End (CFE).

PCF funziona in questo modo:

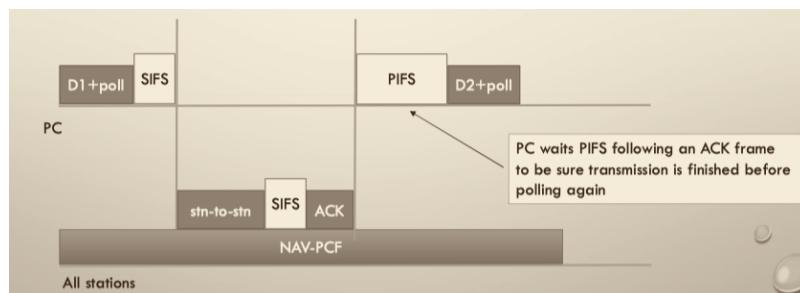
- Il Point Coordinator attende PIFS tempo e poi se il mezzo è ancora libero invia un beacon frame sul mezzo di comunicazione.
- Le varie stazioni quando ricevono il beacon verranno a sapere la lunghezza del contention free period.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Il Point Coordinator attende tempo SIFS e poi invia alle stazioni i dati e/o un frame con un poll poi attende tempo SIFS. In questa situazione le stazioni possono solamente rispondere se sono "pollate" da un Point Coordinator.
- La destinazione che vuole utilizzare il mezzo può, quando riceve i dati del Point Coordinator, inviare a sua volta dei dati al coordinatore oppure inviare un ACK.
- Il Point Coordinator attende tempo SIFS e poi può inviare a sua volta il suo ACK o anche altri dati.



- Volendo la stazione quando viene "pollata" dal Point Coordinator può iniziare una comunicazione con un'altra stazione, quando questa comunicazione finisce allora viene inviato un ACK al Point Coordinator.



Con questo sistema un Point Coordinator può anche inviare dei dati ad una stazione che non conosce il PCF. I messaggi possono essere frammentati come in DCF, le performances non sono brillanti.

1.4 Mobility in Cellular Networks

Consideriamo ora una rete cellulare, ha le seguenti caratteristiche:

- Ogni cella della rete cellulare copre una specifica regione geografica.
- In ogni cella abbiamo una base station (BS).
- In ogni cella abbiamo un utente che si collega alla rete utilizzando la base station.
- Tra la base station e il dispositivo mobile c'è una "air-interface".
- Abbiamo un mobile switching center che connette le celle e gestisce la mobilità.

Ci sono due modi differenti per condividere il canale di comunicazione in una rete cellulare e in particolare per la comunicazione tra il dispositivo mobile e la base station:

- Possiamo dividere lo spettro in canali e poi ogni canale in slot temporali.
- Possiamo usare CDMA ovvero il Code Division Multiple access.

Cosa intendiamo per mobilità? Possiamo vedere 4 tipologie:

- Alta mobilità: in questo caso siamo nella situazione in cui il dispositivo si muove da un access point all'altro mantenendo sempre attiva la connessione, come succede quando utilizziamo i cellulari.

- Bassa mobilità: Il dispositivo si connette e si disconnette da una rete esistente utilizzando DHCP.
- Zero mobilità: Il dispositivo rimane sempre all'interno della stessa rete.

1.4.1 Definizioni

- Home Network: è la "casa" permanente del dispositivo mobile ovvero la cella a cui si è connesso inizialmente.
- Permanent address: questo è l'indirizzo all'interno dell'Home Network.
- Home Agent: si tratta di una entità che gestisce lo spostamento dei nodi all'interno della rete.
- Visited Network: questa è la rete in cui il dispositivo mobile si trova attualmente.
- Care of Address: questo è l'indirizzo all'interno del visited network.
- Foreign Agent: questa è l'entità all'interno del visited network che gestisce la mobilità.

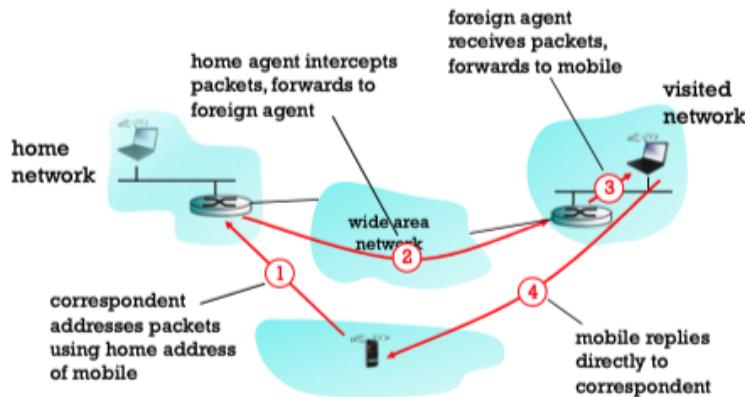
1.4.2 Come funziona la comunicazione?

Se vogliamo comunicare con un nodo all'interno della rete mobile abbiamo due possibilità:

- Potremmo avere una routing table in cui memorizziamo gli indirizzi di tutti gli altri nodi e ovviamente quando un nodo si muove in un'altra cella dobbiamo modificare il suo indirizzo all'interno della routing table. Questo approccio ovviamente non scala.

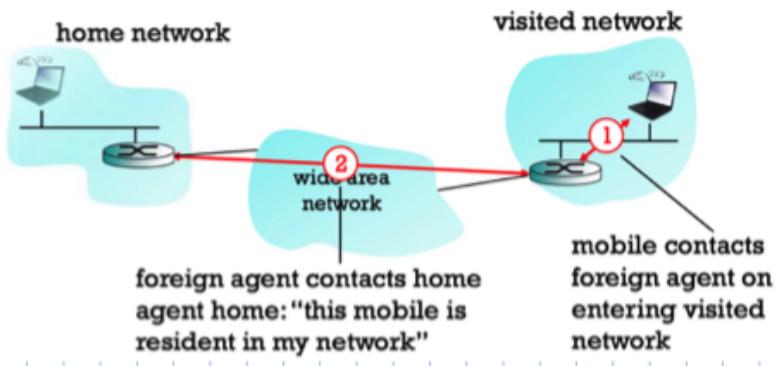
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Possiamo contattare l'home agent e se il dispositivo si è spostato in un'altra cella possiamo contattare il dispositivo tramite la nuova cella.



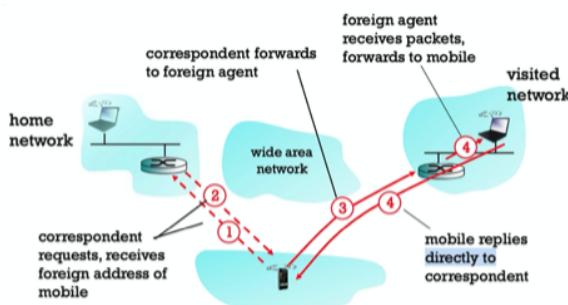
Il funzionamento è semplice, il mittente contatta l'home network, l'home network, contatta il visited network e in particolare il foreign agent del visited network. Questo inoltra il messaggio al dispositivo mobile che ora potrà rispondere al mittente della richiesta.

Per fare tutto ciò è necessario che quando il nodo si sposta in una rete differente da quella iniziale avvenga una comunicazione tra il visited network e l'home network in modo da comunicare questo cambiamento.



Questa prima soluzione, chiamata indirect routing è interessante perché scala ma allo stesso tempo è un problema perchè non è troppo efficiente se consideriamo che i due utenti che vogliono comunicare potrebbero già essere all'interno della stessa rete.

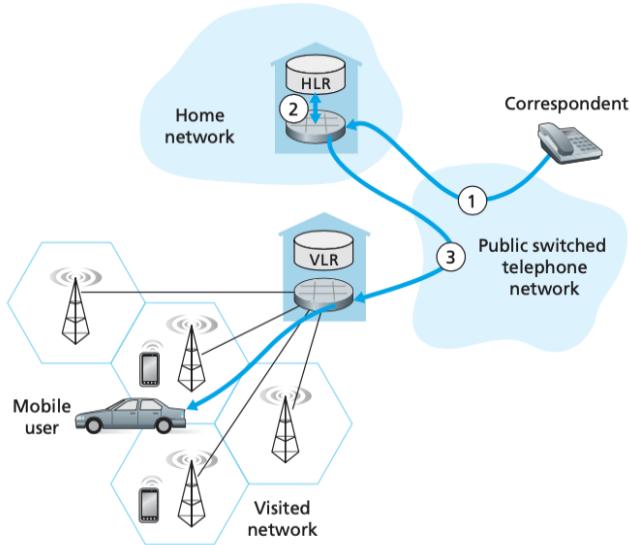
Una possibile alternativa all'indirect routing è il direct routing:



In questo caso non abbiamo una comunicazione tra home e foreign agent perchè l'utente che vuole comunicare con il nodo che si è spostato riceve informazioni riguardo alla nuova posizione dall'Home Network e poi contatta direttamente il Visited Network. In questo caso il problema può verificarsi quando il care of address cambia dopo che abbiamo avuto l'indirizzo dall'home agent.

1.4.3 Managing Mobility in Cellular Networks

Ora che abbiamo visto come funziona la mobilità in una rete vediamo come funziona lo stesso tipo di comunicazione in una rete cellulare. Ci concentriamo in particolare sulla mobilità utilizzando l'architettura di GSM. Anche GSM come nel caso di mobile IP abbiamo un approccio di routing indiretto ovvero quando dobbiamo fare una richiesta comunichiamo prima con l'home network e poi con il visited network.



In particolare in GSM abbiamo:

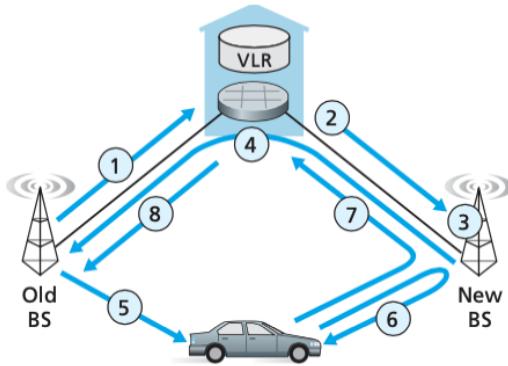
- Home Network: è la rete del provider con cui abbiamo il nostro abbonamento
- Visited Network: è la rete in cui attualmente il dispositivo si trova
- L'home network mantiene un database che è chiamato Home Location Register (HLR) che contiene i numeri di telefono di ogni abbonato e la posizione corrente di questo abbonato. Se il dispositivo è in roaming in una rete di un altro provider l'HLR contiene le informazioni per ottenere un indirizzo all'interno del visited network per fare in modo di inoltrare una eventuale chiamata a quell'utente. Quando devo effettuare la chiamata viene contattato l'Home Mobile Switching Center (home MSC) che è presente nell'home network del destinatario della chiamata.
- All'interno del visited network abbiamo un Visitor Location Register (VLR) che contiene una entry per ogni utente che attualmente si trova all'interno della porzione di rete servita dalla rete di quel

provider. All'interno del VLR i dati quindi vengono aggiunti e rimossi a mano a mano che gli utenti entrano ed escono dalla rete.

Quando un dispositivo si sposta da una rete all'altra avviene uno scambio di messaggi tra il dispositivo e il VLR. Il VLR quando riceve il messaggio comunica all'HLR del dispositivo un aggiornamento della localizzazione. Con questo aggiornamento comunichiamo sia il numero di roaming a cui dovrebbe essere contattato il dispositivo mobile sia l'indirizzo del VLR.

Handoff in GSM

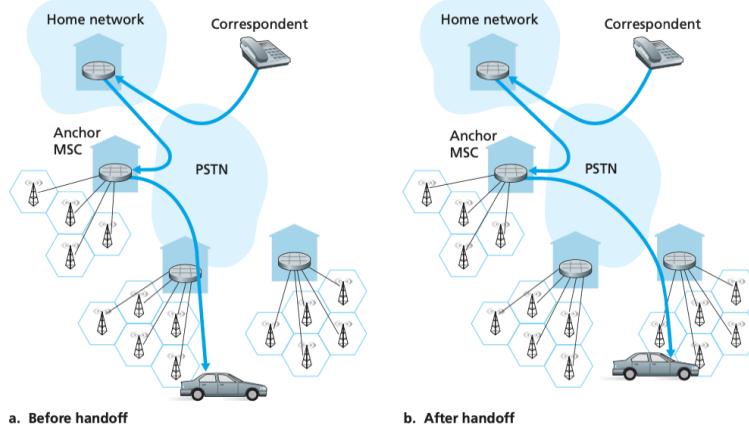
L'handoff è l'operazione che avviene quando un dispositivo mobile cambia la stazione a cui è associato durante una chiamata. Inizialmente la chiamata viene inoltrata da una prima Base station (old Base station) all'utente, poi dopo lo spostamento e dopo l'handoff abbiamo che la chiamata è inviata da un'altra base station (new Base station) all'utente. Notare che l'handoff non avviene solamente nella trasmissione da una base station al dispositivo ma anche nel rerouting della chiamata da uno switching point all'interno della rete verso una nuova base station. Assumiamo inizialmente che old BSS e new BSS condividono lo stesso Mobile Switching Center e che deve avvenire il rerouting. Il rerouting avviene sia perchè il segnale tra la base station e il dispositivo mobile potrebbe essersi deteriorato sia perchè la cella potrebbe essere diventata sovraccarica. Vediamo come funziona l'handoff tra due base station con un Mobile Switching Center MSC in comune:



- La old BSS informa il MSC che deve essere eseguito l'handoff e comunica anche la new BSS verso cui il dispositivo mobile si muoverà.
- Il MSC contatta la nuova BSS e comunica che sta per avvenire un handoff in modo che la nuova BSS sia preparata.
- La nuova BSS attiva un canale radio da far utilizzare dal dispositivo mobile che deve svolgere l'handoff.
- Il nuovo BSS segnala al MSC (che inoltra al vecchio BSS) che è pronto per l'handoff e quindi che il dispositivo mobile può eseguire l'handoff. Vengono comunicate tutte le informazioni necessarie per eseguire l'handoff del dispositivo.
- Il dispositivo mobile viene informato dell'handoff, fino a questo momento ne era all'oscuro.
- Il dispositivo mobile e il nuovo BSS si scambiano dei messaggi per attivare il nuovo canale.
- Il dispositivo mobile comunica che l'handoff è avvenuto correttamente al nuovo BSS, questo inoltra il messaggio al MSC e lui poi lo inoltra al vecchio BSS che rilascia le risorse che erano state allocate per quel dispositivo mobile.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Vediamo anche il caso in cui invece abbiamo un differente MSC:



In questo caso definiamo il concetto di Anchor MSC che è il MSC visitato dal dispositivo mobile quando inizia la chiamata. Questo anchor MSC non cambia durante la chiamata quindi quello che succede è che la chiamata viene inoltrata prima all'home MSC e poi all'anchor MSC. Dall'anchor MSC poi la chiamata viene inoltrata al visited MSC dove il dispositivo mobile si trova attualmente. In ogni momento ci stanno quindi al più 3 MSC durante la chiamata perché quando il dispositivo si sposta la chiamata viene inoltrata dall'anchor MSC verso il nuovo MSC.

Chapter 2

Mobility in Ad Hoc Networks

Le ad-hoc network sono una tipo di wireless network che però sono decentralizzate.

2.1 Routing nelle Ad Hoc Network

Perchè è necessario avere un protocollo per il routing in queste ad-hoc network?

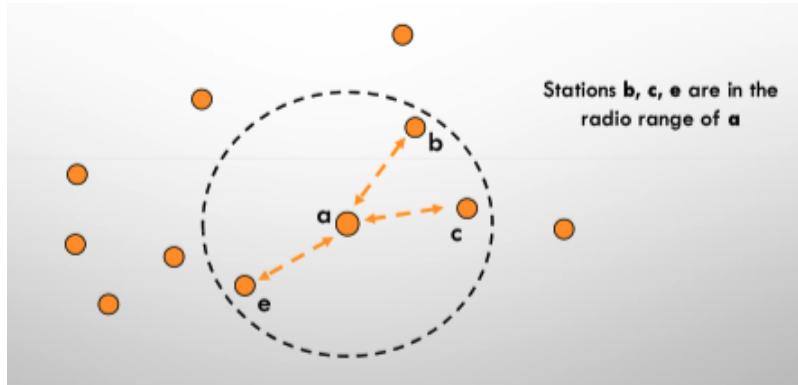
- Abbiamo la necessità di adattarci ai cambiamenti della rete.
- Vogliamo un sistema che sia decentralizzato.
- Vogliamo un sistema in cui non ci siano loop.

Ci sono vari possibili approcci per il routing:

- Approccio proattivo, in questo caso facciamo un tentativo di mantenere consistenti le operazioni di routing.

- Approccio Reattivo: in questo caso scopriamo una nuova route quando ci serve.

Le ad-hoc network possono essere rappresentate utilizzando un grafo:



- In questo grafo ogni nodo è un terminale.
- Se due nodi sono connessi con un grafo diretto allora le due stazioni possono comunicare direttamente.
- Se vogliamo inviare un messaggio da un nodo all'altro e i due nodi non sono connessi allora dobbiamo fare una comunicazione multi hop.
- Assumendo che la comunicazione tra i due nodi è bidirezionale allora possiamo dire che il grafo sarà non diretto.
- Ci serve un protocollo che mi permetta di gestire la comunicazione tra i nodi e mi permetta di gestire i cambiamenti all'interno del grafo.

2.1.1 DSR

Il protocollo DSR è decentralizzato, ha un basso overhead e reagisce velocemente ai cambiamenti che possono avvenire all'interno della rete.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

DSR fa alcune assunzioni:

- Abbiamo dei cooperative nodes: tutti i nodi partecipano attivamente e inoltrano i pacchetti destinati agli altri nodi.
- Abbiamo bisogno di un piccolo diametro all'interno della rete, vogliamo fare solamente pochi hop alla volta per muoverci da un nodo all'altro.
- I nodi possono muoversi all'interno della rete senza dirlo a nessuno.
- È poco probabile che un nodo si allontani da un altro nodo durante la fase di route discovery, se anche il nodo si muove i cambiamenti sono piccoli rispetto all'area degli altri nodi e quindi non abbiamo cambiamenti topologici.

In DSR vengono eseguite due operazioni:

- Route Discovery: proviamo a cercare un percorso per il pacchetto, questa procedura viene utilizzata solamente quando il percorso è sconosciuto.
- Route Maintenance: conosciamo già il percorso tra due nodi S e D ma il nodo S si accorge che c'è un cambiamento nella topologia della rete e non possiamo utilizzare il percorso precedente. In questo caso S può conoscere un percorso alternativo oppure esegue di nuovo la Route Discovery.

2.1.2 Route Discovery in DSR

Vediamo come funziona il protocollo per la route discovery in DSR:

- Il nodo mittente S vuole inviare un messaggio al nodo destinazione D.

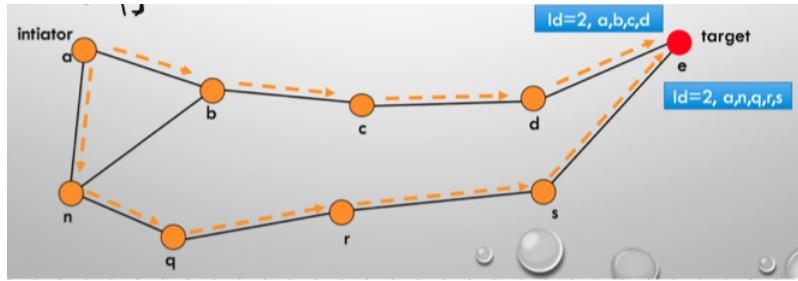
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- S cerca per prima cosa un percorso nella sua cache e poi se trova il percorso lo utilizza per mandare il pacchetto a D.
- Se S non conosce il percorso allora inizia la route request (RREQ) dove S è il nodo che avvia la route request mentre D è il target.
- La RREQ viene inviata a tutti i nodi che sono vicini di S, e contiene informazioni riguardo al mittente, al destinatario, un ID univoco e una lista di nodi a cui la richiesta è stata inoltrata.
- Quando un nodo N riceve la RREQ controlla se il suo ID è già nella lista dei nuovi visitati, se c'è allora elimina la RREQ altrimenti aggiunge il suo ID nella lista e inoltra la richiesta.
- Se il nodo target riceve la richiesta RREQ risponde ad S con una RREP che include la lista dei nodi che sono stati accumulati durante la richiesta.
- Una possibile ottimizzazione per questo sistema, se un nodo intermedio conosce già il percorso verso il nodo target allora può mandare indietro questo percorso senza inoltrare la RREQ, possiamo utilizzare questa ottimizzazione solamente se il path non contiene un loop.

Quando il nodo finale riceve il pacchetto allora ha due possibilità:

- Deve rimandare indietro la RREP e conosce un possibile percorso per raggiungere il nodo mittente.
- Deve rimandare indietro la RREP ma non conosce il percorso per raggiungere il mittente quindi deve eseguire una route discovery per poter trovare il mittente. Per questo procedimento può aiutarsi con i dati che trova nella RREQ ma solamente se i collegamenti tra i nodi sono bidirezionali.

Vediamo un esempio di funzionamento:



- Qua abbiamo che ogni nodo riceve la RREQ e inoltra il messaggio aggiungendo il suo ID alla lista.
- Se alla fine il nodo target riceve più richieste, sceglierà il percorso migliore e alla fine invierà la RREP.

2.1.3 Route Maintenance

Vediamo come funziona il protocollo di Route Maintenance:

- Abbiamo un nodo A che vuole inviare un pacchetto al nodo E.
- Il nodo A conosce il percorso da A ad E.
- Ogni nodo è responsabile di ricevere l'ack dal prossimo hop. Se un nodo non riceve l'ack allora può inviare il pacchetto di nuovo per un certo numero di volte.
- Se un collegamento non funziona allora il nodo riceverà un errore di broken link e il mittente rimuoverà il percorso dalla sua cache. Se conosciamo un altro percorso possiamo utilizzarlo per inviare il messaggio altrimenti possiamo avviare una route discovery per trovare un possibile percorso.

2.1.4 Altre feature della Route Discovery

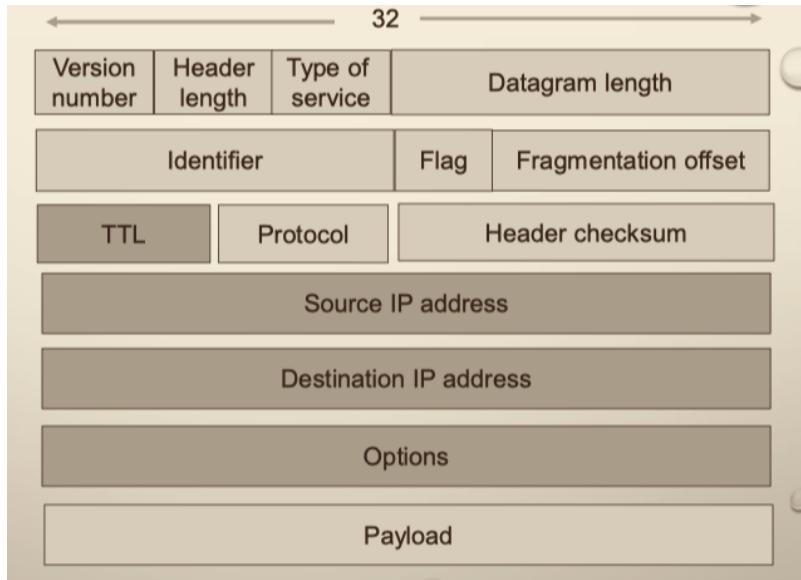
- Possiamo fare una ottimizzazione per la route discovery, quando un nodo intermedio tra il mittente e il target riceve il pacchetto RREQ e conosce già un percorso per arrivare al destinatario può rispondere al mittente con il percorso che conosce già, questo percorso non deve contenere dei cicli.
- Prima di inviare indietro un percorso da un nodo che è vicino al mittente e che conosce il target al mittente della RREQ dobbiamo attendere un certo tempo per evitare di avere collisioni, il tempo è

$$D = L(H - 1 + R)$$

dove la L è un ritardo costante, la H è il numero di salti e R è un numero random.

- Possiamo avere un TTL per i messaggi di route request.

DSR è un classico pacchetto IP che però utilizza un header speciale con delle opzioni.



2.2 AODV

AODV ha lo stesso obiettivo di DSR ma integra comunicazione unicast, broadcast e multicast.

Qua abbiamo che tutti i nodi vogliono partecipare al protocollo di rete e tutti quanti quindi inoltreranno i messaggi per gli altri nodi, i collegamenti tra i nodi sono bidirezionali.

AODV utilizza delle strutture dati per funzionare:

- Abbiamo una route table per la comunicazione unicast e multicast, abbiamo un percorso differente per ogni destinazione. Ogni percorso scade se non è usato dopo un certo periodo.
- Sequence Number: ogni nodo ha un sequence number che viene aumentato quando viene identificato un cambiamento nella topologia della rete.

2.2.1 Route Discovery

Anche qua quando abbiamo due nodi che vogliono comunicare il nodo S deve controllare se sa come raggiungere il nodo D e poi se non trova nessuna strada può avviare la route discovery.

Anche quaabbiamo un RREQ che viene utilizzato per capire dove si trova il nodo destinazione, la RREQ contiene:

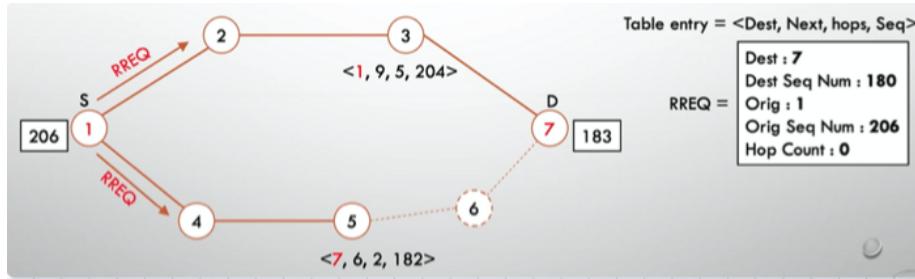
- L'ip e il sequence number di S (il sequence number viene incrementato quando eseguo la RREQ).
- L'ip della destinazione.
- Broadcast ID (viene incrementato quando avviamo una Route Request).
- Abbiamo un contatore degli hop.

Quando un nodo N riceve la RREQ inviata da S deve controllare se ha già visto quella RREQ (ogni nodo mantiene un database con le RREQ), se l'ha già vista allora la elimina altrimenti va avanti con la RREQ. Il nodo N memorizza la fonte e il sequence number della RREQ, il numero di hops per arrivare alla fonte e l'ip del vicino che ha mandato il pacchetto.

Durante la fase di route discovery:

- Il nodo S che non ha informazioni sul destinatario D invia la RREQ ai vicini.
- Il nodo destinazione risponde sempre a queste richieste.
- Il nodo che conosce già come arrivare a D controlla se le sue informazioni sono ancora valide o meno.
- Per evitare di inviare troppi messaggi nella rete settiamo un TTL iniziale e poi avviamo il processo, se non abbiamo trovato il nodo che stiamo cercando possiamo aumentare il TTL.
- Quando un nodo N riceve una RREP le informazioni contenute nell'RREP sono memorizzate nella routing table del nodo. Aggiungiamo 1 all'hop count e poi inoltriamo il messaggio alla fonte che ci aveva originariamente inviato la RREQ originale.
- Se abbiamo varie RREP allora la RREP può essere inviata solamente se il sequence number della destinazione è più alto (l'RREP è generato da un nodo che aveva un path più recente per la destinazione rispetto al precedente RREP) o se l'hop count è più basso.

Esempio di AODV



Vediamo un esempio di AODV abbiamo il seguente funzionamento:

- Il nodo 1 invia una RREQ ai nodi 2 e 4.
- I nodi 2 e 4 possono aggiungere 1 all'interno della routing table. Quindi, per esempio, aggiungono: < 1, 1, 1, 206 > ovvero la destinazione, il prossimo hop, quanti hop sono necessari per arrivare a destinazione e il sequence number della destinazione.
- Il RREQ viene inoltrato da 2 e da 4 ai nodi 3 e 5. Questi due nodi memorizzano le informazioni riguardanti i nodi 2 e 4. Ad esempio il nodo 3 aggiorna la sua routing table con 1, 2, 2, 206.
- Il nodo 5 conosce un percorso per arrivare a 7, quindi risponde ad 1 con un RREP. In questo RREP inserisce il sequence number di 7 che è 182. 4 riceve le informazioni e memorizza 7, 5, 3, 182.
- Il nodo 3 inoltra il messaggio al nodo 7 che è la destinazione. Il nodo 7 memorizza nella routing table 1, 3, 3, 206 e risponde con un RREP dove scrive che il sequence number di 7 è 184 (prima era 183 ma viene incrementato di 1 quando prepariamo la RREP). 1 riceve il RREP e ora aggiorna la routing table perchè il sequence number è maggiore:

$$< 7, 4, 4, 182 > - >< 7, 2, 3, 184 >$$

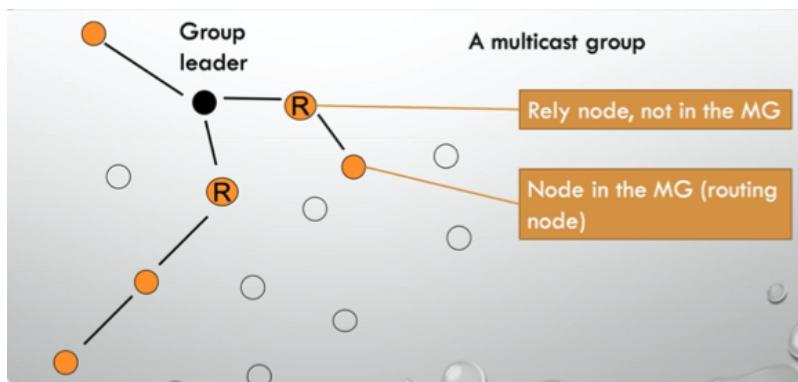
Hello Packet

In AODV i nodi sono anche responsabili del mantenimento delle connessioni tra i vari nodi. Esiste quindi un pacchetto speciale che viene chiamato Hello Packet e che viene utilizzato per informare i vicini che siamo ancora vivi. Quando un nodo riceve un Hello Packet da un nuovo nodo, può aggiornare la routing table. Se non ricevo un hello packet per un certo tempo eliminerò il vicino dalla routing table. Ogni nodo mantiene solamente i percorsi che sono attualmente attivi, in modo da essere sicuri che quando devo inviare qualcosa i percorsi sono ancora attivi.

Se un percorso non è più funzionante quando lo utilizziamo, i pacchetti vengono bloccati e viene inviato un messaggio di RERR al mittente. Il pacchetto RERR contiene la lista delle destinazioni irraggiungibili a causa del collegamento che non è più funzionante. Il RERR viene inoltrato a tutti i predecessori e ogni predecessore segnerà quel percorso come non valido (la distanza dei nodi verrà aggiornata e diventerà infinita). Quando arriva un RERR, il mittente inizia nuovamente la route discovery.

2.2.2 Multicast con AODV

Con AODV è possibile utilizzare il Multicast.



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Il Multicast con AODV ha le seguenti caratteristiche:

- Il gruppo multicast ha un leader, un multicast tree e un sequence number che viene mantenuto dal leader.
- Se vogliamo inviare un messaggio al gruppo iniziamo da uno dei nodi nel gruppo e poi quello lo inoltra agli altri nodi del gruppo.
- Ci sono varie route table per un multicast group, ogni nodo conosce la routing table del gruppo multicast.

Quando dobbiamo entrare in un gruppo multicast dobbiamo inviare una RREQ con una join request. A questa join request possiamo rispondere solamente se siamo all'interno del gruppo multicast, il leader può rispondere sempre, gli altri solamente se il loro sequence number è maggiore di quello nella RREQ. Se il nodo invia una RREQ senza però voler fare la join e vuole semplicemente informazioni sul percorso per quel gruppo, possono rispondere anche i nodi che non sono all'interno del gruppo.

Quando facciamo la procedura di join e dobbiamo aggiungere un nodo all'interno di un multicast group succede che tra il nuovo nodo e il leader del gruppo potrebbero esserci altri nodi che non fanno parte del multicast group. Questo fa sì che alcuni dei nodi siano chiamati "Rely". Questi nodi sono quelli che servono perchè hanno un percorso verso il multicast group ma non fanno parte del gruppo, il percorso verso il gruppo però viene abilitato solamente una volta che l'aggiunta al gruppo è stata confermata.

Quando inviamo la join e il messaggio viene ricevuto da un rely node allora il percorso per il nodo mittente viene inserito nella multicast routing table e risulta disattivato. Quando invece il messaggio di join arriva ad un nodo che non è relay, questo si limita solamente ad inoltrare il messaggio.

Se quando facciamo la join riceviamo vari RREP con vari possibili percorsi per il join del gruppo sceglieremo quello che è migliore per noi ovvero quello che è più vicino al Group Leader.

Come lasciare un gruppo Multicast

Ci sono varie possibili situazioni che possiamo avere quando vogliamo lasciare il gruppo multicast:

- Se il nodo che vuole uscire dal gruppo ed è una foglia del gruppo allora questo può semplicemente mandare un messaggio al padre e dire che sta uscendo dal gruppo.
- Se il nodo che vuole uscire non è un nodo foglia invece la situazione è diversa perchè quel nodo funziona anche da router per gli altri nodi, quindi, il nodo può uscire dal gruppo però rimarrà come router per gli altri.

Maintenance

I nodi devono essere connessi anche se non ci sono messaggi che vengono inviati in modo da mantenere il gruppo, per mantenere il gruppo si utilizzano gli hello packet che abbiamo già visto.

All'interno del gruppo abbiamo il downstream node, che sarebbe il nodo più lontano dal leader, che si occupa di riparare i collegamenti che non funzionano utilizzando di nuovo la RREQ per trovare un nuovo percorso per avere tutto il gruppo collegato.

Se non si riesce a trovare un nuovo percorso per tenere collegata tutta la rete allora dobbiamo dividere il gruppo in due parti ed eleggeremo un nuovo leader per la nuova parte del gruppo.

Questo nuovo group leader continuerà ad inviare route request e proverà a trovare un percorso in modo da creare nuovamente un singolo gruppo. Il messaggio che viene inviato dal group leader è chiamato

GRPH e quando un leader riceve il GRPH allora vuol dire che è di nuovo presente un singolo gruppo.

2.3 Confronto tra DSR e AODV

DSR:	AODV:
<ul style="list-style-type: none"> • ALLOWS MULTIPLE ROUTES • SUPPORTS UNIDIRECTIONAL LINKS • OVERHEARS AND CACHES ROUTING INFO 	<ul style="list-style-type: none"> • DOES NOT REQUIRE LONG HOP LISTS • SUPPORTS MULTICAST • HELLO MESSAGES TO CHECK CONNECTIVITY

Le performances sono differenti tra i due protocolli:

- Se abbiamo poco traffico e poca mobilità hanno delle performances simili e un ritardo accettabile.
- Se abbiamo tanta mobilità e tanto traffico, DSR sarà meno stabile a causa del fatto che abbiamo più possibili percorsi, AODV invece ha un overhead dovuto ai control packets.

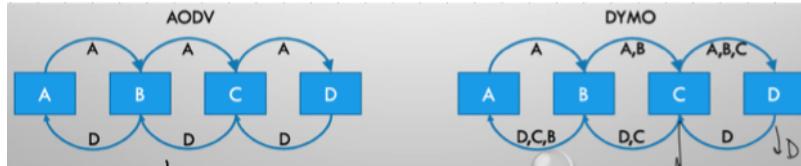
2.4 DYMO

DYMO è in qualche modo la somma dei due protocolli, vogliamo la semplicità di AODV ma vogliamo anche la possibilità di avere vari possibili percorsi come in DSR.

Per DYMO dobbiamo fare alcune assunzioni:

- Il processo di route discovery e di route maintenance è simile a quello di AODV ma non abbiamo gli Hello Packets.

- Prendiamo alcune idee da DSR con RREQ/RREP che portano informazioni ai nodi intermedi. RREQ e RREP sono differenti in DYMO.



Con AODV possiamo aggiornare solamente i dati riguardo la fonte e la destinazione, con DYMO invece possiamo anche utilizzare i percorsi memorizzati fino a quel momento per creare e aggiornare le informazioni memorizzate nei nodi intermedi.

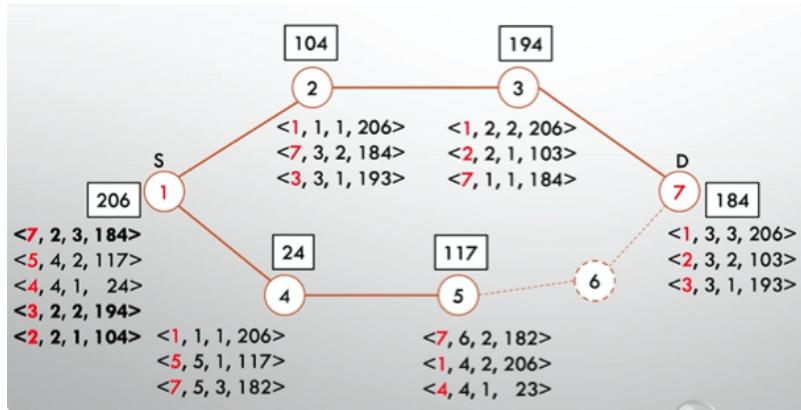
- In DYMO abbiamo dei timer all'interno della routing table, questi quando scattano permettono di fare qualcosa all'interno della routing table:

- **ROUTE_AGE_MIN:** MINIMUM TIME A RT ENTRY SHOULD BE KEPT
- **ROUTE_SEQNUM_AGE_MAX:** TIME AFTER WHICH SEQUENCE NUMBER IN THE RT ENTRY SHOULD BE DISCARDED (TO PURGE OLD INFO)
- **ROUTE_USED:** EVERY TIME A ROUTE IS USED THIS TIMER IS SET TO **ROUTE_USED_TIMEOUT**
- **ROUTE_DELETE:** THIS IS SET TO **ROUTE_DELETE_TIMEOUT** FOR A BROKEN ROUTE, AFTER IT EXPIRES THE ROUTE ENTRY IS REMOVED

Ad ogni nodo viene associato un sequence number come succede in AODV, il sequence number viene incrementato quando inviamo un RREQ e quando inviamo un RREP.

Se un nodo crasha e poi viene riavviato, il sequence number associato parte da 0. Per risolvere il problema dobbiamo memorizzare il sequence number nella memoria permanente altrimenti non potremo eseguire l'ingresso nella rete perché gli altri nodi avranno un sequence number più alto. Prima che i nodi possano entrare nella rete di nuovo devono attendere che scatti un certo timer.

Esempio



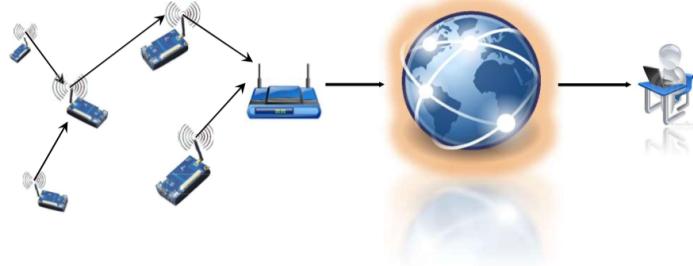
Vediamo questo esempio, qua abbiamo un esempio di RREP/RREQ e possiamo vedere la differenza con AODV.

- Qua abbiamo in ogni nodo i percorsi per ogni nodo che il pacchetto ha visitato.
- Con AODV possiamo trovare meno percorsi rispetto a quello che abbiamo con DYMO.
- La dimensione del pacchetto aumenta con il numero di nodi in DYMO mentre è costante in AODV.
- Se la mobilità è poca allora è meglio DYMO, se invece è alta allora in questo caso abbiamo più chances di avere informazioni errate nella routing table e quindi è meglio utilizzare AODV. Con DYMO abbiamo anche più overhead a mano a mano che la velocità di spostamento aumenta.

Chapter 3

Wireless Sensor Networks: Design Aspects

Data una Wireless Sensor Network una configurazione tipica è la seguente:



All'interno di una configurazione di questo genere abbiamo dei componenti che sono i sensori che sono solitamente dispositivi piccoli, autonomi, con costo basso. Ogni sensore ha una memoria, un processore, una batteria ed è in grado di ricevere e inviare segnali. I principali problemi dei sensori sono i seguenti:

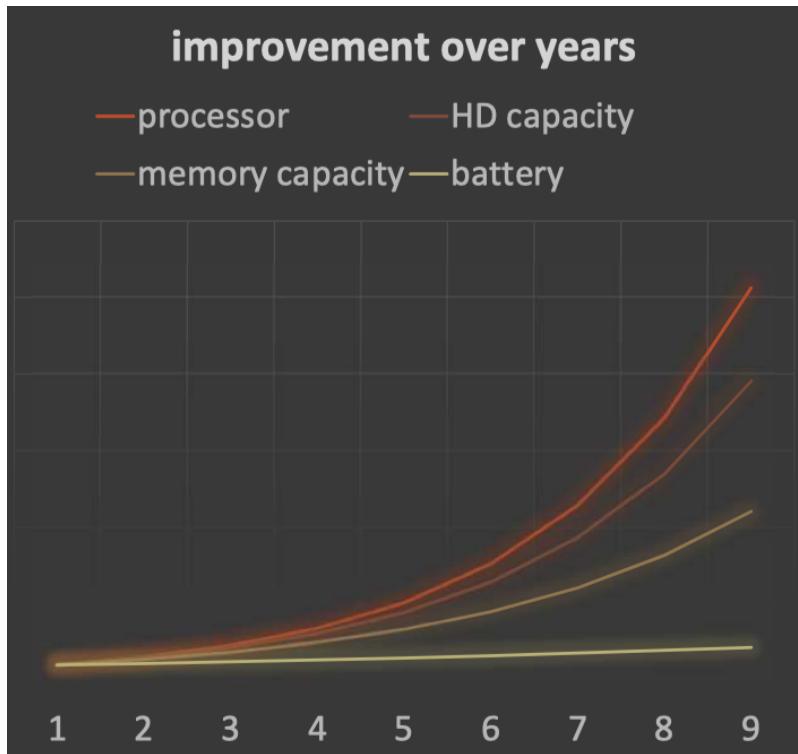
- Abbiamo un problema legato all'efficienza energetica. Le batterie presenti all'interno dei sensori sono piccole e abbiamo bisogno di

soluzioni efficienti per ridurre al minimo il consumo di energia.

- Abbiamo la necessità di avere un protocollo di comunicazione multi hop.
- Abbiamo necessità di un protocollo di routing dinamico perché i dispositivi possono essere mobili all'interno della rete.
- Ci sono anche problemi legati alla sicurezza delle comunicazioni.

3.1 Energy Efficiency

Il problema dell'energy efficiency è quello più importante in questa situazione. Quello che possiamo notare riguardo all'efficienza energetica è che nel corso degli ultimi anni si è visto un aumento delle capacità delle memorie e delle potenze dei processori, allo stesso tempo però le performance delle batterie sono sempre aumentate in modo lineare e questo è un problema perché abbiamo dispositivi sempre più potenti ma con batterie che rimangono sempre uguali.



Una cosa che possiamo notare in un sensore (e che lo differenzia rispetto ad un laptop per esempio) è il modo in cui la batteria viene utilizzata. In un laptop infatti gran parte della batteria viene consumata a causa dello schermo e del chipset mentre in un sensore abbiamo il 40% che viene consumato dal processore, il 40% dalla comunicazione wireless e il restante 20% dipende dai sensori. In particolare la parte dei sensori è abbastanza variabile e dipende molto dall'applicazione che stiamo utilizzando, spesso è qualcosa che non possiamo controllare. Le altre due cause del consumo invece dipendono da come organizziamo l'applicazione e quindi se noi cambiamo queste parti possiamo riuscire a rendere il sistema più efficiente.

Sempre considerando l'efficienza energetica possiamo vedere che sia per quanto riguarda il chip del Wi-Fi sia per quanto riguarda il sensore abbiamo dei consumi differenti a seconda dello stato in cui si trovano

questi componenti:

- Il chip Wi-Fi si può trovare nelle seguenti modalità:
 - Sleep Mode (10 mA): consumo minimo rispetto alle prossime modalità, in questa condizione però non puoi passare da una modalità all'altra velocemente.
 - Listen Mode (180 mA)
 - Receive Mode (200 mA)
 - Transmit Mode (280 mA): in questa situazione serve più consumo a causa dell'antenna.
- Il sensore invece può trovarsi nelle seguenti modalità:
 - Sleep Mode (0.016 mW)
 - Listen Mode (12.36 mW)
 - Receive Mode (12.50 mW)
 - Transmit Mode (da 12.36 a 17.76 mW): in questo caso il consumo dipende dal livello di potenza necessario per trasmettere, se il livello è basso abbiamo un consumo che è anche minore rispetto alla receive mode.

Come possiamo notare quello che consuma maggiormente la batteria è la fase di invio di dati e la fase di ricezione. Questo dipende dal fatto che in questi casi la radio rimane accesa per permettere la trasmissione. L'obiettivo in una sensor network quindi è quello di mantenere spenta questa radio il più a lungo possibile in modo da ridurre al minimo i consumi.

Lo stesso discorso vale anche per il processore, qua però abbiamo una scelta che è diversa rispetto a quella della radio perché mentre per la radio la scelta di spegnere/accendere dipende anche dagli altri device della

rete, per il processore abbiamo una decisione che è locale al dispositivo. Il processore possiamo capire quando può essere spento perchè sappiamo come si comporta il sensore, spegnere la radio invece vuol dire escludere il sensore dal network e se il sensore da solo potesse prendere la decisione di spegnere la radio vorrebbe dire che la rete andrebbe distrutta. Questo fatto che il processore può essere spento ha anche un effetto diretto sul protocollo Mac. Notare che anche lo spegnimento o l'accensione di radio e processore è qualcosa che consuma energia.

Duty Cycle

Considerando che i sensori fanno una vita abbastanza ripetitiva (eseguono una misurazione, processano il dato, lo memorizzano e poi lo inviano) e considerando che questo processo viene ripetuto di continuo, si è pensato di alternare un periodo in cui si lavora con un periodo in cui invece il sensore non è attivo. Durante il periodo in cui il sensore non è attivo allora il consumo dell'energia sarà molto basso perchè processore e radio verranno spenti. Questo periodo di attività/riposo del sensore viene chiamato duty cycle.

Notare che il duty cycle è legato al singolo componente e non al dispositivo completo, quindi possiamo avere un certo duty cycle per il processore e un duty cycle completamente diverso per il sensore. Notare anche che se noi spegnamo processore e sensori abbiamo comunque una perdita di batteria. Consideriamo il seguente esempio, questo è un sensore che viene usato nella ricerca e abbiamo che ci sono vari duty cycle all'interno del singolo sensore, l'altra cosa che notiamo è che il primo modello di sensore lavora di continuo mentre il secondo lavora e poi fa delle pause molto lunghe:

	model 1: 100%DC	model 2: 5%DC	units
Micro Processor			
current (full operation)	100	5	%
current sleep	0	95	%
Radio			
current in receive	50	4	%
current xmit	50	1	%
current sleep	0	95	%
Logger			
write	1	1	%
read	2	2	%
sleep	97	97	%
Sensor Board			
current (full operation)	100	1	%
current sleep	0	99	%

Per la misurazione dell'energia che viene consumata possiamo utilizzare in questo caso il milli ampere come unità di misura e non il Joule perchè possiamo assumere con questi dispositivi che il voltaggio è più o meno sempre costante.

Una volta che abbiamo deciso il duty cycle dei vari componenti del nostro dispositivo possiamo anche andare a fare una stima dei consumi che possiamo avere considerando il consumo che avremmo lavorando a pieno regime e considerando il tempo che passiamo senza lavorare:

- Energy cost microprocessor E_μ :

$$E_\mu = C_\mu^{full} \cdot dc_\mu + C_\mu^{idle} \cdot (1 - dc_\mu)$$

- C_μ^{full} full energy cost microprocessor
- C_μ^{idle} idle energy cost microprocessor
- dc_μ duty cycle microprocessor/100

- Energy cost radio E_ρ :

$$E_\rho = C_\rho^T \cdot dc_\rho^T + C_\rho^R \cdot dc_\rho^R + C_\rho^{idle} \cdot (1 - dc_\rho^T - dc_\rho^R)$$

- C_ρ^T radio transmission energy cost
- C_ρ^R radio receival energy cost
- C_ρ^{idle} idle energy cost

(

- dc_ρ^T transmission duty cycle radio/100
- dc_ρ^R transmission duty cycle radio/100

Una volta calcolati i costi dei singoli componenti possiamo calcolare il costo totale dell'energia necessaria per mantenere funzionante il dispositivo

$$E = E_\mu + E_\rho + E_\lambda + E_\sigma$$

e possiamo calcolare il lifetime che mi indica il livello di batteria che rimane ad un certo punto della vita del dispositivo:

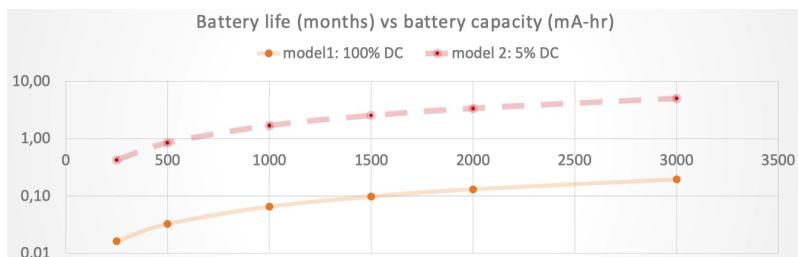
$$\frac{B_0 - L}{E}$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

All'interno della formula del lifetime vediamo che abbiamo B_0 che è la carica iniziale della batteria, L che è un valore che a sua volta dipende dal lifetime e mi indica l'energy leak e la E che è il consumo totale di energia. Date tutte queste formule possiamo trasformarle in una equazione di ricorrenza e se la risolviamo possiamo trovare il livello di batteria al termine di ogni duty cycle n .

$$B_n = B_0 \cdot (1 - \varepsilon)^{n-1} + \frac{E((1 - \varepsilon)^n - 1)}{L}$$

In particolare notiamo che il lifetime del dispositivo al tempo finale n è $B_n = 0$, dato questo risultato possiamo considerare il lifetime delle due applicazioni che stiamo considerando e poi lo possiamo anche plottare.



Nel grafico (con scala logaritmica) possiamo notare che abbiamo i due nostri modelli considerati in precedenza, uno in cui il duty cycle è 100% e uno in cui è 5%. La differenza sta nel fatto che, anche aumentando la capienza della batteria, la durata della batteria del modello con il duty cycle del 100% è di pochi giorni mentre con la stessa capacità il modello con il duty cycle del 5% ha una durata della batteria che può crescere fino a qualche mese.

Un'altra cosa interessante da notare è che dati due sensori con due capacità della batteria, quando andiamo ad aumentare il duty cycle abbiamo che il lifetime del sensore tende a diminuire. Quando il duty cycle

cresce abbiamo che i due dispositivi, pur avendo batterie differenti, raggiungono lo stesso lifetime. Questo ci dice che se vogliamo un dispositivo che sia efficiente, dobbiamo lavorare con un duty cycle che deve essere piccolo perchè questa è l'unica possibilità che abbiamo per far sopravvivere il sensore per mesi.

Quando diminuiamo il duty cycle c'è comunque da considerare che lo spegnimento del processore è una decisione locale che può essere presa dallo scheduler del dispositivo mentre invece lo spegnimento della radio richiede una decisione globale perchè un nodo con la radio spenta non contribuisce alla rete complessiva. Spegnendo il processore abbiamo anche degli effetti sul protocollo.

Mac Protocol

I protocolli Mac sono quelli utilizzati per la comunicazione a basso livello tra i vari sensori. Nelle Wireless Sensor Network i protocolli Mac sono utilizzati anche per implementare strategie legate all'efficienza energetica. In particolare questi protocolli permettono una sincronizzazione dei sensori e permettono di spegnere la radio quando non serve, questo implica che il sensore viene escluso ora dalla rete.

3.2 Multi-Hop Communication

Prendiamo in considerazione il caso in cui abbiamo un nodo A nella rete e un nodo B. Per comunicare devono avere le seguenti caratteristiche:

- A deve essere un nodo con una intensità nell'invio P_A
- B deve avere una intensità in ricezione P_B^r
- B riesce a sentire A solamente se $P_B^r \geq \beta$ dove al β è un valore che dipende dalla qualità della comunicazione che vogliamo ottenere.

La potenza necessaria per fare in modo che i due nodi possano comunicare dipende principalmente dal Path Loss tra A e B ovvero $PL(A, B)$ che a sua volta dipende dalla distanza tra i due nodi e quindi aumenta a mano a mano che i nodi sono più distanti.

$$PL(A, B) \sim d(A, B)^\alpha$$

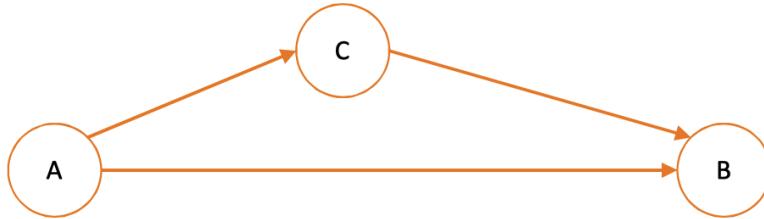
All'interno della formula abbiamo la α che è qualcosa che risulta difficile da stimare perché dipende molto dall'ambiente in cui ci troviamo e in particolare dal tipo di materiale delle mura. Di solito è un valore compreso tra 2 e 6. Per essere sicuri che B riceva è necessario che A trasmetta con intensità:

$$P_A \geq \beta PL(A, B)$$

Questo deriva dalle formule indicate in precedenza:

$P_B^r \geq \beta$ ma $P_B^r = P_A / PL(A, B)$ quindi abbiamo $P_A / PL(A, B) \geq \beta$ che diventa $P_A \geq \beta PL(A, B)$.

La comunicazione Multi-Hop è qualcosa che in queste reti può tornare utile ma non è detto che sia necessaria. Consideriamo il seguente caso, abbiamo A che vorrebbe trasmettere a B.



A può fare due cose:

- Trasmette direttamente a B
- Trasmette a C e poi C inoltra a B

Da un punto di vista dell'efficienza la potenza necessaria per questa trasmissione dipende tutta dalla distanza che abbiamo tra A e B.

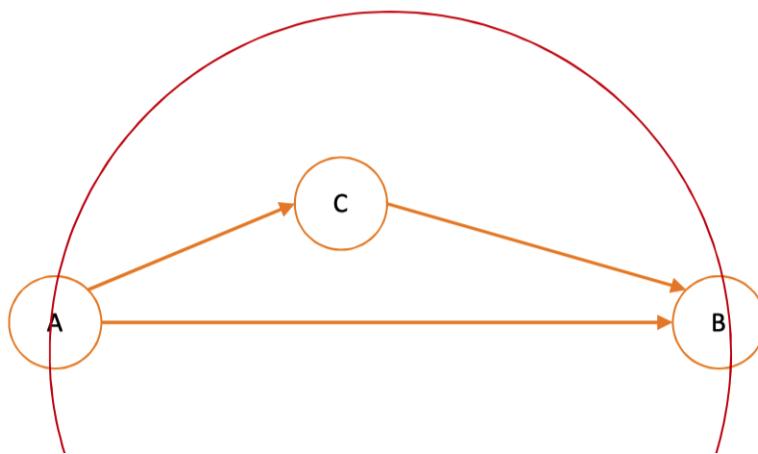
- Se utilizziamo il collegamento tra A e B e trasmettiamo direttamente abbiamo che la potenza necessaria dipende da $P_{A,B} = d(A, B)^\alpha$
- Se utilizziamo invece C avremo:

$$P_{A,C} + P_{C,B} = d(A, C)^\alpha + d(C, B)^\alpha$$

Se consideriamo $\alpha = 2$ avremo:

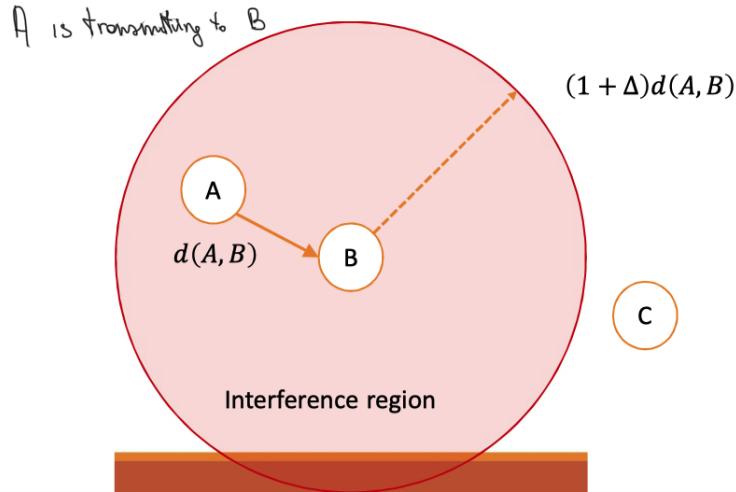
$$P_{A,B} = d(A, B)^2 = \underline{d(A, C)^2 + d(C, B)^2 - \cos \widehat{ACB} \cdot 2 \cdot d(A, C) \cdot d(C, B)}$$

Quindi data la formula sopra possiamo dire che, se il coseno è negativo allora vuol dire che è più costoso trasmettere direttamente a B.

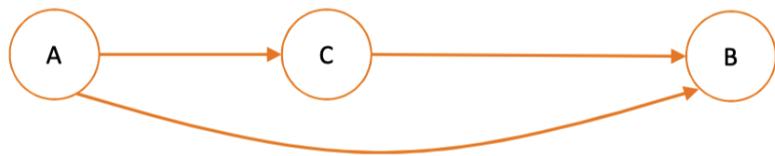


Se invece il coseno è positivo allora vuol dire che è meglio trasmettere direttamente a B perchè trasmettendo a C avremmo un costo maggiore.

Un'altra cosa che possiamo considerare quando si parla di comunicazione MultiHop sono le interference region:



Possiamo fare confronti tra le interference regions, più sono piccole e meglio è, questo vuol dire che possiamo permettere la comunicazione di altri sensori che si trovano all'esterno di queste regioni utilizzando la multi hop communication. Ad esempio possiamo considerare questo grafico:



E qua abbiamo che la distanza tra A e B è:

$$\pi d(A, B)^2 (1 + \Delta)^2$$

Mentre tra AC e CB è la seguente:

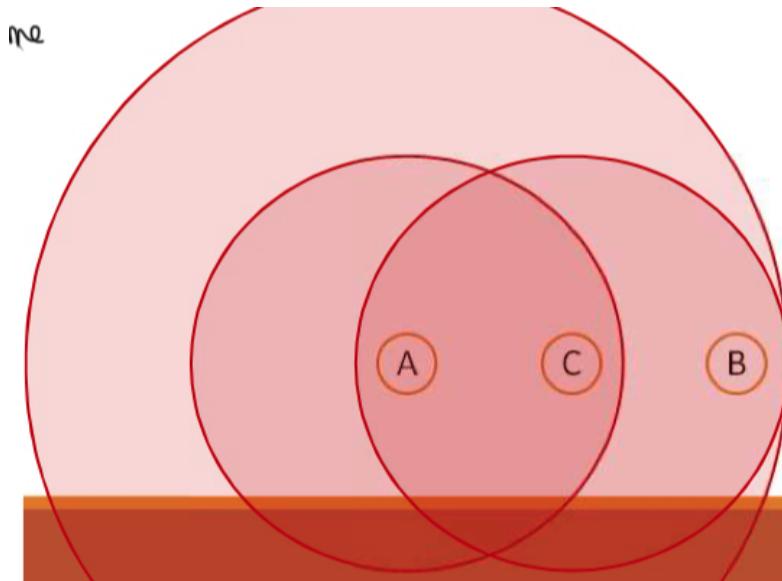
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$\pi d(A,C)^2(1 + \Delta)^2 + \pi d(C,B)^2(1 + \Delta)^2$$

Quindi possiamo fare un confronto tra queste due interference region e arriviamo a dire che è meglio passare per AC e CB invece che usare il collegamento diretto perchè:

$$\begin{aligned} \pi d(A,B)^2(1 + \Delta)^2 &\geq \pi(d(A,C)^2 + d(C,B)^2)(1 + \Delta)^2 \Leftrightarrow \\ d(A,B)^2 &\geq d(A,C)^2 + d(C,B)^2 \end{aligned}$$

In questo caso possiamo anche rappresentare l'interference range:



Queste sensor network originariamente lavoravano con un range molto piccolo mentre con il tempo ci sono stati cambiamenti e quindi ora il range è salito, questo però si paga perchè il throughput è più basso.

La rete di sensori la possiamo rappresentare come un grafo in cui i nodi sono i sensori e gli archi sono le connessioni tra i sensori. È una buona idea passare le informazioni al livello MAC per ottenere un funzionamento migliore della rete soprattutto in termini di efficienza energetica.

3.3 Application Issues

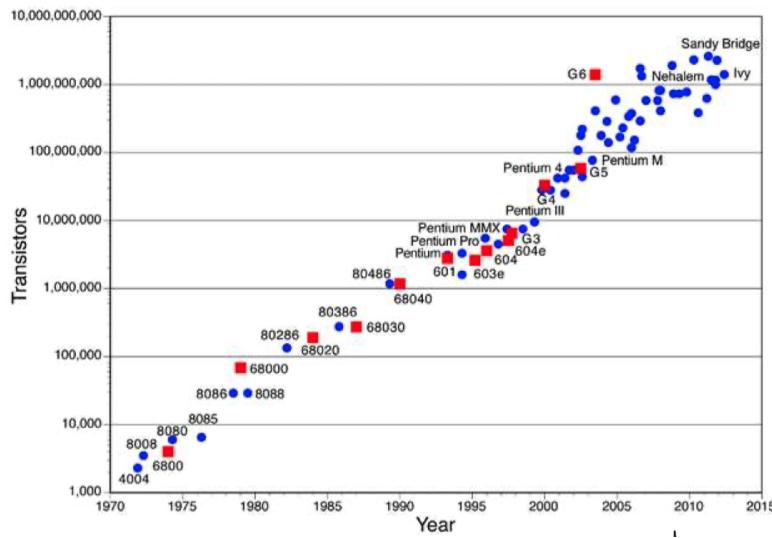
Quando parliamo di wireless sensor network ci sono alcune cose che dobbiamo tenere in considerazione:

- La rete potrebbe essere statica, in questo modo non inseriamo nuovi nodi all'interno della rete.
- La rete può essere dinamica, nuovi nodi possono entrare o uscire dalla rete. In questo caso ogni volta che un nodo lascia la rete o entra nella rete dobbiamo cercare di gestire il problema in modo da riconfigurare di nuovo la rete.
- I nodi presenti all'interno della rete possono offrire un servizio, ad esempio questo è il caso di Zigbee dove, dal punto di vista del sistema, i dispositivi stanno offrendo un servizio.
- Per quanto riguarda la programmazione di questi sensori, solitamente i sensori implementano task abbastanza semplici, possiamo avere la necessità di rendere i codice dinamico, è un altro problema dell'"Application" Layer.

Altri problemi a questo livello sono legati alla sicurezza delle sensor network, ci sono un sacco di problemi che possono avvenire (a vari livelli del nostro stack). I problemi di sicurezza partono dalle comunicazioni radio che possono essere facilmente intercettate, abbiamo poi problemi di integrità dei pacchetti che vengono mossi in questa rete e anche problemi legati all'autenticazione dei nodi all'interno della rete. Il problema principale riguarda la capacità ridotta del sensore, per questa ragione la crittografia simmetrica è la soluzione adottata solitamente.

Moore's Law

Pensando alle wireless sensor network una cosa che possiamo aspettarci è che con il passare del tempo l'evoluzione delle tecnologie possano portare ad una soluzione per tutti questi problemi. Questo è correlato alla legge di Moore secondo cui il numero di transistor che possiamo inserire in un chip cresce esponenzialmente e si raddoppia ogni due anni. Questo è anche quello che rende il settore dei computer diverso dagli altri settori, una lavatrice rimane più o meno sempre quella mentre un computer diventa "vecchio" in un anno. In pratica i produttori di computer lavorano per cercare di rispettare le previsioni della legge di Moore.



Nel caso delle wireless sensor network la legge di Moore ci offre una interpretazione differente:

- Le performance si raddoppiano ogni due anni e il costo rimane uguale.
- La dimensione del chip si dimezza ogni due anni allo stesso costo.

- La dimensione e la potenza del processore rimangono uguali ma il prezzo si dimezza ogni due anni.

Nella wireless sensor network queste tre interpretazioni potrebbero essere vere allo stesso tempo.

Queste evoluzioni della tecnologia fanno sì che quando riduciamo la dimensione del chip andiamo anche a ridurre i consumi del chip e quindi anche questo è un guadagno importante nel caso delle WSN. Un'altra cosa importante è che per alcuni utilizzi nel settore IOT abbiamo bisogno di maggiore potenza e quindi il fatto che i chip migliorino e diventino più potenti è ottimo (ad esempio pensiamo ad una telecamera di sorveglianza smart).

I costi sono un altro aspetto importante nelle WSN e infatti si tende a scegliere l'hardware sufficiente a sostenere il funzionamento dell'applicazione.

Chapter 4

WSN: Procolli

4.1 Protocolli MAC

I protocolli MAC utilizzati nelle Wireless Sensor Network non servono solamente per l'arbitraggio ma anche per garantire efficienza, gli obiettivi principali sono due:

- Ridurre il duty cycle della radio
- Mantenere la connettività all'interno della rete

Se i nodi all'interno della rete fossero sincronizzati tra loro vorrebbe dire che potrebbero spegnere la radio automaticamente e allo stesso momento, quando le radio sono attive allora la rete funziona e tutti sono connessi altrimenti non c'è connessione. Non basta solamente sincronizzare i nodi. Le cose che vanno decise sono il duty cycle e come questo potrebbe influenzare la latenza.

4.1.1 S-MAC

S-Mac è il primo protocollo che vediamo e S sta per sincronizzazione. L'idea chiave di S-Mac è che non c'è una sincronizzazione globale tra i

vari nodi ma solamente una sincronizzazione locale, i nodi alternano tra sleep mode (in questo periodo non possono ricevere messaggi) e listen mode. Questo protocollo è stato implementato anche in TinyOS che è un sistema operativo per sensori che viene usato nel campo della ricerca.

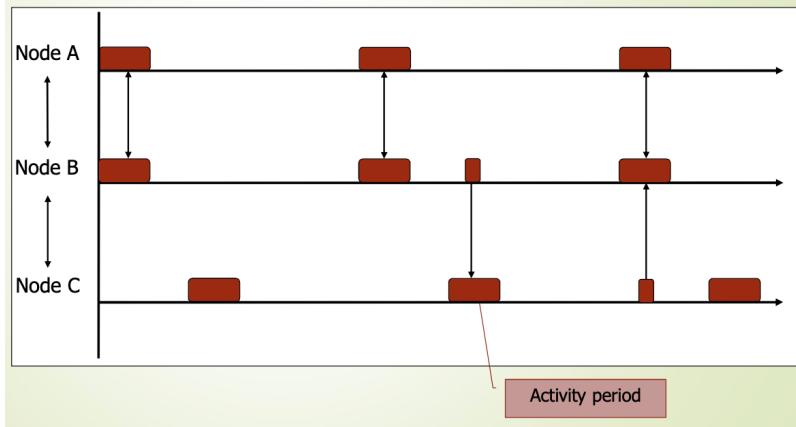
Come funziona la sincronizzazione?

I sensori che sono vicini tra loro possano inviare in broadcast un pacchetto chiamato SYNC che contiene i periodi di sleep/wakeup che lui sta adottando. Quando un sensore riceve un pacchetto SYNC si adeguà e passa ad utilizzare quello dei suoi vicini in modo da avere tutti lo stesso sistema di funzionamento.

Se ad esempio il nodo A vuole inviare un pacchetto a B allora il nodo A deve conoscere il listen period di B e quando A sa che B è attivo allora potrà inviare il pacchetto. Allo stesso modo questo vale per gli altri nodi che sono nei pressi di A.

Il primo nodo che entra nella rete sceglie un suo tempo di sleep/wakeup e poi rimane in attesa che nuovi dispositivi si connettano. Se un nuovo dispositivo si connette potrebbe iniziare a lavorare con un suo tempo e poi una volta ricevuto il SYNC allora passa al tempo indicato dal vicino. In S-MAC c'è anche la possibilità che un sensore che si trova molto lontano dagli altri possa utilizzare un sync period differente.

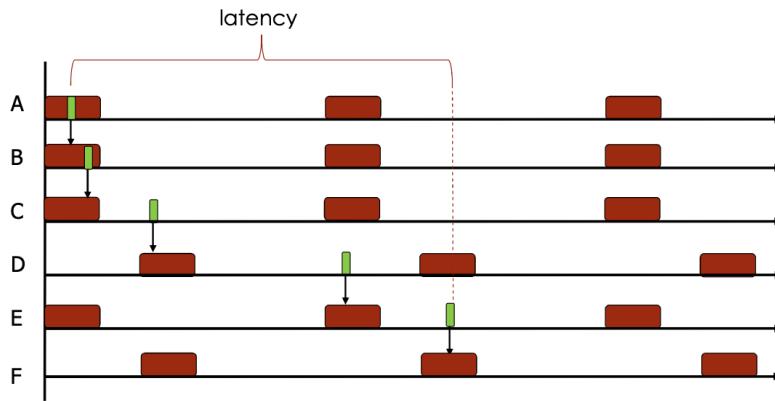
Nella figura dell'esempio abbiamo che A e B non sono sincronizzati con C quindi vengono attivati in tempi differenti:



Se ad esempio B volesse comunicare con C allora B dovrebbe accendere la radio durante l'activity period di C per fare in modo che C possa ricevere il messaggio. Quando un nodo deve comunicare con un altro, deve attendere il listen period del ricevente, prima di tutto però deve controllare il canale per essere sicuro che non ci siano collisioni. Se il canale è già occupato allora si rimanda la trasmissione.

Questo sistema ha però alcuni lati negativi:

- La latenza in questo caso può essere un problema perché se ad esempio ho un pacchetto ed uso il Multihop per inviarlo, potrei dover attendere in ogni nodo intermedio il listen period del nodo successivo. Questo problema viene un minimo mitigato se consideriamo che almeno in teoria i vari nodi della rete dovrebbero convergere ad una organizzazione simile tra loro.



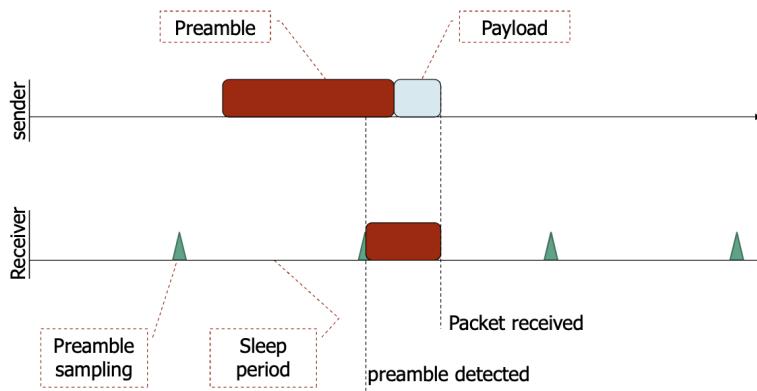
- A seconda della topologia della rete potrebbe essere impossibile mantenere la sincronizzazione dei vari sensori e in alcuni casi potrebbe anche essere necessario trovare dei protocolli che controllino che gli schedule siano ancora validi e in caso mantenerli aggiornati.

4.1.2 B-MAC

B-Mac è il primo esempio di protocollo in cui viene utilizzato il Preamble Sampling. Anche questo è stato implementato in TinyOS e in questo caso non si sfrutta la sincronizzazione dei sensori. Con il B-Mac un nodo presente nella rete può avviare la trasmissione quando vuole. Prima di inviare i dati però il nodo invia un preamble e ogni nodo che sente il preamble poi ascolterà anche i dati che vengono inviati dal nodo. In pratica ricevere il preamble vuol dire mantenere accesa la radio e attendere un messaggio. Tutto questo meccanismo viene chiamato preamble sampling. Per controllare che nell'aria ci sia un preamble, ogni nodo dovrà accendere la radio periodicamente in modo da controllare ed eventualmente ricevere il preamble. Questa attività viene chiamata preamble sampling.

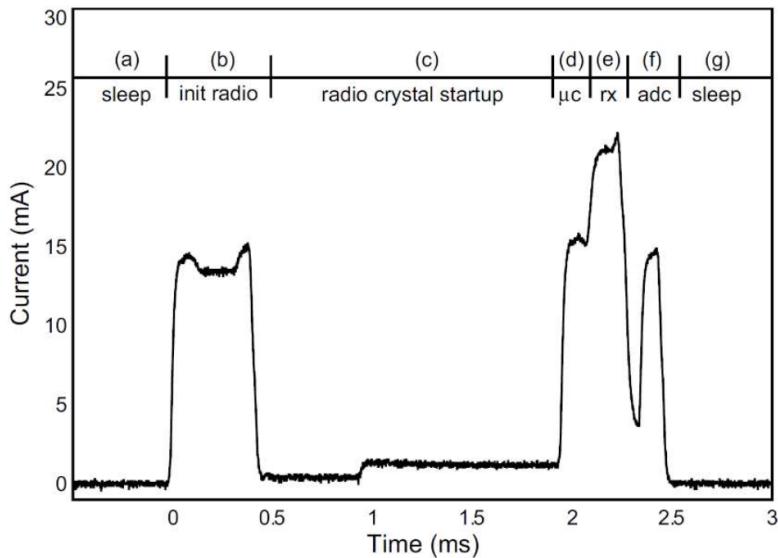
L'idea in questo caso è quella di cercare un bilanciamento tra l'energia che si spende per inviare un pacchetto e l'energia necessaria per riceverlo.

Dato che un solo nodo invia e tanti ricevono, quello che è preferibile è spendere più energia per la trasmissione dei dati e meno energia per la ricezione, ecco il motivo per cui si preferisce tenere attiva l'antenna di chi invia per un tempo più lungo e fare solamente dei preamble sampling negli altri nodi. In particolare una cosa che va tenuta a mente è che il preamble sampling interval deve essere lungo a sufficienza ma allo stesso tempo anche il preambolo deve essere lungo abbastanza da fare in modo che gli altri nodi possano accorgersi della sua esistenza.



Questo è quello che succede nella rete, con il mittente che invia un preamble che poi viene ricevuto dagli altri nodi quando accendono la radio. Quando poi il nodo ricevente si accorge del preambolo allora il mittente dovrà inviare il dato vero e proprio.

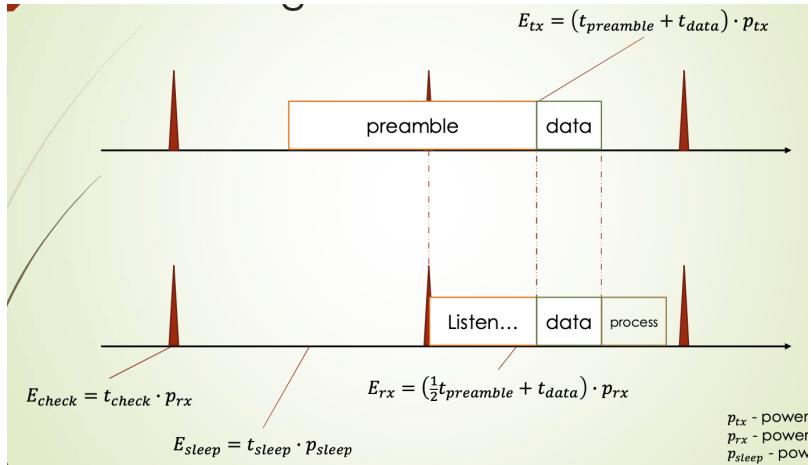
Quello che succede alla radio in pratica è qualcosa di questo genere:



la radio ha vari stati possibili:

- Sleep Phase: La radio parte con un consumo di energia pari a 0.
- Init Radio: il consumo di energia aumenta.
- Radio Crystal Setup: Il consumo poi scende di nuovo.
- Quando c'è da ricevere un pacchetto il consumo aumenta di nuovo.

Possiamo modellare la vita di questo sistema in questo modo e con le seguenti formule:



Ogni sensore deve fare un preamble sampling, anche chi trasmette, anche se poi verrà ignorato il risultato. Quindi i consumi che abbiamo sono i seguenti:

- $E_{check} = t_{check} * P_{rx}$ è il consumo per eseguire un preamble sampling.
- $E_{sleep} = t_{sleep} * p_{sleep}$ è il consumo che abbiamo mentre il nodo è addormentato.
- $E_{rx} = (\frac{1}{2}t_{preamble} + t_{data}) * p_{rx}$ è il consumo che abbiamo per ricevere i dati e il preamble.
- $E_{tx} = (T_{preamble} + T_{data}) * p_{tx}$ è il consumo che abbiamo per l'invio di preamble e dati.

Oltre a calcolare il consumo di energia possiamo anche calcolare i duty cycle di mittente e destinatario, supponendo di avere solamente un mittente e un destinatario.

- ▶ f_{data} frequency of transmitted data ($\frac{1}{sec}$)
- ▶ f_{check} frequency of preamble sampling

Hence, at transmitter:

- ▶ Duty cycle data: $DC_{tx} = f_{data} \cdot (t_{preamble} + t_{data})$
- ▶ Duty cycle check: $DC_{check} = f_{check} \cdot t_{check}$
- ▶ Energy (in Joule) spent in t seconds:

$$ET(t) = t \cdot (p_{tx} \cdot DC_{tx} + p_{rx} \cdot DC_{check} + p_{sleep} \cdot (1 - DC_{tx} - DC_{check}))$$

- ▶ Duty cycle data: $DC_{rec} = f_{data} \cdot \left(\frac{1}{2}t_{preamble} + t_{data}\right)$
- ▶ Duty cycle check: $DC_{check} = f_{check} \cdot t_{check}$
- ▶ Energy (in Joule) spent in t seconds:

$$ER(t) = t \cdot (p_{rx} \cdot DC_{rec} + p_{tx} \cdot DC_{check} + p_{sleep} \cdot (1 - DC_{rec} - DC_{check}))$$

Thus, the lifetime:

- ▶ Transmitter: $lifetime = battery_{charge}/ET(1)$
- ▶ Receiver: $lifetime = battery_{charge}/ER(1)$

Questo B-MAC ha alcuni pro e alcuni contro:

- Una cosa positiva è che è trasparente ai livelli più alti e che non è un protocollo di organizzazione della rete.
- È un protocollo che funziona con un solo parametro, quindi è facile da configurare.
- Una cosa negativa è che il preamble che inviamo è abbastanza lungo.
- In alcuni casi potrebbe essere più costoso di usare una forma di sincronizzazione.

In generale possiamo dire che il lifetime di trasmettitori e ricevitori dipende da quante volte facciamo il check del preamble e da quanti dati vengono inviati.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

4.1.3 X-MAC

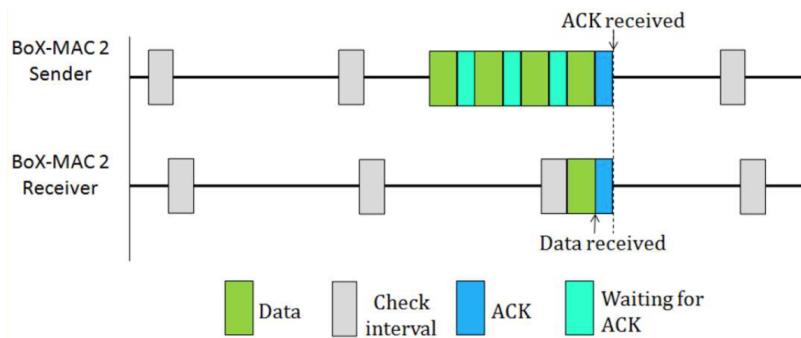
Il problema che si verifica con il B-Mac è che quando abbiamo un preamble molto lungo il ricevente deve rimanere in attesa di ricevere tutto il messaggio completo e questo può comportare uno spreco di energia per il ricevente che deve tenere accesa la radio fino a quando non ha ricevuto tutto il preamble. Per ridurre questo spreco di energia è nata l'idea di X-MAC che invece di mandare un preamble lungo manda tanti preamble brevi, in questo modo quando il receiver riceve il preamble invia subito un ack a chi trasmette in modo che questo possa a sua volta inviare i dati. Questo comporta anche che con X-Mac si possa utilizzare un intervallo per il check del preamble più grande rispetto a quello che avevamo in B-Mac.

Qua con X-Mac si possono verificare dei problemi di sicurezza se l'attaccante va ad eseguire lo spoofing dell'ID del ricevente. Ci sono anche altri possibili attacchi a questi protocolli.

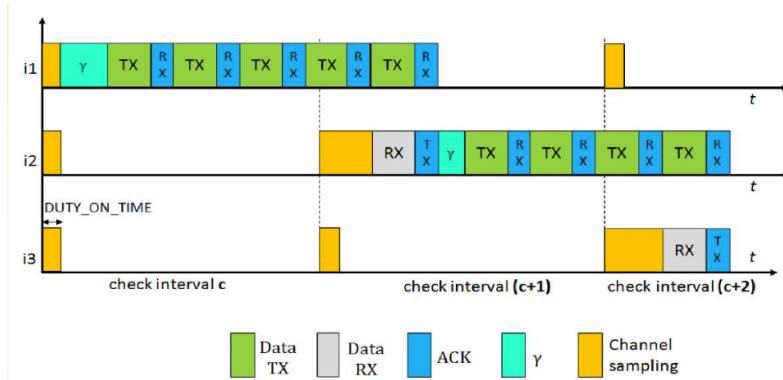
4.1.4 Box-Mac

Box-Mac è un'altra estensione di B-Map che cerca di ridurre l'impatto di un preamble lungo più di quanto venga fatto con X-Mac. Invece di mandare un preamble e poi i dati in Box-Mac viene inviato direttamente il messaggio, questo approccio funziona solamente nelle sensor network perché in queste reti i data frame sono molti piccoli.

Come funziona in questo caso il sistema? Qua noi abbiamo il mittente che invia più volte il dato e poi quando il ricevente fa il preamble sampling allora riceve subito i dati e invia l'ack. Una volta ricevuto l'ack poi il mittente si blocca.



Questo protocollo può essere utilizzato in un routing multi hop, assumiamo di avere un nodo 1 che vuole inviare al nodo 3. Il nodo 1 inizia a inviare i dati, possiamo utilizzare il nodo 2 che riceve i dati da 1 e poi li invia al nodo 3. In questo caso il problema è la latenza perchè ognuno dei riceventi deve attendere il periodo del preamble sampling e poi ricevere i dati, quindi abbiamo un tempo che è 2,5 volte maggiore del normale.



4.2 Polling

Si tratta di un meccanismo implementato ad esempio in Bluetooth, è un approccio difficile da implementare in una rete grande o su larga scala. Viene utilizzato quando abbiamo una organizzazione asimmetrica dei nodi nel senso che abbiamo un dispositivo che fa da master (e che invia

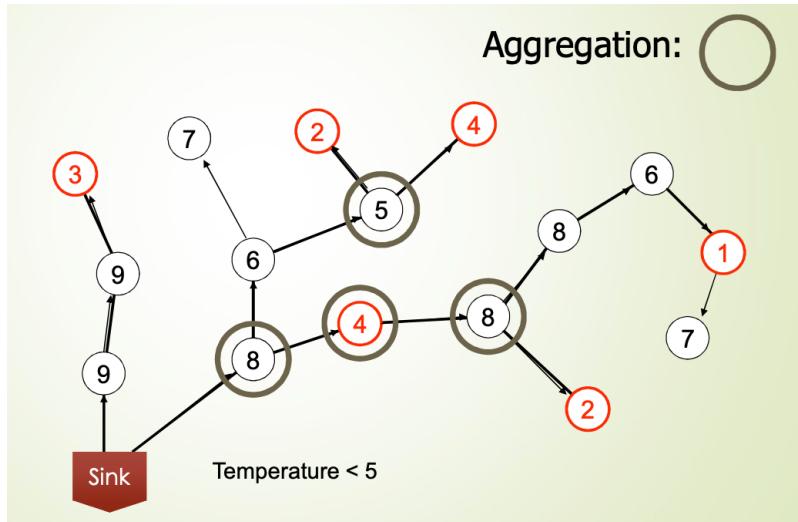
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

un beacon periodicamente) e un dispositivo che fa da slave. Il master riceve il messaggio dallo slave e poi questo si memorizza il messaggio nella sua memoria e avvisa gli altri nodi della presenza del nodo che gli ha inviato il messaggio (tramite l'invio del beacon). Quando uno degli slave attiva la radio attende il beacon, si accorge che c'è un messaggio e poi richiede al master il messaggio.

4.3 Network Protocol

Ora che abbiamo visto i vari protocolli MAC possiamo concentrarci sul network level e possiamo vedere come vengono organizzate le attività in una sensor network.

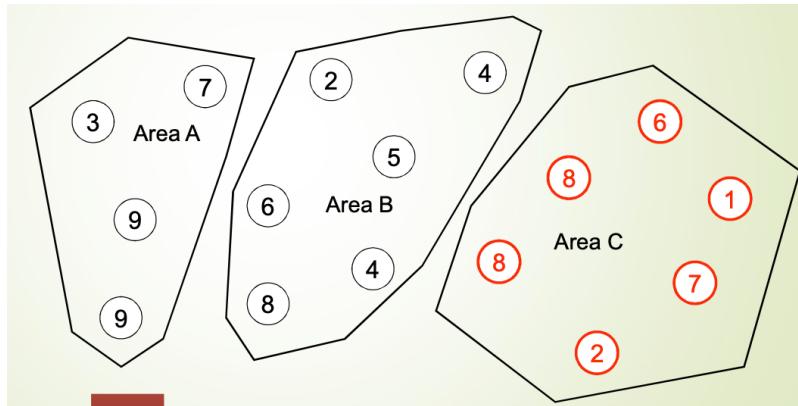
Nelle Wireless sensor network abbiamo i sensori che sono prevalentemente data centric. Se invece consideriamo internet possiamo notare che utilizziamo un indirizzo IP per ognuno dei dispositivi e questo è perchè utilizziamo un server che è il DNS per la trasformazione da symbolic address a IP. Se consideriamo una wireless sensor network non ci interessa conoscere l'indirizzo dei nodi della rete ma ci interessano le capacità di questo nodo, ad esempio il fatto che può produrre informazioni. All'interno della rete siamo particolarmente interessati ai dati che vengono prodotti e una cosa importante da notare è che quando i dati si muovono nella rete allora possiamo avere anche una aggregazione di questi dati che, in alcuni casi sarà utile e in alcuni casi invece non sarà utile.



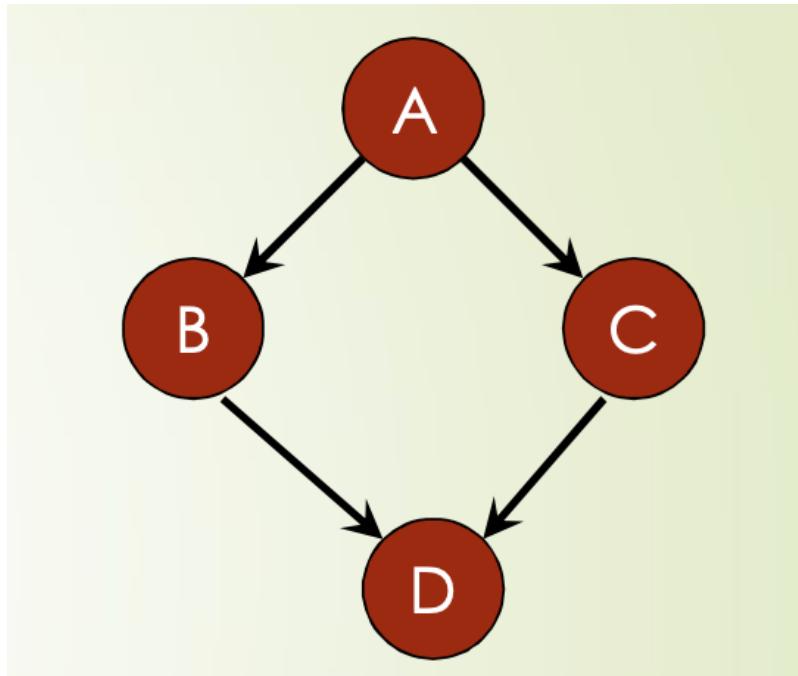
In particolare nell'esempio consideriamo tutti i nodi e ci interessano soprattutto quelli in cui la temperatura è minore di 5. Le informazioni viaggiano nella rete e in alcuni nodi abbiamo dei punti di aggregazione dove uniamo i dati che provengono dagli altri nodi. Notare che il nodo 6 che troviamo tra 5 e 8 non deve fare l'aggregazione perché la fa già il nodo 5 perché è più vicino ai due nodi (2 e 4) che inviano i dati, non fa l'aggregazione anche perché il nodo 7 non gli manda niente al nodo 6 dato che la temperatura nel nodo 7 è maggiore di 5.

Il fatto che conoscere l'ID di un nodo non mi serve a nulla mi porta a pensare che i classici protocolli di routing non sono in realtà interessanti e utilizzabili in queste situazioni.

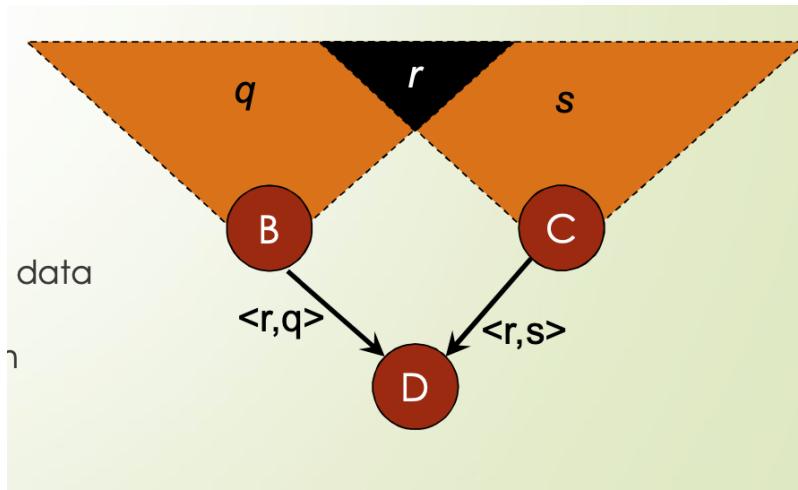
Un'altra cosa che ci permette di ottenere questa sensor network orientata in particolare ai dati è la location awareness ovvero riusciamo a capire dove si trovano i vari nodi della rete e possiamo suddividere la rete in varie aree:



Questi protocolli per le sensor network però hanno alcuni problemi nel momento in cui andiamo a riempire la rete con i nostri messaggi. In particolare quello che si verifica è che se un nodo invia a tutti i vicini un pacchetto, questi poi a loro volta lo inoltrano ad altri nodi e quindi ci saranno nodi che riceveranno più di un pacchetto.



Questo problema si chiama overlap problem, se due sensori coprono un'area che si sovrappone allora i sensori invieranno i loro dati e poi il nodo D riceverà lo stesso pacchetto da B e da C perché B e C non sanno che in realtà il pacchetto è lo stesso:



4.4 Directed Diffusion

Si tratta di un protocollo che mescola routing e application layer, questo protocollo è stato proposto per creare delle reti e per ricevere delle query dal nodo sink, ci permette di inviare delle query nella rete e ricevere delle risposte. L'idea è che la sensor network dovrebbe essere programmata per rispondere alle query. Quando questo sistema è stato proposto le query erano molto semplici, ora abbiamo query molto più complesse e i sensori che abbiamo ora sono in grado di gestirle.

Tutte le comunicazioni servono per richiedere dati o per ottenerli, in particolare se i dati sono associati con un nome allora la richiesta che facciamo può anche specificare un nome. Un nodo può richiedere dati inviando una notifica di interesse per un certo dato, l'unico nodo che può fare richieste è il nodo sink.

Come funziona questo sistema:

- I dati hanno un nome e li possiamo identificare con una coppia *attributo, valore*.
- Il sink (che sarebbe il coordinatore della rete) prepara un messaggio di interesse e codifica uno specifico nome poi lo invia nella rete.
- Ogni sensore ha un gradiente che dipende dall'interested ovvero ogni sensore una direzione lungo la quale i dati dovrebbero essere inoltrati.
- C'è anche un altro elemento che possiamo utilizzare che è il reinforce. Questo è un modo che viene usato per "rinforzare" uno o più percorsi all'interno della rete.

Un esempio per il tracciamento degli animali:

```
type = four-legged animal      // detect animal location
interval = 20 ms               // send back events every 20 ms
duration = 10 seconds          // .. for the next 10 seconds
region = [-100, 100, 200, 400]  // from sensors within rectangle
```

In questo caso il sink indica per prima cosa il tipo di animale che stiamo cercando, in questo caso è un animale con 4 zampe, il tipo è quello che rende il sensore data centric. Poi specifichiamo un intervallo che dice al sensore quale deve essere la frequenza di sampling, queste informazioni possono essere utilizzate per eseguire una cross layer optimization. La duration mi indica per quanto tempo dovrò eseguire il sensing. Le regioni invece riguardano la location awareness e sono definite con un box.

In risposta alla interest request, il destinatario risponde con una lista di dati che seguono uno schema simile.

```

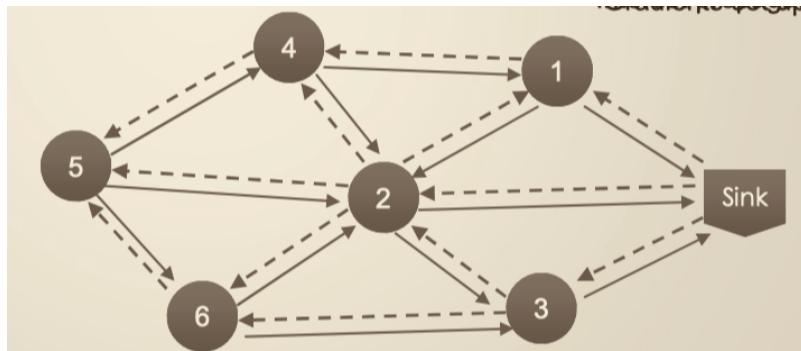
type = four-legged animal    // type of animal seen
instance = elephant          // instance of this type
location = [125, 220]         // node location
intensity = 0.6               // signal amplitude
confidence = 0.85             // confidence in the match
timestamp = 01:20:40          // event generation time

```

Consideriamo che nell'esempio abbiamo animali che hanno al collo un collare che è in grado di segnalare l'animale, la posizione e le altre informazioni. Nella risposta il tipo è quello dell'interest e abbiamo in particolare l'istanza dell'animale che stiamo considerando che, in questo caso, è un elefante. L'istanza è quello che il sensore è stato in grado di rilevare. Il sensore poi mi indica la posizione dell'animale e mi dice quanto il segnale è forte. Tipicamente questa risposta è associata ad un timestamp e ad una confidence che mi indica quanto è vera l'informazione che mi è stata inviata nella risposta.

4.4.1 Funzionamento

Quando il sink è interessato ad una informazione specifica la prima cosa che fa è inviare in broadcast a tutti i nodi della rete un primo "Interest" in cui indica che gli interessa una certa informazione. La comunicazione broadcast però non è affidabile e quindi per specificare meglio il "sensing task" il sink deve inviare nuovamente lo stesso "Interest". Questo secondo invio si chiama refresh e serve per aumentare l'affidabilità della comunicazione broadcast e per rinforzare alcuni percorsi specifici. Quando un nodo riceve il refresh può iniziare l'attività di sensing (anche se è in ritardo rispetto agli altri) e può anche inoltrare l'interest ai vicini.



Questa propagazione degli interest fa sì che si vada a creare un grafo diretto aciclico che possa coprire l'intera rete. I nodi che ricevono un certo interest lo memorizzano nella cache per un certo intervallo di tempo e poi l'interest scade. Per ogni interest che viene ricevuto da un nodo viene calcolato anche un gradiente che mi indica il nodo da cui l'ho ricevuto. Questo è utile perché in questo modo quando dovrò rispondere potremo inviare la risposta a chi aveva mandato l'interest.

Notare che ogni nodo del grafo può ricevere l'interest da più di un vicino quindi potrei dover calcolare più di un gradiente. Prima o poi tutti i nodi della rete riceveranno l'interest. Ad esempio prendiamo il nodo 5, questo riceve i dati da 4, 2 e 6 e quindi dovrà avere il gradiente verso questi tre nodi. La conseguenza è che quando il 5 rileverà un evento che è rilevante per l'interest che ha ricevuto dovrà utilizzare i tre gradienti che ha calcolato per spedire la risposta a tutti e tre.

Seguendo questo ragionamento il sink riceverà varie risposte, per evitare questo, può decidere di dare priorità ad una certa direzione in modo che quella sia quella privilegiata e le altre siano meno importanti.

Vediamo cosa succede quando un sensore rileva un evento che corrisponde ad uno degli interest che ha nella cache:

- Il sensore ha la responsabilità di rilevare gli eventi e di controllare che matchino con quelli memorizzati nella cache.
- Il sensore inizia a inviare l'evento ai suoi vicini (basandosi sulle

informazioni del gradiente che ha memorizzato nella cache), per l'invio si utilizza il sampling rate più alto.

- Se nella cache il sensore ha memorizzati differenti priorità per i vari vicini allora deve andare a utilizzare un sampling rate differente a seconda della priorità.
- Quando un vicino riceve i dati deve a sua volta inoltrarli, per capire se questi dati vanno inoltrati e anche a chi vanno inoltrati il vicino deve vedere nella sua cache e se c'è nella cache un interest per quell'evento allora inoltra i dati ai suoi vicini sempre basandosi sul gradiente.

Reinforcement

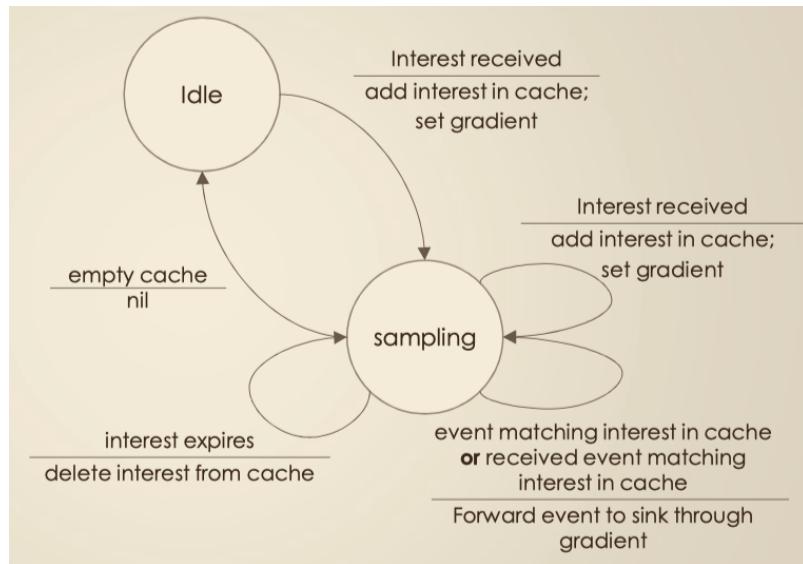
In alcuni casi potrebbe essere necessario migliorare la qualità dei dati ricevuti, per farlo possiamo utilizzare una tecnica chiamata reinforcement. Supponiamo di avere il sink che inizia a ricevere dati che matchano un certo interest da un sensore n. Il sink può "rinforzare" n in modo da migliorare la qualità dei dati ricevuti, in questo modo andiamo ad aumentare il sampling rate lungo quel percorso e facciamo in modo che i dati vengano propagati su un percorso specifico in modo che solamente quel percorso possa sopravvivere.

Affidabilità

Parlando di affidabilità possiamo dire che ogni volta che un nodo rileva un evento deve inoltrare la risposta basandosi sul gradiente, il sink riceverà queste risposte da vari nodi presenti all'interno della rete, questo è un modo per aggiungere affidabilità al sistema perché anche se perdiamo una risposta i dati continuano a muoversi in vari percorsi e quindi arrivano comunque al sink.

Con il reinforcement alcuni percorsi vengono preferiti ma non è un problema perchè comunque i messaggi viaggiano continuamente nella rete e prima o poi arriveranno al sink. Non è un problema neanche avere ridondanza dei dati all'interno della rete, si tratta di una assunzione di Direct Diffusion.

Possiamo rappresentare il funzionamento di Direct Diffusion utilizzando una macchina a stati:



4.4.2 Analisi di Direct Diffusion

Direct Diffusion ha i seguenti pro:

- È semplice da usare e implementare, ogni sensore implementa una macchina a stati finiti. Funziona anche bene su dispositivi non troppo costosi e non troppo potenti, quindi dispositivi con poca memoria e con poche capacità del processore.
- È un sistema scalabile perchè il carico di lavoro viene distribuito tra tutti i sensori presenti nella rete.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Funziona con applicazioni che non richiedono di preprocessare i dati prima di lavorarci perché qua diffondiamo direttamente i dati senza fare nulla prima.
- È anche un sistema robusto perchè utilizziamo una comunicazione broadcast all'interno del grafo.

Direct diffusion è semplice da utilizzare perchè abbiamo solamente un sink che lavora come coordinatore e poi abbiamo il grafo in cui il sink fa da "fonte". Il grafo è diretto e aciclico e i dati che troviamo al suo interno vengono refreshati periodicamente a causa della possibilità di perdere dati. Dobbiamo fare alcune assunzioni in questa rete:

- Abbiamo un solo sink che viene indicato con l'id 0.
- Ogni nodo ha un id univoco.
- Il sink esegue l'inizializzazione della rete e poi mantiene il sistema di routing. L'inizializzazione viene effettuata con una comunicazione broadcast che è simile a quella che viene usata poi per eseguire il refresh.
- Il sink è l'unico che può utilizzare il broadcast per la comunicazione mentre invece tutti gli altri nodi possono utilizzare solamente una comunicazione unicast.
- Assumiamo anche che ognuno dei sensori della rete conosce quello che deve fare e il sink non deve programmare il sensore.

Per il funzionamento del direct diffusion vengono utilizzate le seguenti strutture dati:

- Il sink ha:
 - Un contatore dei messaggi broadcast che sono stati emessi

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Un timer per l'inizializzazione del routing tree
- Un timer che serve per eseguire il refresh della routing tree.
- I nodi presenti all'interno della rete invece hanno:
 - Un contatore di messaggi broadcast ricevuti dal sink
 - Memorizzano gli ID dei nodi padre, ovvero dei nodi verso cui vanno inviate eventuali informazioni che matchano gli interest
 - Un booleano che indica se il routing tree è già stato creato o no.
 - Il timer per il prossimo data sampling.

Quando deve inizializzare la rete il sink invia un messaggio di tipo RTInit, quando poi un nodo deve rispondere al sink perché ha matchato un interest allora utilizza un messaggio di tipo "Data".

Vediamo ora come funziona in generale il sink:

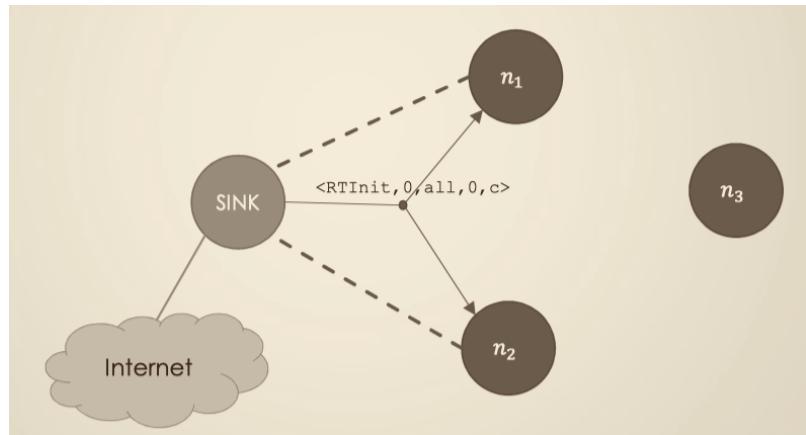
- Abbiamo un metodo init in cui vengono inizializzati i timer e viene inizializzata la costruzione del routing tree. Poi il sink va in sleep mode.
- L'inizializzazione della routing tree viene svolta con una funzione che invia un messaggio a tutti i vicini del sink. Questo messaggio è di tipo RTInit. La prossima cosa che viene fatta è settare il timer per il refresh della routing tree in modo che quando il timer terminerà allora sarà possibile inizializzare di nuovo il routing tree.
- C'è anche una funzione per la ricezione dei messaggi, solamente se il messaggio è di tipo Data viene preso in considerazione, altrimenti viene scartato (ad esempio se è di tipo RTInit).

e un nodo qualsiasi:

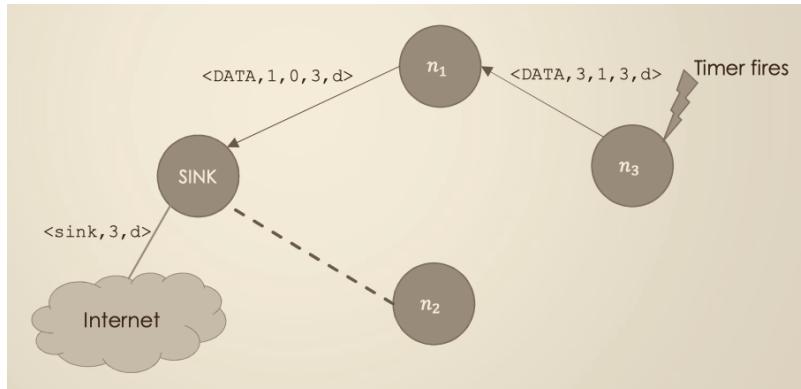
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Anche qua abbiamo una funzione di inizializzazione in cui avviamo il timer del sample time.
- Quando scatta questo timer dobbiamo controllare se il routing tree è formato o meno, se non è formato allora possiamo settare di nuovo il timer altrimenti se è formato leggiamo i dati e inviamo un messaggio di tipo DATA.
- C'è poi anche un'altra funzione che è la funzione di receive che gestisce la ricezione di messaggi, dobbiamo considerare il tipo, se il messaggio è di tipo DATA possiamo inoltrarlo senza cambiare la fonte, se è un messaggio RTInit dobbiamo eseguire di nuovo l'inizializzazione e quindi quello che facciamo è aggiornare la routing tree. Se il messaggio è diverso non viene considerato.

Consideriamo un semplice esempio di Wireless Sensor Network:



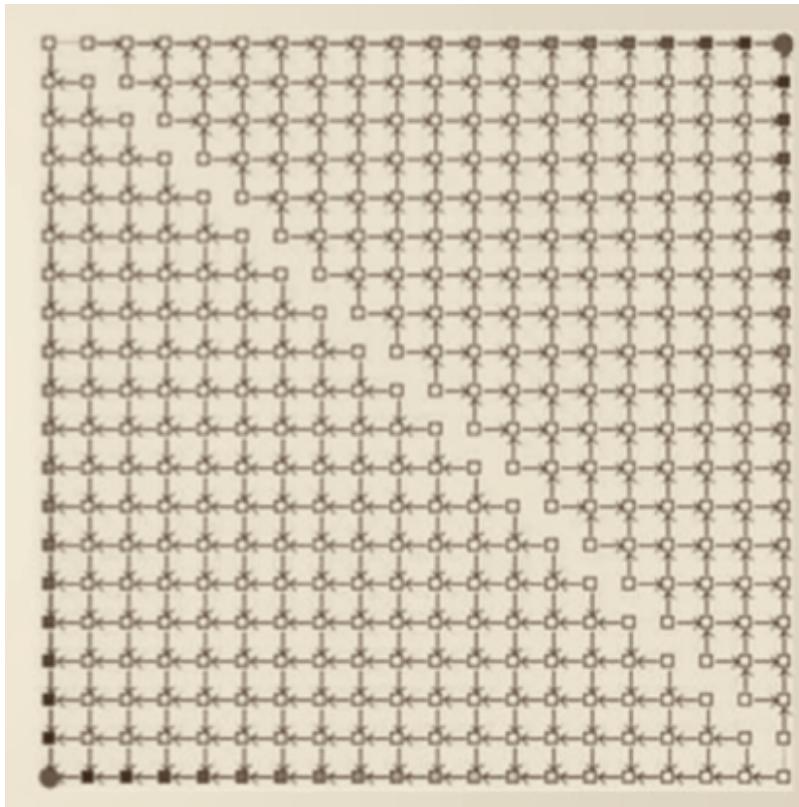
In questa rete abbiamo il nodo sink e altri tre nodi, il nodo sink che vuole inizializzare il routing tree deve inviare un messaggio di RTInit a n_1 e a n_2 . Questi a loro volta inoltreranno il messaggio di inizializzazione a n_3 .



Supponiamo ora che nel nodo n_3 ci sia un timer che scatta, i dati vengono prima inviati al nodo n_1 e poi vengono da questo nodo inoltrati al sink. Il sink a sua volta inoltrerà questi dati verso internet.

4.4.3 Possibili problemi per Direct Diffusion

Abbiamo detto che avere la nostra rete organizzata mediante un albero ci permette di avere scalabilità. Supponiamo di avere una rete connessa come in una griglia e supponiamo di avere due sink, possiamo vedere nella seguente immagine che ci sono alcuni nodi che lavorano più di altri e che quindi hanno un consumo maggiore. Sono i nodi vicino ai nodi sink.



Per risolvere parzialmente questo problema potremmo pensare di utilizzare più sink all'interno della rete, in ogni caso si presenta comunque il problema che i nodi che stanno più vicini ai sink consumano di più.

Un limite di questo sistema è che il sink deve essere permanentemente connesso con la Internet. Un altro limite riguarda poi il processing dei dati, con questa organizzazione della rete non riusciamo ad eseguire un processing dei dati all'interno della rete.

Questi problemi richiedono di utilizzare quindi algoritmi differenti e più complicati.

4.4.4 Applicazioni più complesse

Quando consideriamo le wireless sensor network possiamo anche pensare ad applicazioni più complesse. Una cosa che possiamo pensare è che in alcuni casi potremmo voler rilevare non un evento semplice ma un evento utilizzando tanti sensori che fanno parte di un'area molto grande. Un esempio lo abbiamo con dei sensori di inquinamento, qua noi vorremmo far comunicare i sensori in modo da scambiarsi i dati e vorremmo anche eseguire una aggregazione distribuita dei dati. Con il protocollo attuale non possiamo farlo ma possiamo fare dei cambiamenti al protocollo.

Alcuni esempi di task più complessi

Application:	Input bandwidth	Computational demand	output bandwidth	compression factor
Image (e.g. tracking voice/sound (e.g. speech control)	80 Kbps 256 Kbps	1 GOPS 100 MOPS	0,16 Kpbs 0,02 Kbps	500 x 12800 x
Inertial sensors	2,4 Kbps	10 MOPS	0,02 Kbps	120 x
Biometrics	16 Kbps	150 MOPS	0,08 Kbps	200 x

Con questi task abbiamo la necessità di eseguire un po' di processing sui dati direttamente sul dispositivo che rileva i dati. Ad esempio vediamo che se lavoriamo con le immagini (ad esempio se stiamo usando una smart camera) abbiamo una input bandwidth che è molto alta rispetto all'output perchè abbiamo lavorato sui dati direttamente nel sensore. La differenza input-output ci indica che quindi vale la pena eseguire un po' di computazione direttamente sul sensore.

4.5 GPSR: Greedy Perimeter Stateless Routing

GPSR è un'alternativa a Direct Diffusion e ai protocolli simili che permette di produrre un routing node to node senza mantenere uno stato e

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

avendo un overhead basso.

Anche in questo caso abbiamo delle assunzioni da fare:

- Dobbiamo assumere che i nodi si trovano in uno spazio bidimensionale, per la maggior parte delle applicazioni questa assunzione non è un problema.
- I nodi devono sapere dove si trovano e devono anche sapere dove si trovano i vicini di quel nodo. Per questa assunzione abbiamo bisogno di un sensore GPS all'interno del sensore, questo implica consumi più elevati in termini di energia.
- Il nodo di origine di un certo messaggio conosce il nodo di arrivo del messaggio e conosce anche la sua posizione.

In questo protocollo i nodi mantengono solamente informazioni locali e non c'è necessità di inviare messaggi in broadcast per eseguire la route discovery, abbiamo anche meno messaggi per controllare lo stato della rete.

4.5.1 Funzionamento di GPSR

Ci sono due modalità di funzionamento per GPSR:

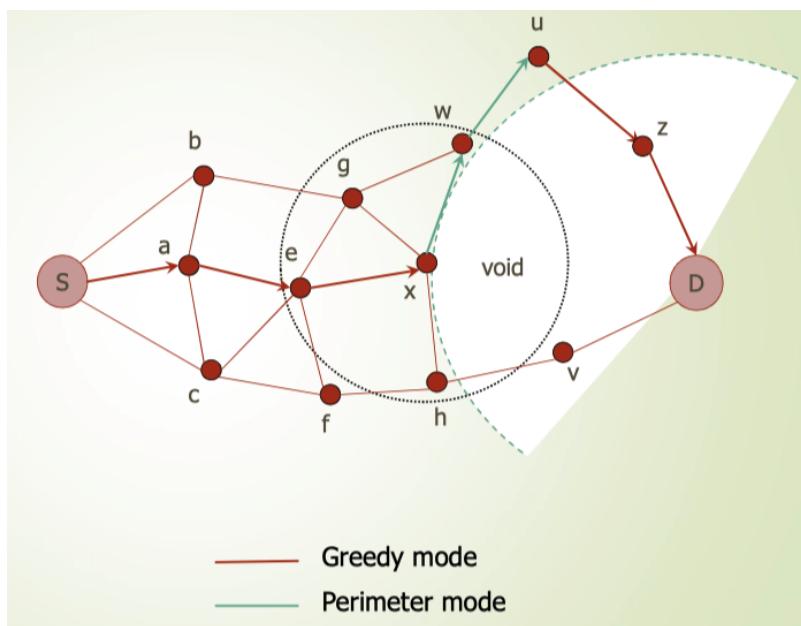
- Greedy Forwarding
- Perimeter Forwarding

Nel caso del Greedy Forwarding dobbiamo considerare un pacchetto che ha una certa destinazione D. Supponiamo di trovarci in un nodo X e di dover inoltrare questo pacchetto al nodo D, dal nodo X dobbiamo selezionare il prossimo nodo a cui inoltrare il pacchetto basandosi su due regole:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Il nodo a cui inoltriamo il pacchetto è più vicino a D rispetto al nodo X.
- Il nodo a cui inoltriamo il pacchetto è il più vicino a D rispetto a tutti gli altri vicini di X a cui X avrebbe potuto inoltrare il pacchetto.

Quando il nodo X non ha nessun vicino che è vicino a D più di lui allora si passa dalla greedy mode alla perimeter mode. Durante la perimeter mode il nodo X identifica i nodi che si trovano sul bordo della zona in cui si trova il nodo destinazione e inoltra il pacchetto a questi nodi.

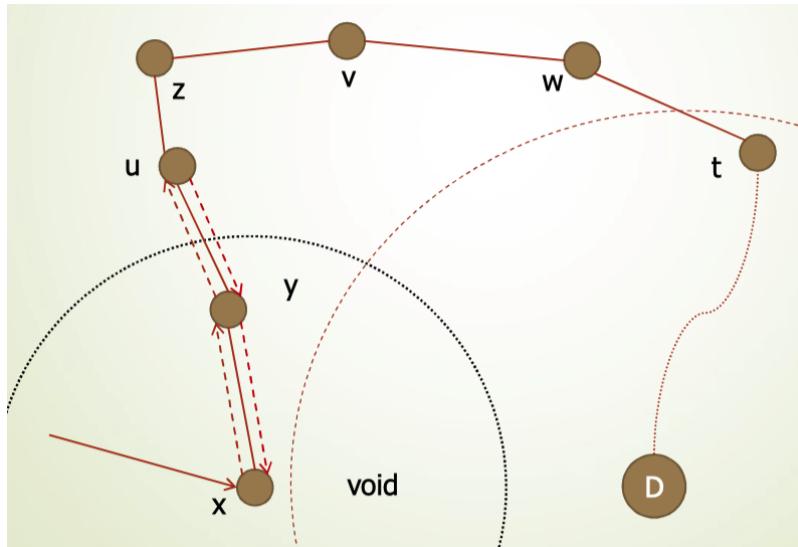


Nel caso della figura fino al nodo X abbiamo la greedy mode mentre invece da X in poi inizia la perimeter mode.

Quando possiamo tornare ad utilizzare nuovamente la greedy mode?

In generale abbiamo una politica abbastanza aggressiva per il passaggio da perimeter mode a greedy mode, vogliamo fare il passaggio il più velocemente possibile.

Consideriamo il seguente esempio?



Prendiamo il caso in cui il dato è in X e deve arrivare a D. Non possiamo inoltrarlo direttamente a D e quindi dobbiamo inoltrare il dato a Y, poi a U, poi Z, V, W, T ed infine D.

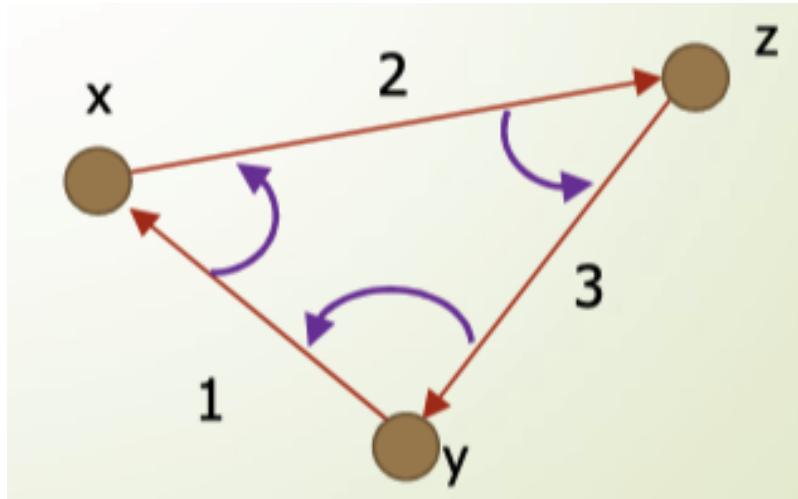
In alcuni punti di questo grafo potremmo chiederci se fosse possibile passare alla greedy mode, ad esempio quando siamo nel nodo Z. Se passassimo alla greedy mode nel nodo Z vorrebbe dire andare a creare un loop perché i pacchetti verrebbero inoltrati di nuovo a U. Il perimeter mode in questo caso viene mantenuto fino a quando non si arriva al nodo W, a questo punto possiamo inoltrare il pacchetto al nodo T.

La regola generale per il passaggio da perimeter mode a greedy mode:

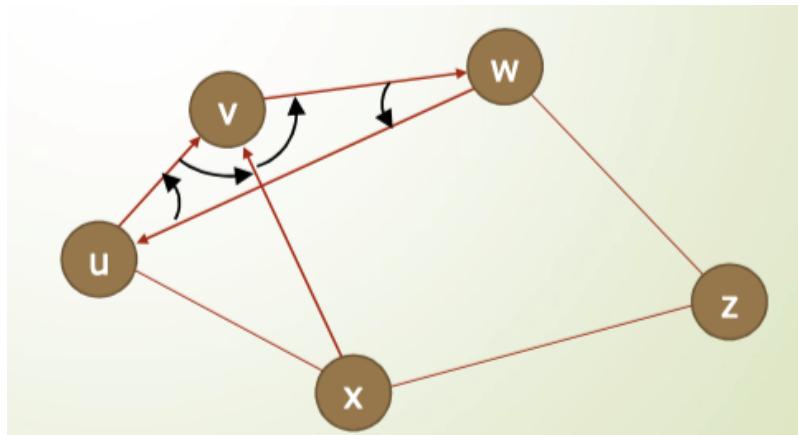
GPSR torna ad utilizzare la greedy mode quando trova un nodo che è più vicino alla destinazione D rispetto al nodo X che per primo è passato alla perimeter mode. Nell'esempio il nodo W trova la T che è più vicina a D rispetto a X e quindi passa di nuovo alla greedy mode.

Per implementare il perimeter forwarding possiamo usare la "right hand rule" o la "left hand rule", la cosa importante è sceglierne una ed

utilizzarla per tutto il tempo necessario.



I pratico quello che succede è che quando mi trovo nel nodo y arriviamo dal nodo z tramite la right hand rule e utilizziamo la right hand rule per andare nel nodo x . Con questo metodo riusciamo a muoverci lungo tutti i nodi del nostro grafo, il metodo funziona se il grafo ha un planar embedding (vuol dire che gli archi del grafo non si devono sovrapporre). Quando ci troviamo a dover lavorare con grafi che non sono planar non possiamo utilizzare le regole viste fino ad ora e non possiamo solamente applicare la right rule o la left rule.



Applicando banalmente queste regole ci troveremmo in un loop e quindi l'unica soluzione è quella di trasformare il grafo non planar in un grafo planar in modo da poter utilizzare la perimeter mode così come è stata spiegata.

Per la trasformazione di grafi non planar in grafi planar possiamo applicare vari algoritmi differenti:

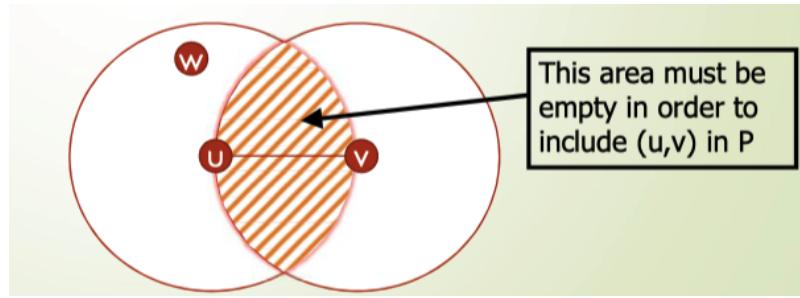
- Relative Neighborhood Graph of G
- Gabriel Graph of G

Dopo aver fatto questa trasformazione valgono alcune proprietà:

- Se G è un grafo connesso allora anche P sarà un grafo connesso.
- P lo otteniamo da G rimuovendo alcuni archi.
- P viene calcolato con un algoritmo distribuito, è un qualcosa che è interamente localizzato all'interno del nodo (usiamo solamente informazioni locali al nodo e riguardanti i vicini del nodo), è importante questo perchè è meno costoso.

Il grafo viene planarizzato appena viene creata la rete, vediamo anche che quando abbiamo la perimeter mode per l'inoltro possiamo utilizzare solamente i link all'interno del grafo planarizzato mentre durante la greedy mode utilizziamo tutti i possibili link.

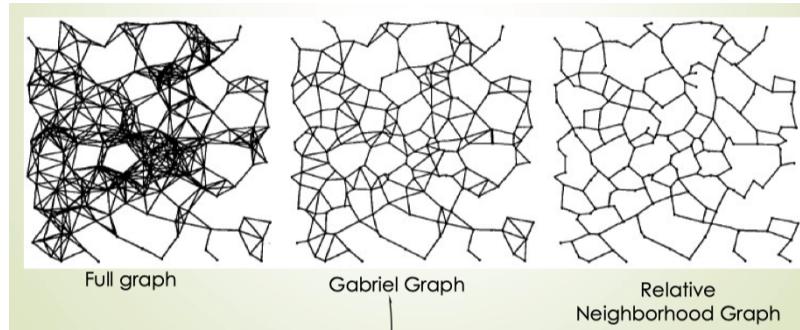
- **Relative Neighborhood Graph:** partiamo dal grafo G e vogliamo ottenere il grafo P. La regola è che se abbiamo due nodi, u e v tracciamo i cerchi con centro in u e in v e consideriamo la loro intersezione. Se all'interno della loro intersezione non ci sono altri nodi allora l'arco tra u e v viene mantenuto, altrimenti lo rimuoviamo.



La regola più formalmente è la seguente:

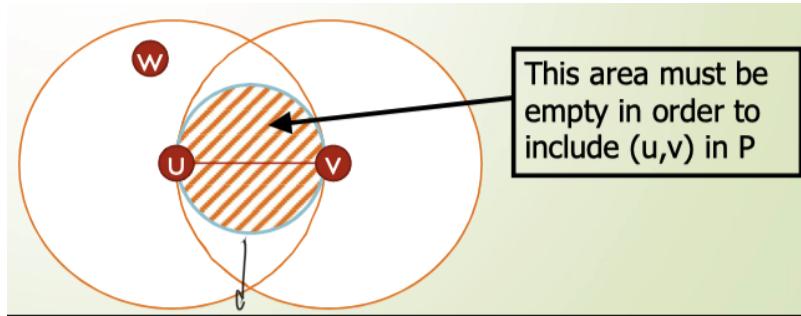
$$\begin{aligned} \text{Edge } (u, v) \in P \text{ iff:} \\ (u, v) \in G \wedge \\ d(u, v) \leq \max_{\forall w \in N(u) \cup N(v)} (d(u, w), d(w, v)) \end{aligned}$$

- **Gabriel Graph:** in questo caso teniamo più nodi rispetto a quelli che abbiamo con l'algoritmo precedente, in particolare possiamo dire che il grafo che otteniamo con Relative Neighborhood Graph è un sottografo di quello che otteniamo con il Gabriel Graph.



In questo caso per rimuovere un certo arco che connette due nodi prima disegniamo i due cerchi centrati nei due nodi e prendiamo la parte comune, poi consideriamo il cerchio che passa per i due nodi

e manteniamo il collegamento solamente se non abbiamo altri nodi all'interno del cerchio:

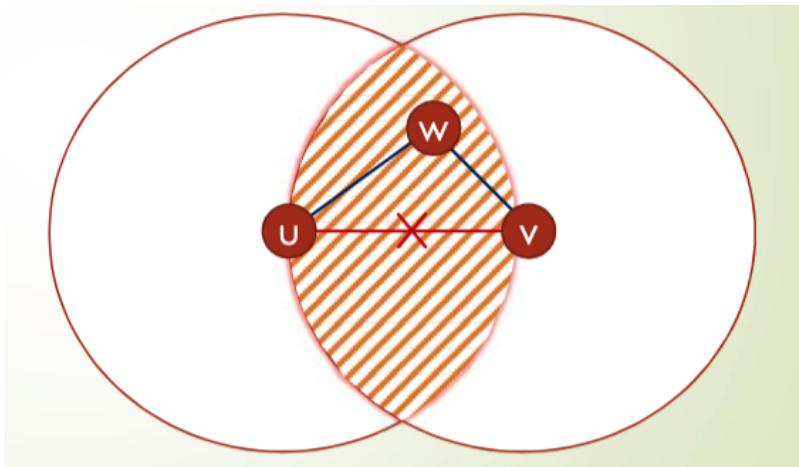


La formula in questo caso è la seguente:

► Edge $(u, v) \in P$ iff:

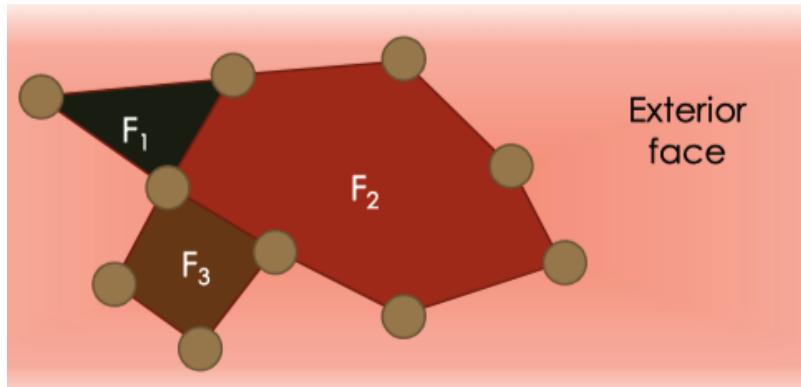
$$(u, v) \in G \wedge d(u, v)^2 \leq d(u, w)^2 + d(w, v)^2 \forall w \in N(u) \cup N(v)$$

In entrambi i casi quando rimuoviamo un arco non ci sono problemi e il grafo rimane connesso perchè se rimuoviamo l'arco vuol dire che all'interno della zona in comune abbiamo comunque un altro nodo che connette i due nodi che erano connessi con l'arco che eliminiamo.



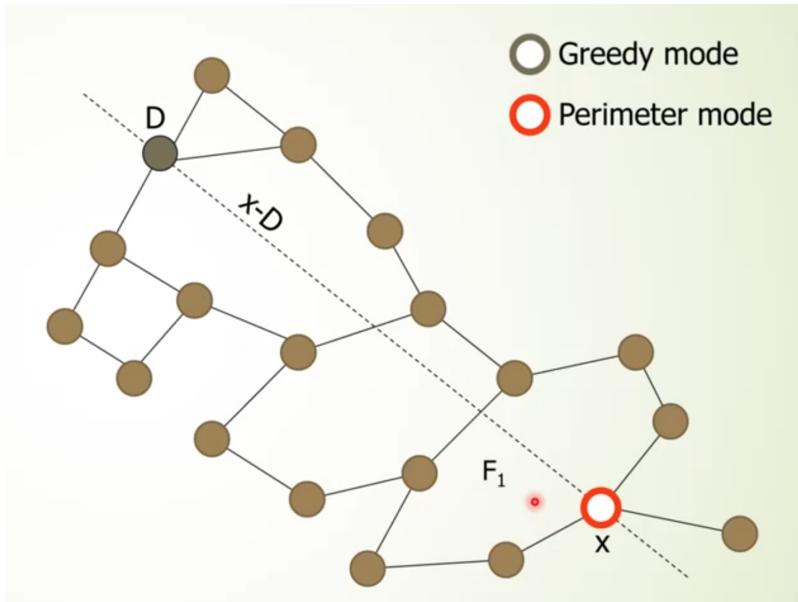
Nel grafo planarizzato (indipendentemente dal tipo di algoritmo usato per la trasformazione) possiamo distinguere tra:

- Interior face: sono le faces che compaiono maggiormente all'interno del grafo. Nell'esempio sotto sono F_1 , F_2 e F_3 . Queste faces sono limitate da nodi e archi del grafo.
- Exterior face: è una sola, è tutta la parte senza confine che sta fuori dal confine esterno del grafo.

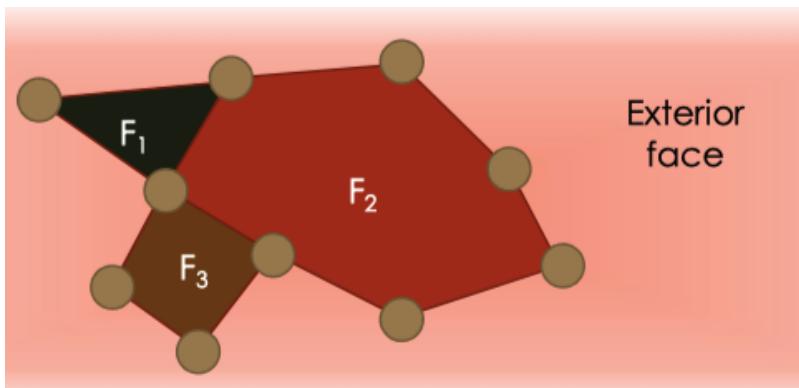


Vediamo come funzionano greedy e perimeter mode in un grafo dove utilizziamo il GPSR.

Abbiamo la fonte che inoltra il pacchetto al nodo X e poi D è la destinazione, il pacchetto. X si trova in una interior face che è X_1 e non ha un greedy forwarding qua e quindi si passa al perimeter mode.



X identifica la linea che la connette alla destinazione, X identifica anche gli altri due nodi (i suoi vicini) che sono sul confine di F_1 e inizia a inoltrare i pacchetti ai suoi vicini (quindi lungo il perimetro di F_1). Assumiamo per il momento che stiamo solamente eseguendo il forwarding con il perimeter mode. Il pacchetto passerà sul confine di F_1 arriva ad un nodo in cui si ferma perché se continuasse il pacchetto arriverebbe nuovamente ad X . In questo caso in particolare vediamo che c'è un punto L_f in cui abbiamo una intersezione tra la linea $X-D$ e il perimetro di F_1 .



Questo è il momento in cui passiamo dall'utilizzo della face F1 all'utilizzo della face F2 e quindi il pacchetto ora viene inviato ai nodi che stanno sul perimetro di F2 fino a che non arriva ad un nodo dove la distanza con D è minore rispetto alla distanza X-D, in questo nodo si passa di nuovo alla greedy mode.

Quindi più formalmente:

- In ognuna delle face GPSR utilizza la right hand rule per raggiungere l'arco che si interseca con la linea X-D.
- Arrivati a questa linea allora GPSR invia il pacchetto alla face successiva che viene attraversata da X-D.
- Ogni volta che il pacchetto entra in una nuova face allora memorizziamo sia il punto in cui c'è stata l'intersezione con la linea X-D sia l'arco che viene utilizzato attualmente.

La regola del trovare l'intersezione e passare poi alla face successiva non viene utilizzata troppo spesso, questo perchè c'è una politica aggressiva per il passaggio dalla perimeter mode alla greedy mode e quindi passiamo alla greedy mode quando il nodo che raggiungiamo è più vicino alla destinazione rispetto ad X.

Quando siamo in perimeter mode l'header del messaggio ha anche necessità di mantenere molte più informazioni che invece non sono necessarie quando siamo in greedy mode.

La destinazione a cui vogliamo spedire il pacchetto ha due possibilità:

- La destinazione è raggiungibile: in questo caso abbiamo che la strada verso la destinazione viene sempre trovata da GPSR.
- Se la destinazione non è raggiungibile invece abbiamo due possibilità: La destinazione è all'interno di una certa face F_i o è nell'exterior face. In entrambi i casi il pacchetto raggiunge la face più vicina alla destinazione e poi inizierà un loop sul perimetro

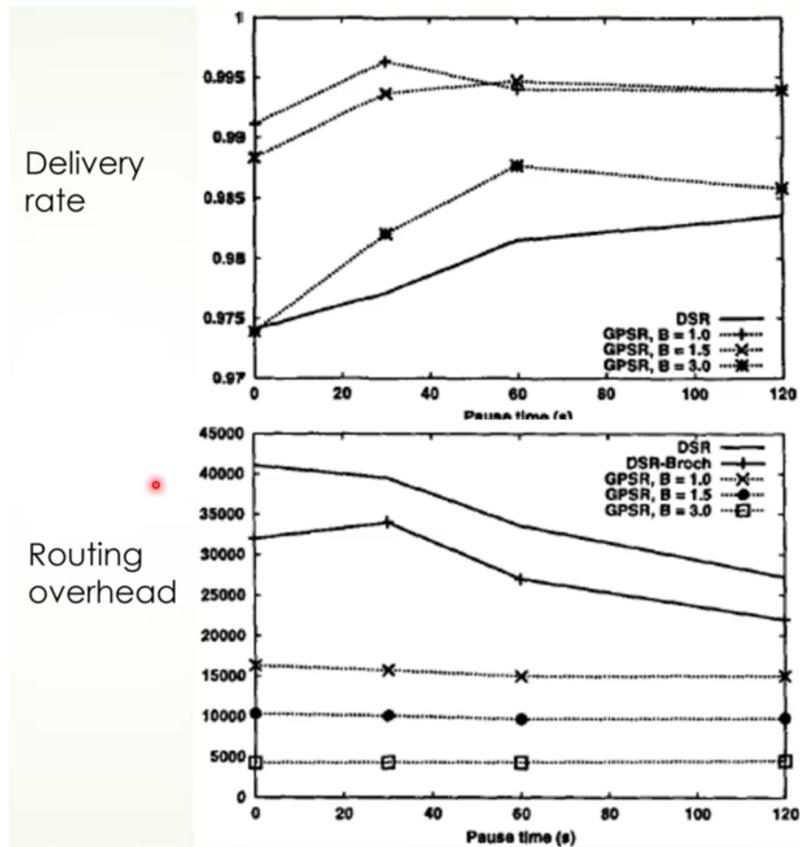
della face. Ad un certo punto il pacchetto passerà nuovamente lungo l'arco che ha già visitato una volta e a questo punto il pacchetto verrà rifiutato.

Abbiamo parlato più volte della planarization, ovvero della trasformazione del grafo che possiamo effettuare, eseguire questa operazione è qualcosa di costoso e che potrebbe diventare eccessivamente costosa quando i cambiamenti all'interno del grafo sono troppi. L'idea è quella di ridurre il più possibile la planarization con un approccio proattivo. Questo vuol dire che ogni volta che abbiamo un cambio all'interno della topologia della rete dovremmo fare una planarization ma la evitiamo perché saranno gli stessi nodi, quando si spostano nella rete a comunicare la nuova posizione e i vicini. Questa informazione viene poi utilizzata per mantenere aggiornata la lista dei vicini e per forzare la planarization solamente quando i cambiamenti sono eccessivi.

Eseguire troppe planarization può anche essere un problema in termini di stabilità.

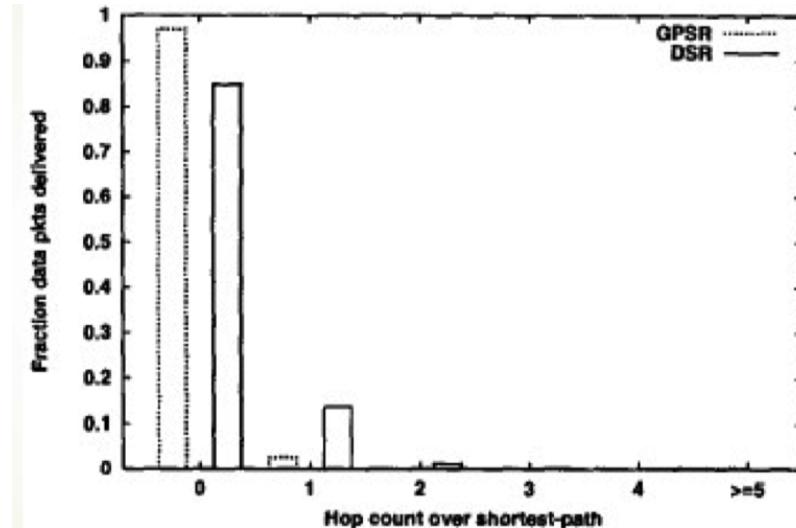
Simulazioni

Sono state fatte delle simulazioni considerando grafi che non sono troppi grandi in modo da fare un confronto con DSR. L'assunzione è che il grafo è mobile, più aumenta il valore sulla X e meno la rete è mobile. GPSR ha un delivery rate che è più o meno sempre migliore rispetto a DSR, anche per la routing overhead abbiamo un valore che è migliore rispetto a DSR.



In questa simulazione però è stato usato un grafo molto denso e questo vuol dire che GPSR non switcha mai alla perimeter mode e quindi questo è molto efficiente.

Un altro confronto che è stato fatto riguarda la differenza di GPSR con DSR in termini di Path lenght:

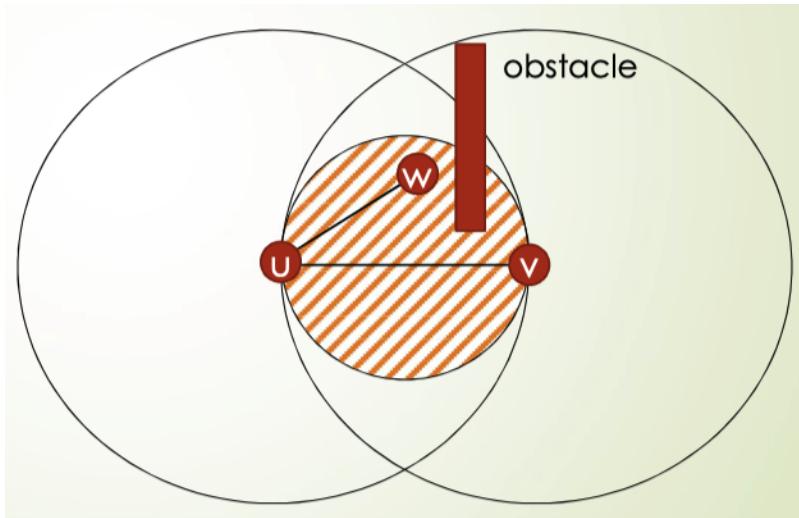


Nel più delle volte GPSR sceglie un path che è il più breve, questo accade meno volte in DSR. Anche qua il fatto è che se il grafo è molto denso usiamo sempre la greedy mode che quindi offre performance migliori.

4.5.2 Limiti e problemi di GPSR

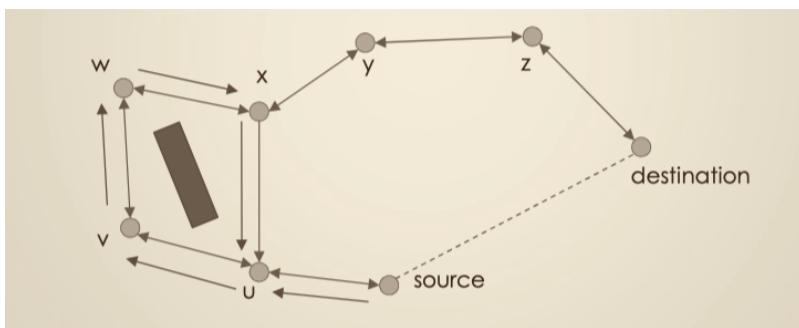
Vediamo ora quali sono i principali limiti di GPSR:

- La planarization può fallire, assumiamo ad esempio che stiamo effettuando qua la planarization:



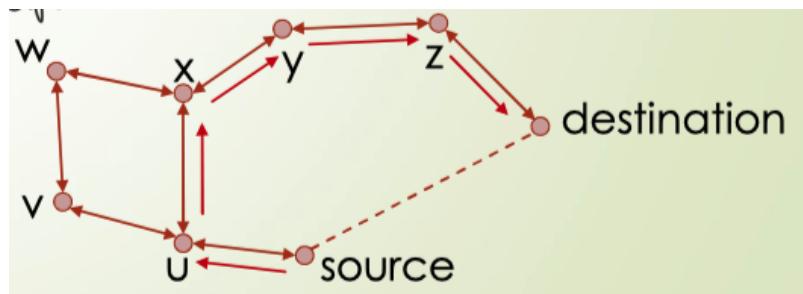
U qua rileva la presenza di un altro nodo, W, quindi non considerà U-V per lo spostamento di informazioni e lascerà questo percorso. V invece, che a sua volta sta implementando la planarization, mantiene quel collegamento perché non può rilevare la presenza di W. Quindi dopo la planarization avremo due collegamenti: U-W e V-U, quello che succede quindi è che la planarization può produrre dei grafi con dei collegamenti che sono unidirezionali.

- In alcuni casi GPSR può fallire e può finire in un loop quando applichiamo l'algoritmo ad alcuni link che non sono bidirezionali. Consideriamo il seguente esempio:

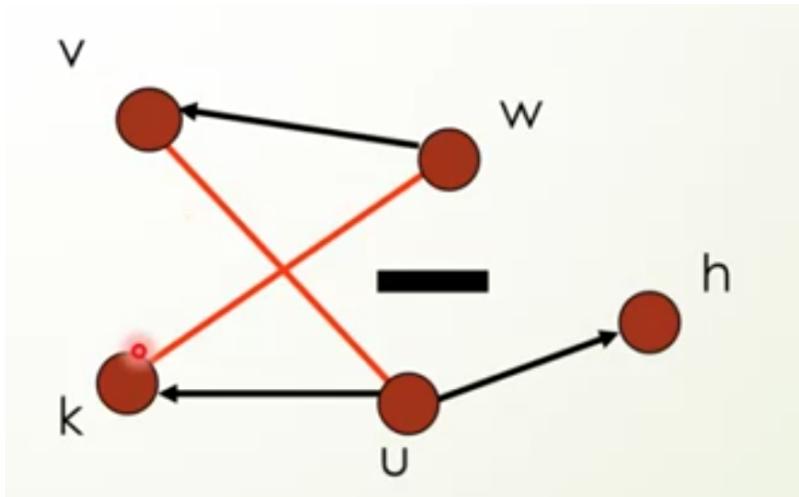


Il problema qua è il collegamento tra U e X perché X non vedrà V nel suo vicinato mentre U vedrà V come vicino e non considererà il collegamento verso X. Quindi il pacchetto parte da Source e seguendo la right hand rule va a U, poi da U non c'è un collegamento verso X perché avevamo cancellato il collegamento durante la planarization. Quindi il pacchetto va da V poi da W poi da X. Una volta arrivato da X il pacchetto viene inviato di nuovo ad U e quindi siamo in un loop.

Una possibile soluzione a questo problema consiste nell'utilizzare la Mutual Witness Rule che è una estensione dell'algoritmo di planarizzazione. Consideriamo il nodo U, V è vicino nella regione, X non è nella regione e dato che non c'è un collegamento da X a V allora viene mantenuto il collegamento tra U e X. In questo modo se utilizziamo lo stesso esempio di prima possiamo usare la right hand rule per mandare i dati verso X.



- La Mutual Witness a sua volta può causare dei problemi, ci sono alcuni cross link che non possono essere rilevati. Consideriamo il seguente esempio:



Il cross link da U-V e W-K non sono rilevabili, dal punto di vista di V abbiamo che W è usato come mutual witness quindi non possiamo rimuovere il collegamento V-U. La stessa cosa vale anche per il nodo K con la sola differenza che qua il nodo U viene usato come mutual witness e quindi non rimuoviamo il nodo K-W. Questo comporta che il grafo che otteniamo non è più planarizzato quindi non possiamo più utilizzare la perimeter mode. Questo non è molto facile che succeda ma comunque può accadere e non è garantito che il protocollo funzioni correttamente.

CLDP

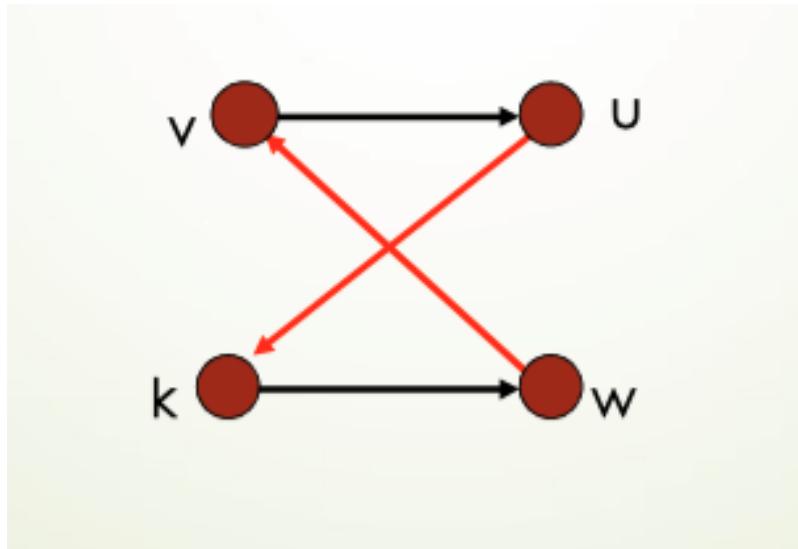
Una possibile alternativa alla planarization è la CLDP ovvero la Cross Link detection Protocol. Questo algoritmo funziona sul grafo completo e non applica la planarization prima di lavorare sul grafo. Ognuno dei nodi presenti all'interno del grafo invia un probe verso ognuno dei suoi vicini. Ogni volta che il probe arriva in un nodo viene aggiunto quel nodo all'interno del probe e poi viene infiltrato. Quando un nodo riceve poi il probe deve anche controllare se ho già ricevuto quello stesso probe e in tal caso per fare in modo di eliminare eventuali crossing link. Il

nodo può fare questo controllo se controlla la lista dei nodi che è scritta all'interno del probe. Una volta che abbiamo trovato un crossing link possiamo eliminarlo e poi consideriamo il successivo.

La rimozione di un collegamento può richiedere una comunicazione aggiuntiva tra i vari nodi e quindi questo può essere un problema che può causare overhead. Per diminuire l'overhead causato da CLDP abbiamo alcune regole semplici che possono essere utilizzate:

- Siamo in un nodo W e abbiamo in uscita il collegamento L che si interseca con il collegamento L'.
- Se non possiamo rimuovere nessuno dei due collegamenti allora dobbiamo mantenerli entrambi.
- Se possiamo rimuoverli entrambi scegliamo di rimuovere L perché richiede meno comunicazione.
- Se possiamo rimuovere solamente uno dei due allora lo rimuoviamo ricordando comunque che rimuovere L' richiederà una comunicazione maggiore.

Prendiamo questo esempio:

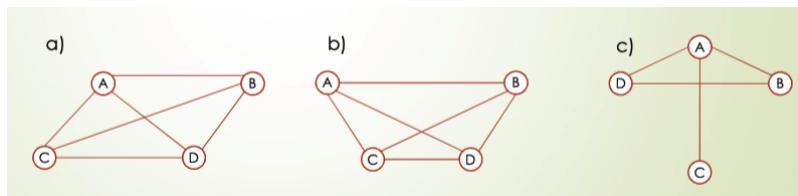


Qua, se siamo in w, possiamo rimuovere il collegamento tra w e v perché comporta una comunicazione minore rispetto a quella che avremmo per rimuovere il collegamento u-k.

Quando utilizziamo il CLDP dobbiamo comunque ricordarci che possiamo rimuovere solamente un collegamento alla volta perché questo causa cambiamenti all'interno della topologia della rete inoltre se rimuovessimo più di un collegamento alla volta potremmo avere una disconnessione nella rete.

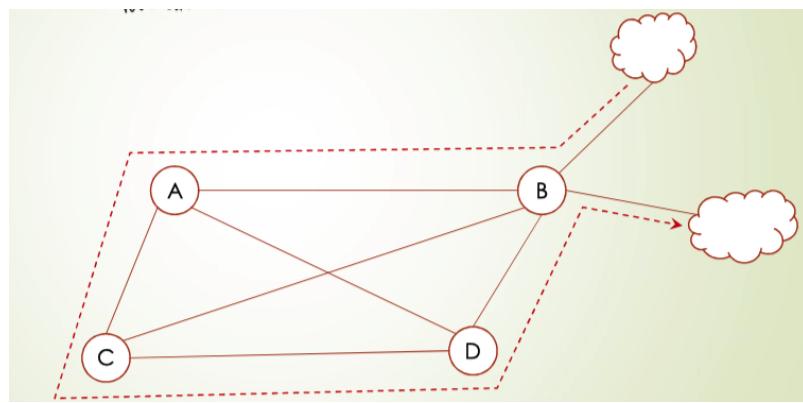
4.5.3 Configurazioni che possono causare intersezioni

Vogliamo fare ora un'analisi più approfondita delle configurazioni che possono causare intersezioni, possiamo avere una delle seguenti configurazioni:



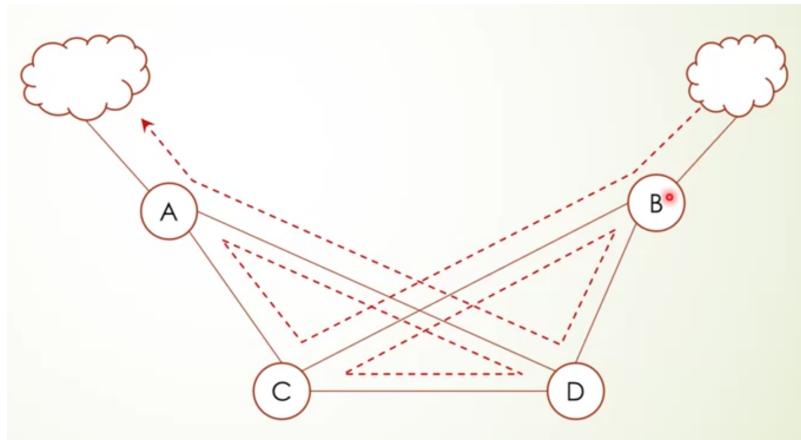
Non tutte queste configurazioni sono problematiche, quelle che vediamo qua sono delle macro strutture che possono essere formate da vari collegamenti (ad esempio tra A e B potrebbero esserci vari altri nodi).

Consideriamo la prima struttura:



Assumiamo che la fonte dei dati sia nella nuvola in alto a destra, in questo caso il pacchetto farà tutto il giro e arriverà alla destinazione finale. Quindi saremo capaci di far arrivare il pacchetto alla destinazione in ogni caso.

Anche con la struttura della prossima figura non abbiamo grossi problemi, siamo più o meno nella precedente situazione. In questo caso potremmo avere il pacchetto che seguendo la right hand rule viaggia un po' attraverso la rete e alla fine raggiunge la destinazione.



Solamente l'ultima configurazione della figura è un problema, questa è la configurazione che avevamo già visto in occasione dello studio della perimeter mode della GPSR. Questa configurazione ad ombrello è l'unica realmente problematica perché si vanno a creare dei loop che non permettono al pacchetto di arrivare a destinazione. Quello che vorremmo utilizzare è trovare le configurazioni ad ombrello all'interno del grafo per poterle eliminare ed ottenere quindi un grafo senza cicli che non comporta problemi. Sfortunatamente non è possibile avere un algoritmo localizzato che funziona all'interno del grafo e che ci permette di eliminare queste strutture ad ombrello. Sarebbe necessaria una conoscenza globale del grafo.

4.5.4 Considerazioni su GPSR

- Se usiamo semplicemente GPSR non abbiamo la sicurezza della consegna dei pacchetti.
- Se lo usiamo con CLDP il protocollo diventa molto complesso. In teoria funziona ma in pratica non troppo.

Ora come ora non abbiamo alcun protocollo di planarization che possa permettere a GPSR di funzionare correttamente.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

4.5.5 Altri protocolli Geo-Routing

Ci sono vari protocolli che sono alternativi a GPSR. Possiamo distinguere tra due tipi principali:

- Un primo gruppo è quello che utilizza la linea X-D come referenza per il protocollo di routing, tra questi ci sono GPSR, Greedy Face Greedy e Compass Routing.
- Altri invece fanno ripartire il perimeter mode ogni volta che ci muoviamo da una face all'altra. Questi sono il Greedy Other Adaptive Face Routing e il Greedy Path Vector Face Routing.

Indipendentemente dal protocollo geografico abbiamo sempre la necessità di avere un protocollo che esegua la planarization. Nonostante la planarization non è neanche sicuro che tutti questi protocolli funzionano correttamente con qualsiasi planar graph.

Rimane anche un altro problema aperto ovvero trovare un protocollo di routing che garantisca la consegna dei pacchetti conoscendo la geografia della rete. C'è anche un altro problema perchè avevamo detto che la fonte del pacchetto doveva conoscere la destinazione, questo è un altro problema che va considerato perchè non è sempre vero che il mittente conosce già il destinatario del pacchetto (In una wireless network la comunicazione è data centric e quindi è completamente differente avere la posizione di un certo nodo nella rete).

4.6 Data-Centric Storage and Geographic Hash Tables

Possiamo risolvere i problemi che abbiamo elencato fino ad ora andando ad adottare una nuova organizzazione della rete che possa rendere più efficiente lo storage e il retrieval dei dati.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

4.6.1 Storage vs Transmission

La prima cosa che dobbiamo analizzare è il tradeoff tra il costo per lo storage e il costo per la trasmissione. L'utente invia istruzioni ai nodi utilizzando il flooding o altri metodi.

Per prima cosa dobbiamo fare delle assunzioni:

- Il costo è trascurabile se il task rimane attivo per lungo tempo.
- Noi all'inizio non conosciamo in quali nodi verranno prodotte informazioni rilevanti.
- Possiamo fare una stima in termini di costi asintotici in termini di numeri di nodi sia per il flooding sia per la comunicazione unicast:
 - Se abbiamo n nodi nella rete allora abbiamo un costo $O(n)$ per il flooding e $O(\sqrt{n})$ per la comunicazione unicast perché dobbiamo attraversare la rete da una parte all'altra.

Ogni nodo poi può lavorare in modi differenti quando riceve il nuovo task:

- Quando il nodo riceve i dati può inviare tutto al sink che poi fornisce il supporto allo storage. In questo caso paghiamo un costo in maniera proattiva per implementare un singolo storage. Per ogni singolo evento abbiamo un costo di $O(\sqrt{n})$.
- Ogni nodo può memorizzare al suo interno i dati. In questo caso tutti gli eventi sono memorizzati nel sensore, non paghiamo niente per la comunicazione ma paghiamo per il recupero dei dati dalla memoria. Con questa strategia noi non sappiamo dove si trovano i dati quindi quando abbiamo bisogno di dati dobbiamo mandare una query in flooding, il costo per ogni query è $O(n)$. Oltre a questo per ognuna delle risposte che viene inviata abbiamo un costo di $O(\sqrt{n})$.

- Possiamo adottare una strategia data-centrica. Per memorizzare ogni singolo evento abbiamo un costo di $O(\sqrt{n})$, le query sono dirette solamente verso quel nodo che memorizza uno specifico evento basandosi sul nome di quel nodo. Il costo qua è $O(\sqrt{n})$ per la query e $O(\sqrt{n})$ per la risposta.

Per il calcolo dei costi possiamo considerare due metriche differenti:

- Total Usage: il numero totale di pacchetti della Wireless Sensor Network.
- Hotspot Usage: Il numero massimo di pacchetti inviati o ricevuti all'interno di una Wireless Sensor Network.

E possiamo anche considerare vari parametri:

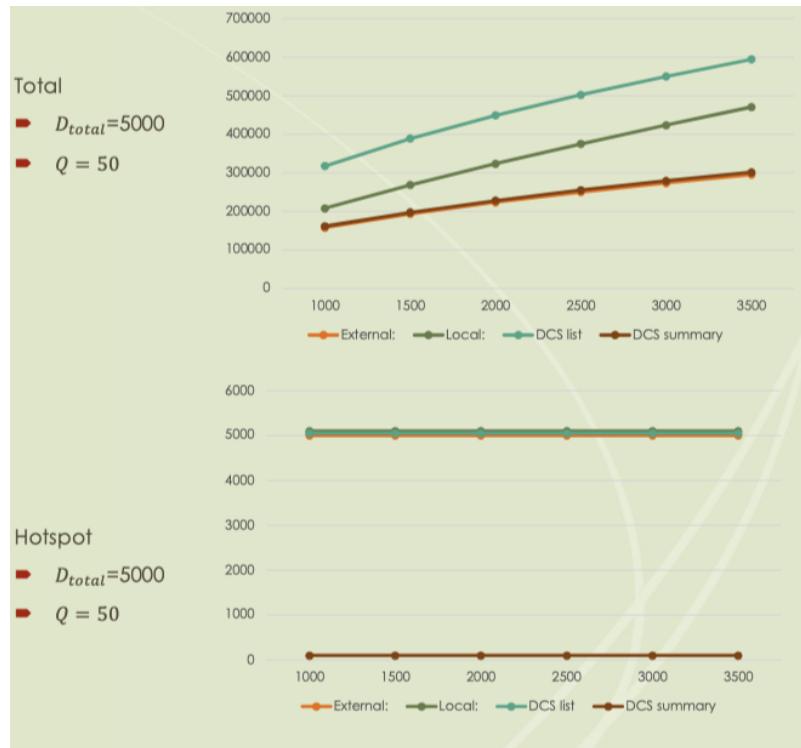
- Abbiamo n nodi nella rete
- Abbiamo T tipi differenti di eventi, ognuno con il suo nome (meta-data), un nodo si occuperà di memorizzare questi eventi.
- Abbiamo il numero totale di eventi che sono stati rilevati D_{total} .
- Abbiamo il numero di risposte per la singola query D_q .
- Abbiamo il numero di query Q che sono state generate dal sink.

Ora considerando i parametri che abbiamo elencato e considerando anche le due possibili metriche possiamo eseguire una valutazione complessiva dei costi dei vari sistemi di "gestione" dei task dei vari nodi:

- Se utilizziamo l'external storage abbiamo:
 - Total: $D_{total} * \sqrt{n}$ perchè abbiamo il numero di eventi che rileviamo e poi per ognuno dobbiamo inviare al sink i dati.

- Hotspot: D_{total} qua il costo sarà questo perchè il sink riceverà solamente D_{total} pacchetti.
- Local Storage:
 - Total: $Q * n + Q * D_q * \sqrt{n}$
 - Hotspot: $Q + Q * D_q$
- Data Centric Storage: in questo caso abbiamo due possibilità. Una possibilità è che ogni dato prodotto deve essere memorizzato in uno specifico sensore, avremo un costo per la memorizzazione che è $D_{total} * \sqrt{n}$.
 - Total: c'è una prima versione in cui mandiamo indietro come risposta tutti gli eventi che matchano una query, in questa versione però mandiamo tutti gli eventi separatamente. Il costo è $D_{total} * \sqrt{n} + Q * \sqrt{n} + Q * D_q * \sqrt{n}$
 - Summary: restituiamo un solo pacchetto con tutti gli eventi. Qua il costo diventa: $D_{total} * \sqrt{n} + Q * \sqrt{n} + Q * \sqrt{n}$
 - Hotspot: nel caso della lista il costo è $Q + Q * D_q$. Qua generiamo una query e riceviamo $Q * D_q$ risposte.
 - Hotspot: Nel caso del summary è $2Q$

La differenza tra i due metodi, Hostpot e Total la possiamo vedere in questo grafico in cui confrontiamo le varie prestazioni:



4.6.2 Data Centric Storage e GHT

La prima proposta per l'organizzazione le reti come uno storage data centric ce l'abbiamo avuta nel 2003. L'obiettivo principale è quello di avere una rete che funziona anche quando il sink non è disponibile o non è connesso. Questo Data Central Storage si basa su protocolli di routing geografico come potrebbe essere ad esempio GPSR. Il DCS prende anche alcune idee dalle geographic hash table, i nodi della rete infatti dovrebbero essere coscienti della loro posizione e dovrebbero anche conoscere il confine della rete.

Nel Data Centric Storage gli eventi sono chiamati con un certo nome (una chiave) e nella rete memorizziamo i dati proprio utilizzando queste chiavi. Grazie alla presenza di questa chiave le query le possiamo direzionare verso un certo nodo che memorizza gli eventi relativi a quella

chiave. Il DCS supporta due operazioni:

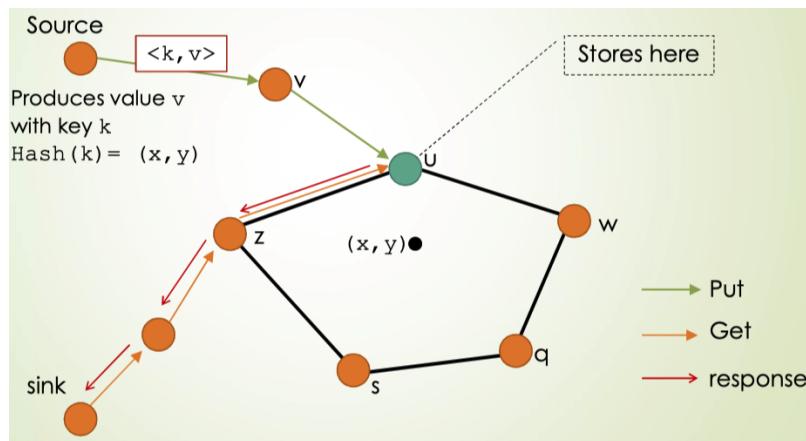
- Put(k, v) memorizza i dati utilizzando la chiave k . Questa operazione viene svolta da un sensore. L'operazione Put(k, v) funziona nel modo seguente:
 - Per prima cosa viene calcolato l'hash di k ovvero $(x, y) = \text{hash}(k)$.
 - La coppia k, v viene inviata al punto (x, y) usando GPSR.
 - Si considera il nodo più vicino al punto (x, y) , la coppia k, v è memorizzata in questo nodo.
- Get(k) è l'operazione della query, recuperiamo i dati associati con la chiave K . Questa operazione viene effettuata dal sink. Anche qua dobbiamo per prima cosa calcolare l'hash di k . Si usa la stessa funzione che era stata usata durante la Put e poi si utilizza GPSR per inviare una richiesta al punto (x, y) .

Noi vorremmo che il DCS avesse alcune proprietà:

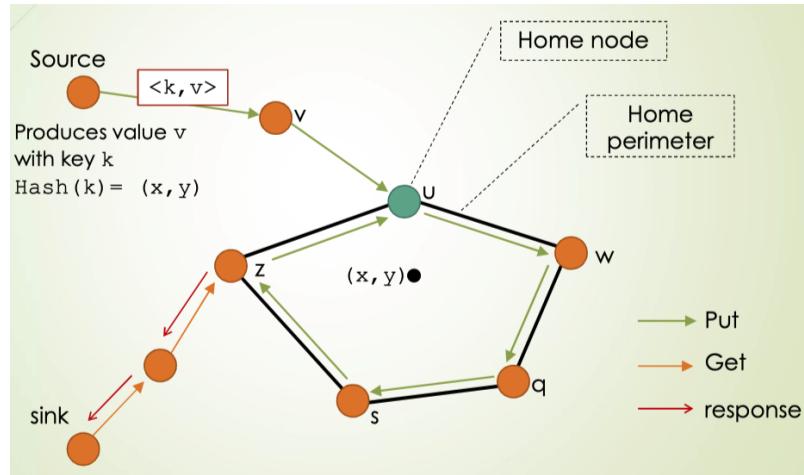
- Il DCS deve essere tollerante ai fallimenti dei nodi.
- Quando cambia la topologia della rete a causa di un fallimento di un nodo DCS non ne risente troppo.
- È un sistema scalabile.
- È un sistema efficiente dal punto di vista energetico.
- Vorremmo un sistema persistente, nel senso che quando memorizzo una coppia (k, v) vorremmo che questa rimanga disponibile indipendentemente dai fallimenti dei nodi che possiamo avere nella rete.

- Vorremmo avere la consistenza. Se generiamo una query per la chiave K vorremmo poter raggiungere il nodo in cui tutte le coppie associate a quella chiave K sono memorizzate. Quindi vorremmo ottenere tutte le coppie (k, v) .
- Anche se ci stanno tanti valori memorizzati non deve essere sovraccaricato solamente un nodo.

Un esempio di funzionamento del DCS:

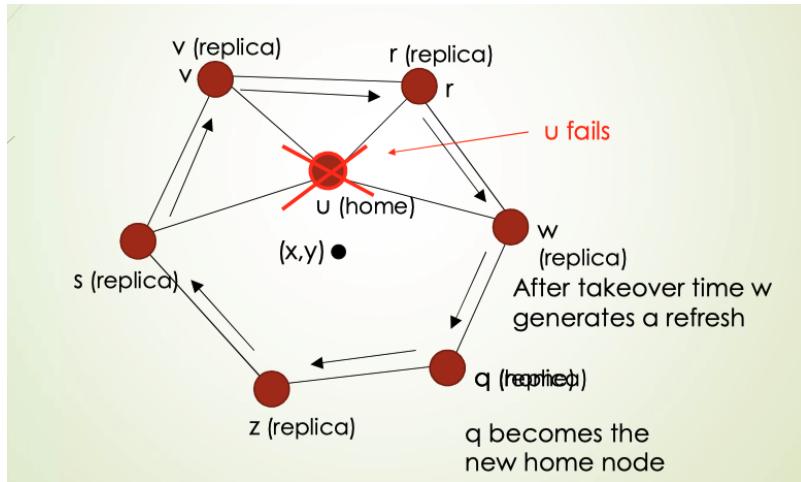


Lo schema di DCS funziona bene solamente se la rete non fallisce, possiamo utilizzare un protocollo per il refresh del perimetro in modo da garantire la persistenza dei dati all'interno della rete. Una soluzione potrebbe essere ad esempio memorizzare i dati in più di un nodo contemporaneamente (i nodi devono essere sul perimetro). In questo modo se un nodo cade e questo sarebbe quello che avrebbe dovuto mantenere i dati, ne prendiamo un altro vicino al punto x, y e riusciamo comunque ad arrivare al dato che stavamo cercando. Il perimetro attorno a cui si trova il punto x, y viene trovato da GPSR. Un esempio di come funziona questo metodo con il perimeter mode:



A differenza dell'esempio precedente qua abbiamo una replica dei dati, se non utilizzassimo questo perimeter mode andremmo a perdere dei dati nel caso di fallimento di un nodo e quindi potremmo perdere consistenza e persistenza dei dati. Il protocollo per il controllo del perimetro funziona in questo modo: Il nodo in cui è stata memorizzata la chiave k genera periodicamente dei pacchetti di refresh, ognuno di questi pacchetti viene mandato alla coordinata $x, y = \text{hash}(k)$, questo è importante perché il nodo originale in cui avevamo memorizzati il dato potrebbe essersi spostato. Allora quello che si fa è girare sul perimetro attorno a x, y e si fa in modo che i vari nodi abbiano la coppia che stiamo prendendo in considerazione.

Il funzionamento di questo protocollo di Refresh:



Se il refresh packet arriva ad un nodo che è più vicino a $x, y = \text{hash}(k)$ rispetto a quanto non lo fosse il nodo leader originale allora questo nodo genera un nuovo refresh packet e quando lo riceve nuovamente diventa il nuovo leader. Quando diventa nuovo leader imposta anche un timer per fare in modo che dopo un tot di tempo si vada di nuovo ad eseguire il refresh.

Ognuna delle coppie che memorizziamo all'interno del nostro grafo è associata ad un timer che ne indica la "morte", infatti queste coppie di dati non verranno memorizzate per sempre.

Cosa succederebbe se tutti i nodi andassero a produrre la stessa chiave per una certa coppia di dati? Avremmo i sensori sul perimetro attorno al punto x, y che sarebbero sovraccaricati. La soluzione a questo problema è la Structured Replication (SR) che ha l'obiettivo di distribuire il carico di dati che devono essere memorizzati. Il protocollo SR (Structured replication) funziona nel modo seguente:

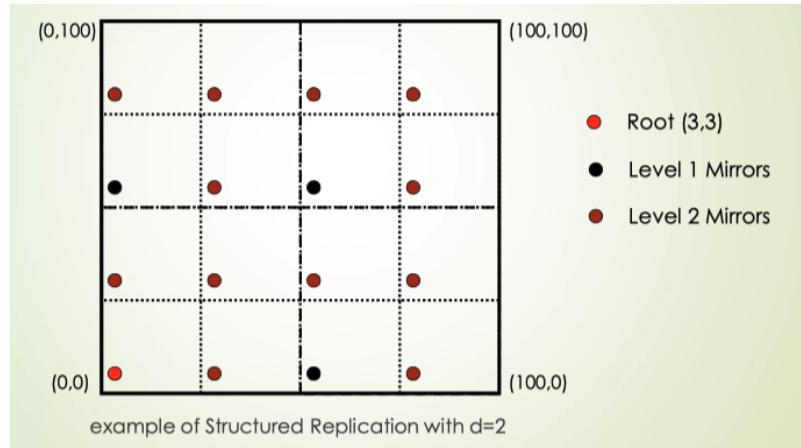
- Viene utilizzata una gerarchia per gestire i dati. A capo di questa gerarchia abbiamo un nodo root che è quello più vicino al punto $x, y = \text{hash}(k)$.
- Per ogni chiave che viene calcolata associato un root e $4d - 1$ nodi

mirror, dove la d indica la profondità di questa gerarchia.

- La coppia di dati k, v viene memorizzata nel nodo mirror più vicino a $x, y = \text{hash}(k)$.
- Recuperare un valore implica la necessità di eseguire una query al nodo root e possibilmente anche ai nodi mirror. Prima la query viene rivolta al nodo root e poi questo nodo la invia al nodo mirror interessato (può accadere che questa query poi la devo anche inviare agli altri nodi mirror). In generale il recupero di una coppia di dati k, v sarà abbastanza costoso.

Vediamo il seguente esempio di Structured Replication, il problema che vogliamo risolvere è quello del load balance. Assumiamo di avere la rete rappresentata in questa griglia, quello che possiamo fare è suddividere la griglia in vari quadranti:

- Inizialmente abbiamo un quadrante unico e il punto arancione è il nodo root di tutti gli altri.
- Poi facciamo una prima divisione e oltre al nodo arancione abbiamo i nodi neri che sono i nodi mirror di primo livello presenti all'interno delle altre sottosezioni.
- Poi possiamo fare una ulteriore suddivisione e otteniamo un'ultima suddivisione in quadrati, in questo modo per ogni sezione abbiamo un nodo mirror di livello 2.



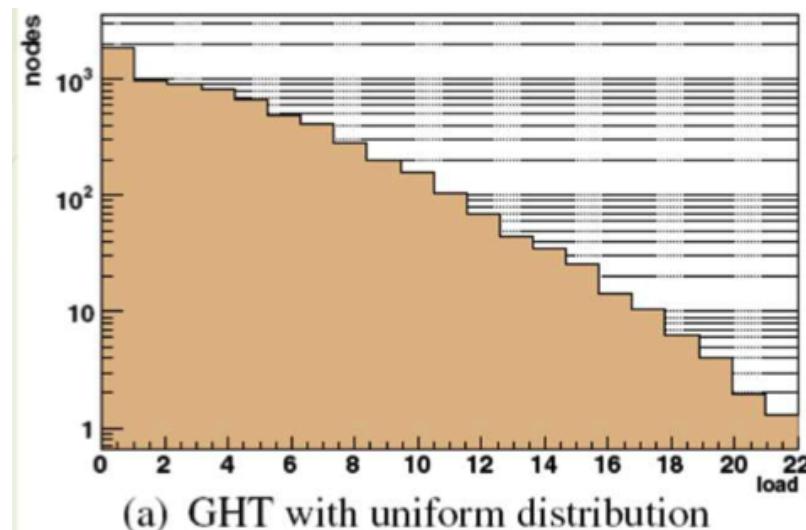
Questa divisione in quadranti la possiamo fare ricorsivamente, in questo modo il carico di dati che devono essere memorizzati verrà suddiviso tra tutti i quadranti. Il sink quando invia la query non conosce questa suddivisione in quadranti e quando il messaggio inviato dal sink arriva al root, questo verrà poi inoltrato al mirror di livello 1 che a sua volta può nuovamente inoltrare il messaggio.

4.6.3 Possibili problemi con DCS e GHT

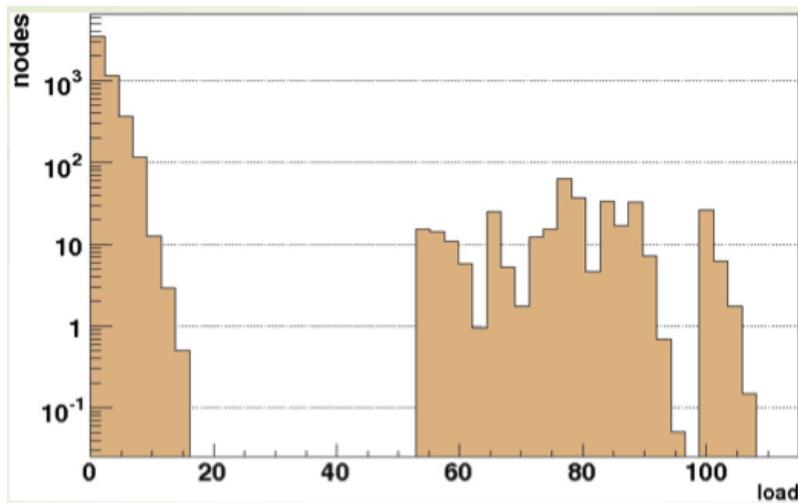
Ci sono comunque alcuni problemi che possono insorgere con l'utilizzo di DCS e di GHT:

- Nonostante l'utilizzo di una replicazione dei dati, non c'è possibilità di controllare il grado di replicazione dei dati presenti all'interno della rete. Non conosciamo la dimensione del perimetro a priori e quindi non sappiamo neanche in quanti sensori andremo a memorizzare i dati. Questo potrebbe indurre una instabilità nel funzionamento del protocollo. Potremmo avere dei problemi ad esempio se il calcolo dell'hash di una certa chiave K dovesse restituire un punto che si trova fuori dai confini della rete.

- Un altro problema ce l'abbiamo con la quantità di nodi che memorizzano qualcosa e quelli che invece non memorizzano niente. Il seguente grafico infatti mostra quanti nodi memorizzano dati e quanti no, abbiamo tantissimi nodi che non stanno memorizzando niente mentre abbiamo pochi nodi che memorizzano tantissime informazioni.



Avendo una distribuzione gaussiana dei sensori, per il carico medio nei vari sensori abbiamo una situazione molto simile:



Sono stati proposti anche altri problemi che possono permettere di risolvere i problemi che abbiamo indicato in precedenza. Potremmo considerare GPS e GPSR con un modo differente di organizzare la rete rispetto a Direct Diffusion. I nodi sono autonomi, il problema è "Come vengono gestite le coordinate fisiche?".

4.7 Physical and Virtual Coordinates

I protocolli tradizionali di routing per le reti ad hoc non sono molto pratici perchè abbiamo routing table molto grandi e perchè la dimensione dell'header dei pacchetti potrebbe essere troppo grande. La soluzione da adottare potrebbe essere il geographic routing che dipende dalle coordinate. Le coordinate sono obbligatorie per supportare il routing geografico, per garantire una relazione tra posizione e dati raccolti e per implementare Data centric storage. Questi sensori dovrebbero quindi essere equipaggiati con un modulo gps che comporterebbe quindi dei costi aggiuntivi per il sensore. Avere un modulo GPS non è sempre fattibile e non è neanche sempre fattibile avere la possibilità di ottenere dati riguardanti la localizzazione, basta pensare ad un ambiente chiuso

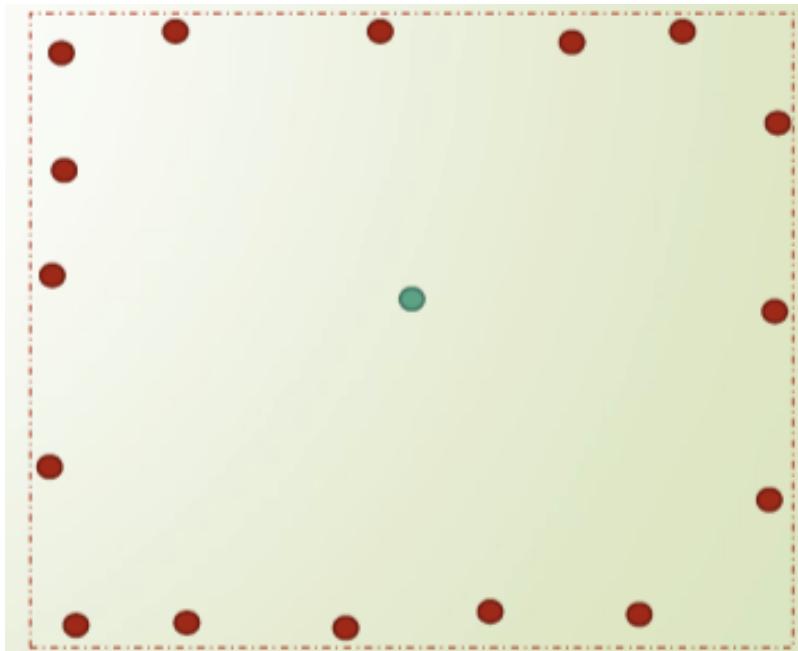
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

o anche a posti isolati. Per risolvere questo problema abbiamo due possibilità:

- Gli anchor node conoscono la loro posizione, gli altri possono calcolarla con alcuni metodi.
- Virtual Coordinates: non sono legate alle coordinate fisiche ma in alcuni casi potrebbero anche funzionare meglio. Quello che succede è che si viene creare un sistema che permette di inviare pacchetti in ogni condizione.

Come possiamo fare ad approssimare il calcolo delle coordinate? Facciamo per prima cosa delle assunzioni, diciamo che i nodi che stanno sul confine della rete sono conosciuti e loro conoscono la loro posizione. I nodi interni invece non conoscono la loro posizione. Supponiamo quindi di avere, un nodo interno i con coordinate x_i, y_i e con un set di vicini N_i . Ora vediamo il funzionamento dell'algoritmo per l'approssimazione:

- Partiamo con una situazione che è la seguente e poi vogliamo eseguire delle iterazioni in cui i nodi interni (che ora sono tutti sovrapposti) possono aggiornare la loro posizione.



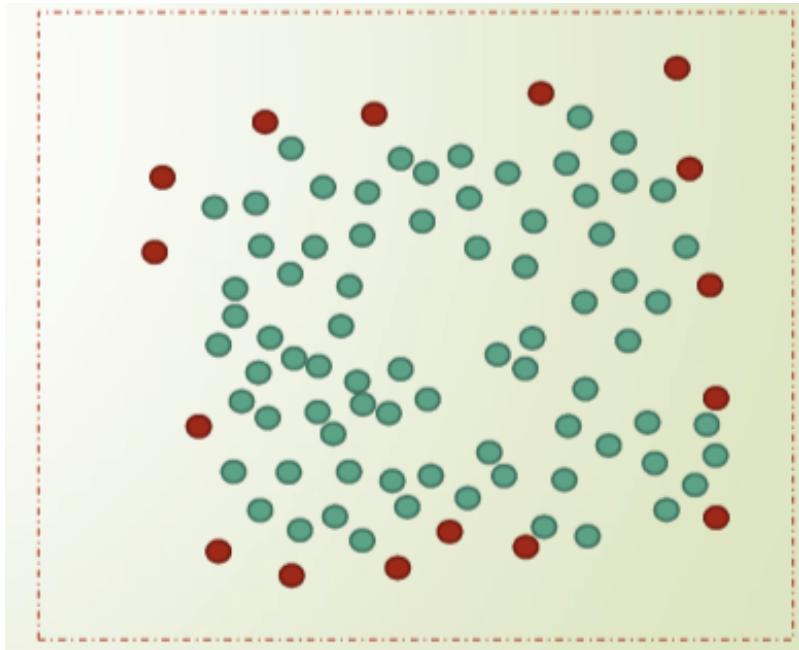
- Di iterazione in iterazione i nodi interni alla rete approssimano la loro coordinata virtuale. L'iterazione viene eseguita per d volte e ogni volta il centro di questo gruppo di nodi si riduce perché i nodi si diffondono sempre di più.

Utilizzando questo sistema di coordinate virtuali possiamo eseguire delle simulazioni per capire il miglior valore possibile per la d (numero di iterazioni). Quello che otteniamo con questa simulazione è che il routing rate che abbiamo con il physical mode è addirittura migliore rispetto al routing rate che abbiamo quando utilizziamo le coordinate fisiche. Da notare però che in questa simulazione abbiamo utilizzato un approccio greedy per il routing.

Fino ad ora abbiamo assunto che noi conosciamo le coordinate dei punti sul perimetro, se però non le conosciamo non possiamo utilizzare il protocollo che abbiamo visto fino ad ora.

Se però conosciamo quali sono i nodi che sono sul bordo della rete possiamo risolvere un problema di ottimizzazione per cercare di trovare

le coordinate virtuali dei vari nodi della rete. In questo caso succede che ogni nodo sul perimetro invia in broadcast un messaggio di Hello a tutta la rete. In ognuno di questi messaggi viene scritto il numero di nodi che sono stati visitati. Una volta che il messaggio si è propagato anche gli altri nodi che stanno sul perimetro riceveranno quel messaggio di Hello e in questo modo i nodi del perimetro saranno in grado di capire la distanza con gli altri nodi del perimetro. La distanza tra i nodi del perimetro viene memorizzata all'interno del perimeter vector. Ora che conosciamo la distanza tra i nodi del perimetro, ogni nodo del perimetro deve svolgere un problema di ottimizzazione in cui va a minimizzare la distanza tra se stesso e gli altri nodi della rete. Si esegue con questo processo una triangolazione per calcolare le coordinate virtuali dei nodi del perimetro, una volta ottenute queste coordinate possiamo tornare al caso precedente in cui conoscevamo la localizzazione dei vari nodi.



Anche in questo caso e anche dopo aver fatto l'ottimizzazione riusciamo comunque ad avere delle performances decenti. Quindi in questo

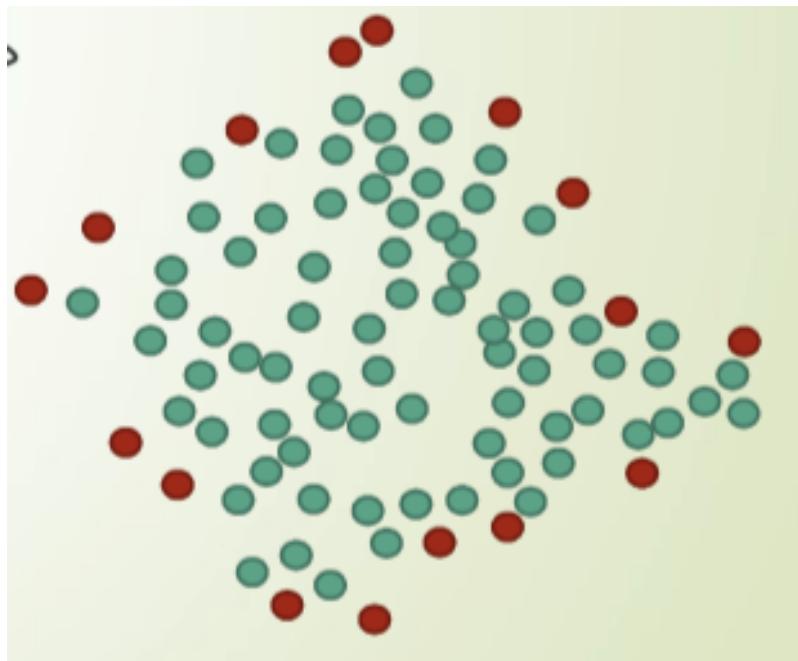
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

caso noi conoscevamo solamente l'elenco dei nodi sul bordo ma non la loro posizione, cosa possiamo fare se sappiamo neanche quali sono i nodi sul bordo?

Quello che possiamo fare è aggiungere uno step all'algoritmo per fare in modo che si possano identificare i nodi sul bordo. L'algoritmo funziona in questo modo:

- Scegliamo due nodi bootstrap random e gli facciamo inviare il messaggio di Hello. In questo modo i due nodi inviano per primo i messaggi ai vicini.
- Ogni volta che un nodo riceve il messaggio di Hello, riesce a capire la distanza del nodo dal nodo bootstrap.
- I nodi che hanno la massima hop distance vengono classificati come boundary nodes.

Si tratta di una euristica che ci permette quindi di trovare i nodi del perimetro, più o meno funziona ma non è perfetta, in ogni caso riusciamo ad ottenere un risultato decente ed eventuali errori non sono drammatici. Una volta che abbiamo ottenuto la lista dei nodi del perimetro e la loro posizione virtuale possiamo eseguire di nuovo l'algoritmo precedente.



4.7.1 Pro e Contro

- Pro: il protocollo resiste alla perdita di messaggi, le informazioni che riguardano i nodi presenti sul perimetro sono tante (forse anche troppe).
- Pro: possiamo sostituire GPS con un algoritmo distribuito.
- Contro: non garantisce comunque la consegna dei pacchetti.
- Contro: il protocollo comunque è costoso.
- Contro: dobbiamo risolvere un problema di ottimizzazione.

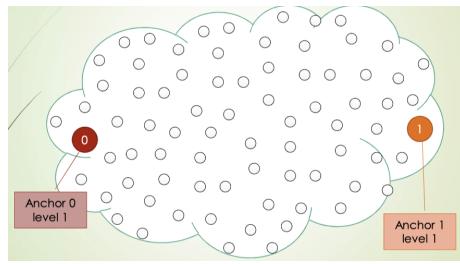
4.7.2 Full Virtual Coordinates

Un’alternativa a quello che abbiamo visto fino ad ora è l’utilizzo di un protocollo che possa assegnare delle coordinate virtuali complete senza

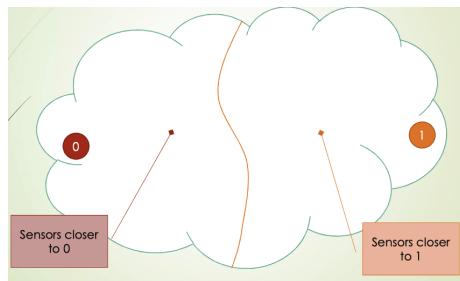
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

avere la necessità di fare approssimazioni. Il protocollo è basato sulla bipartizione ricorsiva del grafo e garantisce anche la consegna dei pacchetti.

Vediamo come funziona il protocollo.

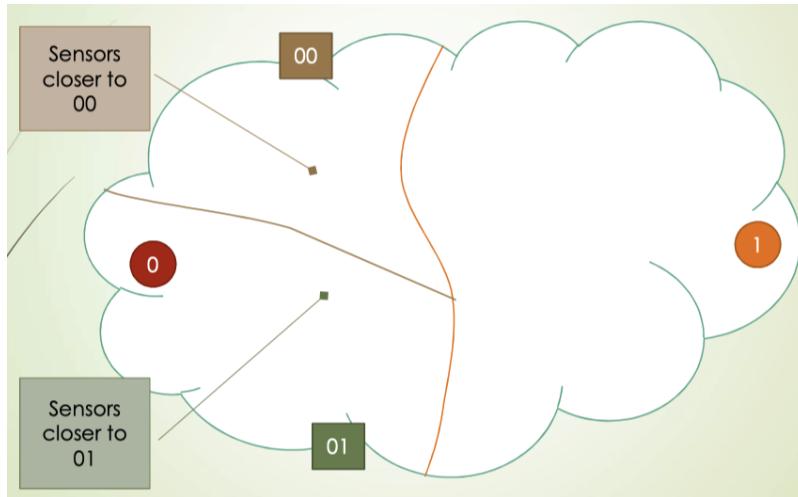


Abbiamo due anchor, l'anchor 0 a livello 1 e l'anchor 1 a livello 1. Se riuscissimo ad identificare i nodi che sono più vicini a 0 e i nodi che sono più vicini a 1 riusciremmo a suddividere in due parti il grafo. Possiamo dividere i vari nodi in due sottografi basandoci sulla distanza dei nodi da 0 e da 1.



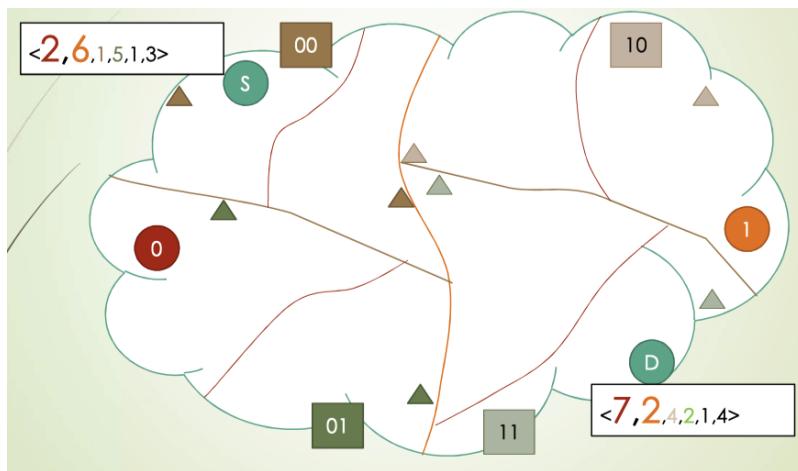
Per calcolare la distanza e misurarla in hop dal nodo 0 e dal nodo 1 possiamo inviare un Hello message in broadcast e creare quindi un sistema di coordinate. Questa prima divisione non è sufficiente, possiamo infatti eseguire altre ripartizioni in modo ricorsivo.

Quindi prendiamo ciascuno dei due sottografi che abbiamo creato e per ogni sottografo andiamo a scegliere due nodi anchor. Una volta scelti i nodi anchor dobbiamo poi fare la stessa cosa fatta in precedenza e possiamo dividere anche quella parte di grafo in due parti.



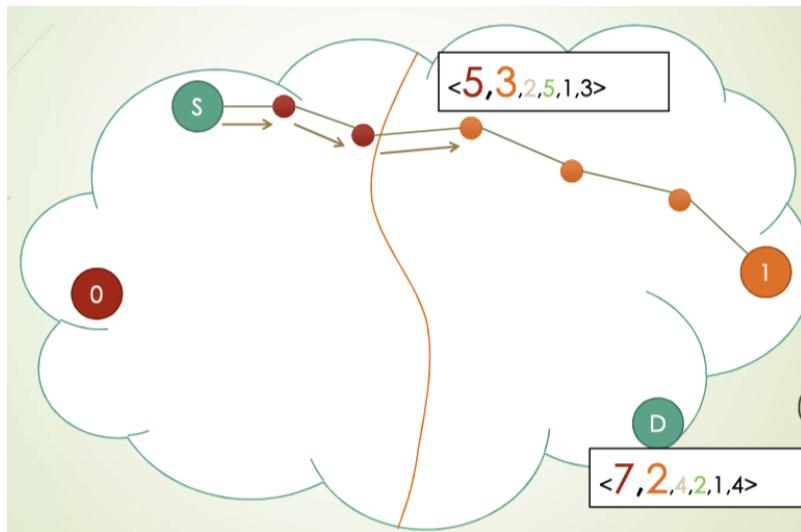
I nodi anchor che sceglieremo per questa seconda bipartizione sono nodi di secondo livello. In generale possiamo andare avanti ricorsivamente fino a creare h livelli di nodi.

Ora anche il protocollo di routing si semplifica, ad esempio, abbiamo il nodo Source e il nodo Destinazione. Sappiamo che dobbiamo raggiungere la zona 11 per consegnare il pacchetto.



Il protocollo di routing parte inviando il pacchetto al livello 1, poi considera le coordinate del livello 2, dato che siamo nella parte sbagliata

del livello 2 dobbiamo spostarci.



Andiamo avanti ricorsivamente fino a quando il pacchetto non arriva alla destinazione.

Con questo metodo garantiamo la consegna dei pacchetti, potremmo perdere pacchetti solamente a causa di collisioni o di buffer overflow, questa soluzione non necessita neanche di avere delle reti troppo dense.

4.8 Conclusione

Molte delle soluzioni viste fino ad ora sono praticamente solo teoriche e funzionano bene solamente se le configurazioni delle reti fossero statiche. Con la mobilità e i possibili fallimenti, problemi come ad esempio quello delle coordinate virtuali andrebbero a non funzionare più come dovrebbero in modo rapido.

Garantire la consegna dei pacchetti è un qualcosa che possiamo fare ma ad un prezzo molto alto utilizzando ad esempio CLDP.

4.9 Clustering nelle Wireless Sensor Network

All'interno di una wireless sensor network in generale i sensori possono implementare un protocollo di clustering che permette di assegnare ai vari nodi della rete un certo ruolo in modo dinamico. Il protocollo di clustering viene eseguito periodicamente e all'interno della rete ai nodi possono essere assegnati due ruoli differenti andando quindi a formare una gerarchia:

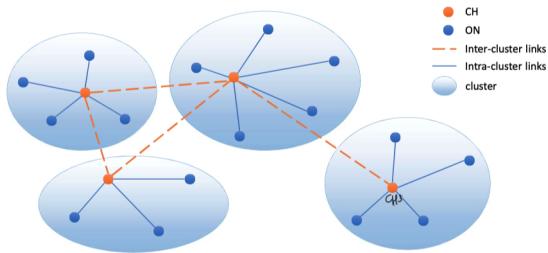
- Cluster Heads (CH)
- Ordinary Node (ON)

Il protocollo di clustering decide in che modo assegnare i ruoli ai nodi e i meccanismi per mantenere la gerarchia all'interno della rete. In generale l'utilizzo del clustering va a semplificare il funzionamento dei protocolli della rete, in particolare per quel che riguarda il routing e la scoperta di nuovi path all'interno della rete.

All'interno di ogni cluster di nodi abbiamo un cluster head che è "a capo" di quel cluster e permette di migliorare (in termini di efficienza) le seguenti operazioni:

- La comunicazione tra i vari cluster.
- L'accesso al canale per la trasmissione dei pacchetti (livello MAC). In questo caso possiamo anche ottimizzare lo sleep time per i nodi all'interno del cluster.
- Si occupa di coordinare anche l'aggregazione e la memorizzazione dei dati dei nodi nel cluster.

Vediamo un primo esempio di Clustered WSN:



Vediamo nell'esempio che in ognuno dei cluster abbiamo un Cluster Head (in arancione) e questi sono gli unici nodi che possono mantenere una connessione con i cluster head degli altri cluster. Questi collegamenti tra nodi seguono una topologia. È conveniente che i collegamenti tra i vari cluster head dei vari nodi siano implementati utilizzando i collegamenti fisici. Il cluster head di ciascuno di questi cluster deve anche gestire il routing, in particolare qua nell'esempio abbiamo il nodo U che vuole comunicare con il nodo V, sarà il cluster head del cluster dove si trova e poi i cluster head dei cluster successivi a gestire questa comunicazione e a capire come far comunicare i due nodi.

In particolare nel protocollo di routing per la maggior parte dei nodi (quelli che sono ordinary nodes) il problema principale consiste nella comunicazione con il cluster head, quindi l'intero protocollo è spostato sui cluster head. Nella rete ce ne sono vari di cluster head ma sono una piccola frazione rispetto al numero complessivo di nodi quindi è facile coordinarsi e creare delle routing table. Il protocollo per l'invio di un messaggio da un nodo ad un altro è diviso in tre fasi:

- Intra Cluster: il messaggio viene inviato dal mittente al suo cluster head.
- Inter Cluster: il messaggio viaggia tra i vari cluster passando da un cluster head all'altro fino a quando non arriva al cluster head del cluster di destinazione.

- Intra Cluster: il messaggio va dal Cluster Head di destinazione fino alla destinazione finale Vz.

Il Cluster Head crea e mantiene una backbone che sarebbe un grafo in cui memorizziamo i link che connettono i vari cluster tra di loro. La fase di route discovery avviene tutta utilizzando questa backbone e i Cluster Head che sono su questa backbone.

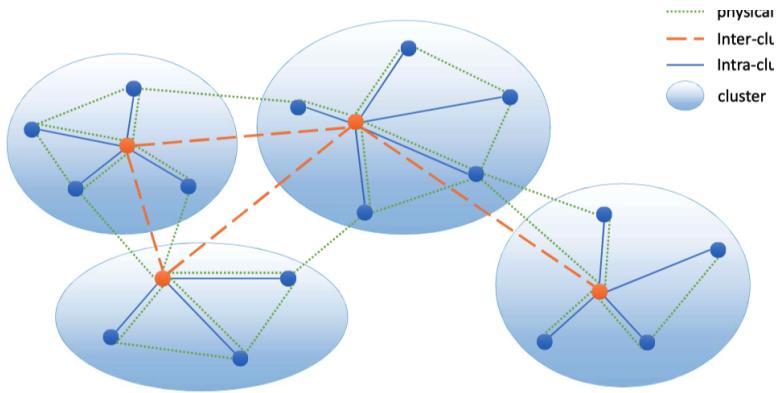
4.9.1 Design Aspects

Nel creare un cluster ci sono una serie di aspetti a livello di design che vanno considerati. Per quanto riguarda la comunicazione all'interno del cluster dobbiamo considerare che solamente alcuni dei collegamenti tra i vari nodi che sono in un cluster sono collegamenti fisici, molti sono invece collegamenti logici, cosa dovremmo fare in questo caso? Un altro problema riguarda il modo di comunicazione dei nodi all'interno del cluster, i nodi dovrebbero comunicare direttamente tra di loro oppure utilizzando il Cluster Head come mediatore?

Sempre parlando di logical link e di physical links dobbiamo considerare che anche la backbone è implementata utilizzando link logici e quindi c'è da trovare un modo per implementarla realmente con link fisici. Come possiamo fare poi per assegnare ruoli ai vari nodi e ogni quanto tempo questi ruoli vengono controllati nuovamente per essere sicuri che non siano cambiati?

Logical and physical links

Cominciamo la parte relativa al design dei cluster parlando di logical e physical links.

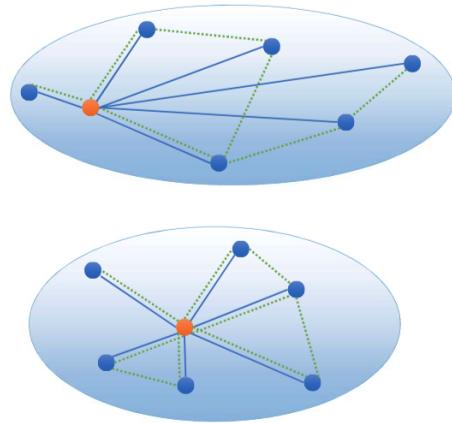


Nell'esempio vediamo le linee tratteggiate di verde che sono i link fisici mentre quelli blu sono i link logici. In questo caso abbiamo all'interno dei cluster due cluster head che sono collegati con un physical link a cui però non corrisponde un logical link. In questo caso per la comunicazione possiamo passare per gli altri link fisici.

C'è anche un'altra considerazione da fare, vediamo ad esempio che all'interno dei cluster abbiamo dei logical link che non corrispondono a link fisici, anche in questo caso la comunicazione dovrebbe essere implementata utilizzando i link fisici. Quando abbiamo questo invio di informazioni verso il cluster head quindi passiamo da un altro nodo, questo non vuol dire che stiamo permettendo una comunicazione diretta tra i due ordinary nodes ma solamente che l'ordinary node fa da "ponte" per questa comunicazione. Ad esempio nel caso di Bluetooth il protocollo è abbastanza stringente per quel che riguarda l'arbitraggio del canale e quindi non possiamo far comunicare direttamente due nodi ma dobbiamo utilizzare comunque il cluster head che si occupa dell'arbitraggio (altrimenti la comunicazione tra U e V potrebbe collidere con la comunicazione di altri nodi della rete). Ci sono poi delle possibili ottimizzazioni che, in alcuni casi, possono consentire di organizzare il cluster in modo che si abbia una comunicazione diretta tra i nodi all'interno del cluster.

Nei limiti del possibile la regola mi dice che per implementare i logical links si tende a sfruttare il più possibile la sovrapposizione con i physical

links.



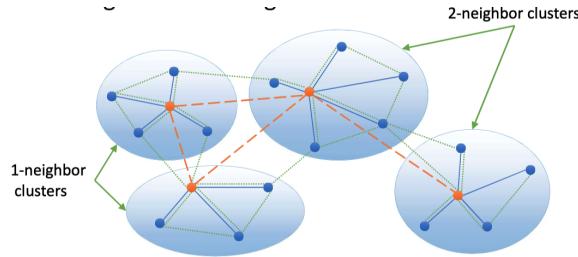
Nella figura sopra in particolare abbiamo nella prima immagine un cluster in cui c'è poca sovrapposizione tra i physical link e i logical links e questo non è buono mentre nella seconda immagine invece abbiamo molta sovrapposizione tra physical e logical links e questo è ottimo.

Un'altra configurazione per l'implementazione della comunicazione tra i cluster, qua noi la implementiamo utilizzando i link fisici già esistenti, in altri clustering possiamo avere dei link fisici che connettono direttamente i cluster head. Un'altra possibilità è che supportiamo la comunicazione tra i cluster head utilizzando delle interfacce radio a lungo raggio. In questo caso però i cluster head vengono assegnati in modo statico perché devono avere queste interfacce radio. Noi siamo più concentrati sulla situazione in cui i cluster sono assegnati in modo dinamico.

4.9.2 K-Neighbor Clustering

Nel caso del K-neighbor clustering siamo nella situazione in cui abbiamo un parametro k che mi limita la distanza tra gli Ordinary nodes e i Cluster Heads a k hops. Il parametro k è solitamente piccolo, il caso particolare è quello di $k = 1$. Questo caso particolare è quello che abbiamo ad esempio in Bluetooth e in Zigbee, qua abbiamo che tutti i link logici

sono in realtà dei link fisici e quindi c'è una sovrapposizione completa che porta quindi ad avere un cluster ottimo.

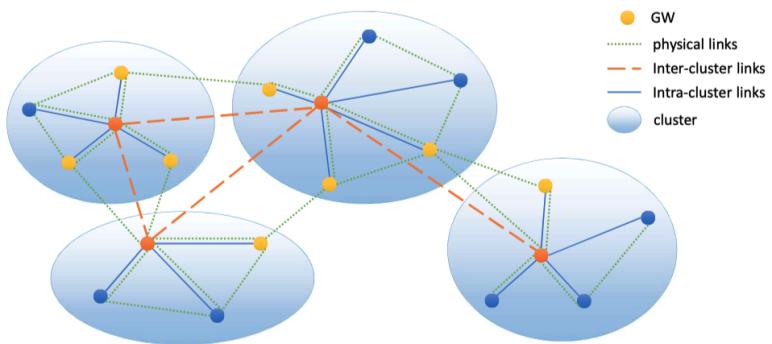


Prendiamo l'esempio della foto sopra, qua noi abbiamo dei cluster con $k = 1$ e dei cluster con $k = 2$ ovvero dei cluster dove permettiamo la distanza di 2 hop tra i nodi.

La cosa da sottolineare è che con la k noi ci riferiamo al numero di link fisici e non al numero di link logici tra i vari nodi.

Anche per quanto riguarda il collegamento tra i vari cluster possiamo utilizzare un ragionamento simile. In questo caso utilizziamo un parametro che chiamiamo h e che viene scelto piccolo ma con il vincolo che $h > k$.

Qua per collegare due cluster dobbiamo utilizzare dei nodi intermedi che sono sul percorso fisico tra i due cluster. Questi nodi che sono sul percorso intermedio prendono il nome di nodi gateway. Un esempio del funzionamento del clustering con la $h = 3$ lo possiamo vedere in questa immagine:



4.9.3 Assegnare i ruoli ai nodi

Per quanto riguarda l'assegnazione dei ruoli ai nodi dei vari cluster possiamo intanto dire che i cluster partizionano la rete in varie sotto reti (questo lo assumiamo noi ma non è una cosa necessaria) e che (solitamente) ognuno dei nodi si trova in un solo cluster alla volta (ad esempio questo è valido in Zigbee ma ci sono alcune eccezioni perchè per esempio in Bluetooth un dispositivo può fare parte di due piconet).

Il protocollo prima di tutto deve identificare i cluster e questo implica che devono essere trovati i nodi all'interno della rete. Tipicamente i nodi vengono inseriti all'interno dei cluster in modo che si abbia la maggiore connessione possibile.

Ci sono varie possibili metriche che possiamo utilizzare per creare questi cluster, abbiamo già visto una di queste che è rappresentata dal matching migliore tra link logici e link fisici. Un'altra metrica possibile riguarda il load balancing dei vari cluster. I vari cluster dovrebbero essere tutti della stessa dimensione, questo può essere estremamente difficile e in alcuni casi potrebbe anche non essere possibile ottenerli. Una dimensione bilanciata del cluster implica anche che il Cluster Heads avrà da svolgere una quantità di lavoro che è più o meno bilanciata (tra i vari cluster).

Un altro aspetto da considerare per la divisione è la stabilità. Un cambiamento all'interno della rete, con un nodo che fallisce o che si sposta potrebbe comportare un cambiamento anche nella clusterizzazione. Dato che siamo in una configurazione dinamica allora questo comporta che l'algoritmo di clustering va eseguito periodicamente. Questo comporta che nella creazione dei cluster andrebbero preferiti quei cluster che sono più stabili nel tempo e che ci potrebbero garantire di effettuare meno cambiamenti alla clusterizzazione, tutto questo perchè fare questi cambiamenti ha un costo e noi vogliamo avere costi bassi e quindi limitare il più possibili queste modifiche della clusterizzazione.

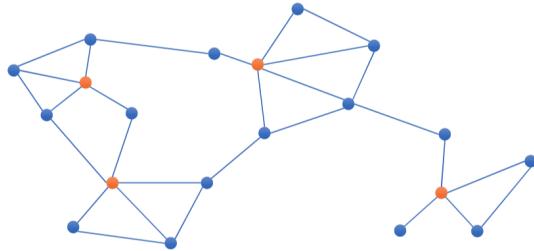
4.9.4 Dominating Set

Il dominating set D sottoinsieme di N è tale che:

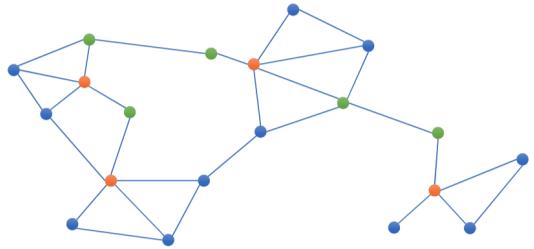
$$\forall u \in N \setminus D$$

there exists $v \in D$ such that $(u, v) \in E$

In pratica stiamo dicendo che ogni nodo che non è nel dominating set D deve essere connesso con un nodo del dominating set. Se il grafo G rappresenta una WSN in cui $k = 1$ allora vuol dire che possiamo considerare come dominating set l'insieme dei Cluster Head perché per ogni nodo che non è nel dominating set abbiamo un collegamento con il cluster Head. Ad esempio:



Un altro concetto è quello del Connected Dominating set (CDS), quando noi abbiamo il grafo G in cui abbiamo un dominating set che corrisponde alla clusterizzazione con $k = 1$, questo dominating set diventa un connected dominating set se consideriamo anche $h = 1$ ovvero se le Cluster Heads formano una backbone.



La definizione più formale di connected dominating set è che dato un grafo $G=(N, E)$ il connected dominating set C contenuto in N per il grafo G è tale che C è un dominating set e il sottografo $G' \in G$ creato con i nodi di C è connesso.

Nell'esempio sopra possiamo vedere che di base il grafo non sarebbe un connected dominating da solo perchè i nodi arancioni non sono direttamente connessi con $h=1$ (non formano un sottografo connesso in cui tutti i nodi sono connessi), per renderlo connected dominating dobbiamo aggiungere i nodi verdi al set di nodi.

In un grafo ci sono vari connected dominating set, quello che vogliamo fare è ottenere il minimo connected dominating set (ovvero quello che ha il numero minore di nodi), potremmo anche avere più di un minimo ma vorremmo comunque trovarne uno solo.

Trovare il minimum connected dominating set ci permette di ridurre il consumo di energia dei nodi. Questo perchè trovare il minimum connected dominating set consiste nel trovare il minimo numero di cluster heads, ogni cluster head coordinerà i nodi nel suo cluster in modo da poter ridurre il consumo di energia. Il cluster head avrà un consumo di energia maggiore perchè deve coordinare gli altri nodi, gli altri nodi però avranno un consumo di energia minore. Il problema è che risolvere il problema del minimum connected dominating set è un problema NP Hard, quindi non possiamo semplicemente risolvere distribuendo il lavoro nei vari nodi della rete ma dobbiamo trovare alternative calcolando ad esempio delle soluzioni approssimate.

4.10 Algoritmi di Clustering: DMAC

DMAC è uno degli algoritmi di clustering, in questa situazione quello che si fa è assegnare dei pesi ai vari sensori che abbiamo nella rete. Il peso rappresenta quanto quel determinato nodo è "disposto" a diventare un Cluster Head.

I pesi possono dipendere da vari parametri come ad esempio:

- Il grado del nodo nella rete in modo da favorire i cluster più grandi.
- Il livello di energia rimanente, potremmo eleggere un nodo che ha più batteria rimanente rispetto al cluster head.
- Il Node ID, se due nodi adiacenti hanno lo stesso grado allora possiamo considerare l'utilizzo dell'identificatore del nodo.

4.10.1 Funzionamento del protocollo

DMAC assegna ad ogni nodo:

- Lo status del nodo, che può essere Cluster Head (CH), ON (nodo normale) o UN (undecided, ovvero i nodi che non hanno un cluster).
- All'inizio i nodi dei vari cluster sono tutti undecided e quindi non sono assegnati ad un cluster. Poi viene assegnato ad un cluster.

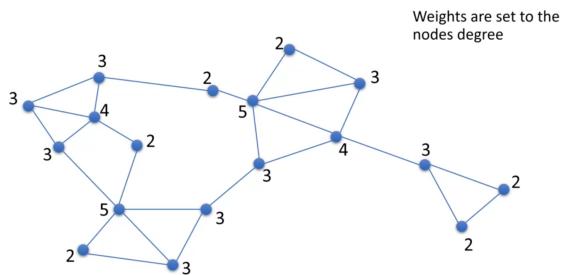
Quando il protocollo inizia a funzionare, ogni nodo invia in broadcast ai suoi vicini un messaggio di "Hello" che contiene un node ID, il peso del nodo e il suo status. Una volta che il nodo ha ricevuto l'Hello Message da tutti i vicini allora può decidere il suo status in base ad una decision rule:

- Se il suo peso è maggiore rispetto a tutti i pesi di tutti i nodi che stanno nel cluster (compreso anche il Cluster head) allora il nodo diventa Cluster Head.
- Una volta che questo nuovo nodo diventa cluster head deve informare tutti gli altri, quindi viene inviato un messaggio in broadcast in modo che tutti gli altri nodi del cluster possano diventare nodi ON.

Quando il nodo deve entrare nella rete, sceglie il cluster di appartenenza in base ad una association rule, in particolare sceglie il cluster dove si trova il CH che ha il massimo peso.

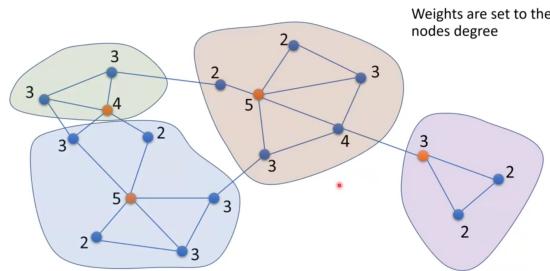
Il protocollo DMAC non garantisce che vengano creati cluster di cardinalità minima ma mi permette di avere una rete in cui la distanza massima tra nodi e cluster head sia $k = 1$. Quindi ogni nodo presente all'interno della rete o è un Cluster Head o è un vicino di un Cluster Head. Una regola è che due cluster head non possono essere adiacenti e questo è per via della seconda decision rule del protocollo (quando cambia il cluster head gli altri nodi del cluster diventano ON). La conseguenza è che all'interno dei cluster il parametro h sarà invece $h \geq 2$ o $h \leq 3$.

Vediamo come funziona il protocollo:



Partiamo con una rete di questo tipo, qua noi diciamo che i vari nodi hanno un certo peso che dipende da quanti vicini hanno nel grafo. I nodi che localmente hanno il peso più alto sono forzati a diventare dei cluster head.

In questo caso abbiamo 4 cluster head che formano quindi 4 cluster in base al protocollo di DMAC:



L'overhead dovuto alla comunicazione è minimo perché in pratica abbiamo un sistema che è totalmente localizzato all'interno del grafo, questo richiede in particolare una comunicazione di 1 solo "hop" alla volta. Quindi in pratica quello che si fa è scambiarsi $O(n)$ messaggi dove n è il numero di nodi all'interno della rete.

Il protocollo funziona bene anche perchè supporta l'aggiunta, la rimozione e lo spostamento di nodi, quindi supporta una rete che è dinamica. In particolare possiamo vedere che se un nodo se ne va dalla rete allora semplicemente diminuiamo il peso dei vicini, il peso cambia anche se il nodo si muove.

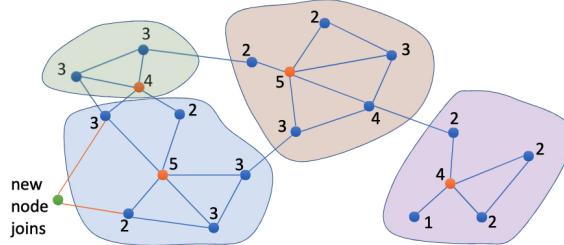
Nel caso in cui un nodo entra nella rete:

- Supponiamo che la rete sia già clusterizzata, al nodo che entra viene assegnato inizialmente uno status di UN (undecided).
- Il nodo inizia ad inviare messaggi ai vicini.
- Il nodo riceve risposte dai vicini che gli inviano i loro pesi.
- Il nodo diventa ON se uno dei suoi vicini è un CH e se questo ha un peso maggiore, altrimenti sarà lui il nuovo CH.
- Se il nodo deve creare un nuovo cluster allora lo crea e poi comunica in broadcast ai vicini che ora c'è un nuovo CH e un nuovo cluster.

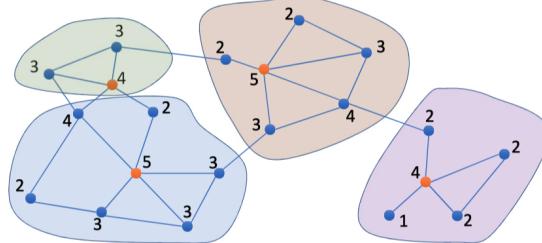
Quando un nodo entra nella rete (ma anche quando un nodo ha più batteria del CH che comunque consuma più rispetto ad un semplice nodo

o quando c'è un fallimento o un nodo si muove) abbiamo dei cambiamenti all'interno della rete che comportano un cambiamento nei pesi dei nodi presenti all'interno del grafo. Ogni tanto quindi è necessario eseguire una riconfigurazione della rete che potrebbe cambiare i vari ruoli dei nodi ma potrebbe anche cambiare i cluster della rete. Se un nodo entra nella rete il cambiamento in termini di peso è permanente mentre se il peso è legato alla batteria del nodo è un qualcosa che può modificarsi spesso. Se appena troviamo un nodo che ha più batteria del CH andiamo subito a fare la sostituzione del CH potremmo avere dei problemi di overhead perché rischieremmo di fare questa operazione varie volte. Quello che conviene fare è eseguire il clustering periodicamente in modo da evitare cambiamenti continui.

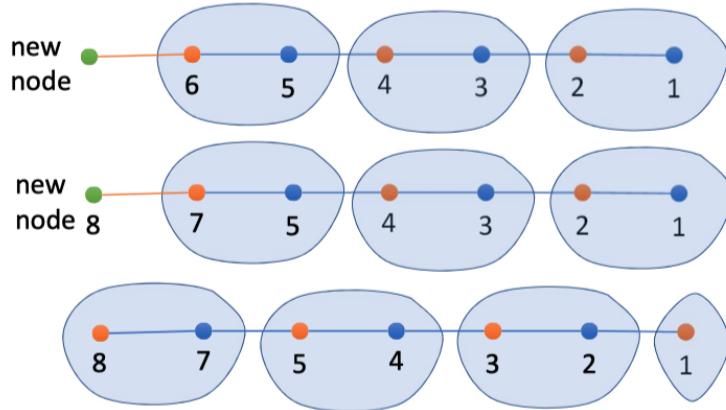
Esempio di ingresso di un nuovo nodo:



Abbiamo il nuovo nodo, che è quello verde che entra nella rete, questo va a modificare i pesi dei vicini di questo nuovo nodo che entra nella rete e potrebbe indurre una reazione a catena perché tutti gli altri nodi della rete potrebbero cambiare peso o potrebbero cambiare il cluster di appartenenza.



In generale noi vorremmo evitare di avere una reazione a catena perché vorrebbe dire che i vari nodi dovrebbero riconfigurarsi e quindi verrebbe introdotto molto overhead. La chain reaction la possiamo avere in modo più evidente quando i cambiamenti riguardano l'intero grafo, il caso particolare è quello in cui abbiamo i nodi che sono messi tutti in ordine su una sola linea e abbiamo una alternanza di nodi CH e di nodi ON con il peso dei vari nodi che decresce a mano a mano che ci spostiamo da destra verso sinistra.



Come conseguenza qua abbiamo che l'aggiunta del nuovo nodo mi comporta una riorganizzazione della rete, tutti i nodi devono cambiare il loro ruolo, questa è una situazione poco probabile ma comunque è possibile che accada.

Tipicamente abbiamo più probabilità di avere una chain reaction se aggiungiamo più nodi all'interno della rete e tipicamente la chain reaction

è limitata ai cluster che sono più vicini. In particolare se i nuovi nodi aggiunti nella rete vengono eletti come CH del cluster abbiamo ancora più problemi perchè i vicini dovranno modificare il loro status, se invece rimangono ON non abbiamo problemi.

4.11 Conclusione

Tutto quello che abbiamo visto sono principalmente ricerche che comunque hanno un impatto nell'utilizzo delle WSN. Nel caso di Zigbee i cluster sono utilizzati in modo statico e i nodi hanno i loro ruoli. Nel caso di Bluetooth abbiamo un singolo cluster nella rete e questo è una combinazione di statico e dinamico.

Chapter 5

Embedding Programming - Case Studies

I sistemi embedding sono sistemi differenti rispetto ai classici computer che conosciamo. Si basano su microcontroller ovvero su microprocessori che vengono utilizzati per la gestione dell'I/O. Le differenze principali rispetto ai computer sono:

- Non è possibile scrivere il codice che deve girare su questi microcontroller direttamente sul microcontroller e compilare il codice direttamente per il microcontroller. Dobbiamo utilizzare un computer classico per scrivere il nostro codice e per compilarlo per una piattaforma differente, si utilizzano dei cross compiler. A tempo di compilazione si deve specificare il tipo del sistema che stiamo andando ad utilizzare poi si esegue l'upload dell'eseguibile sul dispositivo aggiungendo altre informazioni, ad esempio potrebbe essere necessario l'identificativo del dispositivo per implementare, ad esempio, il protocollo di routing.
- Solitamente abbiamo un simulatore del dispositivo sul nostro computer per provare il codice prima di inviarlo proprio al dispositivo.

- Non è possibile controllare l'esecuzione del programma da una interfaccia utente.
- È solitamente più complesso eseguire il debug del codice.
- Su questi dispositivi non c'è un sistema operativo convenzionale, abbiamo solamente un set di librerie. Durante la compilazione questo set di librerie viene linkato staticamente al nostro codice. In sistemi più potenti, come ad esempio i RaspberryPi possiamo avere un sistema operativo vero e proprio. Nel corso consideriamo principalmente gli embedded system più semplici ovvero quelli che non hanno un OS ma solamente un set di librerie.
- Quando scriviamo un programma per un OS classico tipicamente implementiamo la funzione main. In un embedding system questo non lo facciamo perchè il main è fornito dal sistema operativo. Una volta che avviamo il dispositivo viene eseguita una funzione main che esegue il setup del dispositivo. Tutte queste operazioni non sono necessarie su un computer classico perchè in quel caso abbiamo il sistema operativo che se ne occupa.
- Il software del sistema embedding è organizzato tramite un loop, abbiamo anche qua un duty cycle che consiste in:
 - Lettura dai transducers.
 - Il sistema prende una decisione.
 - Vengono controllati gli attuatori.
 - Si comunica con altri dispositivi.

Alla fine di questo ciclo di eventi il dispositivo viene messo in sleep mode fino a quando non inizia il prossimo ciclo. Questo loop solitamente non lo dobbiamo scrivere noi ma è già scritto all'interno della main function del dispositivo.

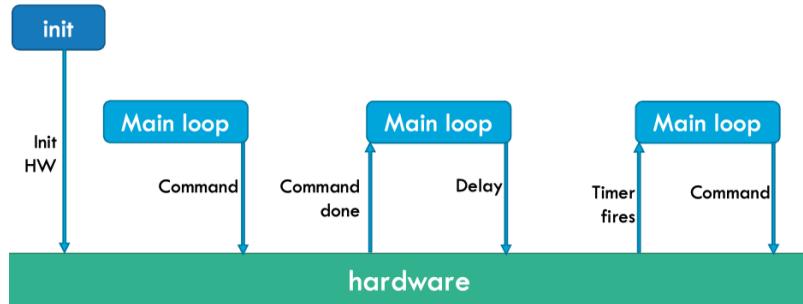
- Il sistema embedded deve anche interagire con hardware esterno e in alcuni casi ha anche una memoria interna. Il sistema supporta l'utilizzo di hardware esterno e di memoria fornendo un set di comandi che possono essere utilizzati (ad esempio per leggere i segnali dai transducers). Un comando tipico consiste nell'attivazione dell'hardware e questo tipicamente è un processo che richiede del tempo. Questo è diverso rispetto ai sistemi operativi classici dove questo tipo di comandi vengono messi a disposizione tramite della system call. Durante queste system call tipicamente c'è da attendere e quindi il thread che invoca la system call viene sospeso, durante questo periodo la CPU viene assegnata ad un altro thread. In pratica quello che succede è che le operazioni di I/O vengono mascherate dal sistema operativo.
- Abbiamo detto che nei classici sistemi operativi abbiamo thread che possono sospendersi e riattivarsi, questo richiede che questi thread abbiano a disposizione uno stack in cui memorizzare i loro dati. In un embedding system questo è un problema perché la memoria che abbiamo a disposizione è limitata e potremmo non avere spazio per questi dati. Prendiamo ad esempio il caso dell'Arduino 1, abbiamo solamente 4KB di ram che va utilizzata per memorizzare le variabili, i dati per la comunicazione e la lettura e lo stack dei thread (il codice è nella flash memory). La conseguenza è che i thread potrebbero semplicemente invocare un comando e in caso di attesa dovrebbero solamente attendere facendo attesa passiva perché non avrebbero spazio per i dati. Le soluzioni per questo problema sono due e dipendono dal tipo di embedded system che stiamo utilizzando:
 - Con arduino abbiamo un event loop.
 - Con tiny OS abbiamo invece una programmazione ad eventi.

5.1 Arduino

Vediamo per primo l'approccio utilizzato da Arduino, abbiamo una single loop function in cui il loop viene eseguito ripetutamente su un singolo thread che non si sospende mai quindi non abbiamo bisogno di uno stack per gestire la sospensione del thread. Durante l'esecuzione del codice possono esserci degli accessi all'hardware esterno e quindi ad esempio l'utilizzo dell'I/O. Questo comporebbe delle attese ma in ogni caso nessun altro thread viene eseguito nello stesso momento, si attende fino a che è necessario. Da codice possiamo anche decidere di ritardare l'esecuzione del programma in modo esplicito aggiungendo delle wait.

Questo è il funzionamento del loop di Arduino:

- Abbiamo all'inizio la chiamata (fatta dal sistema e non dal programmatore) di una init function che si occupa di inizializzare la comunicazione con l'hardware.
- Alla fine della init function inizia l'esecuzione del Main loop, qua noi possiamo inviare dei comandi all'hardware. Per attendere la risposta dell'hardware il thread viene messo in attesa fino a che non arriva dall'hardware una interruzione che mi indica che l'esecuzione del comando è terminata.
- Una volta che l'esecuzione del comando è terminata il main loop viene eseguito normalmente e la CPU torna a funzionare. Se a questo punto abbiamo terminato l'esecuzione della parte "attiva" del codice del main loop, possiamo invocare una funzione di attesa che invia un comando al timer, la CPU viene bloccata nuovamente fino a quando il timer viene riattivato e il loop viene riavviato.



Questo modello è molto semplice e questo è uno dei motivi per cui Arduino ha avuto una forte espansione. Arduino offre tante librerie per lavorare con sensori, originariamente non era stato pensato per la comunicazione con altri dispositivi e infatti era possibile solamente la comunicazione tramite la serial line. Poi con il tempo sono stati prodotti altri moduli per la comunicazione, tipo quello Wi-Fi e con loro altre librerie per garantirne il funzionamento. La comunicazione con altri dispositivi comporta altri problemi che devono essere risolti, potremmo infatti ricevere una comunicazione asincrona da uno degli altri dispositivi. Pensiamo ad esempio ad una WSN, i sensori avranno il loro duty cycle per il processore ma devono anche avere un duty cycle per la radio in modo da ricevere segnali da altri dispositivi. Il problema che si può verificare è che possiamo ricevere un messaggio da un altro dispositivo nel momento in cui la CPU del nostro dispositivo è in sleep mode e poi quando ripende il loop non c'è modo di riprendere quel segnale che abbiamo ricevuto perché non ce lo aspettavamo. Quindi l'unico modo per avere la comunicazione è introdurre dei meccanismi per la gestione di eventi asincroni, come avviene per esempio in TinyOS. Arduino offre anche questa possibilità (simile a TinyOS), offre infatti la possibilità di intercettare e gestire le interruzioni direttamente dal codice. Questa possibilità di gestire le interruzioni era presente quando la comunicazione avveniva solamente con le serial line ma in quel caso era gestita delle librerie fornite dal sistema.

5.2 TinyOS

In TinyOS non abbiamo il concetto di Event Loop ma abbiamo una gestione asincrona degli eventi, definiamo in particolare tre entità:

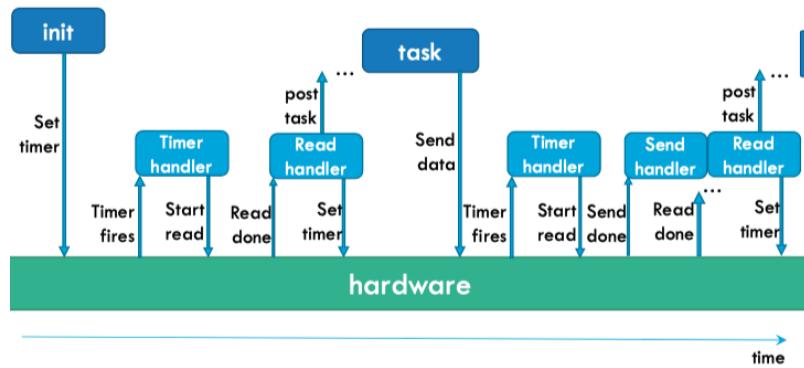
- Commands: sono le funzioni che vengono offerte per programmare e attivare l'hardware.
- Events: Le interruzioni vengono astratte sotto forma di eventi e queste sono molto simili alle system call del sistema operativo. Si tratta di una sorta di up call e il programmatore ha il compito di definire il modo in cui questa upcall verrà gestita.
- Tasks: i task sono eseguiti in modo sequenziale e non attendono durante l'esecuzione, quando inizia l'esecuzione poi viene anche portata a termine. I task possono essere attivati come conseguenza di un evento. Quando finisce l'esecuzione di un task e non ce ne sono altri da eseguire allora il processore va in idle. In questo approccio non abbiamo la necessità di memorizzare lo stato e il contesto di un task e questo permette di risparmiare memoria.

L'unica eccezione al normale funzionamento del task l'abbiamo quando arriva una interruzione, in questo caso l'interruzione dovrà essere subito gestita dall'event handler e dovrà completare il suo lavoro per due motivi:

- Perchè durante l'esecuzione di un event handler non puoi ricevere un'altra interrupt quindi andresti a rimandare la ricezione di altre interrupt.
- Perchè l'event handler non è pensato per implementare task complessi, aggiorna solamente le strutture dati e poi se c'è più lavoro da fare lo inserisce in un nuovo task che deve essere eseguito.

Vediamo un esempio di esecuzione con TinyOS, l'implementazione del duty cycle qua è più complessa rispetto a quella che abbiamo con Arduino perchè gestiamo tutto con dei timer, inoltre anche le operazioni di I/O verranno spartite in due fasi:

- la prima fase: qua noi scriviamo il comando.
- La seconda fase: qua ci assicuriamo che il comando sia eseguito e poi andiamo a gestire l'output del comando.



Anche in questo caso abbiamo un init in cui viene settato il timer, l'init è definito dal programmatore. L'attività vera e propria viene avviata nel momento in cui scatta il timer.

Quando scatta il timer viene gestito questo timer e ad esempio viene effettuata la lettura dall'hardware che a sua volta causerà una upcall nel momento in cui la lettura è completata. Quando si finisce di effettuare la lettura poi il task viene inserito in una task queue. All'interno del task possiamo inserire comandi più complessi come ad esempio la trasmissione di dati.

5.3 Caso di studio: Arduino

Arduino è un progetto open source che fornisce una coppia hardware, software facile da utilizzare e destinato principalmente a progetti di hobby per coloro che sono interessati a creare oggetti o ambienti interessanti.

Esistono vari tipi di Arduino, in particolare l'Arduino Uno è stato il primo e aveva solamente 2KB di ram e 32KB di memoria interna. Per l'I/O è possibile utilizzare 13 pin digitali, in particolare quelli che possono inviare delle interruzioni dall'esterno sono solamente i pin 3 e 2. Poi dopo l'arduino Uno ce ne sono stati anche altri. Il software da eseguire sull'Arduino viene scritto in un IDE (open source) e poi viene inviato alla scheda. Quando inviamo il codice alla scheda e lo compiliamo dobbiamo selezionare la scheda che stiamo utilizzando in modo che il codice venga compilato per quella specifica scheda.

Il programma che scriviamo e che viene eseguito sull'arduino si chiama sketch, questi vengono programmati utilizzando un linguaggio simile al C. Durante l'esecuzione del programma possiamo accedere ai vari input collegati alla scheda tramite i PIN, da ognuno di questi pin dobbiamo leggere un valore analogico compreso tra 0 e 1023, ad esempio un valore di questo genere indica la luce di un certo led o la velocità di un motore.

All'interno degli sketch dobbiamo definire due funzioni:

- Setup: è la funzione che viene eseguita solamente una volta quando lo sketch viene avviato.
- Loop: questa funzione viene eseguita di continuo quando eseguiamo il programma, subito dopo il main verrà eseguito il codice nel loop.

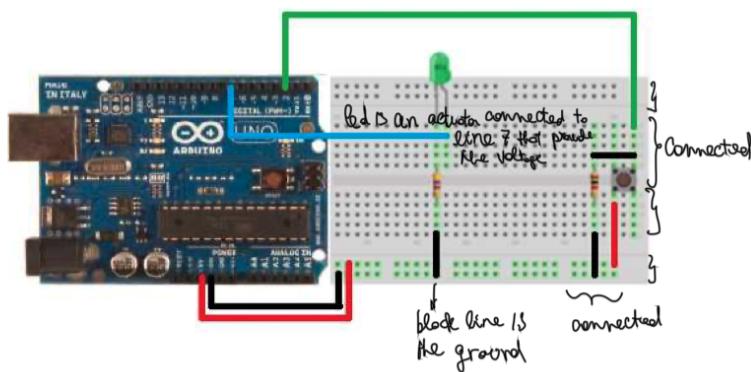
Negli sketch possiamo utilizzare delle variabili e delle funzioni che sono già definite, esistono poi tante librerie per il controllo di sensori o dispositivi collegabili ad Arduino.

Anche se di base lo schema di funzionamento di Arduino consiste nell'utilizzo di sketch che leggono i sensori in modo sincrono, esiste anche la possibilità di accedere in modo asincrono ai sensori. Arduino in particolare fornisce una interfaccia per le interruzioni.

In particolare con Arduino abbiamo tre tipi di interruzioni:

- Esterne: dal punto di vista del programmatore noi dobbiamo pensare a gestire solamente le interruzioni esterne. Possiamo utilizzare la funzione `attachInterrupt`, in questo modo Arduino può gestire due linee di interruzione. Tramite queste due linee (INT0 e INT1) che sono mappate, come visto, con i pin 2 e 3, possiamo connettere un dispositivo esterno e possiamo riconoscere un cambio nello stato in modo asincrono.
- Dal timer: queste sono interruzioni che vengono gestite direttamente dal sistema operativo.
- Device: queste sono interruzioni che vengono gestite direttamente dal sistema operativo.

Un esempio di connessione di un dispositivo esterno con Arduino:



Possiamo notare dall'immagine come sono collegate le varie parti delle

breadboard, in particolare abbiamo le parti orizzontali che sono connesse tra loro, così come quelle verticali.

Vediamo un esempio di codice scritto per funzionare con Arduino:

```


    /**
     * prova interrupt
     */
    volatile int greenLed=7;
    volatile int count=0;

    void setup()
    {
        Serial.begin(9600);
        pinMode(greenLed, OUTPUT);

        digitalWrite(greenLed, LOW);

        attachInterrupt(0,
            interruptSwitchGreen,
            RISING);
        // 0 because it is pin 2
    }

    void loop()
    {
        count++;
        delay(1000);
        // interrupts are received
        // also within delay!

        Serial.print("waiting:");
        Serial.println(count);

        if ( count == 10 )
        {
            count = 0;
            digitalWrite(greenLed, LOW);
            Serial.println("now off");
        }
    }

    void interruptSwitchGreen()
    {
        digitalWrite(greenLed, HIGH);
        count=0;
        Serial.print("now on");
    }
}


```

La prima cosa che possiamo notare è che le due variabili che definiamo (greenLed e count) sono dichiarate come volatile. Se non le dichiarassimo volatile il processore le allocherebbe in un registro ma questo comporterebbe che, quando una di queste variabili viene aggiornata, l'aggiornamento non avrebbe effetto nel loop in cui viene utilizzata (ad esempio count viene portata a 0 e se non fosse volatile non me ne accorgerei). In questo modo il compilatore viene forzato a memorizzare la variabile in memoria, c'è anche un'altra variabile che è GreenLed che è volatile anche se non è mai stata scritta, è in pratica una costante e quindi in questo caso potremmo anche evitare di definirla volatile.

Il codice accende un led che inizialmente è spento. Noi nel codice del setup gestiamo una interruzione e quando arriva andiamo ad accendere il led, nella funzione loop poi questo led viene solamente spento e per spegnerlo andiamo a resettare un contatore.

5.4 Case Study: TinyOS

TinyOS è un sistema operativo open source minimale e destinato alle sensor network. È programmato con un linguaggio generato dal C che è molto semplice e il modello d'esecuzione è quello event based. Come detto il linguaggio che viene utilizzato è una estensione del C e per questa ragione sarebbe anche possibile utilizzare la malloc. Anche se è possibile, l'utilizzo delle malloc è una cattivissima idea, così come anche l'utilizzo della ricorsione.

Una volta che abbiamo il programma scritto per TinyOS, il codice viene caricato nel firmware e solamente un programma alla volta può essere eseguito dal sistema. Il programma è formato da un set di componenti che sono connessi tramite interfacce, i componenti sono linkati tra di loro, alcuni sono definiti dall'utente, altri sono definiti dall'OS. La configurazione delle componenti è quello che è necessario per il compilatore per creare un eseguibile TinyOS. Ogni componente ha una interfaccia che definisce comandi ed eventi, i comandi definiscono una funzionalità che viene offerta mentre gli eventi rappresentano le notifiche di un evento che sta avvendendo.

Possiamo vedere gli eventi come delle upcall mentre i comandi possiamo considerarli come delle downcall. I componenti che usano una interfaccia possono invocare i comandi. Il componente che offre l'interfaccia deve implementare tutti i comandi definiti all'interno dell'interfaccia e può segnalare tutti gli eventi definiti nell'interfaccia.

I componenti sono delle unità software che definiscono un modulo, il modulo è una specifica dell'interfaccia che viene utilizzata dal componente o dall'interfaccia offerta dal componente e include una implementazione offerta dal componente. I componenti sono anche associati ad una configurazione che definisce quali componenti vengono usati da un modulo e in che modo vengono utilizzati.

Un esempio di TinyOS, vogliamo solamente accendere un flash, qua possiamo vedere la configurazione:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```
configuration BlinkAppC {
}
implementation {
    components MainC, BlinkC, LedsC;
    components new TimerMilliC() as Timer0;
    components new TimerMilliC() as Timer1;
    components new TimerMilliC() as Timer2;

    BlinkC.Boot -> MainC.Boot;
    BlinkC.Timer0 -> Timer0;
    BlinkC.Timer1 -> Timer1;
    BlinkC.Timer2 -> Timer2;
    BlinkC.Leds -> LedsC;
}
```

Questo codice viene eseguito dal Main di TinyOS, possiamo vedere che vengono definiti tre componenti che sono tre timer. Poi nelle righe finali utilizziamo l'operatore `->` e vediamo che a sinistra viene scritto il componente che utilizza l'interfaccia mentre a destra abbiamo il componente che offre l'interfaccia che stiamo utilizzando. Ad esempio l'interfaccia Boot che viene usata nel modulo BlinkC è offerta al componente MainC.

Un'altra parte del codice è quella del modulo che possiamo scrivere per l'implementazione, ogni volta che viene acceso il dispositivo viene eseguita la funzione `booted()`

```

implementation
{
  event void Boot.booted() {
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
  }
  event void Timer0.fired() {
    call Leds.led0Toggle();
  }
  event void Timer1.fired() {
    call Leds.led1Toggle();
  }
  event void Timer2.fired() {
    call Leds.led2Toggle();
  }
}

```

Localsense con TinyOS

Un altro esempio che possiamo considerare con TinyOS è quello del localsense, qua noi leggiamo un valore ogni 500 millisecondi, per capire quando dobbiamo leggere il valore si utilizza un timer, quando scatta il timer usiamo il comando read. Se il valore che leggiamo è inferiore ad un certo threshold cambiamo lo stato del led. Qua la lettura che viene effettuata è una lettura completa e mai intermedia, solamente quando ho letto tutto quello che doveva essere letto allora possiamo trasformare la lettura in un evento.

A grandi linee il codice è il seguente:

```
Configuration SenseAppC{}
```

```
Implementation{
```

```

  Component senseC, MainC, LedsC;
  Component new timer Millic() as timer;
  Component new DemoSensorC() as sensor;

  SenseC.Boot;
  .
  .
}

}
```

Ogni volta che scatta il timer facciamo un sample utilizzando il comando read, il risultato non è immediato, solamente quando il risultato della lettura sarà completa allora potremo trasformare la lettura in un evento.

L'interfaccia *Read < uint16_t >* che utilizziamo per la lettura non è qualcosa di specifico per un sensore, è una interfaccia generica e i vari sensori possono implementarla. Nella implementazione attendiamo un evento, quando il timer scatta allora invochiamo la lettura dall'interfaccia di lettura. È interessante notare come funziona la read per evitare che il thread attenda più del dovuto. Dopo la lettura viene invocato l'evento ReadDone, questo ha come parametro il risultato della operazione di read e un codice di errore se il risultato rappresenta un fallimento.

Il comportamento dell'operazione di Read è molto differente in TinyOS da come è per esempio in C. In C quando facciamo la lettura abbiamo una operazione bloccante che non termina fino a che non è stato letto tutto quello che era necessario leggere. Qua invece viene utilizzata una strategia chiamata Split Phase in cui prima facciamo la lettura poi nella seconda fase riceviamo il risultato dell'operazione di lettura tramite un evento.

Questi eventi che vengono utilizzati sono eventi asincroni, la concorrenza viene gestita all'interno di TinyOS tramite due meccanismi che vengono chiamati Tasks e atomic statement:

- Task: i task sono funzioni void che non possono prendere parametri, per passare un parametro a queste funzioni possiamo utilizzare il contesto, sono utilizzati per computazioni lunghe. Per memorizzare i task che devono essere eseguiti abbiamo una coda di task di tipo FIFO, abbiamo un thread che gestisce questa coda, un task viene attivato (ovvero viene inserito all'interno della coda) tramite la keyword post. Solitamente mettiamo solamente una istanza del task alla volta all'interno della coda, in ogni caso se stiamo eseguendo il task allora il task stesso può inserire una nuova istanza

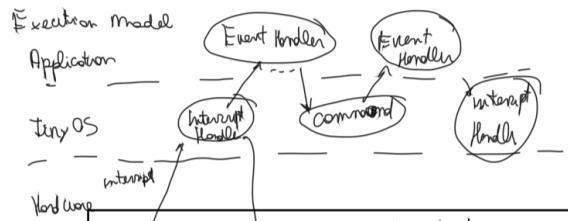
del task all'interno della coda.

Vediamo il modello di esecuzione:

- EventX: Abbiamo un thread che gestisce gli eventi, non abbiamo bisogno di tenere il contesto completo per questo thread.
- TaskX: Abbiamo un altro thread che invece gestisce i task e li esegue uno dopo l'altro. Questo thread ha una priorità minore rispetto all'EventX.

La cosa migliore da fare è rendere gli eventi il più corti possibili e in particolare quello che dobbiamo fare è evitare di utilizzare all'interno degli eventi le chiamate al sistema. Quando dobbiamo eseguire calcoli un po' più pesanti e che necessitano di più tempo allora non utilizziamo un evento ma utilizziamo un task.

Il modello di esecuzione lo possiamo rappresentare con questo schema:



vediamo una spiegazione:

- Dal livello hardware arriva l'interruzione.
- L'interruzione viene gestita dal livello di TinyOS che invia poi l'evento corrispondente al livello applicazione.
- Al livello applicazione viene gestito l'evento e viene inviato da qua al livello TinyOS il comando che deve essere eseguito.

- Dal livello di TinyOS poi si passa di nuovo al livello applicazione in cui gestiamo l'evento e poi si passa di nuovo al livello di TinyOS in cui gestiamo l'interruzione.

Abbiamo però un problema, cosa succede se, durante la gestione di un evento riceviamo la notifica da un altro evento che utilizza le stesse variabili? La soluzione è utilizzare un atomic statement ovvero, alcune parti del codice, dette critical accedono a variabili che sono in comune tra i vari eventi, queste parti vanno inserite in una parte di codice detto "atomic". La "regola" è che queste parti di codice atomico non dovrebbe essere troppo lungo per evitare di non poter accettare interruzioni per un tempo troppo lungo.

5.4.1 Communication Stack

TinyOS offre uno stack per la comunicazione che è gestito con "Active Messages", in particolare i dispositivi più "tipici" forniscono una comunicazione radio ma la comunicazione può avvenire anche tramite USB. Indipendentemente dall'interfaccia che viene utilizzata, avremo dei canali di comunicazione e possiamo definirne vari differenti a seconda del tipo di messaggio che vogliamo inviare. Tramite questi canali poi il messaggio che riceviamo viene inoltrato alla applicazione corretta. I messaggi in particolare vengono filtrati dal livello fisico. Ogni pacchetto che inviamo viene caratterizzato da un *AM_type*.

```
typedef nx_struct message_t {
    nx_uint8_t header[sizeof(message_header_t)];
    nx_uint8_t data[TOSH_DATA_LENGTH];
    nx_uint8_t footer[sizeof(message_footer_t)];
    nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```

Nella immagine sopra vediamo un esempio di messaggio che viene inviato:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- All'interno del pacchetto abbiamo il footer e i metadata
- Abbiamo gli array di bytes che vengono definiti come nx_uint8_t , il problema è che differenti dispositivi con differenti processori potrebbero avere un modo differente di rappresentare i bytes. Utilizzando $nx_\underline{}$ non stiamo definendo i dati come semplici caratteri ma stiamo utilizzando una rappresentazione univoca. I dispositivi con processore differente vedranno il messaggio nello stesso modo.

TinyOS fornisce interfacce per inviare/ricevere/esplorare i pacchetti. Usando lo splitcontrol possiamo implementare anche strategie efficaci dal punto di vista energetico.

Chapter 6

Signal Theory - Introduction to Theory of signals

Noi dobbiamo gestire dei sensori e dobbiamo eseguire il sampling dei dati registrati dal sensore.

Il sample che facciamo lo possiamo definire in un dominio temporale, dobbiamo fare letture periodiche e possiamo renderle digitali, alla fine il sensore dovrà anche trasmettere questo dato.

Per l'invio dei dati possiamo utilizzare la comunicazione wireless, il pacchetto di dati deve essere convertito in un segnale analogico, ad esempio onde radio, quando poi viene ricevuto il pacchetto viene nuovamente trasformato in un segnale digitale.

Per capire bene il funzionamento dei segnali dobbiamo considerare alcune domande:

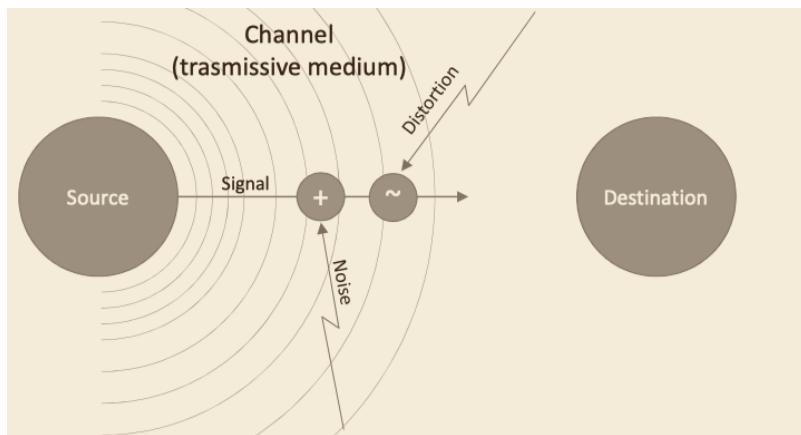
- Come possiamo decomporre un segnale?
- Come possiamo ricostruire un segnale?

- Come possiamo fare il sample?

Per dare una risposta a queste domande si devono comprendere le serie di Fourier.

6.1 Trasmissione

La trasmissione dei messaggi coinvolge vari componenti che solitamente sono organizzati in una rete. Possiamo assumere che i messaggi che devono essere spediti sono generati prima della trasmissione e in generale assumiamo anche che il supporto fisico per questi messaggi è un segnale. I segnali possono essere sia analogici (ad esempio la voce di una persona che parla) che digitali (ad esempio un documento in un pc), noi utilizziamo i segnali digitali e dobbiamo fare una trasformazione in segnali analogici.



Consideriamo la trasmissione delle informazioni, la foto sopra rappresenta il funzionamento generale, in particolare abbiamo una fonte e abbiamo una destinazione che vorremmo ricevesse il segnale che stiamo inviando. Durante la trasmissione al segnale viene spesso aggiunto del rumore che è in pratica un altro segnale che aggiungiamo al segnale che

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

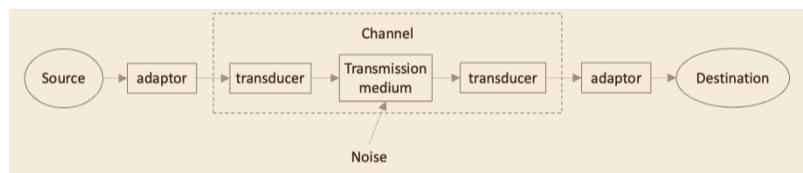
stavamo già trasmettendo. Un altro segnale che talvolta viene aggiunto a quello originale è la distorsione.

Quando viene trasmesso il segnale passa in un canale e questo canale potrebbe alterare il segnale, questa alterazione può essere rappresentata da un parametro che possiamo utilizzare per esprimere la qualità del canale. I cambiamenti che il messaggio ha quando viene trasmesso attraverso un canale sono chiamati Signal to noise ratio (SNR). I segnali possono avere natura differente a seconda del canale, abbiamo onde elettromagnetiche, onde sonore, onde ottiche, tutte quante sono adatte per la trasmissione.

Durante la trasmissione utilizziamo i transducers per convertire i messaggi da una forma all'altra e la conversione accade sia dalla parte della trasmissione sia dalla parte di chi riceve.

6.1.1 Trasmissione delle informazioni analogiche

Vediamo nel dettaglio in che modo funziona la trasmissione di informazioni analogiche:



All'interno della figura sopra abbiamo i vari componenti che sono utilizzati durante la comunicazione:

- Abbiamo la fonte da cui parte il pacchetto.
- Adaptors: è un componente che è una sorta di amplificatore del segnale e che viene utilizzato per migliorarne la qualità. Questo componente serve a ridurre l'effetto del rumore e della distorsione che solitamente sono presenti all'interno di catene come questa.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

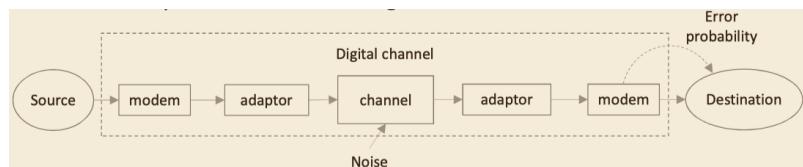
- Una volta che è passato dall'adaptors il messaggio passa attraverso il canale, qua il messaggio passa prima in un transducers che si occupa di convertire il messaggio in un segnale poi entra in un canale vero e proprio dove potremmo avere rumore e distorsione. Quando il messaggio esce dal canale viene convertito di nuovo da un transducers.
- Una volta finito questo viaggio il segnale passa di nuovo all'interno di un adaptor che si occupa di migliorare nuovamente la qualità del segnale.

Finita tutta questa catena il segnale può essere finalmente utilizzato dalla destinazione.

6.1.2 Trasmissione digitale

Vediamo ora come funziona per la trasmissione di segnali digitali, i messaggi che si scambiano in questo caso sono dei messaggi rappresentati da una sequenza di simboli. In questo caso, anche se la propagazione del segnale lungo il canale rimane simile, viene utilizzato un componente in più che si occupa della conversione del segnale da digitale ad analogico e viceversa.

Possiamo rappresentare anche in questo caso la comunicazione in questa figura:



- Come detto abbiamo un modem che è necessario quando dobbiamo inviare un segnale digitale.

- Abbiamo anche qua un adaptor per amplificare il segnale.
- Abbiamo poi il canale vero e proprio in cui passa la comunicazione, il segnale ora è diventato analogico e passa all'interno di questo canale dove potrebbe essere aggiunto rumore e distorsione. In alcuni casi la quantità di rumore e distorsione che si ottiene potrebbe portare anche a non riuscire a riconoscere il segnale.
- Una volta usciti dal canale abbiamo poi un altro adaptor e un altro modem che mi permette di convertire il segnale da analogico a digitale.

Quando facciamo una trasmissione digitale abbiamo due possibili tipi di encoding che in parte si sovrappongono ma che funzionano in contemporanea:

- Channel encoding: per rendere la trasmissione più robusta vengono trasmessi più simboli di quanti ne dovrebbero essere davvero trasmessi, viene usata la ridondanza. Questa ridondanza potrebbe essere utile per capire se ci sono stati errori di trasmissione e in alcuni casi, a ricostruire i simboli originali.
- Source encoding: la fonte potrebbe codificare i simboli che vengono generati, questo per esempio potrebbe tornare utile per eseguire una compressione dei dati oppure per avere una ridondanza in modo da ottenere una migliore trasmissione.

In alcuni casi il canale per la trasmissione digitale viene anche utilizzato per trasmettere segnali analogici, è il caso ad esempio dei video su Youtube, all'inizio il video è analogico e viene trasformato in un segnale digitale per fare in modo che venga trasmesso tramite internet. Il segnale analogico viene rappresentato da una sequenza di simboli. Per eseguire questa trasformazione e questo invio il trasmettitore deve:

- Eseguire il sampling: nel trasmittor dobbiamo eseguire il sampling dal segnale analogico in modo da leggere da questo segnale.
- Questo implica una quantizzazione del segnale che viene eseguita dall'ADC (Analog Digital Converter).
- Dopo questo processo il segnale digitale è rappresentato da una sequenza di simboli.

Quando il messaggio arriva al ricevente poi i simboli vanno convertiti di nuovo in un segnale analogico per poter essere utilizzati, per fare questo viene utilizzando un DAC ovvero un Digital Analog Converter.

Il processo di sampling e di quantizzazione introduce distorsione del segnale analogico, in generale possiamo dire che perdiamo informazioni nel momento in cui abbiamo il processo di sampling. La perdita di informazione è chiamata "quantization noise", questa dipende da:

- Frequenza di sampling
- Quantità di simboli che abbiamo per rappresentare il singolo segnale analogico.

La quantization noise riusciamo ad abbassarla se aumentiamo la frequenza del sampling e se aumentiamo il numero di simboli differenti che vengono utilizzati per rappresentare un singolo valore analogico.

6.2 Teoria di Shannon

Abbiamo ora una teoria presentata da Shannon in cui si cerca di capire quale sia il limite di informazioni che possono essere trasmesse tramite un canale, si tratta in realtà di un limite che è prevalentemente teorico. La formula della capacità massima del canale è la seguente:

$$C = B \ Log_2\left(1 + \frac{S}{N}\right)$$

all'interno della formula abbiamo:

- B rappresenta la massima bandwidth;
- S rappresenta la potenza del segnale;
- N rappresenta il rumore dalla parte del ricevente;

Alla teoria di Shannon sono legate vari concetti come ad esempio il tempo necessario per la trasmissione, in particolare diciamo che se abbiamo una piccola quantizzazione nel mittente allora avremo una quantità maggiore di informazioni da trasmettere. Se il canale ha capacità minori allora avremo una maggiore distorsione del segnale o un tempo maggiore per la trasmissione.

6.3 Classificare i segnali

I segnali possono essere classificati in due tipologie principali:

- Segnali deterministici: un segnale è deterministico se è conosciuto prima che venga prodotto, Fourier viene applicato ai segnali deterministici.
- Segnali Random: se non conosciamo il segnale prima di applicarlo allora il segnale è random, in questo caso l'analisi del segnale viene eseguita solamente con metodi statistici e non possiamo utilizzare Fourier.

In queste analisi noi ci concentreremo solamente sui segnali deterministici.

Un segnale può essere rappresentato da una funzione

$$f(t) : D \rightarrow C$$

la funzione viene definita con un dominio D e produce valori in un codominio C . Quando lavoriamo con segnali continui abbiamo la D che troviamo all'interno della definizione è il set dei numeri reali. Quando lavoriamo con segnali discreti invece abbiamo che il dominio D è il set degli interi $Z(T)$ dove la $T \in R$ e $Z(T) = nT, \forall n \in Z$ rappresenta il set dei multipli dei numeri interi, ad esempio: $Z(1) = \dots, -2, -1, 0, 1, 2, \dots$

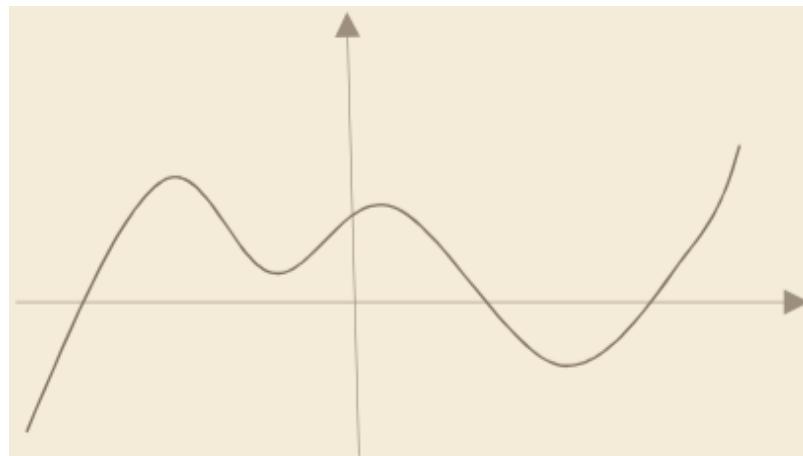
Il codominio dipende dal tipo di segnale che abbiamo:

- Il codominio è un set di numeri reali per i segnali che hanno ampiezza continua.
- Il codominio è un set finito per i segnali con una ampiezza discreta.

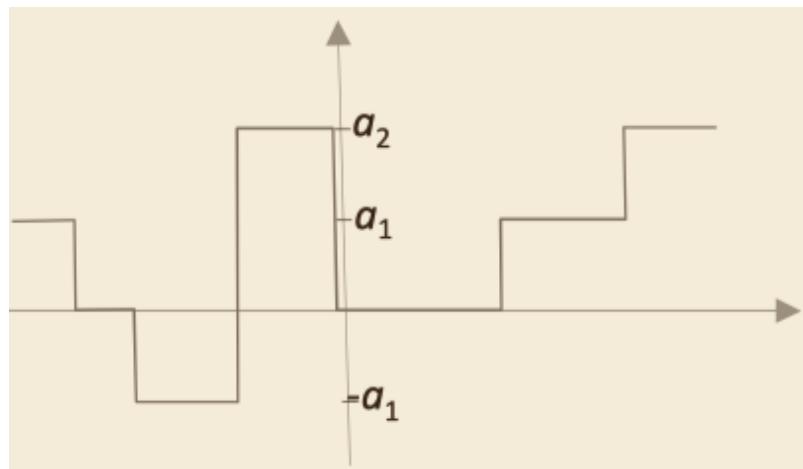
Un esempio di segnale potrebbe essere una immagine che possiamo considerare come un segnale bidimensionale, in particolare qua abbiamo che il dominio è lo spazio (x, y) e il codominio mi permette di esprimere (ad esempio) la luminosità dell'immagine o anche proprietà della luce.

Vediamo ora di classificare meglio i segnali andandoli a vedere su un grafico:

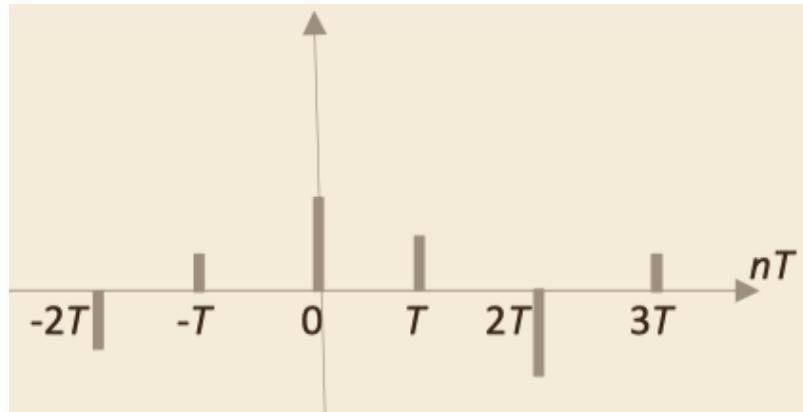
- Il primo caso che consideriamo è quello in cui abbiamo un tempo continuo e una ampiezza continua:



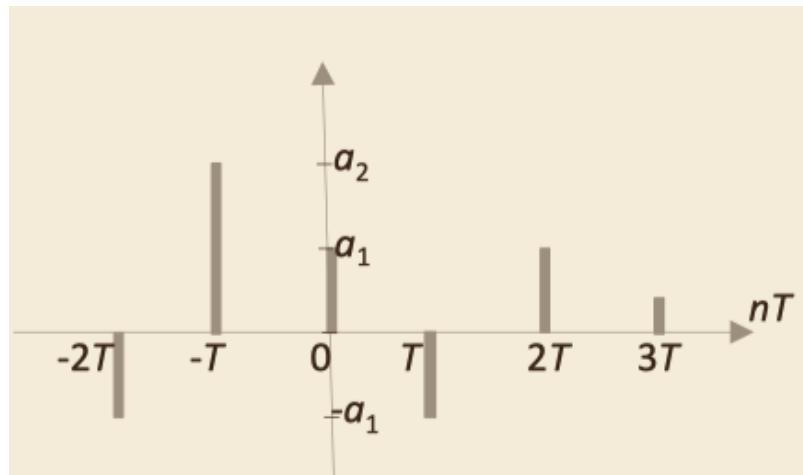
- Il secondo caso è quello in cui abbiamo un segnale continuo ma una ampiezza discreta:



- Abbiamo poi il caso in cui abbiamo un tempo discreto e una ampiezza continua:



- L'ultimo caso è quello del tempo e dell'ampiezza discreti:



6.3.1 Segnali Discreti vs Segnali Continui

Segnali discreti

Un segnale discreto è chiamato digitale se è digitale e se è definito con un set finito di simboli (dobbiamo limitare il numero possibile di simboli), per questa ragione è anche chiamato sequenza simbolica. Un testo è un esempio di sequenza simbolica perché è definito su una sequenza di

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

simboli, abbiamo infatti un alfabeto di 26 lettere e da queste 26 lettere possiamo creare una qualsiasi sequenza di simboli. Possiamo limitare ancora di più l'alfabeto di simboli, invece di 26 ne bastano anche solamente 2 che sono $B = 0, 1$, possiamo rappresentare con questi due qualsiasi sequenza di testo che potevamo rappresentare anche con i 26 caratteri, questo perchè possiamo assegnare ognuno dei 26 simboli ad una sequenza binaria. In questo modo il segnale digitale diventerà una sequenza di cifre binarie.

Utilizzando l'alfabeto binario e volendo spedire le stesse informazioni (codificando le varie lettere) possiamo calcolare il tempo necessario per la spedizione, ammettiamo di avere un canale che ci permette di spedire f simboli al secondo. Ammettiamo di voler spedire 8 caratteri (scritti con il primo alfabeto) che diventano 40 con il secondo alfabeto. Abbiamo quindi un tempo per la spedizione che è $8/f$ oppure $40/f$.

Possiamo anche considerare altri dettagli della trasmissione, ad esempio potremmo considerare che la fonte esegue il sample di f_c sample al secondo, ogni sample poi viene quantizzato in M bit, questo vuol dire che la fonte ha un throughput di $f_c * M$ bit al secondo perchè in ogni secondo abbiamo f_c nuovi sample e ogni sample viene rappresentato con M bit.

Segnali continui

Un segnale continuo è un segnale periodico

$$s(t) : R \rightarrow R$$

con periodo T se vale la seguente equazione:

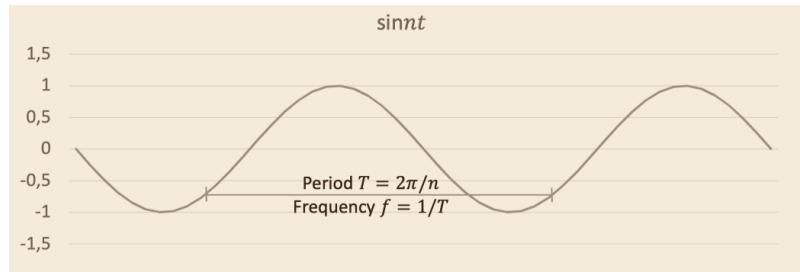
$$s(t) = s(t + T) \quad \forall t \in R$$

l'equazione mi dice che il segnale si ripete perchè il valore corrispondente alla $s(t)$ ricompare nuovamente al tempo $s(t + T)$, quindi abbiamo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

una periodicità che dipende dalla T . Un segnale non periodico viene chiamato aperiodico.

Esempi classici di segnali periodici sono ad esempio il seno e il coseno con un periodo $T = 2\pi$:



La periodicità del segnale in realtà non è solamente $T = 2\pi$ perché il segnale è periodico anche con periodo $T = \frac{2\pi}{n}$. Notare anche che la combinazione lineare di segnali che sono periodici sarà a sua volta periodica con un certo tempo T .

Abbiamo nominato prima i segnali aperiodici, consideriamo un segnale aperiodico all'interno di un intervallo $[a, b]$, il segnale s può diventare periodico se lo ripetiamo più volte dopo il punto b. Il risultato diventa in questo modo periodico. L'estensione del segnale aperiodico ad un segnale periodico s^* possiamo esprimere come:

$$s^*(t) = \sum_{n=-\infty}^{\infty} s(t - nT)$$

dove $T = b - a$.

Se il segnale aperiodico che vogliamo ripetere per farlo diventare periodico è un segnale che è definito in un intervallo $[-\infty, +\infty]$ allora possiamo considerare solamente una parte di questo intervallo e possiamo ripetere solamente questa specifica parte.

6.4 Energia di un segnale

Consideriamo un segnale $s(t)$ definito in un intervallo $[-\frac{T}{2}, \frac{T}{2}]$, l'energia del segnale viene definita come:

$$E_s(T) = \int_{-\frac{T}{2}}^{\frac{T}{2}} |s(t)|^2 dt$$

L'energia che calcoliamo in questo modo ha un significato fisico perché indica l'energia che viene dissipata dal resistore in un periodo T.

Se consideriamo il limite dell'energia che tende a infinito possiamo fare una suddivisione:

$$E_S = \lim_{T \rightarrow \infty} E_s(T) = \int_{-\infty}^{\infty} |s(t)|^2 dt$$

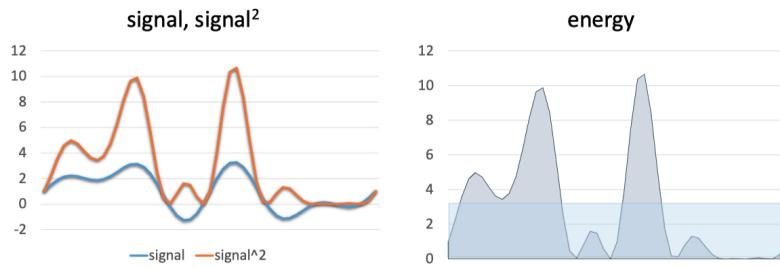
- Se abbiamo il segnale $s(t)$ e consideriamo il limite definito sopra allora, se $E_S > 0$ (quindi l'integrale deve avere valore maggiore di 0) e se $E_S < +\infty$ allora il segnale ha energia finita. Qua sono inclusi sia segnali con durata finita sia segnali con durata infinita.
- Se invece l'energia fosse infinita l'integrale andrebbe a 0, in pratica però tutti i segnali hanno energia finita.
- $s(t)$ è un segnale di potenza se l'integrale converge (>0) e il limite $P_S(T) = \lim_{t \rightarrow \infty} \frac{1}{2T} \int_{-T}^T |s(t)|^2 dt$ risulta non nullo.

Notare che:

- Se il segnale $s(t)$ è segnale di energia, $E_s \in]0, +\infty[\Rightarrow P_s = 0$, quindi non è di potenza.
- Se il segnale $s(t)$ è segnale di potenza, $P_s \in]0, +\infty[\Rightarrow E_s = 0$, quindi non è di energia.

In pratica in realtà ogni segnale che consideriamo ha energia finita perché nessun segnale dura per sempre.

Possiamo considerare il rapporto tra il segnale e il segnale al quadrato e poi la relazione che c'è con l'energia nel seguente grafico:

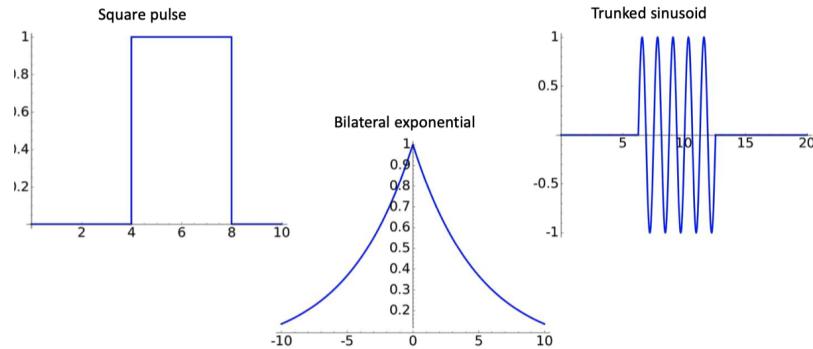


6.4.1 Impulso

Un "impulso" in un segnale è un cambiamento rapido dell'ampiezza del segnale che passa da un valore basso ad un valore alto tornando poi al valore iniziale. In questo caso possiamo considerare di nuovo una definizione simile a quella che abbiamo visto prima con l'energia, qua però quando calcoliamo il limite non abbiamo il segnale al quadrato. L'impulso viene definito come:

$$0 < \int_{-\infty}^{\infty} |s(t)| dt < +\infty$$

Possiamo vedere alcuni esempi di impulso:



6.5 Potenza di un segnale

Dato un segnale $s(t)$ possiamo considerare un intervallo $[-\frac{T}{2}, \frac{T}{2}]$ e possiamo poi calcolare la potenza media del segnale all'interno di questo intervallo:

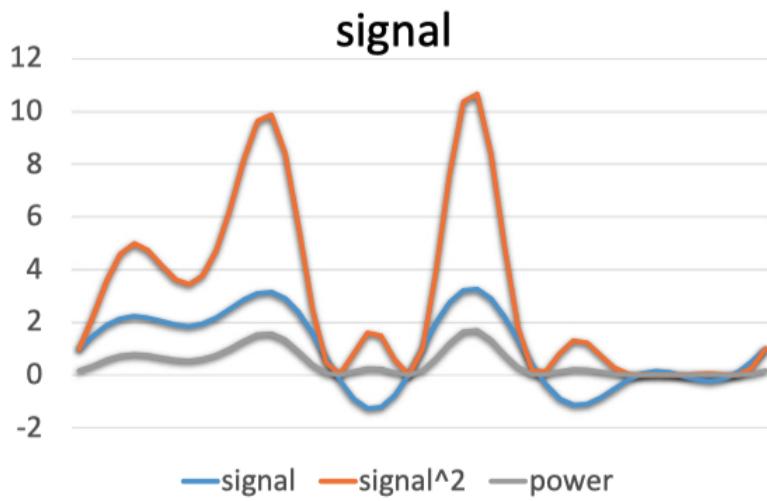
$$P_f(T) \triangleq \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} |s(t)|^2 dt$$

Possiamo definire che un segnale ha potenza finita se è definito il seguente limite:

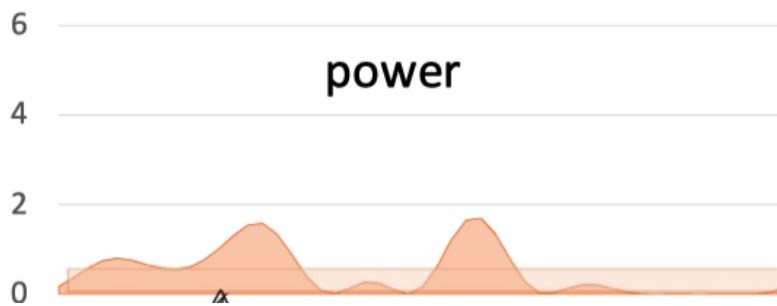
$$P_s = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} |s(t)|^2 dt > 0 \quad \text{&&} \quad P_s < +\infty$$

Se prendiamo i segnali periodici possiamo dire che questi rientrano nella classe dei segnali con potenza finita, la potenza media di questo tipo di segnali è uguale alla potenza media del loro periodo, inoltre hanno energia infinita e questo è utile per l'utilizzo con Fourier.

Abbiamo visto in precedenza la relazione tra il segnale e l'energia, ora possiamo vedere la relazione con la potenza. La potenza è qualcosa di istantaneo e possiamo calcolarne la media, nella figura qua sotto abbiamo in blu il segnale e in arancione il segnale al quadrato e poi abbiamo in grigio la potenza che è definita come $segnale^2 / tempo$.



Possiamo anche rappresentare la potenza all'interno di un suo grafico dedicato e in particolare è anche possibile calcolare l'area sotto la curva applicando l'integrale:



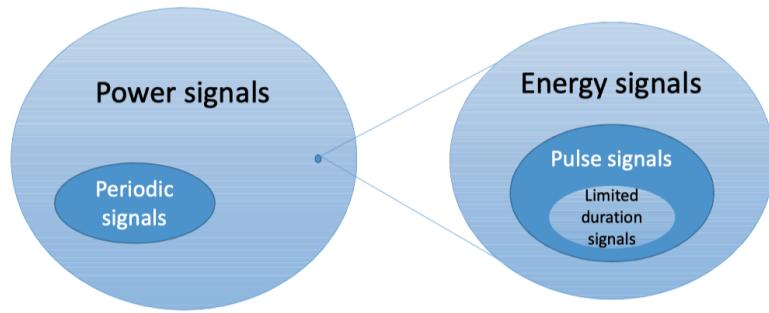
Se un segnale ha energia finita allora la sua potenza media è zero, questo mi porta a delle conseguenze e possiamo quindi distinguere due

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

classi di segnali:

- Abbiamo una prima classe di segnali che hanno energia finita e sono chiamati energy signals.
- Abbiamo una classe di segnali che hanno una potenza finita e sono chiamati power signal.

È importante notare che queste classi di segnali sono separate:



Nella figura possiamo vedere che abbiamo i periodic signal che sono una sottoclasse dei power signal e rappresentano i segnali periodici. Abbiamo poi gli energy signal che sono un sottoinsieme dei power signal e qua abbiamo che la potenza media è 0. All'interno della classe degli energy signal abbiamo poi la classe dei Pulse Signals e all'interno abbiamo i Pulse Signal che hanno una durata limitata.

Quindi riassumendo abbiamo:

- Il pulse è un energy signal.
- Un segnale con durata limitata è un pulse.
- Un segnale periodico è un power signal ma non un enery signal.
Ha energia infinita.

Vediamo alcuni esempi:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Esempio 1

Abbiamo un segnale esponenziale che definiamo in questo modo:

$$s(t) = \begin{cases} 0 & \forall t < 0 \\ ae^{-bt} & \forall t \geq 0 \end{cases}$$

questo è definito per $t \geq 0$ altrimenti è 0. Per calcolare l'energia possiamo restringere il calcolo ai valori positivi della funzione perchè per valori negativi la funzione ha valore 0:

$$E_s = \int_0^{\infty} |s(t)|^2 dt = \frac{a^2}{2b} < \infty$$

Possiamo poi calcolare anche la potenza media che è calcolata utilizzando il limite dell'integrale, in questo caso il segnale ha una potenza pari a 0:

$$P_s = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^{\frac{T}{2}} |s(t)|^2 dt = \lim_{T \rightarrow \infty} \frac{a^2}{2bT} = 0$$

Esempio 2

Consideriamo ora il seguente segnale periodico:

$$s(t) = cost \forall t$$

Questo segnale periodico ha una energia infinita e ha una potenza media finita, lo possiamo vedere se la calcoliamo:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$\begin{aligned}
 P_s &= \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} \cos^2 t dt = \frac{1}{2\pi} \int_0^{2\pi} \cos^2 t dt = \\
 &= \frac{1}{4\pi} \int_0^{2\pi} dt + \frac{1}{4\pi} \int_0^{2\pi} \cos 2t dt = \frac{1}{2}
 \end{aligned}$$

Possiamo vedere che il segnale è periodico e si ripete nel periodo $[0, 2\pi]$, se calcoliamo l'integrale di \cos^2 possiamo trasformare la formulazione e possiamo ottenere la somma di due integrali, il coseno è un power signal perchè è un segnale periodico.

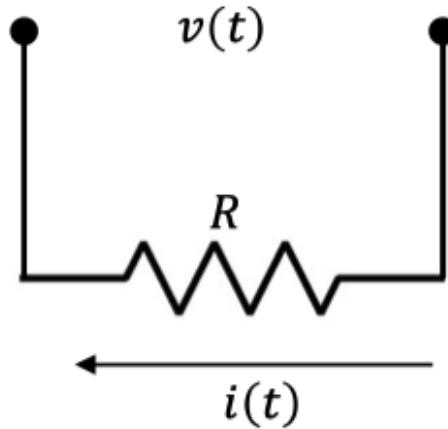
Se continuiamo a sviluppare il calcolo possiamo vedere che $t = \frac{1}{2} + \frac{1}{2}\cos 2t$ e quindi possiamo ottenere.

$$\begin{aligned}
 \frac{1}{2\pi} \int_0^{2\pi} \cos^2 t dt &= \frac{1}{2\pi} \int_0^{2\pi} \frac{1}{2} + \frac{1}{2}\cos 2t dt = \\
 &= \frac{1}{4\pi} \int_0^{2\pi} dt + \frac{1}{4\pi} \int_0^{2\pi} \cos 2t dt = \\
 &= \frac{1}{4\pi} 2\pi + \frac{1}{4\pi} [\sin 2t]_0^{2\pi} = \frac{1}{2}
 \end{aligned}$$

Eseguendo i calcoli sopra riusciamo ad arrivare a dimostrare che il segnale ha una potenza media finita.

6.6 Aspetti fisici di potenza ed energia

Consideriamo il voltaggio $v(t)$ applicato ad un resistore R.



Possiamo definire la corrente $i(t) = \frac{v(t)}{R}$, e da questa definizione possiamo ottenere la definizione di potenza istantanea assorbita dal resistore:

$$P(t) = v(t) * i(t) = \frac{v^2(t)}{R} = i^2(t) * R$$

questa potenza istantanea viene calcolata in Watt mentre l'unità di misura della $v(t)$ è il *volt*² quello che facciamo è moltiplicare il voltaggio con la $i(t)$.

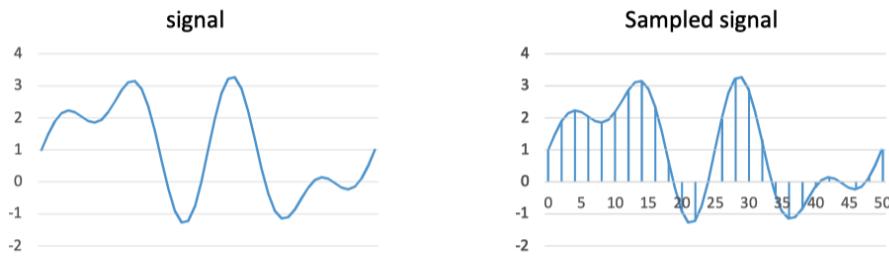
L'energia che viene assorbita dal resistore R in un periodo T è la seguente:

$$E(T) = \int_0^T P(t) dt [Joule]$$

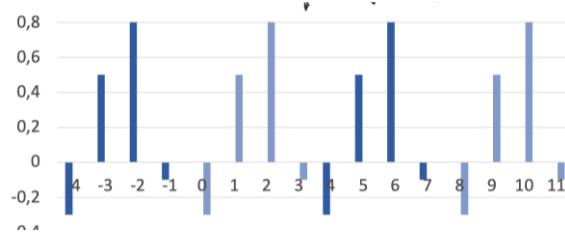
dove 1 Joule = 1 Watt * 1 Second.

6.7 Sampling di segnali discreti

Spesso possiamo ottenere i segnali discreti $g(nT)$ con un sampling uniforme da un segnale continuo $s(t)$. Supponiamo di avere una situazione del genere:



Quello che possiamo fare qua è prendere il segnale continuo e fare il sample del segnale discreto andando a fare delle letture periodiche del segnale ad intervalli regolari T . Il segnale discreto $g(t)$ poi sarà l'intersezione della linea (segnale continuo) con le varie linee. Anche per i segnali discreti si applica la stessa definizione di periodicità che si applicava ai segnali continui:



Nella immagine della figura sopra abbiamo che il segnale è dato dai punti che troviamo nell'immagine, notare che il periodo della periodicità non è uguale al periodo del segnale.

In particolare abbiamo la seguente definizione, un segnale $g(nT)$ si dice periodico con periodo NT se:

$$g(nT + NT) = g(nT) \forall n \in \mathbb{Z}$$

Notare che se abbiamo un segnale periodico continuo $s(t)$ con periodo NT e da questo facciamo il sample di un segnale discreto $g(nT)$ allora anche il segnale discreto sarà periodico con periodo NT . Possiamo dimostrarlo:

$$\begin{aligned}s(t) &= s(t + NT) \\ g((n + N)T) &= s(nT + NT) = s(nT) = g(nT)\end{aligned}$$

Questo vuol dire che $g(nT)$ è periodico perché $g(nT) = g(nT + NT)$.

Dato che un segnale discreto è diverso da 0 in un certo numero di punti, allora la sua energia e la sua potenza saranno nulli. Energia e potenza possono comunque essere calcolati considerando il segnale continuo $s'(t)$ che mantiene il valore di $g(nT)$ durante il periodo del sampling.

Come possiamo definire energia e potenza per un segnale discreto?

Noi stiamo eseguendo il sampling di un segnale, il segnale continuo è definito per ogni valore del suo dominio. Quindi possiamo definire l'energia con questa formulazione:

$$E_g \triangleq \sum_{n=-\infty}^{\infty} T |g(nT)|^2$$

All'interno della formula abbiamo la n che rappresentano i punti che sono multipli del sampling, per ogni valore di n andremo a prendere un sample del segnale e otterremo poi il segnale g . Il segnale g è però definito solamente su un dominio discreto.

La N la utilizziamo quando un segnale discreto è periodico con periodo T (anche il periodo deve essere un multiplo del periodo di sampling). N è la costante che definisce la periodicità del segnale. Se abbiamo che $g(nT + NT) = g(nT)$ allora possiamo prendere $n = 1$ e possiamo sommare la NT che è la lunghezza del periodo, ad esempio se è uguale a 4 andremo dal punto 1 al punto 5.

La potenza di un segnale discreto poi possiamo definirla in questo modo:

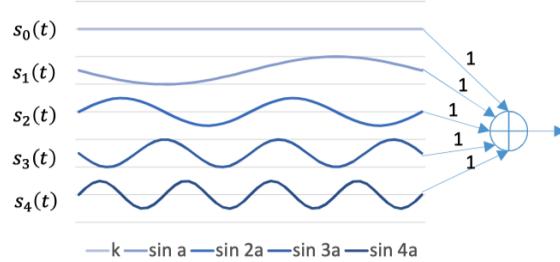
$$P_g \triangleq \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{n=-N}^N |g(nT)|^2$$

Se la media è definita allora il segnale discreto è un power signal e questo vuol dire che è anche un energy signal.

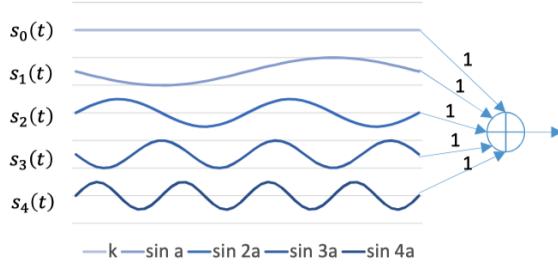
Chapter 7

Signal Theory - Serie di Fourier

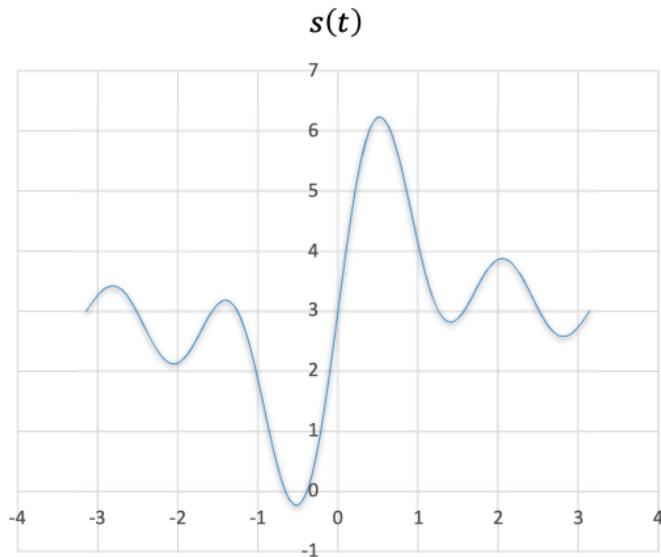
Consideriamo questi segnali:



Il primo segnale che troviamo in questo gruppo è un segnale continuo che è sempre uguale a se stesso, gli altri sono segnali periodici, in questo caso abbiamo un segnale fondamentale che è il s_1 e poi abbiamo tutti gli altri che sono ottenuti moltiplicando il primo per una costante. Nella immagine possiamo anche vedere che tutti questi segnali vengono sommati e il risultato è un segnale che sarà a sua volta periodico. In questo caso ad esempio quello che otteniamo è il seguente segnale:



Dato il segnale che abbiamo ottenuto sommando tutti gli altri, come facciamo ad invertire questo processo? Una possibile soluzione a questo problema è quella proposta da Fourier. Con invertire il processo intendiamo trovare un modo, partendo dal segnale finale che abbiamo ottenuto, di trovare i segnali che lo hanno generato facendo la somma.



7.1 Serie di Fourier

Le serie di Fourier permettono di decomporre un segnale come somma di un numero infinito di funzioni continue che oscillano a frequenze differenti, queste funzioni continue sono sinusoidi e cosinusoidi. In alcuni casi

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

questa decomposizione rappresenta un cambio di coordinate, in questo caso usando Fourier abbiamo la rappresentazione in termini di frequenza. Il set di funzioni continue che utilizziamo sono la base della decomposizione, facciamo qualcosa di molto simile alle serie di Taylor solo che nel caso di Fourier abbiamo dei segnali periodici che oscillano. Il set di funzioni continue che utilizziamo per la decomposizione devono essere ortogonali, questo set di funzioni che vengono usate per le serie di Fourier sono dette basi.

Ci sono vari metodi differenti che possono essere utilizzati per eseguire questa decomposizione del segnale, in generale ognuno di questi tipi di decomposizione viene utilizzata per evidenziare una delle feature che vogliamo evidenziare del segnale:

- Abbiamo l'analisi armonica che si basa su funzioni trigonometriche;
- Compressione: abbiamo una scelta della base che permette di avere anche un supporto temporale.
- Signal detection: qua la base viene scelta a seconda della feature del segnale che vogliamo evidenziare.

In generale quindi abbiamo varie possibilità ma ci concentreremo soprattutto sull'analisi armonica.

7.1.1 Definizione

Dato un segnale continuo $s(t)$ periodico nell'intervallo $[-\Pi, +\Pi]$ allora la serie di Fourier è definita come:

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} (a_n \cos nt + b_n \sin nt)$$

Nella definizione abbiamo la prima parte che è costante perchè è definita come la metà della a_0 . In particolare la a_0 è definita nel modo seguente:

$$a_0 = \frac{1}{\Pi} \int_{-\Pi}^{\Pi} s(t) dt$$

ed è la media del segnale.

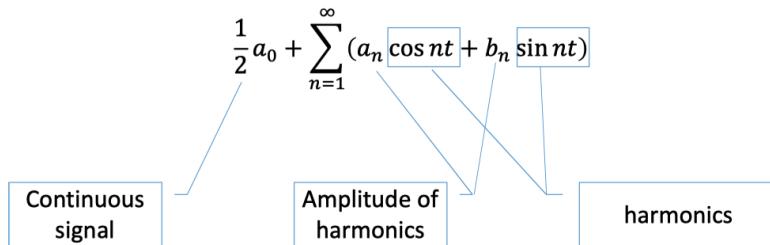
Abbiamo la seconda parte che è una sommatoria infinita di seno e coseno che prendono nt come parametro, ognuna di queste funzioni oscillano ad una frequenza differente. All'interno della sommatoria abbiamo la

$$a_n = \frac{1}{\Pi} \int_{-\Pi}^{\Pi} s(t) \cos nt dt$$

e la

$$a_n = \frac{1}{\Pi} \int_{-\Pi}^{\Pi} s(t) \sin nt dt$$

quest'ultima è la media del segnale moltiplicato per la sinusoide di nt .



7.1.2 Condizioni

Un primo problema da considerare è "Quali sono le condizioni per cui è possibile utilizzare le serie di Fourier?"

Non è facile capire quando possiamo dividere una $s(t)$ utilizzando le serie di Fourier, non conosciamo delle condizioni necessarie ma solamente delle condizioni sufficienti:

- Il segnale che prendiamo in considerazione deve essere un segnale periodico perchè in questo modo l'energia è infinita. Se avessimo un segnale non periodico non potremmo decomporlo perchè avrebbe energia finita.
- La $s(t)$ potrebbe anche essere non continua ma l'importante è che abbia delle discontinuità che sono contabili e ben definite.

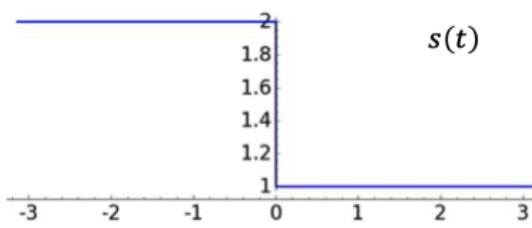
Queste due condizioni mettono forti limitazioni ma nel momento in cui sono entrambe valide allora possiamo dire che la serie di Fourier relativa alla $s(t)$ esiste e converge.

7.1.3 Esempi

Consideriamo qualche esempio, abbiamo la seguente funzione che è periodica e ha periodo 2π :

$$s(t) = \begin{cases} 2 & \text{if } -\pi < t < 0 \\ 1 & \text{if } 0 \leq t < \pi \end{cases}$$

E abbiamo la corrispondente rappresentazione:



Ora per prima cosa possiamo calcolare i coefficienti, quindi calcoliamo a_0 , a_n e b_n . a_0 viene calcolato utilizzando l'integrale del segnale, possiamo dividere l'integrale in due parti, il risultato di questa espressione è 3 in questo caso:

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} s(t) dt = \frac{1}{\pi} \left(\int_{-\pi}^0 2 dt + \int_0^{\pi} 1 dt \right) = 3$$

a_n è la media del periodo e in questo caso è 0:

$$a_n = \frac{1}{\pi} \left(\int_{-\pi}^0 2 \cos nt dt + \int_0^{\pi} 1 \cos nt dt \right) = 0$$

Abbiamo poi la b_n in cui facciamo una differenziazione in base alla n abbiamo infatti un risultato differente se la n è pari o se è dispari.

$$b_n = \frac{1}{\pi} \left(\int_{-\pi}^0 2 \sin nt dt + \int_0^{\pi} 1 \sin nt dt \right) = \begin{cases} 0 & \text{if } n \text{ is even} \\ -2/n\pi & \text{if } n \text{ is odd} \end{cases}$$

Questi tre coefficienti sono necessari per il calcolo della serie di Fourier.

Mettendo tutto insieme possiamo utilizzare la seguente formula in cui in questo caso all'interno abbiamo la b_n che è 0 se la n è pari altrimenti è $-2np$.

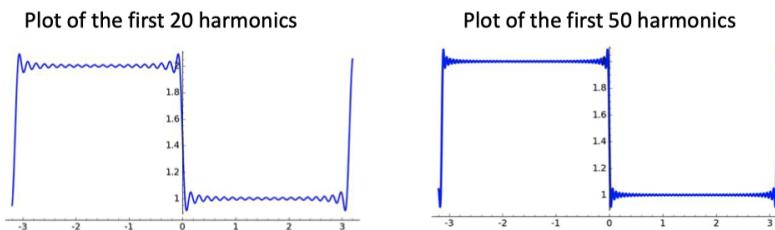
$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} (a_n \cos nt + b_n \sin nt)$$

In questo caso possiamo fare una semplificazione e indichiamo $n = 2k + 1$ per ogni $k > 0$ e otteniamo:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$s(t) = \frac{3}{2} - \sum_{k=1}^{\infty} \left(\frac{2}{(2k-1)\pi} \sin((2k-1)t) \right)$$

In particolare possiamo anche rappresentare il risultato e possiamo notare che quello che otteniamo è un segnale che non è molto simile a quello originale (segnale a sinistra), se però facciamo la somma e rappresentiamo i primi 50 valori allora abbiamo una approssimazione migliore:



7.1.4 Segnale pari o dispari

Se il segnale è pari allora il segnale è simmetrico, la definizione di segnale simmetrico è la seguente:

$$s(t) \quad e' \quad pari \quad se \quad s(t) = s(-t)$$

Se invece il segnale è dispari abbiamo la seguente definizione

$$s(t) \quad e' \quad dispari \quad se \quad s(t) = -s(-t)$$

un esempio in questo caso è il *sint* che è un segnale sinusoidale, è simmetrico rispetto al top left e al bottom right.

Ci sono delle regole che derivano dai segnali pari o dispari:

- Il prodotto di due segnali pari è a sua volta pari:

$$h(t) = s(t)f(t) = s(-t)f(-t) = h(-t)$$

- Il prodotto di due segnali dispari è pari:

$$\begin{aligned} h(t) &= s(t)f(t) = -s(-t)(-f(-t)) \\ &= s(-t)f(-t) = h(-t) \end{aligned}$$

- Il prodotto di un segnale pari e di un segnale dispari è dispari:

$$\begin{aligned} h(t) &= s(t)f(t) = s(-t)(-f(-t)) \\ &= -s(-t)f(-t) = -h(-t) \end{aligned}$$

Serie di Fourier per segnali pari

Consideriamo un segnale $s(t)$ che è pari, allora vuol dire che dato che $\sin nt$ è dispari avremo: $s(t)\sin nt$ dispari.

$$s(t)\sin nt = -s(-t)\sin(-nt) \Rightarrow \int_{-\Pi}^0 s(t)\sin nt dt = -\int_0^{-\Pi} s(t)\sin nt dt$$

La conseguenza è che se calcoliamo b_n utilizziamo l'integrale da $-\Pi$ a Π e dato che le due funzioni sono opposte abbiamo che i due termini hanno somma 0 e quindi $b_n = 0$.

$$\begin{aligned} b_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} s(t) \sin nt dt = \\ &\frac{1}{\pi} \left(\int_{-\pi}^0 s(t) \sin nt dt + \int_0^{\pi} s(t) \sin nt dt \right) = 0 \end{aligned}$$

La serie di Fourier per segnali pari non contiene il sin perchè abbiamo la seguente formula:

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos nt$$

all'interno della formula abbiamo i seguenti coefficienti:

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} s(t) dt = \frac{2}{\pi} \int_0^{\pi} s(t) dt;$$

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} s(t) \cos nt dt = \frac{2}{\pi} \int_0^{\pi} s(t) \cos nt dt$$

Vediamo un esempio:

$$s(t) = \begin{cases} 1 & \text{if } -\pi < t < -1 \\ 0 & \text{if } -1 < t < 1 \\ 1 & \text{if } 1 < t < \pi \end{cases} \quad \text{Period} = 2\pi$$

Compute the Fourier Series.

$s(t)$ is even if $s(t) = s(-t)$

is odd if $s(t) = -s(-t)$

In this case we have

$$s\left(\frac{\pi}{2}\right) = 1 \quad s\left(-\frac{\pi}{2}\right) = 1$$

$\Rightarrow s(t)$ is even.

$b_n = 0$ and we can write the Fourier series:

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos nt$$

$$a_0 = \frac{2}{\pi} \int_0^{\pi} s(t) dt = \frac{2}{\pi} \int_1^{\pi} s(t) dt = \frac{2}{\pi} \int_1^{\pi} 1 dt$$

$$= \frac{2}{\pi} (\pi - 1) = \frac{2\pi - 2}{\pi}$$

$$a_n = \frac{2}{\pi} \int_0^{\pi} s(t) \cos nt dt = \frac{2}{\pi} \int_1^{\pi} 1 \cdot \cos nt dt$$

$$\frac{2}{\pi} \left(\frac{\sin \pi n}{n} - \frac{\sin n}{n} \right) =$$

$$\frac{2 \sin \pi n - 2 \sin n}{n\pi}$$

Se invece il segnale è dispari abbiamo la Serie di Fourier che non contiene il coseno e la formula possiamo scriverla come:

$$s(t) = \sum_{n=1}^{\infty} b_n \sin nt$$

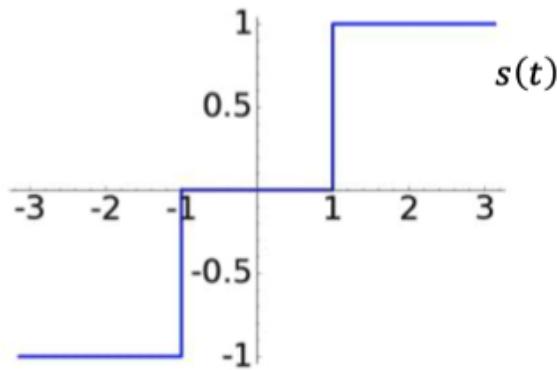
I coefficienti all'interno della formula sono $a_0 = 0$, $a_n = 0$ e poi abbiamo la definizione della b_n :

$$b_n = \frac{2}{\pi} \int_0^\pi s(t) \sin nt dt$$

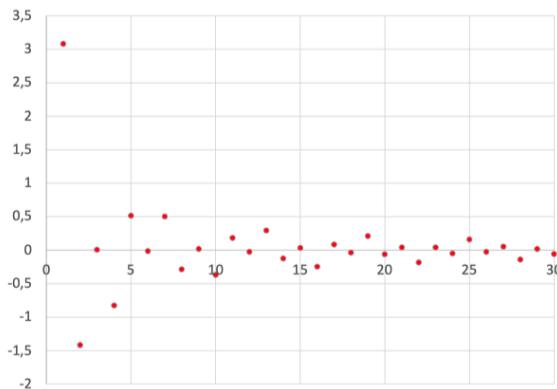
Consideriamo ora il seguente esempio:

$$\begin{aligned} s(t) &= \begin{cases} -1 & \text{if } -\pi < t < -1 \\ 0 & \text{if } -1 < t < 1 \\ 1 & \text{if } 1 < t < \pi \end{cases} \\ a_0 &= 0 \quad a_n = 0 \\ b_n &= \frac{2}{\pi} \int_0^\pi s(t) \sin nt dt = \frac{2}{\pi} \int_1^\pi \sin nt dt \\ &= \frac{2(\cos n - \cos \pi n)}{n} \\ s(t) &= \sum_{n=1}^{\infty} \left(\frac{2(\cos n - \cos \pi n)}{n} \sin nt \right) \end{aligned}$$

La funzione che utilizziamo nell'esempio la possiamo rappresentare nel grafico come:



possiamo utilizzare la serie di Fourier per poterla rappresentare all'interno del grafo e avremmo un punto per ognuno degli "harmonic" nel grafo originale. Quello che vediamo nel diagramma è lo stesso che vediamo nell'altro grafico:



7.1.5 Segnali di Fourier con periodi arbitrari

Supponiamo di avere un segnale che è periodico in un periodo arbitrario $[-\frac{T}{2}, \frac{T}{2}]$ possiamo applicare un cambio di variabile:

$$x = \frac{2\pi t}{T}$$

Il segnale considerando questo cambio di variabile diventa periodico nell'intervallo $[-\Pi, \Pi]$. Possiamo fare questo cambio di variabile per usare lo stesso metodo in vari periodo differenti ed arbitrari.

La serie di Fourier associata alla $s(t)$ quindi diventa:

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{2\pi n t}{T} + b_n \sin \frac{2\pi n t}{T} \right)$$

Dove i coefficienti sono definiti nel modo seguente:

$$\begin{aligned} a_0 &= \frac{2}{T} \int_{-T/2}^{T/2} s(t) dt \\ a_n &= \frac{2}{T} \int_{-T/2}^{T/2} s(t) \cos \frac{2\pi n t}{T} dt \\ b_n &= \frac{2}{T} \int_{-T/2}^{T/2} s(t) \sin \frac{2\pi n t}{T} dt \end{aligned}$$

Quello che cambia nel calcolo dei coefficienti è solamente l'intervallo dove calcoliamo l'integrale e la frequenza dell'harmonic. Possiamo anche fare delle riduzioni nel caso in cui il segnale sia pari o dispari:

- Se $s(t)$ è pari allora possiamo non considerare la parte con il sin all'interno della formula e quindi avremmo:

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos \frac{2\pi n t}{T}$$

E le seguenti definizioni per i coefficienti, con la $b_n = 0$:

$$a_n = \frac{4}{T} \int_0^{T/2} s(t) \cos \frac{2\pi n t}{T} dt$$

$$a_0 = \frac{4}{T} \int_0^{T/2} s(t) dt$$

- Se invece la $s(t)$ è dispari, definiamo la serie di Fourier in questo modo:

$$s(t) = \sum_{n=1}^{\infty} b_n \sin \frac{2\pi n t}{T}$$

E all'interno abbiamo $a_0 = 0$, $a_n = 0$ e invece la b_n ha un valore differente definito come:

$$b_n = \frac{4}{T} \int_0^{T/2} s(t) \sin \frac{2\pi n t}{T} dt$$

Vediamo un esempio:

$$S(t) = \begin{cases} 1 & \text{if } |t| < 1 \\ 0 & \text{if } |t| \leq 4 \end{cases}$$

La funzione è periodica in un periodo = 8
La funzione è pari, quindi abbiamo:

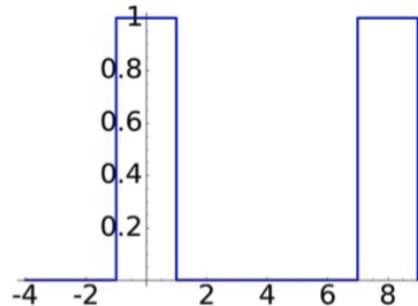
$$\boxed{a_n} = \frac{1}{2} \int_0^4 S(t) \cos \frac{\pi n t}{4} dt = \frac{1}{2} \int_0^4 \cos \frac{\pi n t}{4} dt$$

$$= \underline{2 \sin \left(\frac{1}{4} \pi n \right)}$$

$$a_0 = \frac{1}{2} \int_0^4 S(t) dt = \frac{1}{2} \int_0^4 1 dt = \frac{1}{2}$$

$$b_n = 0$$

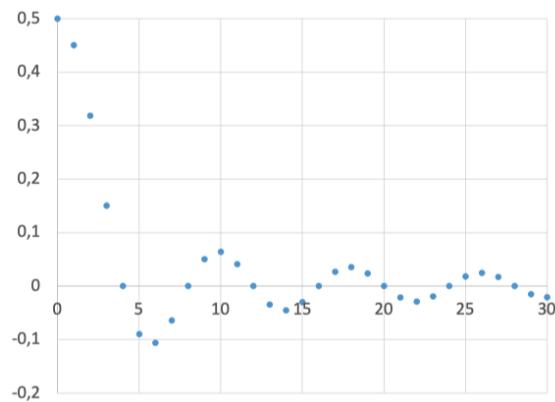
Alla equazione originale è associato il seguente grafico:



Dati i calcoli possiamo procedere a scrivere la serie di Fourier come:

$$s(t) = \frac{1}{4} + \sum_{n=1}^{\infty} \left(\frac{1}{\pi n} 2 \sin \left(\frac{1}{4} \pi n \right) \cos \frac{\pi n t}{4} \right)$$

Possiamo anche rappresentare i segnale tramite l'"harmonic", la forma del segnale in questo dominio è discreta:



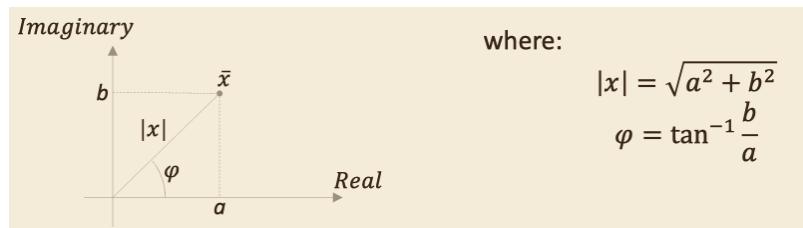
Chapter 8

Signal Theory - Fourier Transform

8.1 Introduzione

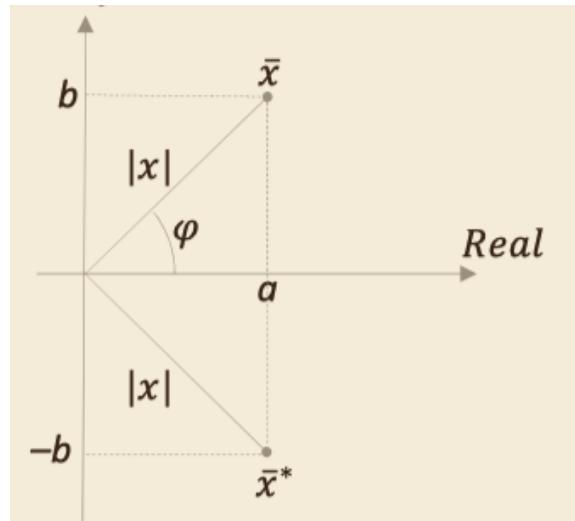
8.1.1 Numeri complessi

Un numero complesso $x = a + jb$ è composto da una parte reale a e da una parte immaginaria jb . La parte immaginaria j è definita come $j = \sqrt{-1}$. Possiamo rappresentare il numero complesso all'interno del piano:



Con i numeri complessi valgono le stesse operazioni che abbiamo con i numeri classici a cui siamo abituati, possiamo anche definire il coniugato

del numero reale come: $x^* = a - jb$. Anche il coniugato lo possiamo rappresentare nel piano:



8.1.2 Esponente di Eulero

L'esponente di Eulero è un numero complesso definito come:

$$e^{\pm j\varphi} = \cos \varphi \pm j \sin \varphi$$

Tenendo conto della definizione sopra e anche del fatto che all'interno della formula abbiamo seno e coseno possiamo definire seno e coseno in termini del coefficiente di Eulero:

$$\begin{aligned}\cos \varphi &= \frac{e^{j\varphi} + e^{-j\varphi}}{2} \\ \sin \varphi &= \frac{e^{j\varphi} - e^{-j\varphi}}{2j}\end{aligned}$$

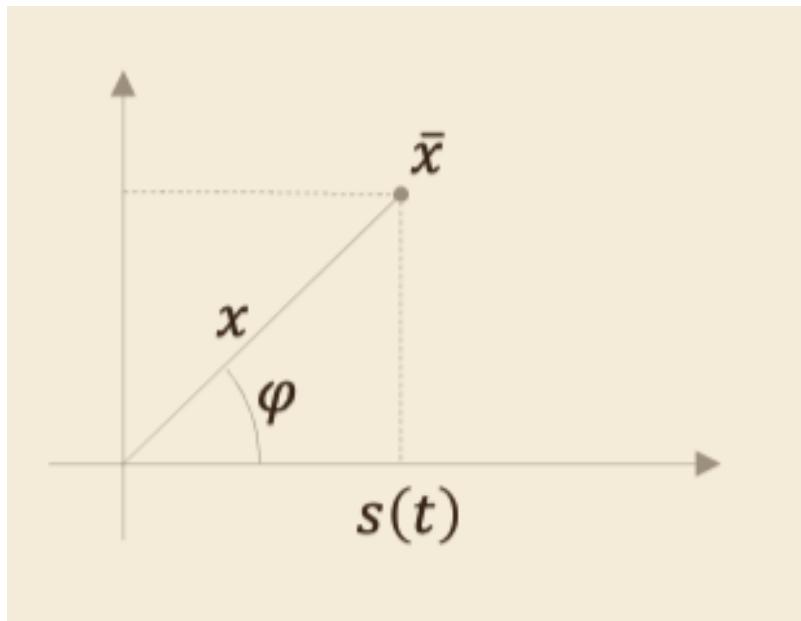
Data la definizione che abbiamo dato sopra di coefficiente di Eulero, possiamo riscrivere il numero complesso x come $x = |x|e^{j\Phi}$

Questo perchè $x = a+jb = |x|\cos\Phi + j|x|\sin\Phi = |x|e^{j\Phi}$. Il coniugato del numero complesso x diventa quindi $x^* = |x|e^{-j\Phi}$

8.1.3 Fasore

Un segnale $s(t) = x\cos(2\Pi f_0 t + \Phi)$ possiamo rappresentarlo completamente tramite un numero complesso che è chiamato fasore:

$$\bar{A} = |x|e^{j\Phi}$$



Dato il fasore è possibile tornare al segnale originale:

By the relationship $s(t) = \Re(\bar{x} \cdot e^{j2\pi f_0 t})$

$$\begin{aligned}\Re(\bar{x} \cdot e^{j2\pi f_0 t}) &= \Re(|x| \cdot e^{j(2\pi f_0 t + \varphi)}) = \\ &= \Re(|x| \cos(2\pi f_0 t + \varphi) + j|x| \sin(2\pi f_0 t + \varphi)) = \\ &= |x| \cos(2\pi f_0 t + \varphi) = s(t)\end{aligned}$$

Il segnale originale viene decomposto in sinusoidi e cosinusoidi rappresentate da fasori, per ottenere il segnale origiale sommo il fasore con il suo coniugato invece di prendere la parte reale del numero complesso che rappresenta il segnale.

8.2 Serie di Fourier con base esponenziale

La serie di Fourier viene espressa nel dominio dei numeri complessi (solitamente):

$$s(t) = R - > C$$

La base che viene utilizzata è il set di funzioni:

$$e^{j2\pi nFt}$$

Queste funzioni sono una combinazione di coseno e seno:

$$e^{j2\pi nFt} = \cos(2\pi nFt) + j\sin(2\pi nFt)$$

e non siamo lontani dalle serie di Fourier che abbiamo definito con i numeri reali.

Quali sono le proprietà che abbiamo con queste funzioni:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Ogni funzione $e^{j2\pi nFt}$ è periodica con periodo $\frac{1}{nF}$ quindi:

$$e^{j2\pi nF(t+\frac{1}{nF})} = e^{j2\pi nFt} + e^{j2\pi} = e^{j2\pi nFt}$$

- Le funzioni $e^{j2\pi nFt}$ sono ortogonali in $[0, T]$ dove $T = \frac{1}{F}$:

$$\frac{1}{T} \int_0^T e^{j2\pi nt/T} e^{-j2\pi mt/T} dt = \begin{cases} 1 & \forall n = m \\ 0 & \forall n \neq m \end{cases}$$

Supponiamo di avere un segnale periodico $s(t) = s(t + T)$, quindi abbiamo che il periodo in questo caso è T e la frequenza è $\frac{1}{T}$ e viene espressa in Hertz. Consideriamo la serie:

$$S_n = \frac{1}{T} \int_0^T s(t) e^{-j2\pi nFt} dt$$

Possiamo rappresentare il segnale $s(t)$ utilizzando la seguente combinazione lineare:

$$s(t) = \sum_{n=-\infty}^{\infty} S_n e^{j2\pi nFt}$$

All'interno della sommatoria abbiamo quello che viene chiamato componente armonico di $s(t)$, ognuno di questi ha frequenza nF . Notare che conoscere la serie S_n equivale a conoscere anche il segnale $s(t)$, questo segnale possiamo vederlo come una composizione di un numero infinito di segnali periodici. Notare anche che il coefficiente $S_0 = \frac{1}{T} \int_0^T s(t) dt$ viene chiamato componente continuo.

8.2.1 Interpretazione geometrica delle serie di Fourier

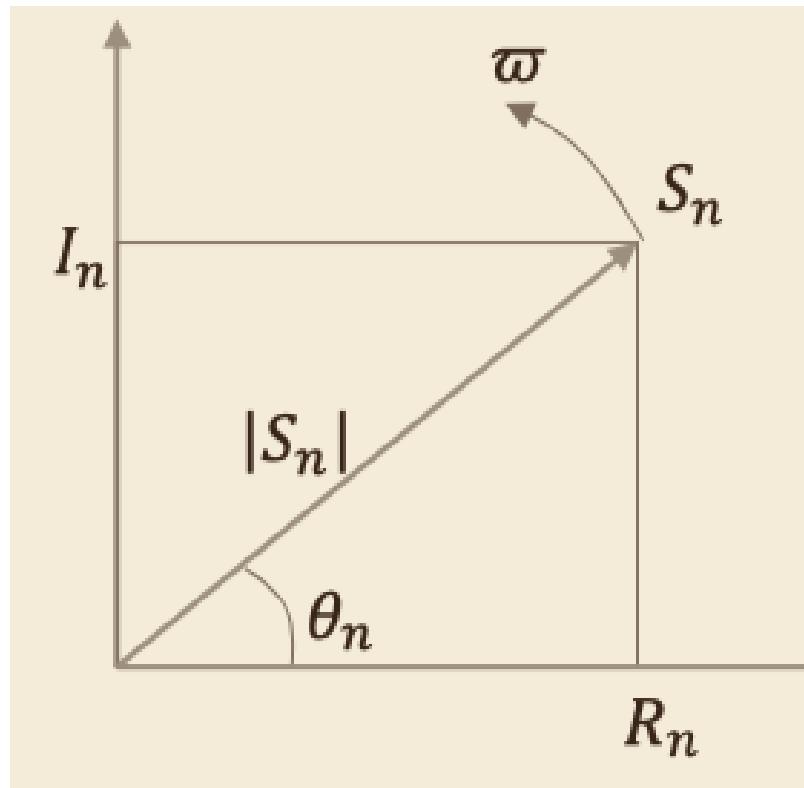
$$s(t) = \sum_{n=-\infty}^{\infty} S_n (\cos(2\pi n F t)) + j \sin(2\pi n F t)$$

All'interno della formula della serie abbiamo la parte $(\cos(2\pi n F t)) + j \sin(2\pi n F t)$ che mi rappresenta l'armonica mentre invece la S_n indica l'ampiezza dell'armonica.

In particolare la $s(t)$ la possiamo riscrivere e riscrivendola teniamo anche conto della definizione di S_n .

$$s(t) = \sum_{n=-\infty}^{\infty} S_n e^{j2\pi n F t}; \quad S_n = \frac{1}{T} \int_0^T s(t) e^{-j2\pi n F t} dt$$

S_n è un numero complesso che possiamo rappresentare come il fasore $S_n = |S_n| e^{j\Phi_n}$.



8.3 Serie di Fourier e Teorema di Dirichlet

Dirichlet fornisce alcune condizioni per cui è possibile scrivere una serie come una serie di Fourier:

- La serie di Fourier mi fornisce lo stesso valore di $s(t)$ quando la $s(t)$ è continua.
- Possiamo calcolare la serie di Fourier per ogni segnale finito in un certo intervallo temporale T , facendo questo calcolo però noi il segnale lo rendiamo periodico.

Per quanto riguarda la convergenza, il teorema di Dirichlet fornisce

delle condizioni sufficienti, non si conoscono invece le condizioni necessarie:

- Il segnale $s(t)$ deve essere periodico
- Il segnale $s(t)$ è più o meno continuo
- I punti di discontinuità hanno sia il limite a destra che a sinistra.
- La derivata del segnale $s(t)$ è limitata nell'intervallo $[0, T]$

Tutte queste condizioni fanno sì che la serie di Fourier converge a $s(t)$ in ogni punto di continuità mentre nei punti di discontinuità converge alla media tra il limite sinistro e il destro, questo preserva l'energia del segnale.

8.4 Serie di Fourier con segnali reali

Le serie di Fourier sono state sviluppate per i numeri complessi ma funzionano anche con il codominio con numeri reali. Se prendiamo la serie di Fourier dei segnali reali i coefficienti negativi sono i coniugati dei coefficienti positivi. In questo modo otteniamo un'altra rappresentazione del segnale:

$$s(t) = \sum_{n=-\infty}^{\infty} S_n e^{j2\pi n F t}$$

con il suo coefficiente:

$$S_n = \frac{1}{T} \int_0^T s(t) e^{-j2\pi n F t} dt$$

In questo caso quindi succede che $S_{-n} = S_n^*$.

I termini della serie per $n < 0$ sono dipendenti dai termini per cui $n > 0$ e possiamo riscrivere la serie di Fourier come:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$s(t) = R_0 + 2 \sum_{n=1}^{\infty} \left(R_n \cos \frac{2\pi n t}{T} - I_n \sin \frac{2\pi n t}{T} \right)$$

All'interno della formula abbiamo solamente il coseno se $s(t)$ è pari altrimenti se è dispari abbiamo solamente il seno all'interno della formula.

Una forma alternativa di questa formula è la seguente:

$$s(t) = S_0 + \sum_{n=1}^{\infty} [2|S_n| \cos(2\pi n F t + \theta_n)]$$

All'interno della formula abbiamo:

- $2|S_n|$ rappresenta l'ampiezza dell'armonica.
- nF rappresenta la frequenza dell'armonica.
- Θ_n rappresenta la fase dell'armonica.

8.5 Teorema di Parseval

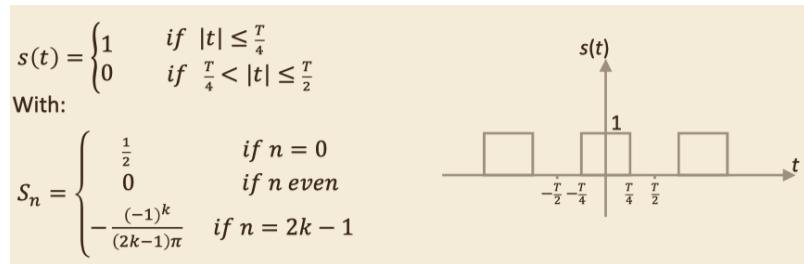
Il teorema di Parseval è la conseguenza della convergenza della serie di Fourier al segnale corrispondente. Dato un segnale periodico $s(t)$ con potenza media P_s abbiamo:

$$P_s = \frac{1}{T} \int_0^T |S(t)|^2 dt = \sum_{n=-\infty}^{\infty} |S_n|^2$$

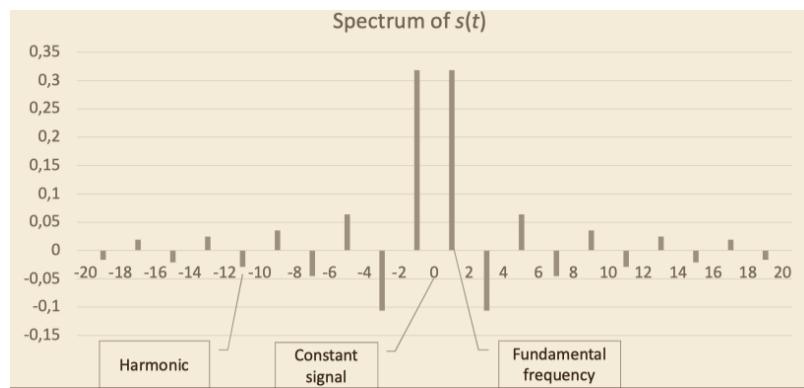
La potenza del segnale è uguale alla somma della potenza dei suoi componenti periodici. Qui è vero che la potenza della somma è la somma delle potenze perchè le funzioni nella base di Fourier sono ortogonali.

8.6 Spettro del segnale

La serie di Fourier preserva la potenza del segnale. Se abbiamo il nostro segnale $s(t)$ tramite l'utilizzo di Fourier possiamo calcolare i coefficienti della serie S_n , da questi coefficienti possiamo poi ricostruire $s(t)$. La sequenza dei coefficienti ordinati S_n da $-\infty$ a $+\infty$ è lo spettro di $s(t)$, lo spettro è un segnale discreto.



Nella formula sopra possiamo considerare la sequenza dei coefficienti ordinati S_n che è chiamata spettro del segnale $s(t)$. Possiamo rappresentare anche lo spettro del segnale nel piano, nel grafico vediamo lo stesso segnale $s(t)$ ma definito sulle frequenze.

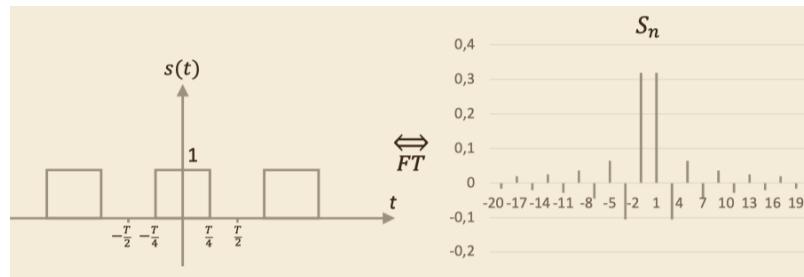


8.7 Trasformata di Fourier

Il passaggio dall'avere il segnale continuo $s(t)$ allo spettro del segnale (ovvero al suo segnale discreto S_n) è chiamato Trasformata di Fourier.

$$\begin{aligned}
 s(t) &\xleftrightarrow{\text{FT}} S_n \\
 S_n &= \mathcal{F}(s(t)) \quad \text{Transform} \\
 s(t) &= \mathcal{F}^{-1}(S_n) \quad \text{Inverse transform}
 \end{aligned}$$

Con questa trasformazione passiamo da $s(t)$ in cui siamo nel dominio del tempo (t esprime il tempo) a S_n in cui invece siamo nel dominio discreto delle frequenze, è un frequency domain discreto perchè abbiamo un set discreto di frequenze.

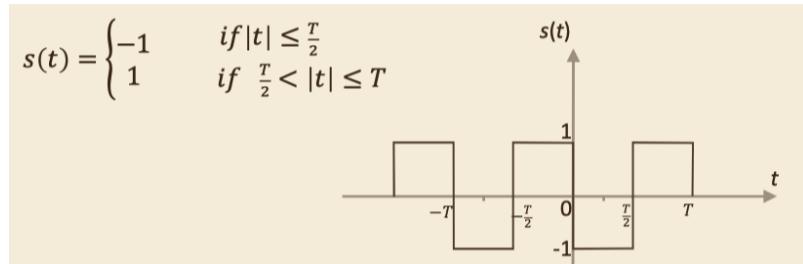


I coefficienti di S_n sono solitamente numeri complessi anche se $s(t)$ è un reale, da questo deriva anche il fatto che non sempre è possibile rappresentare lo spettro con un diagramma 2D. Lo spettro del segnale S_n è anche rappresentabile con un'altra forma:

$$S_n = |S_n|e^{j\Theta_n}$$

all'interno della formula abbiamo $|S_n|$ che è l'ampiezza dell'armonica e la $j\Theta_n$ che è la fase dell'armonica.

Vediamo un esempio:



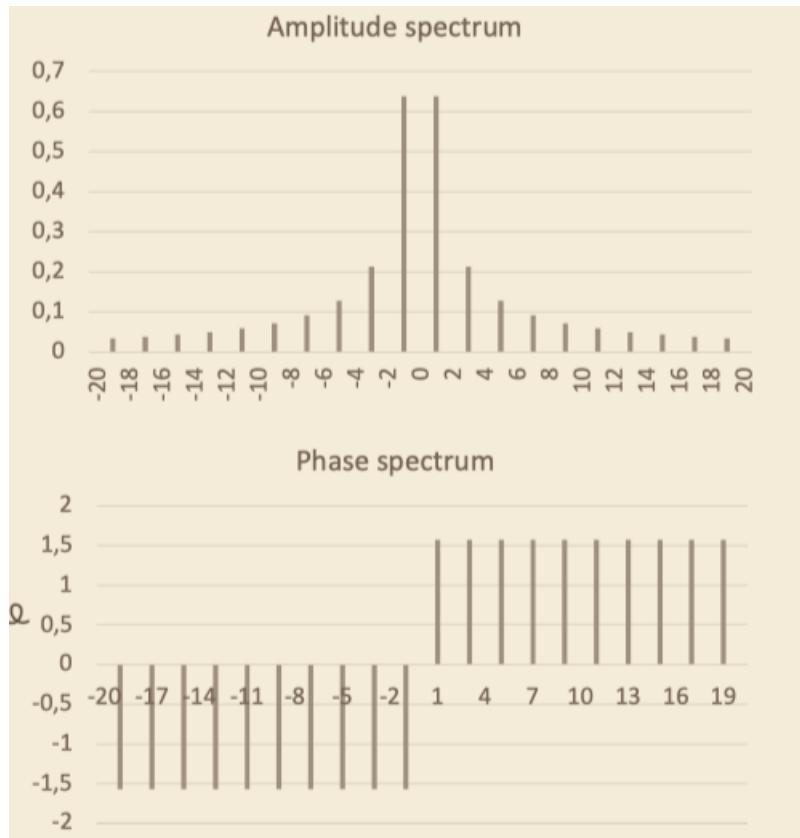
In questo caso abbiamo un segnale periodico e possiamo calcolare i coefficienti che sono 0 quando la n è pari e $j\frac{2}{n\pi}$ quando n è dispari.

$$\begin{aligned} S_n &= \frac{1}{T} \int_0^T s(t) e^{-j2\pi n t/T} dt = \frac{1}{T} \int_{-\frac{T}{2}}^0 e^{-j2\pi n t/T} dt - \frac{1}{T} \int_0^{\frac{T}{2}} e^{-j2\pi n t/T} dt = \\ &= \begin{cases} 0 & \text{if } n = 0 \\ j \frac{1 - \cos n\pi}{n\pi} & \text{if } n \neq 0 \end{cases} = \begin{cases} 0 & \text{if } n \text{ even} \\ j \frac{2}{n\pi} & \text{if } n \text{ odd} \end{cases} \end{aligned}$$

Quindi se n è dispari possiamo riscrivere la S_n in questo modo:

$$\begin{aligned} S_n &= |S_n| e^{j\theta_n} = \left| \frac{2}{n\pi} \right| (\cos \theta_n + j \sin \theta_n) \\ \text{if } n > 0, n \text{ odd: } \frac{2}{n\pi} \sin \theta_n &= \frac{2}{n\pi} \Rightarrow \theta_n = \sin^{-1} 1 = \frac{\pi}{2} \\ \text{if } n < 0, n \text{ odd: } -\frac{2}{n\pi} \sin \theta_n &= \frac{2}{n\pi} \Rightarrow \theta_n = \sin^{-1} -1 = -\frac{\pi}{2} \\ \text{Then:} \quad S_n &= \begin{cases} 0 & \text{if } n \text{ even} \\ \frac{2}{n\pi} e^{j\frac{\pi}{2}} & \text{if } n > 0, n \text{ odd} \\ -\frac{2}{n\pi} e^{-j\frac{\pi}{2}} & \text{if } n < 0, n \text{ odd} \end{cases} \end{aligned}$$

Graficamente possiamo rappresentarlo in questo modo:



La conseguenza è che possiamo definire l'ampiezza e la fase nel modo seguente:

$$|S_n| = \begin{cases} 0 & \text{if } n \text{ even} \\ \frac{2}{n\pi} & \text{if } n > 0, n \text{ odd} \\ -\frac{2}{n\pi} & \text{if } n < 0, n \text{ odd} \end{cases}$$

$$\theta_n = \begin{cases} 0 & \text{if } n \text{ even} \\ \pi/2 & \text{if } n > 0, n \text{ odd} \\ -\pi/2 & \text{if } n < 0, n \text{ odd} \end{cases}$$

8.8 Trasformata di Fourier per segnali non periodici

La trasformata di Fourier sarebbe applicabile solamente ai segnali periodici, in realtà tutti i segnali sono finiti nel tempo. Possiamo applicare la trasformata di Fourier se il segnale non è periodico considerando la periodicità come se tendesse ad infinito. Si tratta di una astrazione, se consideriamo il segnale con il periodo infinito allora abbiamo di nuovo un segnale periodico.

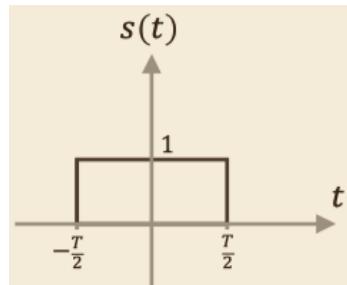
Un segnale non periodico $s(t)$ possiamo esprimere come:

$$s(t) = \int_{-\infty}^{\infty} S(f) e^{j2\pi f t} df$$

dove la f è la frequenza continua che sta nel range $[-\infty, \infty]$ e la $S(f)$ è l'ampiezza della frequenza f che calcoliamo come:

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft} dt$$

Questo spettro $S(f)$ è un segnale continuo, la trasformazione da $s(t)$ a $S(f)$ è una trasformazione di Fourier continua ed è una generalizzazione della trasformazione di Fourier. Per vedere come funziona la trasformata di Fourier continua consideriamo il pulse signal:



Definito come:

$$s(t) = \begin{cases} 1 & \text{if } |t| \leq \frac{T}{2} \\ 0 & \text{otherwise} \end{cases}$$

I coefficienti della serie di Fourier sono:

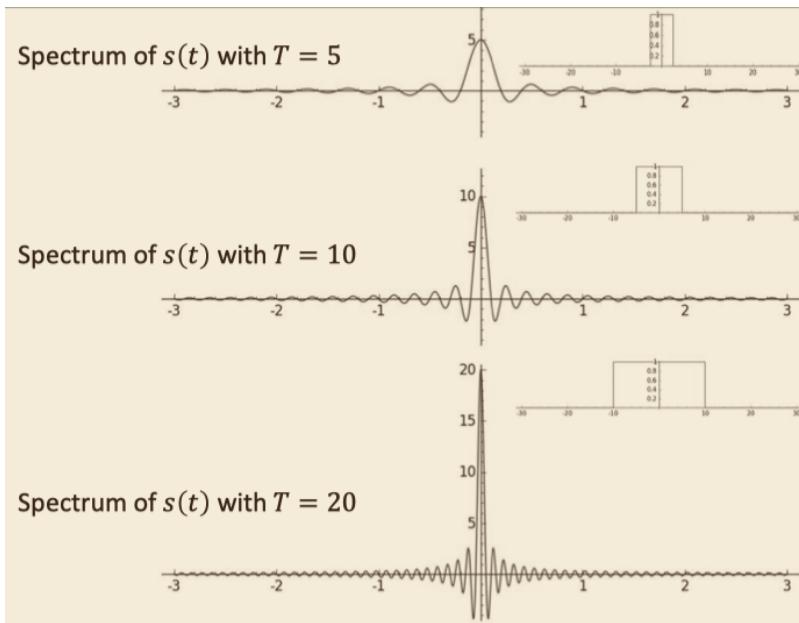
$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi f t} dt = \int_{-T/2}^{T/2} e^{-j2\pi f t} dt = \left[\frac{e^{-j2\pi f t}}{-j2\pi f} \right]_{-T/2}^{T/2}$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Ovvero possiamo riscriverlo come:

$$S(f) = \frac{e^{\frac{j2\pi fT}{2}} - e^{-\frac{j2\pi fT}{2}}}{j2\pi f} = \frac{\sin(\pi fT)}{\pi f}$$

Per ogni frequenza f possiamo rappresentare il segnale solamente utilizzando lo spettro.



Ad esempio nella immagine sopra consideriamo tre valori differenti per la T , ovvero 5, 10 e 20. In tutti e tre i casi otteniamo lo spettro del segnale. Qua non rappresentiamo la fase perché la parte immaginaria della $s(f)$ scompare e alla fine otterremo un coefficiente che è formato solamente dalla parte reale e non da quella immaginaria.

8.9 Energia di un segnale continuo

Anche nel caso della trasformazione di Fourier continua vale un risultato simile a quello del teorema di Parseval, dato un segnale $s(t)$ la sua energia E_s la possiamo ottenere tramite la sua trasformazione:

$$E_s = \int_{-\infty}^{\infty} |S(f)|^2 df$$

Questa formula mi indica che l'energia di un segnale è uguale alla somma delle energie dei suoi componenti e questo vale perchè le funzioni che formano la base di Fourier sono ortogonali.

Da qua possiamo calcolare che l'energia di un componente a frequenza f è calcolata come:

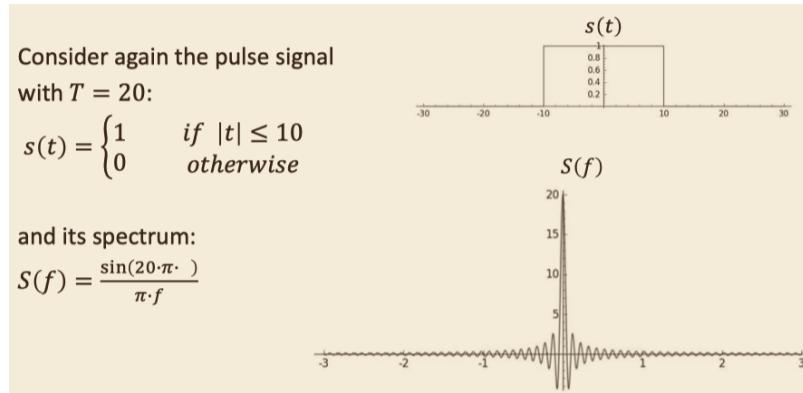
$$E_s(f) = |S(f)|^2$$

L'energia della banda di frequenza tra f_1 e f_2 è calcolata come:

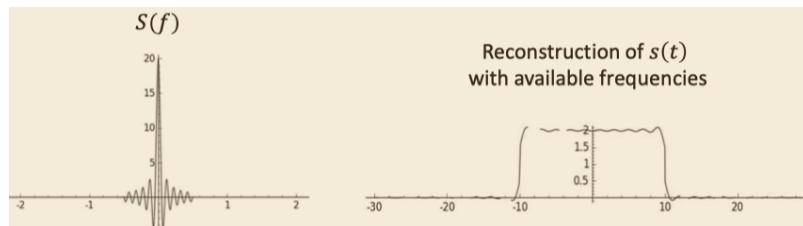
$$\int_{f_1}^{f_2} |S(f)|^2 df$$

In generale lo spettro del segnale copre da $-\infty$ a $+\infty$, se il segnale è periodico lo spettro è discreto e contiene solamente l'armonica della frequenza principale. Possiamo filtrare alcune delle frequenze dello spettro ottenendo un segnale limitato nella larghezza di banda escludendo le frequenze alte (low pass filter) o le basse (high pass filter) o entrambi (bandpass filter).

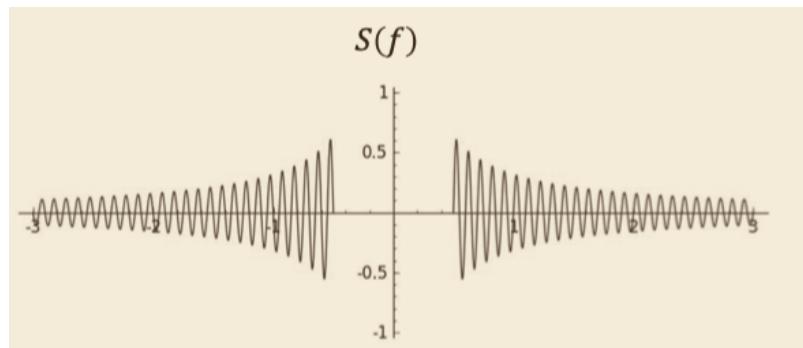
Consideriamo un altro esempio, abbiamo il pulse signal e il suo spettro:



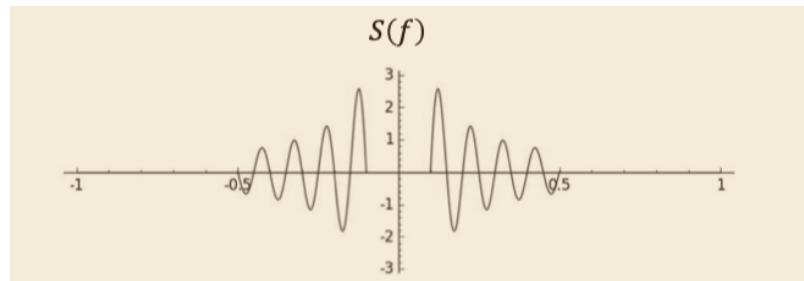
se applichiamo un low pass filter per le frequenze sotto a 0.5 otteniamo solamente il componente che vediamo nella immagine in basso a sinistra. A destra invece quello che otteniamo con la ricostruzione della $s(t)$ utilizzando le frequenze che sono disponibili otteniamo una approssimazione del segnale originale.



Se invece applichiamo un filtro high pass invece manteniamo solamente valori positivi e negativi:



Se applichiamo il band pass filter per le frequenze $[0.1, 0.5]$ invece otteniamo:



In pratica qua stiamo considerando solamente una parte dei filtri, con il band pass in particolare consideriamo solamente il segnale che è sotto o sopra un certo valore, l'effetto che otteniamo è che alcune frequenze che sono sopra o sotto vengono tagliate.

Chapter 9

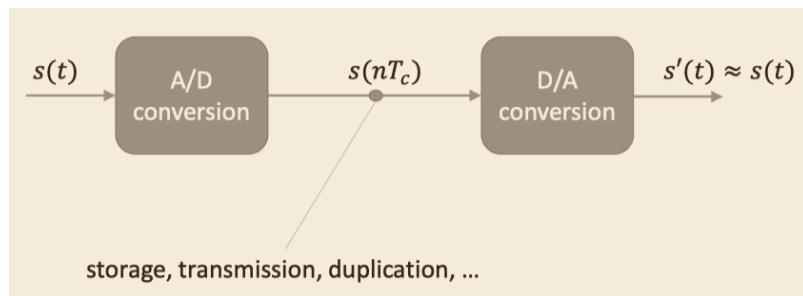
Signal Theory - Analog to Digital Conversion

I computer possono gestire solamente segnali digitali, supponiamo di avere un segnale $s(t)$ e di voler inviare il segnale tramite rete, è necessario eseguire una conversione del segnale da analogico a digitale ovvero è necessario trasformare il segnale originale in una sequenza di interi in modo che possano essere memorizzati e trasmessi da un dispositivo digitale. I componenti fondamentali di questa operazione di Analog to Digital conversion sono:

- Sampling: tramite il sampling partiamo da una sequenza $s(t)$ e poi estraiamo una sequenza di valori numerici che vengono estratti ad intervalli definiti (spazio uniforme). L'intervallo che passa tra un sample e il successivo è detto sampling period. Quindi quello che succede è che partiamo con $s(t)$, eseguiamo il sampling ottenendo una serie di numeri reali che non possiamo gestire.
- Quantization: ora che abbiamo svolto il sampling e che abbiamo ottenuto una sequenza di numeri reali possiamo eseguire la conversione di questi numeri in una sequenza di numeri accettati dalle

macchine, quindi ad esempio interi e float. Il tipo di conversione che svolgiamo dipende dal task che stiamo svolgendo. Il sampling ci permette di mantenere il segnale originale sotto alcune ipotesi, la quantizzazione invece implica una perdita di informazioni questo vuol dire che non è possibile ricostruire precisamente il segnale dopo la quantizzazione.

La conversione da analogico a digitale è fatta per fare in modo che il segnale diventi digitale e sia utilizzabile quindi via software, questa tecnica viene utilizzata nei dispositivi moderni, quindi per la musica ad esempio. Dato il segnale $s(t)$ prima facciamo la conversione da analogico a digitale che è la parte più complessa e poi otteniamo un segnale che è quello originale in cui leggiamo i valori in tempi che sono multipli di T_c . Poi dobbiamo fare nuovamente la conversione da digitale ad analogico e vogliamo che il segnale che otteniamo dopo questa seconda conversione sia il più simile possibile a quello originale.

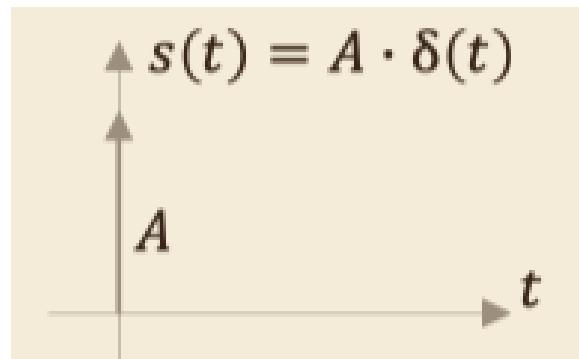


9.1 Sampling

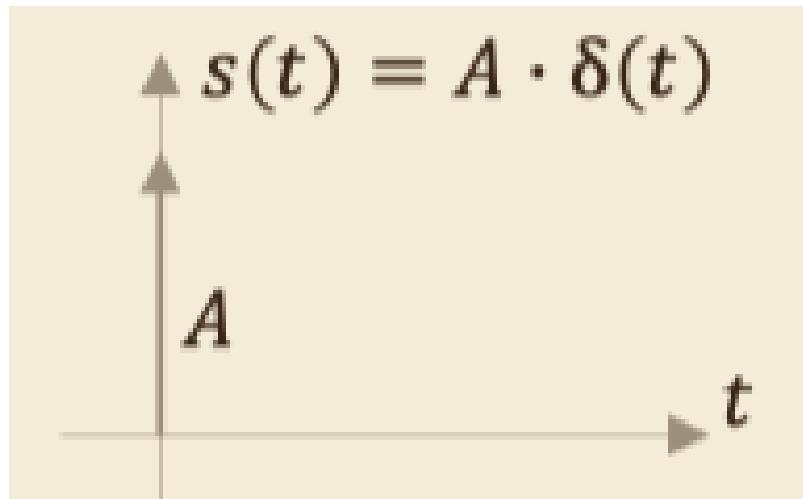
La Dirac Delta è un segnale ideale che vale 0 ovunque tranne che per $t = 0$, l'area sotto al segnale è 1.

$$\delta(t) = \begin{cases} \infty & \text{if } t = 0 \\ 0 & \text{if } t \neq 0 \end{cases}$$

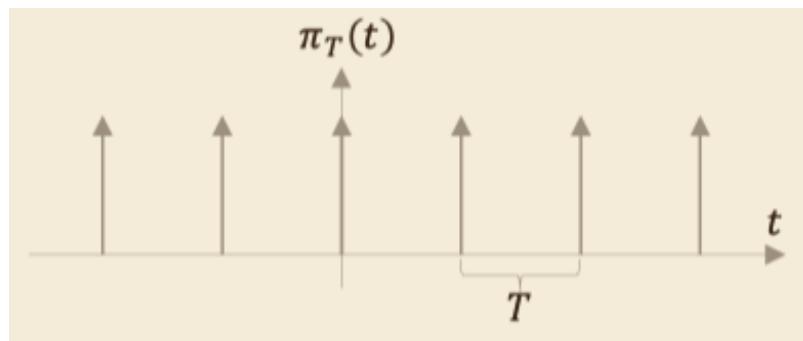
Questa dirac delta non è una funzione ma è una distribuzione, se calcoliamo $A * \delta(t)$ vediamo che otteniamo A, solitamente questo lo possiamo rappresentare con una freccia:



La lunghezza di questa freccia rappresenta il valore di A. Data la dirac delta la sequenza di delta con periodo T è un segnale periodico che definiamo come:

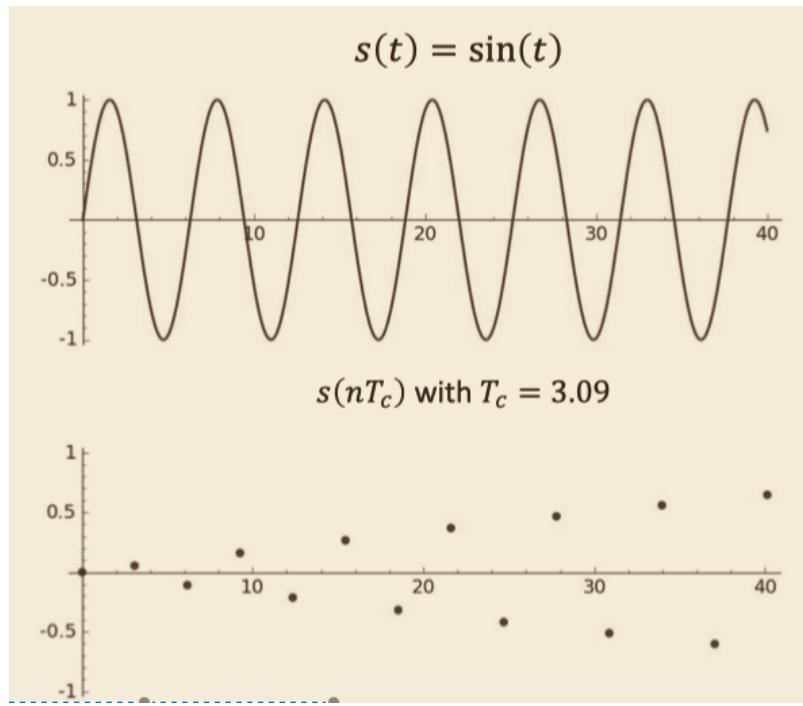


Possiamo anche rappresentarla graficamente come:



Il sampling è esso stesso un segnale $s(nT_c)$ che prende un valore per ogni multiplo del sampling period T_c . Il segnale viene rappresentato tramite i suoi sample presi a distanza di tempo discreto, il singolo sample è il valore del segnale preso ad un tempo preciso.

Vediamo un esempio di sampling, abbiamo il nostro segnale $s(t) = \sin(t)$, la frequenza di $s(t)$ è $f = \frac{1}{2\pi} = 0.159$, assumiamo di eseguire il sample del segnale con periodo $T_c = 3.09$, la frequenza del sampling è più grande del segnale originale.



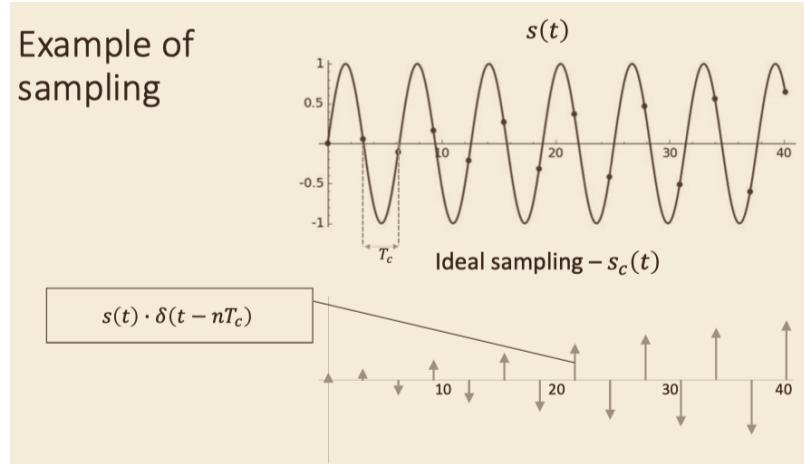
Il segnale originale è un segnale potenza mentre invece il segnale che otteniamo con il sampling è formato dai valori che prendiamo a distanza T_C e non è un segnale potenza e non ha energia.

9.1.1 Ideal Sampling

Nel sampling ideale i valori di $S(nT_c)$ sono estratti da $S(t)$ senza distorsione. Il sample ideale $S_c(t)$ è il prodotto del sample moltiplicato per il delta $\delta(t - nT_c)$. Quindi quello che otteniamo è:

$$\begin{aligned}
 s_c(t) &= \sum_{n=-\infty}^{\infty} s(nT_c) \cdot \delta(t - nT_c) = \\
 &= s(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT_c) = s(t) \cdot \pi_{T_c}(t) \\
 s_c(t) &= s(t) \cdot \pi_{T_c}(t)
 \end{aligned}$$

Eseguire il sampling ideale è impossibile, possiamo rappresentarlo in questo modo, per ragioni pratiche invece di utilizzare le dirac delta utilizziamo le squared pulses:



9.1.2 Teorema del sampling

Consideriamo un segnale $s(t)$ con spettro nullo alle frequenze sotto f_M . In pratica stiamo limitando la banda delle frequenze di $s(t)$, questa assunzione è irrealistica ma se è vera allora il segnale lo possiamo rappresentare

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

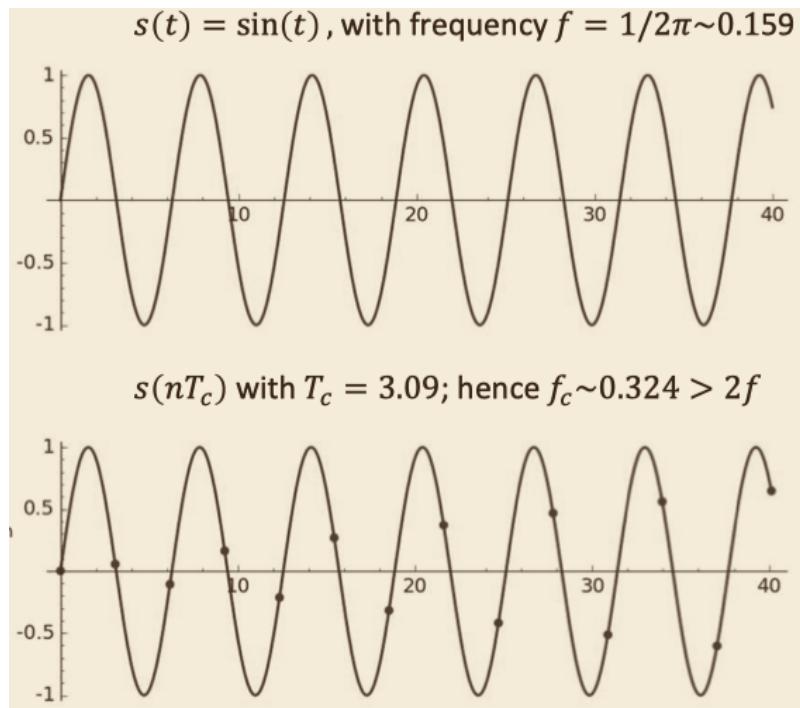
con il segnale preso a intervalli regolari. Quindi $s(t)$ è rappresentabile con samples presi a intervalli regolari $t_n = nT_C$ con un sampling period $T_C \leq \frac{1}{2f_m}$. Da questi sample possiamo ottenere il valore di $s(t)$ per ogni t:

$$f_{Cmin} = \frac{1}{T_{Cmax}} = 2f_m$$

questa f_{Cmin} che abbiamo ottenuto viene chiamata Nyquist frequency. La Nyquist frequency è la minima frequenza a cui è possibile fare un sample del segnale $s(t)$ limitato nella banda. Possiamo eseguire il sample a frequenze più alte ma non più basse. $s(t)$ sotto queste condizioni può essere ricostruito usando l'interpolazione:

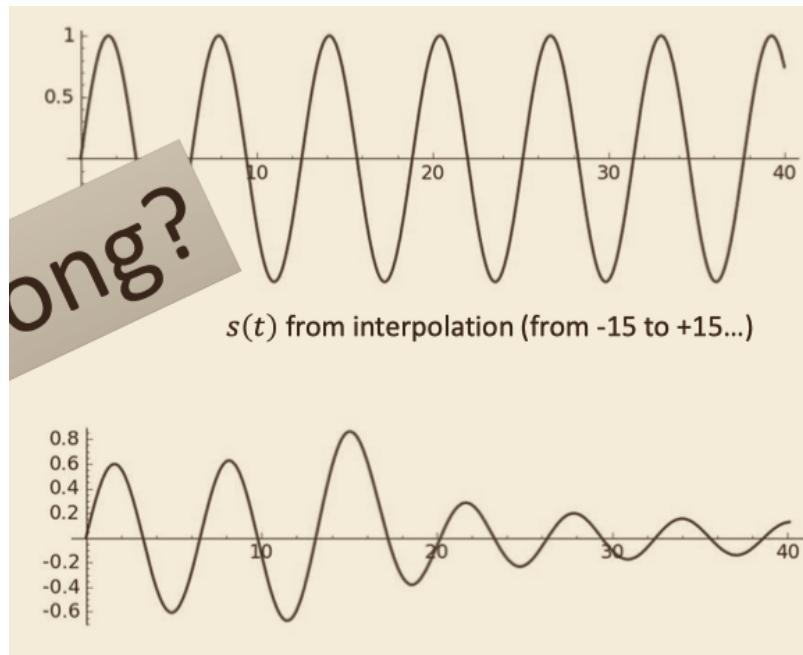
$$s(t) = \sum_{n=-\infty}^{\infty} s(nT_c) \cdot \frac{\sin(\pi \cdot f_c \cdot (t - nT_c))}{\pi \cdot f_c \cdot (t - nT_c)}$$

Qua nella formula vediamo che l'interpolazione viene espressa tramite il seno. Vediamo un esempio:

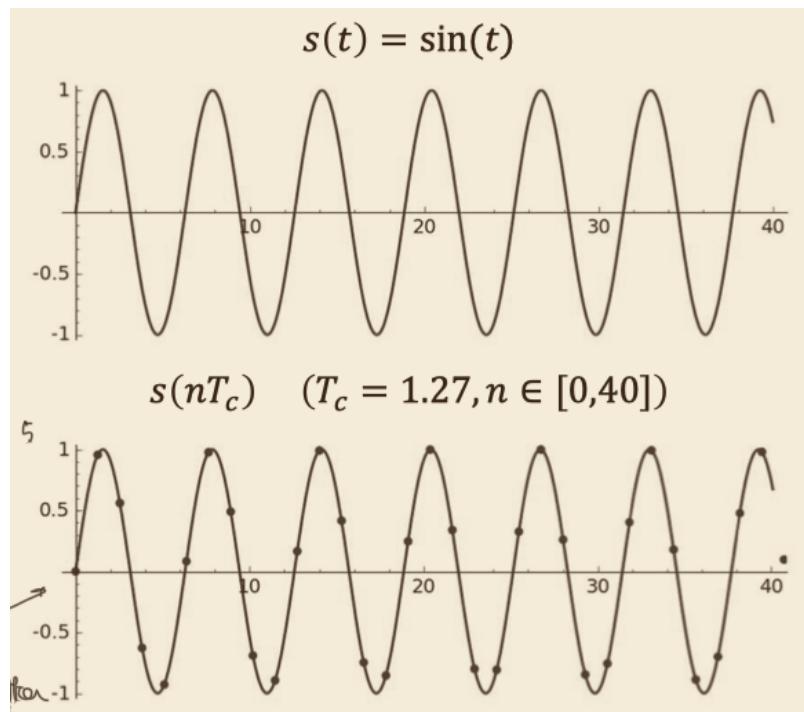


Abbiamo il nostro segnale iniziale $s(t)$ che ha una frequenza $\frac{1}{2\pi}$, se eseguiamo il sample di questo segnale con una certa sampling frequency otteniamo quello che abbiamo nel secondo grafico che potrebbe andare bene per ricostruire il segnale originale.

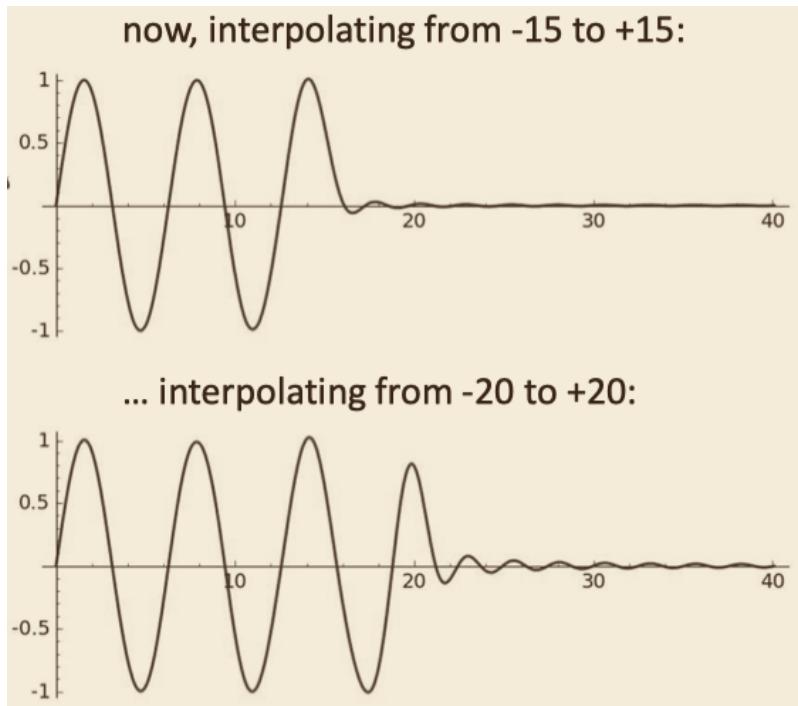
Se però cerchiamo di ricreare il segnale $s(t)$ iniziale tramite interpolazione tra -15 e +15 allora ci troviamo con un problema perché la ricostruzione in questo caso non è perfetta.



Una possibile soluzione a questo problema consiste nel cercare di aumentare il sampling rate, se prendiamo lo stesso segnale $s(t)$ che avevamo anche nell'altro esempio e poi eseguiamo il sampling in questo caso otteniamo più sample rispetto a prima e in questo modo possiamo eseguire l'interpolazione ottenendo risultati migliori.



Ad esempio se interpoliamo di nuovo tra -15 e +15 otteniamo il primo grafico mentre se aumentiamo il range dell'interpolazione otteniamo una approssimazione migliore.

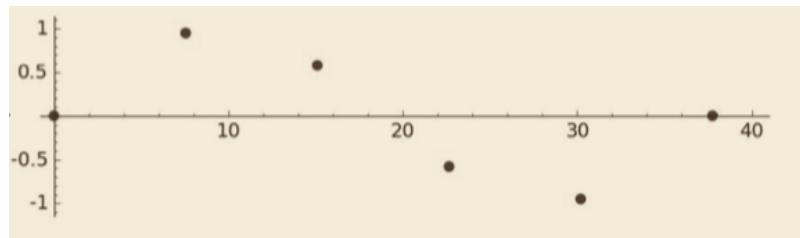


9.1.3 Problemi con Nyquist

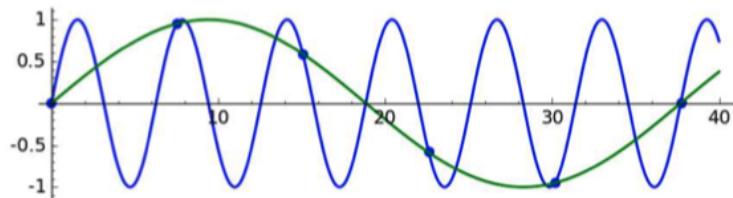
Consideriamo le ipotesi di Nyquist, nessun segnale reale è perfettamente limitato nella banda, un segnale di questo tipo non dovrebbe avere energia oltre una certa banda di frequenza ma per questo dovrebbe estendersi all'infinito. Per i segnali non periodici eseguiamo la trasformazione assumendo un periodo T che tende a infinito. Anche se poi il segnale è limitato nella banda non c'è modo di eseguire il sample dal mondo reale per ricreare perfettamente il segnale a meno che non prendiamo un numero infinito di sample. Dato che non abbiamo tempo infinito, un modo per migliorare il sample del segnale è andare ad aumentare la sampling frequency.

9.2 Aliasing

L'aliasing è il fenomeno che si produce quando applichiamo il teorema di campionamento e i requisiti non sono soddisfatti. Supponiamo di avere il segnale $s(t)$, se ignoriamo quello che c'è tra i sample vuol dire che non stiamo considerando molte informazioni. Se abbiamo solamente i sample da soli non è detto che il segnale da solo abbia la stessa frequenza. Ad esempio noi potremmo aver preso dei sample da un certo segnale e li possiamo rappresentare nel piano:



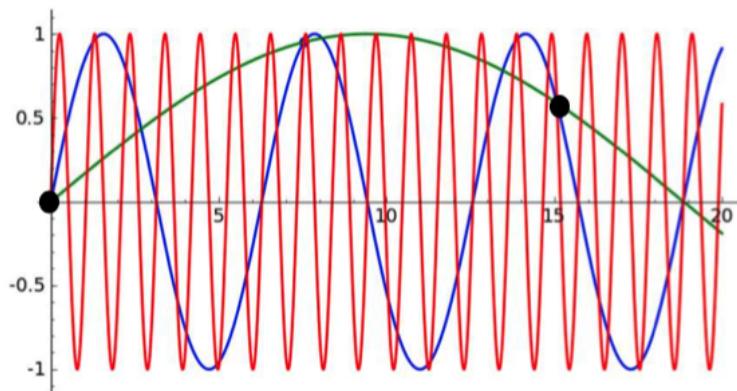
Il problema è che ora partendo da questi sample possiamo anche ricostruire altri segnali che sono differenti da quello che avevamo all'inizio.



Questo effetto che otteniamo è chiamato aliasing e questo rende indistinguibili i segnali nel momento in cui è stato eseguito il sampling. Dati i vari sample quindi potremmo generare più segnali che sono compatibili con quell'insieme di sample.

In pratica quello che succede è che abbiamo dei sample e partendo da questi sample possiamo andare a ricostruire un numero infinito di

armoniche e non possiamo distinguere quella che è realmente corrispondente a quel sampling.



Ci serve un modo per capire quali sono compatibili con quel sample.

9.2.1 Da dove arriva l'aliasing

Dato un segnale $S(t)$ limitato alla frequenza f_M il sample del segnale è $S(nT_c)$ e la corrispondente serie di impulsi è:

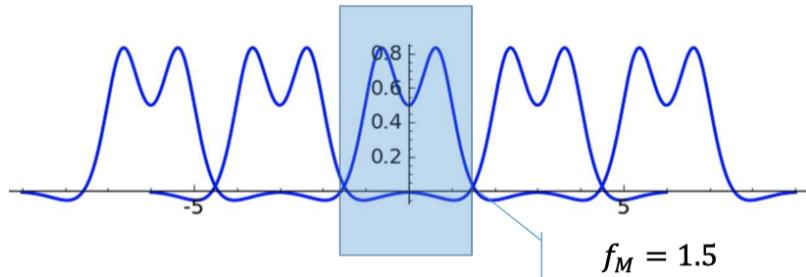
$$S_c(t) = s(t) * \pi_{Tc}(t)$$

Se svolgiamo la trasformazione del sample ideale, quindi la trasformazione $S_c(f)$ di $s_c(t)$ allora otteniamo:

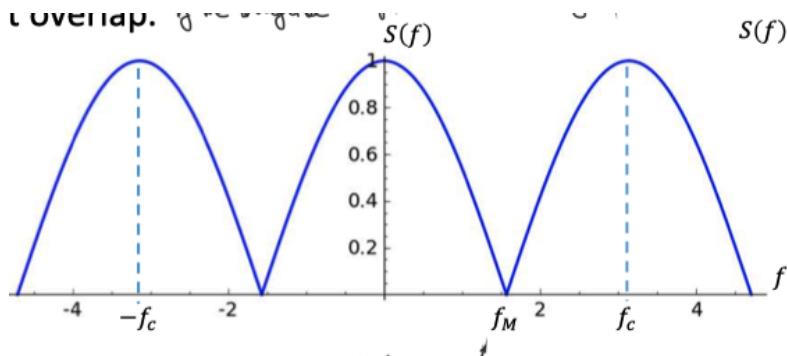
$$S_c(f) = F_c(s_c(t)) = f_c \sum_{n=-\infty}^{\infty} S(f - nf_c)$$

Otteniamo questa somma infinita che è la trasformazione di Fourier del segnale originale. Lo spettro di $s_c(t)$ è dato dalle infinite repliche dello spettro di $s(t)$ tutte centrate alle frequenze multiple della frequenza di sampling f_c .

Se noi assumiamo di avere $s(t)$ limitato nella banda a frequenza $f_M = 1.5$ possiamo non considerare tutte le repliche che hanno una frequenza più alta.



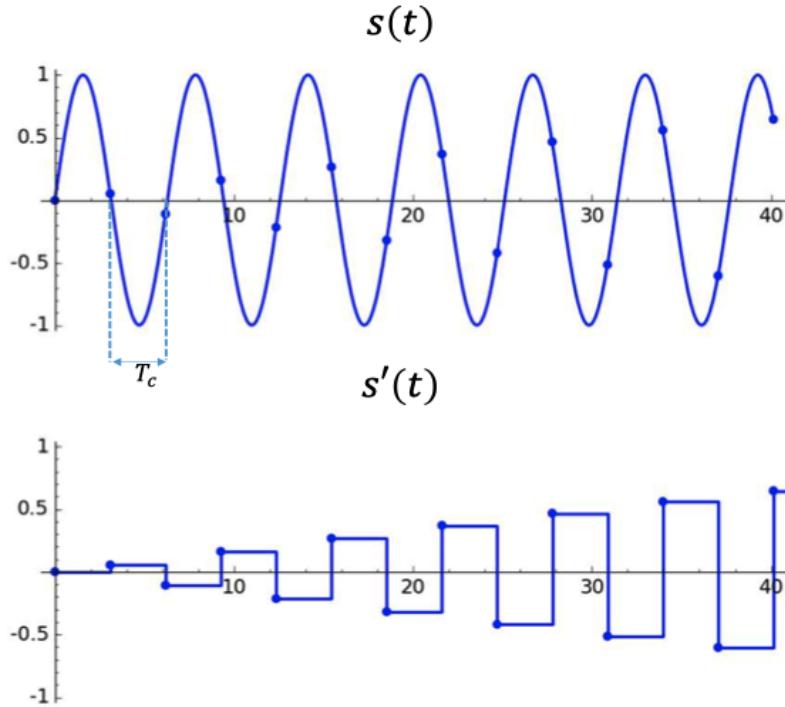
Questo è il motivo per cui Nyquist vuole che $s(t)$ abbia una frequenza nulla sotto f_M , se la $s(t)$ è strettamente limitato nella banda, possiamo scegliere f_c per essere sicuri che le repliche dello spettro di $s(t)$ non si sovrappongono.



9.2.2 Sampling

Nella vita reale nessun segnale è veramente limitato nella banda, altrimenti questo vorrebbe dire che il segnale è illimitato nel tempo. Molti segnali quindi sono quasi limitati nella banda che vuol dire che lo spettro oltre quella banda è piccolo. In tutti i sample possiamo aspettarci una

distorsione causata da questo fatto, questo problema lo possiamo risolvere con un sampling fatto bene, in questo modo la distorsione non sarà più visibile.



Quindi nella immagine sopra abbiamo il nostro segnale originale $s(t)$ e facciamo il sample, poi tramite i sample possiamo ricostruire il segnale $s'(t)$ utilizzando le square pulses.

9.3 Possibili errori con Nyquist

Consideriamo alcuni possibili errori che possiamo fare quando consideriamo Nyquist.

Devo fare un sample a 8KHz e quindi devo usare un filtro con un cutoff a 4KHz:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Nyquist non dice che se facciamo il sample al rate N poi possiamo usare un anti aliasing con un cutoff $f = N/2$.
- Il risultato in questo caso sarebbe quello di avere un alias che disturberebbe il segnale.
- Cosa possiamo fare?
 - Possiamo usare un filtro anti aliasing più complesso.
 - Possiamo fare oversample (questo accade nelle linee telefoniche analogiche)

Abbiamo un segnale a 1KHz che dobbiamo controllare quindi possiamo eseguire il sample a 2Khz:

- Nyquist dice che un segnale che si ripete N volte al secondo ha una bandwidth di N hertz.
- La powerline è a 60HZ ma in realtà c'è tanta distorsione.
- Se non vogliamo usare l'anti aliasing filter dobbiamo capire la massima frequenza del segnale.

9.4 Quantizzazione

Come abbiamo già detto dopo il sampling i numeri che otteniamo sono solitamente reali, in alcuni casi anche la rappresentazione in floating point è troppo costosa. La quantizzazione indica una approssimazione del valore del sample in modo che possa essere rappresentato come un intero nell'intervallo $[0, 2^R - 1]$.

Vediamo come funziona la quantizzazione:

- Abbiamo un segnale $s(t)$ con una frequenza di sampling f_c .

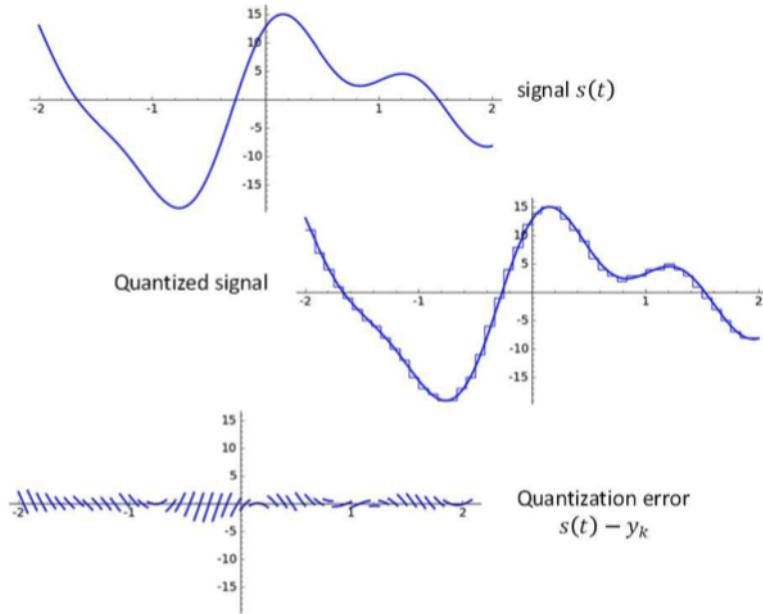
- x_k è il sample letto nell'intervallo di tempo I_k .
- La quantizzazione codifica X_k in un valore intero $y_k \in [0, 2^R - 1]$.
- La quantizzazione richiede R bit per ogni sample.
- Il bit rate per un segnale in cui facciamo il sample a frequenza f_c è:

$$R_b = R * f_c \text{bit/sec}$$

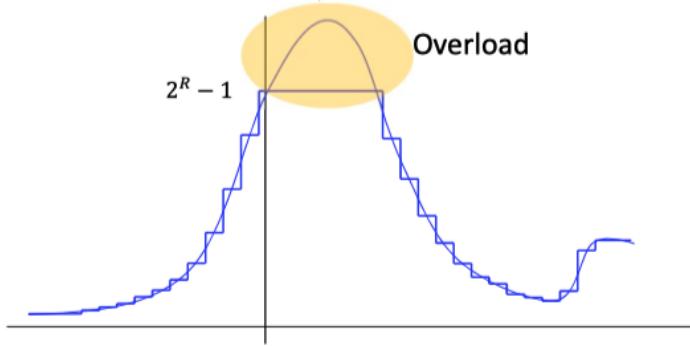
dove la f_c è la frequenza del sample.

- La quantizzazione è chiamata scalare.

L'immagine qua sotto mostra il risultato della quantizzazione, abbiamo il segnale iniziale $s(t)$ e svolgiamo il sampling ottenendo una sequenza di sample x_k che vengono letti ad intervalli I_k . La quantizzazione codifica questo intervallo di sample in una sequenza y_k . Il segnale quantizzato lo mostriamo nel grafico al centro mentre nell'ultimo mostriamo invece la differenza che abbiamo tra il segnale quantizzato e il segnale originale $s(t)$.



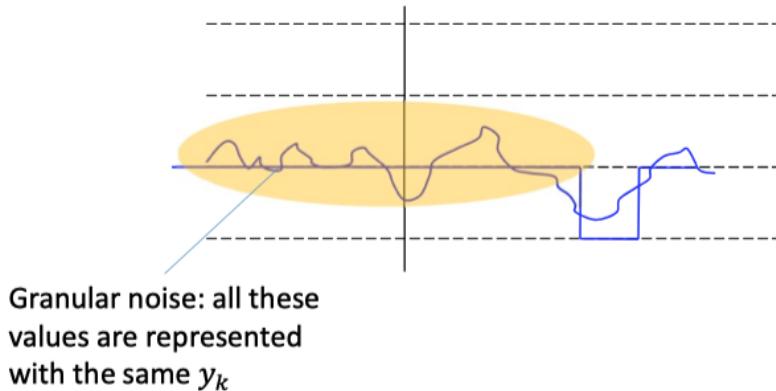
Un altro tipo di quantization noise è l'overload, questo tipo di errore ce l'abbiamo quando abbiamo $s(t) > 2^R - 1$ per una qualche t . In questo caso in pratica il valore del segnale al tempo t eccede la rappresentazione della quantizzazione, questo è graficamente quello che avviene quando abbiamo l'overload:



Un altro tipo di errore che si verifica è il granular noise, in questo caso abbiamo tutti i valori in un certo intervallo vengono rappresentati

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

con un unico valore quantizzato y_k .



Vediamo un esempio:

- La voce ha una banda di 20KHz.
- Accettando una degradazione della qualità possiamo avere un livello per il taglio attorno a 4KHz.
- Quindi abbiamo che il sampling rate è $f_c = 8KHz$.
- Ognuno dei sample vengono quantizzati a 8bits quindi R=8.
- Il bit rate quindi è $R_b = 64Kbps$, il bitrate diventerà questo quando solitamente è quello supportato dai protocolli.

Nel caso della qualità dei CD invece si utilizza l'oversampling:

- La qualità dell'audio di un cd ha una bandwidth di 20KHz.
- Con un piccolo oversampling prendiamo i sample con un rate di $f_c = 44.1KHz$.
- La quantizzazione viene fatta con 16 bit per ogni sample.

- Abbiamo 2 canali.
- Quindi il bit rate è 1411Kbps e per ognuno dei canali è 705Kbps.

Chapter 10

Thingspeak

Thingspeak è una piattaforma per l'internet of things. Con piattaforma intendiamo un servizio che permette all'utente di connettere vari sensori e vari componenti di un sistema IoT. In questo campo ci sono tanti software di questo tipo perchè il settore è in continua evoluzione e le cose cambiano molto velocemente. Quello che vediamo con Thingspeak non è ancora "vecchio" ma era moderno 3/4 anni fa, è comunque utile partire da qua perchè è più facile da essere utilizzato rispetto ad altri.

La piattaforma IoT deve essere in grado di:

- Raccogliere i dati, i vari dispositivi inviano alla piattaforma i dati raccolti prima di processarli.
- I dati devono essere scambiati in modo sicuro perchè comunque sono dati sensibili, ad esempio potrebbero rivelare dove ti trovi in quel momento.
- Una volta che i dati sono arrivati alla piattaforma, la piattaforma dovrebbe essere in grado di mostrarli ma anche di aggregarli perchè vari sensori producono molti dati e quindi i dati si accumulano rapidamente e dobbiamo trovare il modo di aggregarli bene in modo da gestirli nel modo corretto.

- Dobbiamo poter creare dei trigger in base al tipo di dati che riceviamo e in base al tipo di dato ricevuto possiamo eseguire una certa azione.

Queste attività della piattaforma sono spesso supportate dal cloud:

- In alcuni casi come SaaS (Software as a Service).
- In altri casi abbiamo Paas (Platform as a Service), qui abbiamo probabilmente una API Rest, nel primo caso possiamo avere qualsiasi altra cosa.
- Possiamo anche utilizzare una architettura serverless, come ad esempio AWS che possiamo utilizzare per processare i dati prima di inviarli ad un database.

Tutte le maggiori aziende hanno progetti dedicati all'IoT ma ci sono anche delle compagnie che lavorano su altri servizi commerciali:

- Amazon
- Microsoft
- IBM con Watson
- Cisco si focalizza su un approccio Edge Computing. Loro provano a diminuire il traffico dalle varie unità al server centrale e al database e questo è interessante perché lavorano su dispositivi di questo genere e quindi l'idea è di mettere dell'"intelligenza" all'interno di questi dispositivi rendendo meno pesanti le comunicazioni (Viene chiamata anche FOG computing).
- Salesforce ha un'offerta di dispositivi per dispositivi IoT "da casa", come Domoticz ad esempio.

Se non vogliamo utilizzare questi tipi di servizi possiamo utilizzare dei servizi che possono essere installati sulla nostra macchina, ad esempio Kaa o anche ThingSpeak che è un servizio commerciale pensato specificatamente per l'IoT che offre anche un server open source (che però non è mantenuto bene).

10.1 ThingSpeak

ThingSpeak ci fornisce un servizio e un middleware. Il web service permette di utilizzare un piano gratuito che ha alcune limitazioni. Il server di ThingSpeak ci fornisce:

- Un'interfaccia REST/HTTP per interagire con il servizio.
- La possibilità di processare i dati e di visualizzarli utilizzando Matlab, questa possibilità è solo per i piani premium.
- Possiamo embeddare i dati e i grafici in documenti.
- Possiamo integrare altri servizi con ThingSpeak, ad esempio noi vorremmo interagire anche con altri servizi in internet, un esempio è Twitter.
- ThingSpeak può generare, basandosi su trigger e su altre condizioni, richieste GET e POST.
- Abbiamo anche la possibilità di avere dei feedback dai dispositivi IoT che abbiamo connesso.

10.1.1 REST

ThingSpeak ci fornisce una interfaccia REST che possiamo utilizzare per interagire con la piattaforma. REST è caratterizzato dall'essere

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

stateless, non c'è una memoria, questo è comodo per il caso dei dispositivi IoT perchè non vogliamo affidarci al fatto che venga mantenuto uno stato perchè potremmo avere dei fallimenti e la nostra applicazione avrebbe dei problemi se dovessimo basarci solamente sullo stato.

Le librerie REST e HTTP esistono per tanti dispositivi quindi non c'è necessità di riscriverle da 0.

Utilizzare uno standard di questo genere come HTTP permette anche di semplificare il lavoro perchè utilizziamo delle porte standard.

Con questo sistema abbiamo anche la possibilità di utilizzare dei protocolli standard che ci permettono di avere una interoperabilità con altri sistemi, perchè qua utilizziamo ad esempio HTTP o JSON.

10.1.2 Channel

L'astrazione che sta alla base di ThingSpeak sono i channel che rappresentano il valore di varie variabili durante il tempo (8 variabili al massimo), ogni canale ha un ID.

L'accesso al canale è protetto da una chiave delle API, per avere l'accesso al canale dobbiamo aggiungere l'URL nel campo Header, scrivere l'API key per aggiungere dati e leggere la API Key per accedere ai dati (se è privato dobbiamo specificarla, se è pubblico no).

Un singolo canale può rappresentare valori che arrivano da vari sensori, un canale può essere condiviso tra vari sensori ad esempio. Dispositivi distinti potrebbero infatti caricare dati sullo stesso canale.

Ogni variabile all'interno del channel viene chiamata field, possiamo avere ad esempio un canale che rappresenta dati che arrivano da sensori del meteo o rappresentare con ogni field un valore differente. Possiamo caricare uno o più valori alla volta con una operazione che è chiamata feed, un singolo feed può aggiornare vari field alla volta. Possiamo implementare l'operazione feed in due modi:

- Possiamo fare una GET per aggiornare il canale in cui inseriamo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

l'url e nell'url specifichiamo i field e il valore corrispondente.

- Possiamo fare una POST e nel body della richiesta inseriamo le chiavi che vogliamo aggiornare memorizzate come un JSON.

Quando facciamo l'accesso ai dati tramite le API possiamo anche decidere di visualizzare solamente una parte dei dati, in questo caso il preprocessing viene effettuato direttamente da ThingSpeak che invia solamente i dati che richiediamo.

Tramite ThingSpeak possiamo anche embeddare una rappresentazione dei nostri dati all'interno di un altro documento, ad esempio possiamo embeddarlo all'interno di una documento HTML.

10.1.3 Thingspeak come Relay Point

Possiamo anche utilizzare ThingSpeak come un Relay Point ovvero un punto in cui le informazioni transiscono (possiamo anche chiamarlo Proxy ma Relay Point è meglio). Possiamo ad esempio aggiungere dei dati che poi vengono letti da un attuatore.

Per aggiungere un nuovo comando utilizziamo una richiesta POST aggiungendo il comando che vogliamo inviare all'interno del body della richiesta.

10.2 ThingSpeak integration

Un sistema IoT è formato da varie periferiche, sensori e attuatori, i dati sono memorizzati e possiamo lavorare su questi dati. ThingSpeak può essere utilizzato come infrastruttura IoT si occupa di unire tutto questo in un'unica piattaforma.

Una volta raccolti i dati possiamo analizzarli, filtrarli e visualizzarli, possiamo usare Matlab per questo ad esempio. I dati possono anche essere aggregati e processati, per l'aggregazione abbiamo a disposizione

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

delle API Rest e per abbiamo anche la possibilità di integrarci con altri servizi.

Integrando ThingSpeak con altri servizi possiamo estendere delle funzionalità che sono già presenti aggiungendone di nuove, l'integrazione necessita di comunicazione tra interfacce che devono essere compatibili. Nel nostro caso ThingSpeak solitamente utilizza REST per comunicare, si integra in questo modo con Twitter ad esempio ma anche con qualsiasi altro servizio che offre una interfaccia REST. Una volta che abbiamo il modo di interagire con altri servizi dobbiamo trovare il modo di triggerare questa comunicazione. Abbiamo tre tipi di trigger che possiamo utilizzare:

- Time Event: ad un certo momento possiamo invocare un certo servizio REST o magari possiamo twittare qualcosa, questo è periodico.
- L'altra possibilità si basa sui dati che riceviamo, a seconda dei dati che riceviamo triggeriamo un certo evento.
- Abbiamo poi dei trigger con i tweet. Per questo scopo si utilizza ThingTweet, prima di tutto dobbiamo collegare ThingSpeak con il nostro account Twitter poi linkiamo la nostra applicazione con Twitter. Ad un certo punto ThingSpeak può andare a twittare qualcosa quando succede qualcosa che noi decidiamo possa triggerare l'azione. Posso decidere di inviare il tweet quando faccio io la richiesta oppure anche quando succede qualcosa, ad esempio se la temperatura scende sotto ad un certo livello. Per triggerare la pubblicazione del tweet possiamo anche utilizzare uno script scritto in Python.

Un'altra possibile integrazione è ThingHTTP, quando invochiamo ThingHTTP possiamo generare una richiesta ad un altro servizio REST oppure anche a ThingSpeak stesso. Quando utilizziamo ThingHTTP abbiamo i seguenti campi che devono essere riempiti:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Abbiamo l'url dell'endpoint.
- Quando accediamo una certa API può essere utile utilizzare anche dei parametri per l'autenticazione.
- Il tipo di richiesta GET/POST/PUT/DELETE.
- Il tipo di contenuto che inviamo.
- Il body.

Quando creiamo una nuova istanza di ThingHTTP dobbiamo associare un identificatore per la funzione e una apiKey.

Anche Thingspeak è integrato con ThingHTTP tramite dei trigger.

10.2.1 Usare un database esterno

Una possibile integrazione è quella di ThingSpeak con i database e in particolare con database NoSQL come MongoDB. Questo è rilevante perchè possiamo essere interessati a prendere i dati o a memorizzarli non solo da ThingSpeak ma anche da database esterni. In questo caso utilizziamo MongoDB che è un database NoSQL che sono interessanti per quel che riguarda l'IoT per varie ragioni:

- Un database NoSQL è in grado di gestire molti più dati rispetto ad un database relazionale.
- Il database è formato da collections, non troppe, ad esempio una per i record dei dipendenti, uno per gli uffici....
- Ogni collection contiene i documenti, ad esempio per descrivere il singolo impiegato.
- Ogni documento è un dato JSON.

- NoSQL è in grado di eseguire delle fuzzy query ovvero delle query che non matchano completamente il contenuto, abbiamo un match parziale.
- I documenti all'interno delle collezioni non necessitano tutti lo stesso formato e questo è possibile perchè noi non definiamo un formato dei documenti nel momento in cui generiamo il database.

Una delle cose che possiamo fare con MongoDB è una replicazione dei dati, in questo caso noi utilizziamo ad esempio Atlas e creiamo un cluster per fare in modo di avere la replicazione. I dati vengono salvati tramite MongoDB e Atlas nel cloud e possiamo scegliere ad esempio AWS come provider del cloud.

Per accedere ai nostri dati possiamo utilizzare il FaaS (Function as a Service). Abbiamo un sistema che è molto simile alle Lambdas di AWS. Una stitch FaaS è una funzione Javascript che possiamo sviluppare direttamente nel browser usando una interfaccia web, questa mi espone una interfaccia REST e non mantiene uno stato. Questo è importante ed è facile, utilizzando questi tool riuscire a non utilizzare un server. Se vogliamo estrarre ad esempio la temperatura di una certa città possiamo creare una funzione che mi esegue questo task. Questa funzione viene attivata quando la chiamo, viene eseguita, prende i dati e restituisce i dati all'applicazione che ha fatto la richiesta HTTP. MongoDB utilizza le stitch function e le stitch espongono una interfaccia REST.

Chapter 11

IEEE 802.15.4 e Zigbee

Zigbee è lo standard per le wireless sensor network sviluppato dalla Zigbee alliance che viene utilizzato in vari campi oltre a quello della home automation, ad esempio Health care o anche industrial automation. Le principali richieste di Zigbee sono:

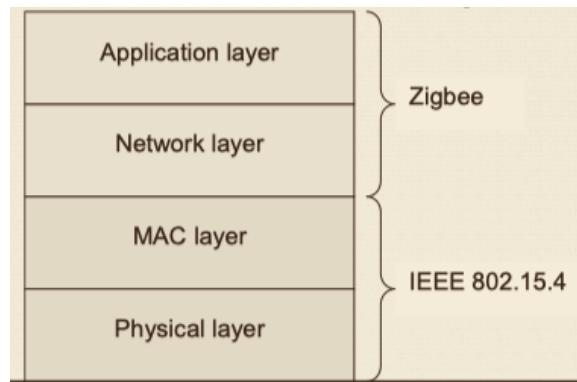
- Vogliamo una rete completamente autonoma senza l'intervento umano.
- Vogliamo far comunicare dispositivi di produttori differenti.
- Vogliono una lunga durata della batteria e uno scambio di non troppi dati.

Le principali caratteristiche di Zigbee sono:

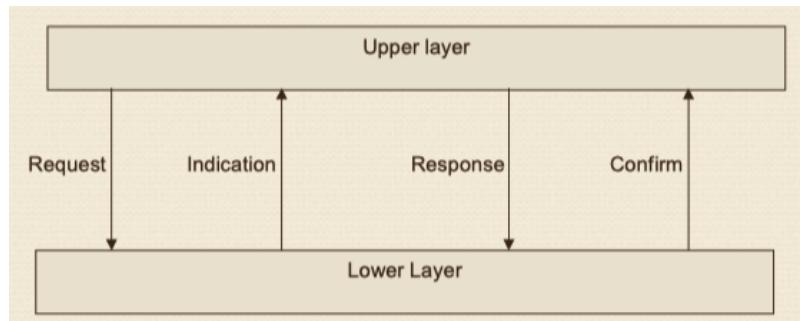
- Si basa su uno standard.
- È low cost.
- Può essere utilizzato globalmente.
- Supporta un gran numero di nodi.
- È facile da deployare.

- È sicuro.
- Garantisce una lunga durata della batteria.

Zigbee è stato sviluppato per funzionare sopra allo stack IEEE 802.15.4, in particolare l'idea è stata quella di utilizzare il Mac e il Physical Layer dell'IEEE 802.15.4 andando poi a standardizzare i livelli superiori ovvero il Network layer e l'application Layer.

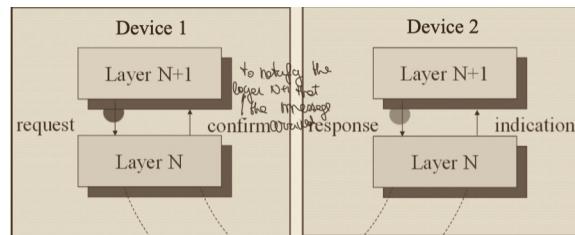


Nella foto sopra vediamo meglio come sono organizzati questi layer, in particolare possiamo dire che ognuno di questi layer fornisce dei servizi al livello superiore, questi servizi sono quello che permette ai vari layer di interagire tra loro. Questi servizi sono specificati con una serie di primitive che possono essere di 4 tipologie:



- Request: è invocata dal layer di livello superiore per invocare una richiesta per un servizio del livello inferiore.
- Indication: è generata dal livello inferiore ed è diretta al livello superiore per notificare che un certo evento è avvenuto.
- Response: viene inviata al livello inferiore per notificare che la procedura che era stata avviata in precedenza è ora terminata.
- Confirm: è un ack che viene generato dal livello superiore e viene inviato al livello inferiore.

Questa organizzazione con layer permette anche di far comunicare vari dispositivi, ad esempio se abbiamo due device e il device 1 vuole comunicare con il device 2, al livello N+1 del device 1 invierà una richiesta al livello N del device 1, questo poi inoltrerà la richiesta al livello N del device 2, tra i livelli poi la comunicazione andrà avanti come se il tutto avvenisse in un singolo dispositivo.



11.1 IEEE 802.15.4

Lo standard IEEE 802.15.4 viene utilizzato per il livello fisico e per il livello MAC nelle Personal Area Networks, è "short range".

11.1.1 Livello Fisico

All'interno del livello fisico dell'IEEE 802.15.4 abbiamo i seguenti servizi:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

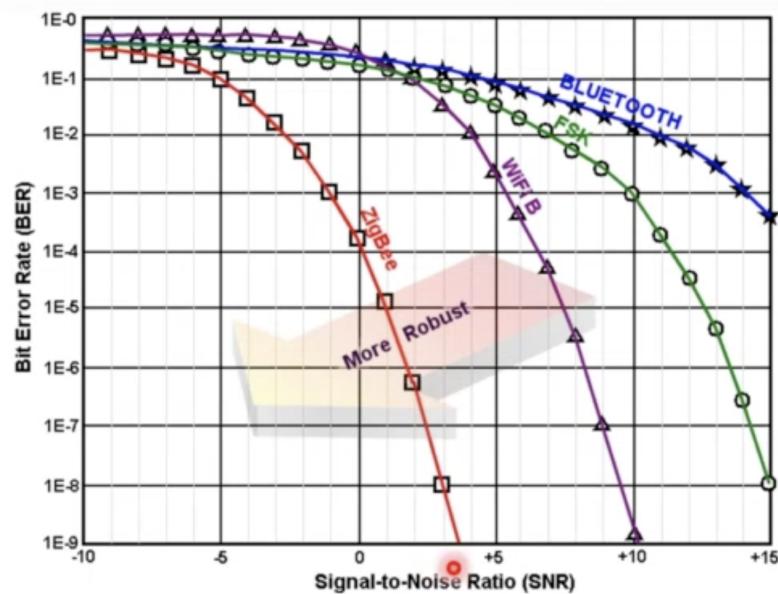
- Data service: questo servizio consente la trasmissione e la ricezione di PPDU (Protocol Data Unit) attraverso il mezzo fisico.
- Management Service: questo è un servizio che gestisce vari aspetti differenti:
 - Per implementare l'efficienza energetica abbiamo un sistema di attivazione/disattivazione della radio.
 - Abbiamo un sistema di energy detection per implementare il carrier sense. Questo servizio di energy detection stima l'energia, calcola una media e poi restituisce indietro il risultato.
 - Abbiamo un Link Quality Indicator che stima la qualità dei pacchetti dati che vengono ricevuti. La qualità dei pacchetti che vengono ricevuti si basa sull'Energy Detection oppure sul rapporto segnale/rumore. Per la qualità dei pacchetti dobbiamo avere almeno 8 livelli differenti. Il valore stimato per la qualità dei pacchetti viene poi inoltrato alla rete e all'application layer.
 - Viene implementato un sistema che possa permettere di capire se un certo canale è libero o è utilizzato. Per capire se il segnale è in uso o no viene utilizzata la Energy Detection.
 - Viene gestita nel Physical layer la gestione dei servizi che riguardano la configurazione del livello fisico.
 - Abbiamo i data services che sono servizi utilizzati per la trasmissione delle data units. Questi servizi possono essere utilizzati per inviare indietro informazioni riguardanti il fallimento o il successo.

Il livello fisico solitamente utilizza 3 bande, la più utilizzata è la 2.4 Ghz dove abbiamo 15 canali.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Supponiamo di avere una rete Zigbee con 30.000 dispositivi, i canali sono assegnati all'intera rete, non al singolo dispositivo. Nel caso di Zigbee la trasmissione è differente rispetto a WiFi e non utilizza la frequency division per l'accesso.

Nel seguente grafico possiamo notare le performance di Zigbee rispetto al Wifi e a Bluetooth. Zigbee è più robusto degli altri a parità di Signal To Noise.



11.1.2 Livello Mac

Anche nel livello Mac vengono messi a disposizione dei servizi simili a quelli del livello fisico, in più qua abbiamo anche dei meccanismi di sicurezza. In particolare per quel che riguarda i servizi abbiamo:

- Data Services: viene gestita la trasmissione e la ricezione di frames al livello MAC attraverso il livello fisico.

- Management Services: permette di gestire la sincronizzazione della comunicazione che è importante per implementare l'efficienza energetica. Inoltre questo servizio permette anche di gestire l'associazione e la disconnessione dei dispositivi dalla rete, la presenza di questo servizio è necessaria per rendere la rete self-configurable e autonoma. Quando i dispositivi vengono accesi si connettono in automatico alla rete senza la necessità di un intervento umano.
- Security Mechanism

Al livello MAC abbiamo necessità di due tipologie di dispositivi:

- Al primo livello abbiamo i Reduced Function Devices che sono dispositivi con capacità ridotte (in termini di memoria e di capacità di comunicazione). Questi dispositivi hanno dei vincoli molto forti e implementano solamente una parte delle funzionalità del MAC layer.
- Ci sono poi dei dispositivi "potenziati" che sono i Full Function Devices e vengono utilizzati per la comunicazione con gli altri dispositivi. Questi implementano tutto il livello MAC e funzionano come coordinatori del PAN o come coordinatori di un generico set di RFDs. Questi coordinatori fanno il setup e gestiscono la rete andando anche a gestire l'associazione e la disconnessione dei dispositivi.

Per quanto riguarda la topologia implementata al livello di rete dai Full Function Devices abbiamo due tipologie:

- Star: abbiamo un solo coordinatore, gli altri sono tutti RFD, il coordinatore poi sincronizza tutta la comunicazione all'interno della rete. In una certa area possiamo avere differenti PAN ognuna allocata su un canale differente e possono avere un identificatore

differenti. Ognuna di queste PAN è del tutto indipendente e non può comunicare con le altre.

- Abbiamo una seconda topologia implementata che è la Peer To Peer, qua in questa rete possono esserci più di un FFD ma solamente uno fa da coordinatore mentre gli altri FFD sono routers all'interno della rete PAN. Gli altri nodi della rete sono RFD che sono poi connessi con un FFD.

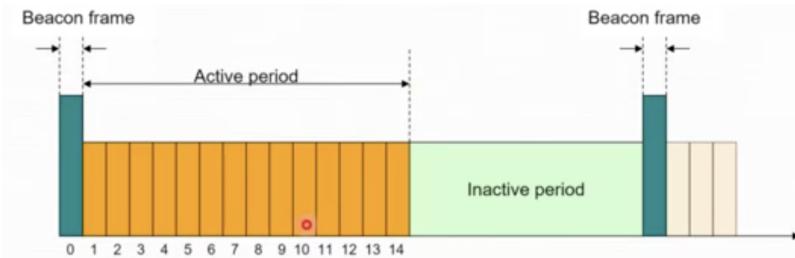
Per quanto riguarda l'accesso al canale al livello MAC abbiamo due approcci possibili, con il superframe e senza:

- La struttura con il superframe viene utilizzata tipicamente con la topologia a stella e garantisce la sincronizzazione tra i nodi abilitando la power saving dei dispositivi. In questo caso è il coordinatore che lavora in modo "aggressivo" per garantire la comunicazione sui nodi.
- Senza il superframe è un meccanismo più generale, se vogliamo implementare l'energy efficiency in questo caso deve essere gestita completamente dal tuo dispositivo e non dal coordinatore. Normalmente la comunicazione qua è gestita in un modo arbitrario Peer to Peer.

I frames sono l'unità base per il trasporto di dati e ne esistono 4 tipologie (data, acknowledgment, beacon e MAC). Una struttura aggiuntiva è il superframe che è definita dal coordinatore e può essere utilizzata nel caso in cui due beacon agiscono come limiti della struttura e garantisce sincronizzazione con gli altri dispositivi così come informazioni di configurazione. Un superframe consiste in 16 slots di uguale lunghezza che possono essere divisi in due parti, active e inactive, tutte le trasmissioni devono terminare prima dell'arrivo del secondo beacon.

Per quanto riguarda l'utilizzo del superframe abbiamo questa situazione:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Qua possiamo notare varie cose:

- Il tempo viene suddiviso in due parti, uno è l'active period e l'altro è l'inactive period. All'inizio dell'active period abbiamo il Beacon Frame che viene inviato dal coordinatore e contiene le informazioni che descrivono questa rete. In questo modo il coordinatore fa sapere sul canale su cui sta lavorando che la sua rete esiste, in particolare fornisce informazioni riguardo all'active period e all'inactive period. Quando questi due periodi sono finiti viene trasmesso un altro Beacon frame.
- Durante l'inactive period il coordinatore può andare in power save mode perchè non deve comunicare con la rete e non deve controllarla. Durante l'active period il coordinatore ha la radio accesa mentre tutti gli altri possono scegliere.
- L'active period è diviso in slots, se un dispositivo vuole comunicare deve attendere l'inizio dello slot, implementare un sistema di accesso al canale e poi comunicare.
- L'active period può essere formato da al più 16 slots.
- L'active period è diviso in due parti uno è la CAP e la comunicazione interna alla CAP è implementata con la CSMA/CA e poi abbiamo un contention free period che contiene degli slot che sono staticamente associati ad un dispositivo specifico. Possiamo

richiedere l'accesso a questi slot del contention free period per poter inviare dati in ogni momento.

- Il Contention Access Period: in questo periodo i dispositivi competono per accedere al canale con CSMA-CA. Una volta che un dispositivo ha ottenuto il suo slot per trasmettere, inizierà a trasmettere e nessun altro potrà trasmettere.
- Il contention free period è usato per le low latency applications o per applicazioni che hanno richieste specifiche in termini di bandwidth. Possiamo dividere questo Contention free period in GTS, ognuna di queste GTS viene assegnata ad uno specifico dispositivo. All'interno di un CFP abbiamo al massimo 7 GTS.
- In ogni caso il CFP non può occupare tutti gli slot di tempo. Per esempio questo (avere i CAP) è utile quando deve entrare un nuovo dispositivo nella rete perché il dispositivo non è ancora nella rete e non può comunicare nel CFP ma può farlo nel CAP. In ogni caso ogni transazione contention-based dovrebbe terminare prima dell'inizio del contention free

Per l'accesso al canale con il superframe, ogni coordinatore PAN crea il suo superframe, l'active period del superframe ha sempre la stessa lunghezza, il coordinatore, dopo aver terminato il setup del superframe, inizia a inviare il beacon. Se la rete ha una topologia Peer To Peer, in questo caso è organizzata come rete multi hop e ogni router avrà dal coordinatore gli stessi parametri del superframe, i parametri sono già utilizzati all'interno della rete e tutti i router andranno a configurarsi con quei parametri ed emitteranno poi a loro volta il beacon frame.

Anche in Zigbee ci sono gli InterFrame Spacing per dare priorità alla comunicazione, prima di spedire un frame un nodo dovrebbe attendere quindi l'inter frame space.

11.1.3 Frames

In 15.4 ci sono 4 tipologie di frame:

- Beacon Frame
- Data Frame
- Acknowledgement frame
- Command frame: utilizzato per gestire la connessione/disconnessione

11.1.4 Rete senza superframe

Se la rete è senza superframe allora abbiamo una Non-Beacon Enabled network, la comunicazione si basa su CSMA-CA non slotted, non abbiamo slot e non abbiamo beacon. Il coordinatore è sempre attivo e pronto a ricevere dati, non c'è modo per il coordinatore di spegnere la radio per risparmiare energia. Al livello applicazione però possiamo comunque implementare qualcosa che ci permetta di spegnere la radio in modo da avere comunque un risparmio di energia.

Il trasferimento di dati dal coordinatore agli end device sarà poll-based. L'end device può spegnere la radio, quindi il coordinatore non può sapere quando è accesa e quando è spenta e quindi non può inviare direttamente i dati all'end device quando vuole. Il meccanismo è indiretto ed è poll based, è il coordinatore o il router che controllano periodicamente l'end device per capire se la radio è accesa oppure no e se è accesa allora i messaggi in attesa vengono spediti.

11.1.5 Data transfer

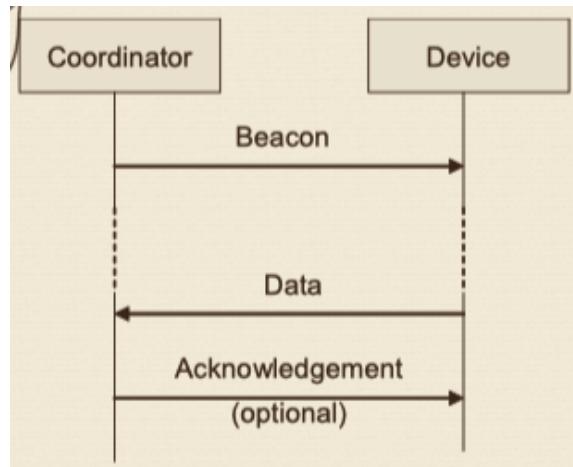
Per quanto riguarda il trasferimento di dati abbiamo vari metodi che dipendono anche dalla topologia della rete:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Abbiamo la comunicazione tra l'end-device e il coordinatore.
- Abbiamo la comunicazione tra il coordinatore e l'end device.
- Abbiamo una comunicazione peer to peer.

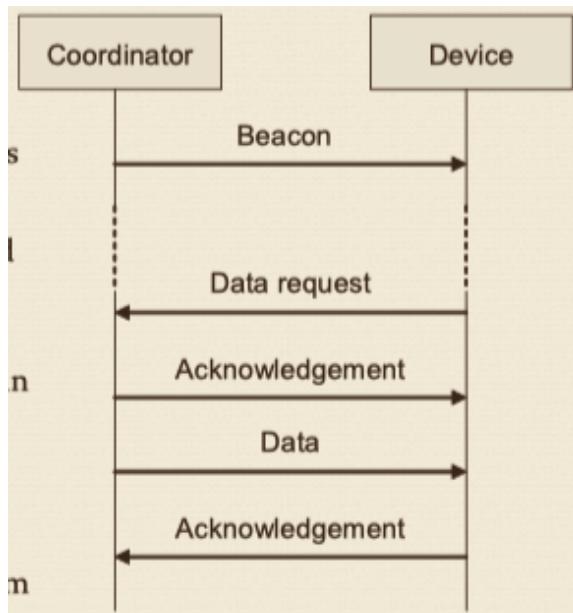
La topologia a stella utilizza solamente le prime due mentre con la topologia peer to peer tutti e tre i tipi di trasferimento dei dati sono possibili. L'implementazione di questi modelli di comunicazioni dipende dal fatto che la rete supporti o meno la trasmissione di beacon:

- Supponiamo di avere una rete che supporta i beacon, il caso più semplice è quello in cui abbiamo un trasferimento di dati dal device al coordinatore. Per prima cosa il dispositivo attende un beacon, quando l'ha ricevuto allora può inviare i dati al coordinatore. Il coordinatore può inviare poi al dispositivo un ACK che in realtà è opzionale.



- La situazione è più complessa se vogliamo avere un trasferimento di dati dal coordinatore ad un dispositivo in una rete con beacon abilitati. In questo caso il coordinatore memorizza i messaggi che

devono essere inviati ai vari dispositivi, ogni tanto il coordinatore invia un beacon al dispositivo (il dispositivo solitamente dorme e ogni tanto si accende per attendere il beacon e vedere i messaggi in attesa), questo fa una richiesta di dati al coordinatore e il coordinatore invierà un ACK al dispositivo. Se il dispositivo non riceve l'ack allora potrà anche inviare nuovamente la richiesta o i dati. Il coordinatore dopo l'ACK invia i messaggi che sono in attesa durante il CAP period. Nello slot di tempo successivo poi il dispositivo invierà al coordinatore un ACK, qua l'ACK è necessario perchè il coordinatore deve sapere se il messaggio è stato ricevuto o meno.

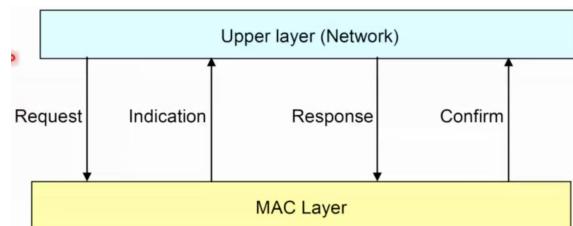


- Se siamo in una rete con i beacon possiamo anche inviare i dati utilizzando un trasferimento P2P. Se il mittente o il ricevente sono dispositivi allora possiamo usare uno dei metodi precedenti, se invece entrambi sono coordinatori allora il mittente deve sincronizzarsi con il beacon di destinazione e si comporterà come un dispositivo.

- Se siamo in una rete senza beacon il trasferimento da device a coordinatore è più semplice, in questo caso abbiamo il dispositivo che invia dati al coordinatore e il coordinatore deve inviare un ACK come risposta per fare in modo che il dispositivo sappia che tutto è andato a buon fine.
- Se siamo in una rete senza beacon e abbiamo il trasferimento da coordinatore a dispositivo allora il coordinatore memorizzerà messaggi che sono destinati ai vari dispositivi poi saranno i dispositivi a inviare una richiesta dati al coordinatore che risponderà con un ACK. Dopo l'ack il coordinatore invierà i messaggi in attesa, se non ci sono messaggi verrà spedito un messaggio vuoto. Il dispositivo invierà poi alla fine un ACK.
- L'ultimo caso è quello in cui abbiamo una non beacon enabled network e vogliamo avere una comunicazione peer to peer. Ogni dispositivo comunica con qualsiasi altro dispositivo presente nel suo range.

11.2 Mac Layer Services

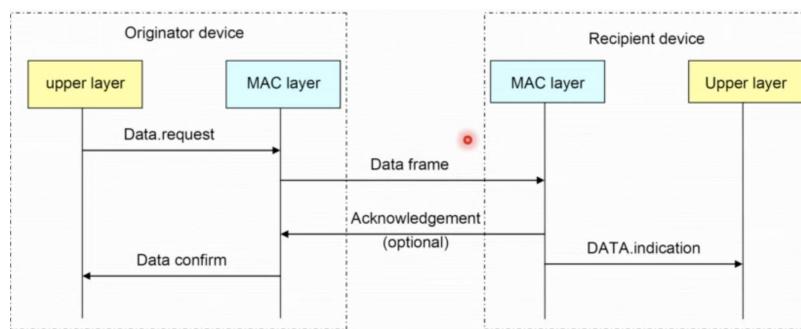
Il Mac Layer offre le sue primitive ovvero Request, Indication, Response e Confirm.



Nel Mac Layer possiamo implementare sia un Data Service sia un Management Service.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Data Service: il data service utilizza solamente le primitive request, confirm e indication. Request viene utilizzata dal livello più alto per inviare un messaggio ad un altro dispositivo. La primitiva Confirm è usata per inviare un report del risultato della trasmissione della precedente Data.Request al livello superiore. La Data.indication primitive invece corrisponde alla primitiva receive, viene generata dal MAC layer alla ricezione di un messaggio dal livello fisico per passare il livello ricevuto al livello superiore. Supponiamo di avere due dispositivi che vogliono comunicare, consideriamo il MAC layer in entrambi i dispositivi e il livello superiore al MAC Layer. Dal livello superiore del dispositivo originale viene inviata una Data.Request al MAC layer del dispositivo, dal MAC Layer del dispositivo 1 poi viene trasmesso un Data frame al MAC Layer del dispositivo di destinazione, qua la trasmissione è semplice ma la trasmissione dipende dai vari parametri della rete e viene utilizzato uno dei metodi che abbiamo visto in precedenza. Nel dispositivo di destinazione possiamo inviare un ACK che in generale è opzionale, dopo questo il Mac Layer del mittente invierà un Data.Confirm al livello superiore per notificare che il messaggio è stato inviato. Nella destinazione dal Mac Layer verrà inviato un Data.indication all'Upper Layer.



- Per quanto riguarda invece il Mac Layer Management Services, qua abbiamo delle funzionalità per l'inizializzazione del PAN, per con-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

trollare l'esistenza di un PAN e per la connessione, disconnessione di un dispositivo.

Il Mac Layer del 15.4 definisce molti altri servizi, i più importanti Management Services sono i seguenti:

Name	Request	Indication	Response	Confirm	Functionality
ASSOCIATE	X	O	O	X	Request of association of a new device to an existing PAN.
DISASSOCIATE	X	X		X	Leave a PAN.
BEACON-NOTIFY		X			Provides to the upper layer the received beacon.
GET	X			X	Reads the parameters of the MAC.
GTS	O	O		O	Request of GTS to the coordinator.
SCAN	X			X	Look for active PANs.
COMM-STATUS		X			Notify the upper layer about the status of a transaction that begun with a response primitive.
SET	X			X	Set parameters of the MAC layer.
START	O			O	Starts a PAN and begins sending beacons. Can also be used for device discovery.
POLL	X			X	Request for pending messages to the coordinator.

All'interno della tabella sopra abbiamo la X e la O per indicare i vari servizi, in particolare la O indica opzionale per i RFD.

11.2.1 Associate Protocol per il MAC Layer

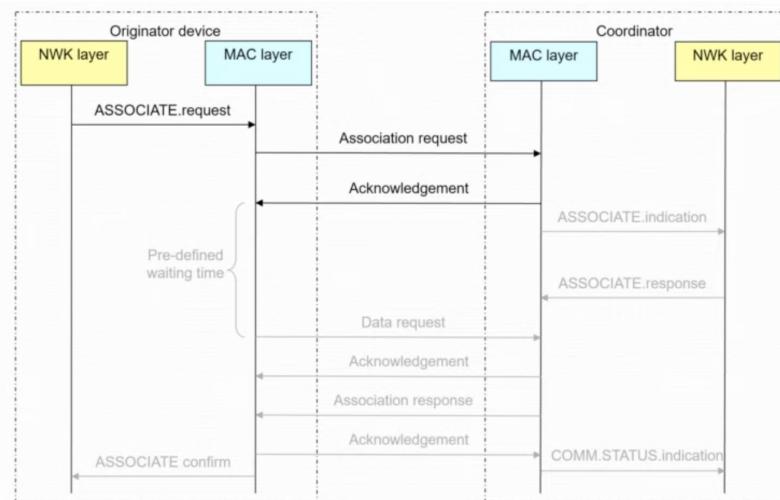
Consideriamo l'associate Protocol per il Mac Layer, è interessante perchè in questo protocollo vediamo come realmente la rete viene formata. Ha due parti:

- Una parte nel coordinatore
- Una parte nel dispositivo che avvia il protocollo quando vuole entrare nella rete.

Qua noi abbiamo il dispositivo che vuole entrare nella rete che deve fare una scansione per capire a quale PAN deve collegarsi. Quando ha

trovato la PAN a cui connettersi può invocare la `Associate.Request` primitive che prende come parametri l'identificatore del PAN e l'indirizzo del coordinatore.

Il coordinatore che riceve la richiesta deve decidere se accettare la richiesta e in caso deve fornire un indirizzo al dispositivo che ha richiesto l'accesso. L'association request viene inviata durante un CAP usando il protocollo CSMA-CA, quando il coordinatore riceve una richiesta di associazione manda subito un ACK ma non vuol dire che l'associazione sia stata accettata.



Vediamo come funziona nel dettaglio l'associate protocol:

- Nel dispositivo che si vuole collegare al coordinatore abbiamo che al livello di rete viene inoltrata una `Associate.Request` al Mac Layer, il Mac Layer del dispositivo trasmette una `association request` al Mac Layer del coordinatore che invia subito un ACK.
- La richiesta di associazione deve essere controllata dal coordinatore e questo avviene quando dal Mac Layer del coordinatore viene inviata una `Associate.indicator` all'upper layer.

- Tramite l'Associate.indicator il coordinatore viene effettivamente a sapere di questa richiesta, il coordinatore può sia rifiutarla sia accettarla. Se l'accetta deve selezionare un indirizzo di 16 bit per il dispositivo che deve connettersi. Dopo l'associazione l'end device dovrà poi sempre utilizzare questo indirizzo al posto del suo indirizzo originale. Per inviare questa risposta il network layer del coordinatore invia una Associate.response.
- Questa network response serve per inviare una risposta al dispositivo che ha chiesto di associarsi. Questa viene trasmessa durante una trasmissione indiretta.
- La trasmissione non può essere diretta perché il dispositivo non ha un indirizzo di rete. Quindi quello che succede è che il dispositivo una volta che riceve l'ACK saprà che la decisione del coordinatore richiederà un certo periodo di tempo, quindi il dispositivo attenderà un certo periodo di tempo, quando il tempo finisce il coordinatore avrà già inviato la Associate.Response. Il dispositivo allora può inviare una Data Request al coordinatore, il coordinatore invia un ACK e poi l'association response message che contiene il nuovo indirizzo di rete del dispositivo. Alla fine il dispositivo invia un ACK al coordinatore. Gli ACK sono necessari per fare in modo che i due MAC layer sappiano che il protocollo sta andando avanti.
- A questo punto il Mac Layer del dispositivo conosce il nuovo indirizzo di rete ma il network layer del dispositivo non lo sa ancora così come non lo sa il network layer del coordinatore. Questa ultima comunicazione viene effettuata tramite un Associate.confirm che viene inviata al network layer dal Mac Layer mentre nel coordinatore non abbiamo più primitive da utilizzare e quindi si utilizza il communication status per inviare al network layer del coordinatore l'informazione della nuova associazione.

11.2.2 MAC Layer Security

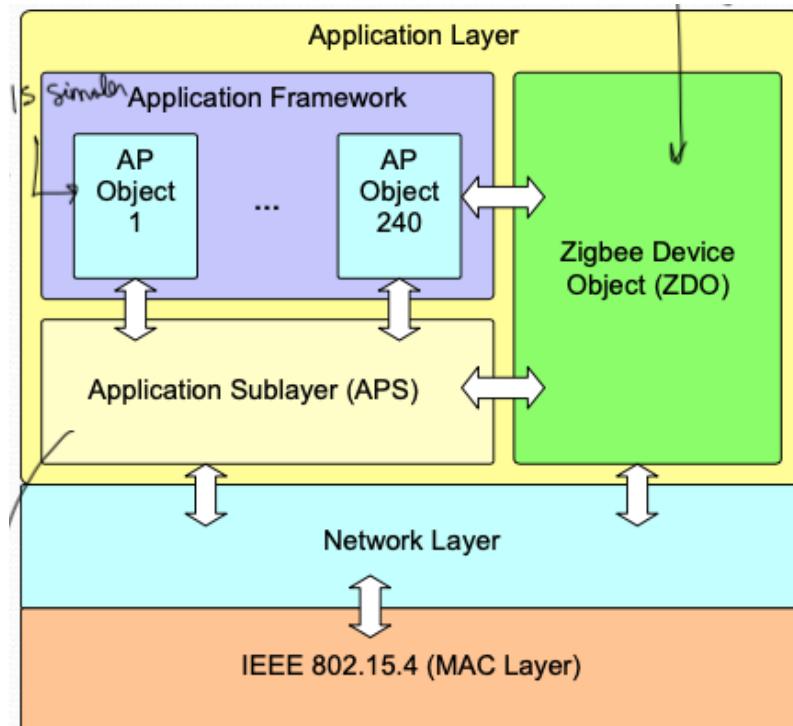
Il Mac Layer garantisce supporto per la sicurezza, solo un supporto base perchè tutto il resto viene lasciato ai livelli superiori. Non vengono implementate autenticazioni dei disposititivi con delle chiavi, vengono implementati solamente dei servizi basati su chiavi simmetriche. Tutti i servizi di sicurezza sono opzionali, si basano su chiavi simmetriche perchè le operazioni di codifica/decodifica sono abbastanza veloci. Questi fanno sì che le reti 15.4 sono più deboli di altri tipi di reti ma consideriamo anche che in questo tipo di reti non riusciremmo a implementare delle forme di crittografia troppo complesse.

Abbiamo anche altri tipi di controlli nel Mac Layer:

- Access Control: ogni dispositivo mantiene una lista di dispositivi con cui è in grado di comunicare.
- Data Encryption: si basa sulle chiavi simmetriche, questo si basa su due chiavi, possiamo usare una singola chiave che è condivisa all'interno della rete (group key) ma possiamo anche eseguire il setup di una chiave specifica per proteggere la comunicazione tra due dispositivi. Queste chiavi sono sempre simmetriche e vengono fornite dal Mac Layer. Se vogliamo usare sicurezza al Mac Layer dobbiamo quindi settare le chiavi e poi useremo la data encryption.
- Frame Integrity: aggiunge un codice calcolato dalla chiave simmetrica per fare in modo che nessun altro può prendere la chiave e cambiarla. Questo fornisce la sicurezza che i dati arrivano da un dispositivo che ha una chiave crittografica, se c'è un dispositivo all'interno della rete che ha la stessa chiave simmetrica degli altri dispositivi allora può attaccare l'integrità dei frame, non c'è modo di proteggersi da questo.
- Sequential Freshness: sono i sequence numbers associati ad ogni frame per evitare di ricevere più volte lo stesso frame.

11.3 Zigbee

Lo standard Zigbee è stato costruito sopra allo standard IEEE 802.15.4, questo specifica il livello rete e il livello applicazione.



In particolare all'interno del livello rete abbiamo il supporto a varie topologie della rete, a stella, ad albero, peer to peer e multi hop. A livello applicazione abbiamo vari componenti, questo livello è un transport layer, all'interno abbiamo tra gli altri l'application framework che è implementato da tutti i dispositivi Zigbee, qua dentro è presente la vera business logic della nostra applicazione. Al livello applicazione abbiamo anche due altri sublayer che sono lo Zigbee Device Object che fornisce servizi per organizzare l'association protocol o la gestione di richieste e l'application sublayer che è una sorta di transport Layer. La conseguenza di questo è che se siamo dei programmatori all'interno di una

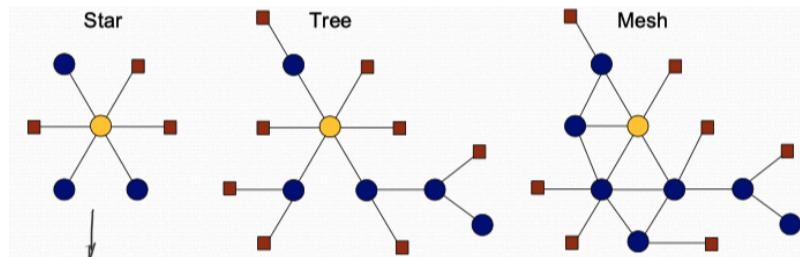
azienda, dovremo implementare solamente l'application framework, se invece vendiamo dispositivi Zigbee per personalizzarli dobbiamo implementare anche lo Zigbee Device, l'APS e il livello rete.

11.3.1 Network Layer

All'interno del network layer di Zigbee si distinguono tre tipi di dispositivi e tre topologie differenti della rete. Per quanto riguarda i tipi di dispositivi abbiamo:

- End-Devices: corrispondono ad un RFD o ad un FFD che lavora come un dispositivo semplice.
- Routers: un FFD capace di eseguire il routing.
- Coordinatore: un FFD che gestisce l'intera rete.

Abbiamo poi tre topologie differenti:



- A stella: questa è un mapping naturale della IEEE 802.15.4, è stata pensata per utilizzare il superframe.
- Ad albero: può utilizzare la struttura superframe.
- Mesh: la comunicazione funziona senza la struttura superframe.

Più nello specifico il Network layer di Zigbee è importante perché offre dei servizi che consentono la gestione dinamica della rete e questo nel caso di Zigbee è importante perché la rete è più dinamica perchè abbiamo tanti dispositivi all'interno della rete. Quello che rende complicato Zigbee rispetto ad esempio ad una LAN è proprio questa dinamicità. Questa dinamicità però rende Zigbee un protocollo smart. I servizi che vengono gestiti dal Livello di rete sono in particolare:

- Trasmissione di pacchetti
- Inizializzazione della rete
- Routing
- Gestione di connessione e disconnessione di dispositivi

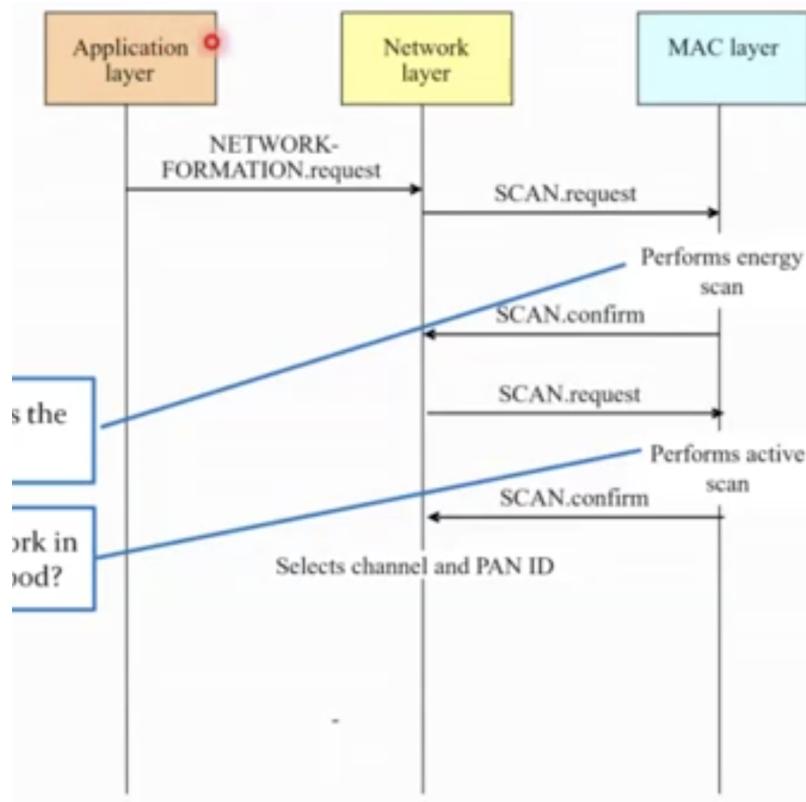
Ci sono anche altri servizi a disposizione, in particolare vediamo come viene creata la rete e come si entra all'interno della rete. In generale prima di comunicare con la rete un dispositivo Zigbee deve essere connesso e questo lo ottiene in due modi:

- Il dispositivo è esso stesso il coordinatore e quindi può creare una rete.
- Il dispositivo è un router o un end-device e quindi deve solamente aggiungersi ad una rete già esistente.

Il ruolo del dispositivo all'interno della rete viene scelto a tempo di compilazione, è una decisione che viene presa dal produttore che assegna il ruolo al dispositivo e questo è importante perchè oltre ad assegnare il ruolo si devono assegnare anche le funzioni e i meccanismi di funzionamento che sono tipici di quel tipo di dispositivo.

La creazione della nuova rete viene effettuata tramite la richiesta Network Formation che viene invocata dal coordinatore, questo coordinatore

non deve essere già in un'altra PAN perchè si può stare solamente in un PAN alla volta. Vediamo come funziona:



- Al livello applicazione viene invocato il Network Formation del Network layer.
- Il network layer esegue un energy scan del canale per capire se c'è rumore nel canale.
- Il Mac Layer restituisce il risultato dell'energy scan, a questo punto il Network layer richiede un secondo scan che è un active scan per cercare dei PAN che sono vicini. La motivazione per questa ricerca è che in una certa area non possiamo avere due reti con lo stesso

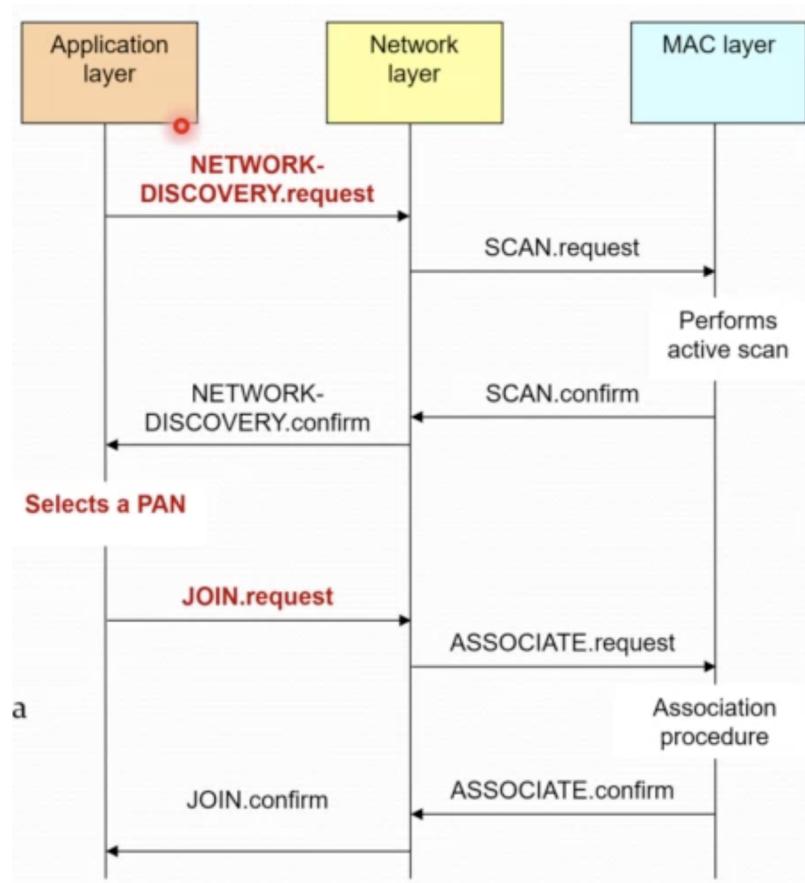
identificatore, quindi facciamo la ricerca in modo da selezionarlo uno differente.

- Il Mac Layer restituisce la lista degli identificatori esistenti e ora il Network layer ha tutte le informazioni per settare il PAN identifier, a questo punto il Network layer configura il Mac layer, da quanto riceve questa richiesta di start del PAN, il Mac layer potrà iniziare a inviare i beacon. Una volta che ha fatto questo il Mac Layer può rispondere con una conferma al Network layer che a sua volta conferma all'application layer che è stato creato il PAN.

Per entrare all'interno di una di queste reti ci sono due possibilità, l'end device o il router può comunicare con il coordinatore per richiedere l'accesso alla rete, questa è la join through association. Esiste poi anche la direct join in cui invece è lo stesso coordinatore presente nella rete o un router presente nella rete a richiedere ad un dispositivo di eseguire la join nella rete. Con questo secondo meccanismo non dobbiamo selezionare il PAN dal dispositivo, questa operazione di direct join però dovrà essere attivata da qualcuno. Noi ci concentriamo sulla join through association.

11.3.2 Join through association

Nello schema qua sotto abbiamo il funzionamento della Join through association dal punto di vista del dispositivo che vuole entrare nella rete.



Dall'altro lato c'è il Mac Layer del coordinatore che risponderà alle richieste del Mac Layer del dispositivo che vuole entrare nella rete. La join viene triggerata dal livello applicazione, per prima cosa però si deve fare una scansione della rete per trovare un canale che sia libero. L'active scan restituisce la lista delle reti che si trovano nelle vicinanze a cui posso collegarmi. All'application layer arriva la lista e devo selezionare un PAN, una volta che il PAN è stato selezionato inviamo una richiesta di join che specifica anche se il dispositivo si deve connettere come router o come end-device. Nel Mac Layer quando riceviamo la richiesta di associazione viene utilizzata la association procedure del mac layer e questo risponderà poi con una conferma al network layer che poi l'inoltrerà all'Application

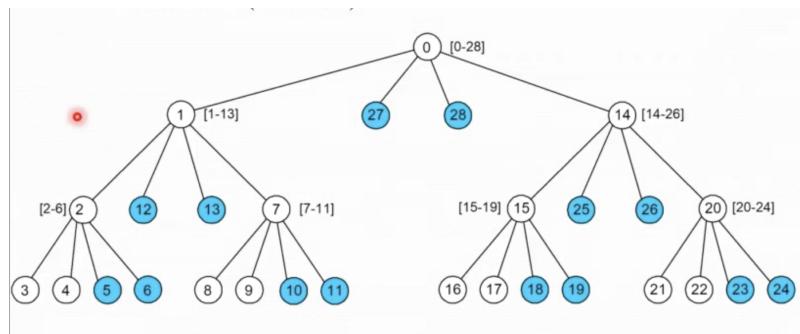
Layer.

Quando ci colleghiamo alla rete, questa può avere la sua topologia, se abbiamo la stella, l'end device si connette solamente come end-device e non come router. Se invece stiamo usando una tree based topology allora il dispositivo si può connettere come router o come end device.

Il meccanismo per associarsi ad una rete definisce in modo implicito un padre e questo vuol dire che la rete avrà una sorta di topologia con un albero in cui gli end-devices saranno le foglie mentre i router saranno i nodi interni e il coordinatore sarà il nodo root. Questa topologia ad albero viene utilizzata per assegnare degli indirizzi ai vari dispositivi nella rete, per assegnare questi indirizzi il coordinatore Zigbee deve decidere vari parametri ovvero il massimo numero di router, il massimo numero di end-devices e la massima profondità dell'albero.

Ad ognuno dei router della rete viene assegnato un set di indirizzi in modo che poi abbiano indirizzi da assegnare ai figli ovvero agli end-devices. Quando un dispositivo viene aggiunto all'interno della rete si cerca di posizionarlo il più alto possibile all'interno dell'albero in modo da minimizzare il numero di hops.

Questo meccanismo per l'assegnazione funziona indipendentemente dal tipo di topologia che viene utilizzata.



Supponiamo di avere una situazione come quella della foto, in cui assumiamo di avere il massimo numero di routers per livello che è $R_m = 2$, il massimo numero di end devices figli di un router che è $D_m = 2$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

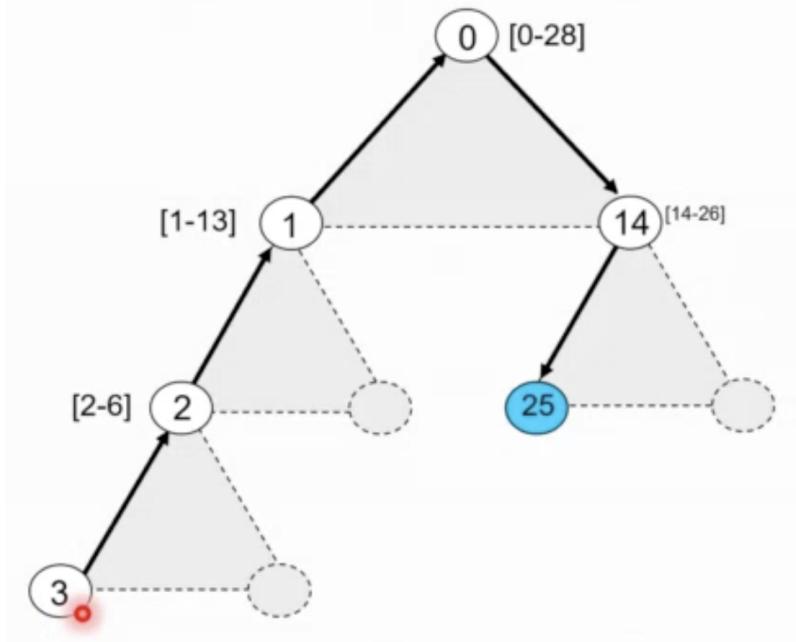
e la massima profondità dell'albero che è $Lm = 3$. I nodi celesti sono end-devices mentre invece i bianchi sono i router. Con questi parametri possiamo determinare la struttura completa dell'albero in cui abbiamo 29 dispositivi, non ne possiamo avere di più con questi parametri, se vogliamo una rete più grande dobbiamo configurare il coordinatore con un numero differente di parametri. Il coordinatore riserva l'indirizzo 0 per se stesso, poi tiene gli ultimi due, 28 e 27 per gli end-devices, gli altri invece sono divisi tra i due router figli, in particolare da 1 a 13 al primo router e 14-26 al secondo router. Ricorsivamente questi due router che sono figli del coordinatore funzioneranno nello stesso modo. Assumiamo che nessun nodo è stato preso come secondo router come secondo figlio del coordinatore 0 e che arriva un altro end-device, questo nodo rimane fuori dalla rete anche se ci sono degli indirizzi liberi perchè nel sottoalbero a sinistra siamo già pieni e siamo già all'altezza massima. Questo può essere un problema e ci sono state delle proposte per rendere questo meccanismo più dinamico in modo da offrire comunque la possibilità di unirsi alla rete anche in questa situazione. Il protocollo utilizza però questo meccanismo quindi c'è da adattarsi, per evitare che questo accada dovremmo definire i parametri in modo che siano un po' più grandi rispetto ai parametri minimi di cui abbiamo bisogno. La seconda considerazione è che questo metodo è stato pensato in modo molto rigido in modo che sia anche efficiente e semplice, il router a cui chiediamo l'accesso alla rete infatti può prendere la sua decisione automaticamente e non ci sono problemi di race condition. Se avessimo utilizzato metodi differenti il router avrebbe dovuto implementare una sorta di meccanismo di consenso distribuito.

Al livello di rete dobbiamo definire anche un protocollo di routing, Zigbee in particolare ci fornisce per questo vari metodi:

- Broadcasting
- Mesh Routing

- Tree Routing

Gli ultimi due sono point to point. Il tree routing è il più semplice, se abbiamo ad esempio un dispositivo che è 3 e che vuole comunicare con 25, dovrà passare il pacchetto a 2 che è il padre. Il nodo 25 è fuori dal range di 2, quindi anche 2 dovrà passare di nuovo il pacchetto al padre, quindi al nodo 1. Il pacchetto arriverà al coordinatore che saprà dove si trova il destinatario del pacchetto quindi inoltra il pacchetto al nodo corretto, quindi a 14, questo poi inoltra direttamente il pacchetto a 25.

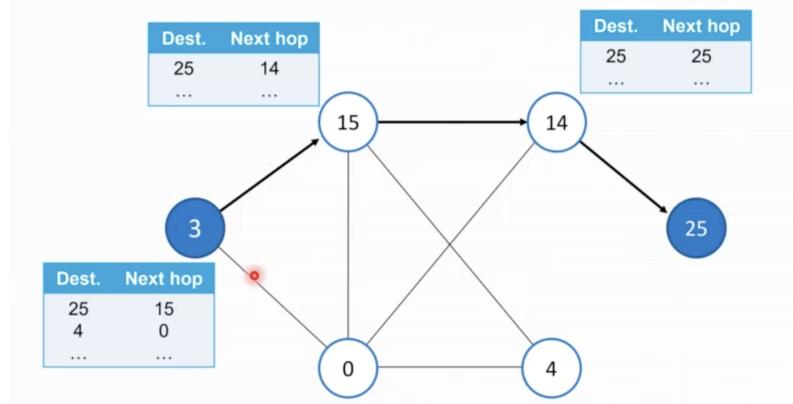


Tutte queste comunicazioni sono tra i router tranne l'ultima che è tra il router e l'end device. Per tutte queste comunicazioni è l'end device che deve implementare il corretto meccanismo di comunicazione. Per l'ultimo hop il 14 potrà avvertire della presenza del messaggio nel suo beacon e poi 25 richiederà il messaggio a 14.

Un'altra considerazione riguardo al tree routing, qua noi utilizziamo dei beacon ma abbiamo un problema perchè due router consecutivi che

sono nel range non possono avere l'activity period che si sovrappone. Zigbee non specifica dei meccanismi per risolvere il problema ma noi dobbiamo essere sicuri che due router vicini abbiano degli active period che sono completamente desincronizzati e quindi l'active period di un router deve capitare durante l'inactive period dell'altro. Questo è importante perchè se abbiamo una collisione poi non si riesce ad avere una comunicazione tra i figli dei due router.

Per quanto riguarda il mesh routing, questo si basa su AODV, ogni router ha una routing table, sono le stesse di AODV, mantiene la destinazione, il prossimo hop e una serie di informazioni riguardo al percorso da seguire.



Se vogliamo fare una comunicazione tra 3 e 25 sappiamo in ogni nodo dove dobbiamo andare per inviare l'informazione.

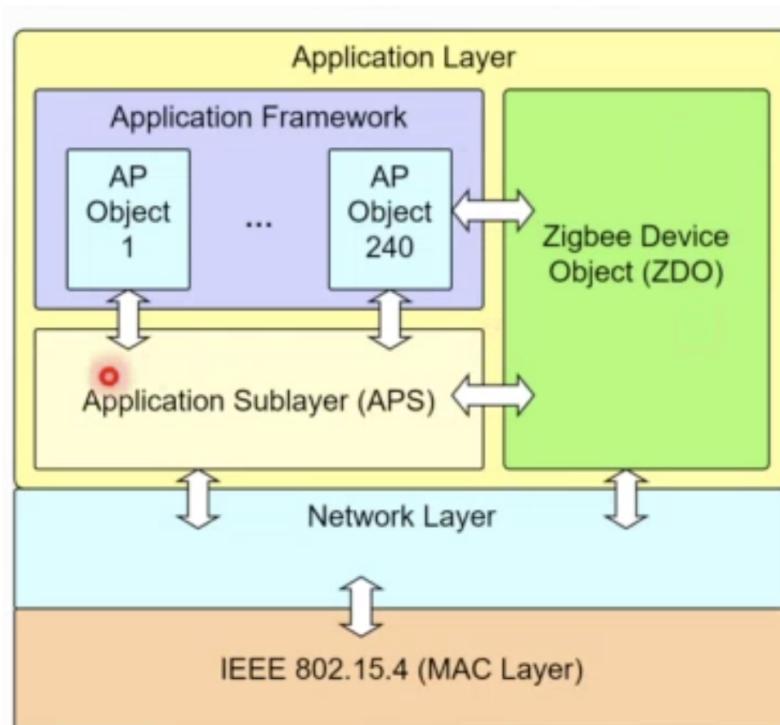
In generale per implementare il mesh routing guardiamo la destinazione nella routing table, se non c'è quella entry allora facciamo una ricerca. Il tree routing e il mesh routing possono essere presenti insieme e possiamo passare da un routing all'altro durante lo scambio dei messaggi. Se usiamo il Mesh routing non possiamo utilizzare il beaconing mentre se abbiamo l'albero si. Se non usiamo il beaconing allora utilizzeremo solamente il mesh routing, se invece usiamo il beaconing e se

abbiamo una struttura ad albero della rete allora possiamo utilizzare il tree routing.

Il route discovery nel mesh routing funziona con il broadcast, contiene il request ID e il path cost inizialmente è 0, viene propagata la richiesta all'interno della rete, ogni router intermedio può rispondere se abbiamo una entry per la destinazione, la destinazione risponderà sempre.

11.3.3 Application Layer di Zigbee

L'application layer comprende vari componenti:

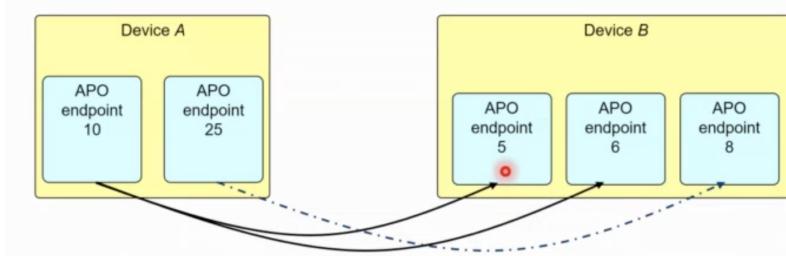


- Zigbee Device Object: è una applicazione specifica per la gestione del dispositivo.

- Application sublayer che permette di gestire il binding e il discovery services.
- Application Framework: è un container per l'application Object. Ogni Application Object definisce una sua logica. I differenti Application Object potrebbero non essere collegati tra loro. L'application object interagisce con l'hardware del dispositivo per leggere o scrivere dati e fornirli ai sublayer.

Application Framework

All'interno dell'Application Framework possono esserci fino a 240 Application Object, ognuno di questi è unicamente identificato all'interno della rete. Nella prima versione di Zigbee a questo livello potevamo inviare query alle APO tramite un Key Value Pair Data Service, questo servizio è poi sparito nelle versioni più recenti di Zigbee. In generali gli APO più complessi possono avere uno stato più complesso. Vediamo un esempio di APO:



Qua abbiamo due dispositivi, il primo con due endpoint e il secondo con tre endpoint, ad esempio un endpoint potrebbe essere una luce e quella collegata potrebbe essere uno switch. Implementare una applicazione semplice di questo genere necessita solamente di un ON-OFF Attribute e poi i due endpoint del dispositivo A possono accendere o spegnere le luci del dispositivo B.

Application Support Sublayer

L'Application Support Sublayer definisce endpoints, cluster, profile IDs e Device IDs. APS è responsabile per:

- I data service forniscono supporto per l'affidabilità della comunicazione e inoltre filtrano i pacchetti.
- I Management services permettono di mantenere una binding table locale, una tabella di gruppi locale e una address map locale.

Per quanto riguarda gli endpoints, sono identificati da un numero compreso tra 1 e 240, un nodo Zigbee può eseguire varie applicazioni contemporanea, una per ogni APO. Un endpoint è visto come una sorta di socket. Per comunicare con un endpoint devo essere connesso con uno specifico APO. Ogni endpoint separa ogni APO e quindi abbiamo che ogni APO potrebbe rispondere ad uno specifico protocollo.

Un altro concetto dell'APS è l'idea del Cluster. Il cluster è un protocollo che può essere utilizzato per implementare le funzionalità di una applicazione. I cluster definiscono gli attributi che contengono le informazioni riguardo all'APO. Un cluster è definito da un identificativo di 16 bit che è locale all'interno del profilo, questo vuol dire che Zigbee in generale definisce profili, ogni profilo comprende varie applicazioni, dentro ad un singolo profilo abbiamo un set di cluster ognuno rappresentato da un identificatore di 16 bit. Profili differenti possono essere all'interno dello stesso cluster o possono essere in cluster differenti.

In particolare esistono vari tipologie differenti di cluster, ad esempio il basic serve per ottenere informazioni basilari sul dispositivo che ha un cluster ID = 0. Poi c'è il power configuration cluster che serve per impostare ad esempio la temperatura minima e massima del dispositivo. L'on-off clustering è quello che ci permette di accendere o spegnere qualcosa, per questa ragione questo cluster potrebbe essere utilizzato in profiles differenti.

Cluster Name	Cluster ID
Basic Cluster	0x0000
Power Configuration Cluster	0x0001
Temperature Configuration Cluster	0x0002
Identify Cluster	0x0003
Group Cluster	0x0004
Scenes Cluster	0x0005
OnOff Cluster	0x0006
OnOff Configuration Cluster	0x0007
Level Control Cluster	0x0008
Time Cluster	0x000a
Location Cluster	0x000b

All'interno dell'APS abbiamo anche l'Application Profile che sono le specifiche del comportamento di una applicazione. Questo in generale riguarda un certo numero di dispositivi, un esempio di Application Profile può essere ad esempio Home Automation in cui abbiamo varie attività che vogliamo effettuare, ad ognuna corrisponderà un certo cluster. Ad esempio avremo un cluster riguardante l'allarme, un cluster riguardo la temperatura, cluster che riguarda il multimedia. L'application profile specifica tutte le attività che possono essere effettuate dai dispositivi che lavorano sotto lo stesso application profile, in particolare specifica un set di dispositivi e un set di cluster che il dispositivo deve implementare all'interno di questo profilo. I profili possono essere visti come un dominio, viene definito un certo numero di public profiles che sono indicati all'interno della libreria di Zigbee. Questi sono specificati da identificatori in un certo range, gli specifici produttori possono poi decidere i loro profili personalizzati, questi possono avere un profile ID in un range differente. Tutti i dati che vengono ricevuti o inviati in una rete Zigbee devono essere taggati con un application profile ID, all'interno di una rete Zigbee possiamo avere un qualsiasi numero di Application Profiles.

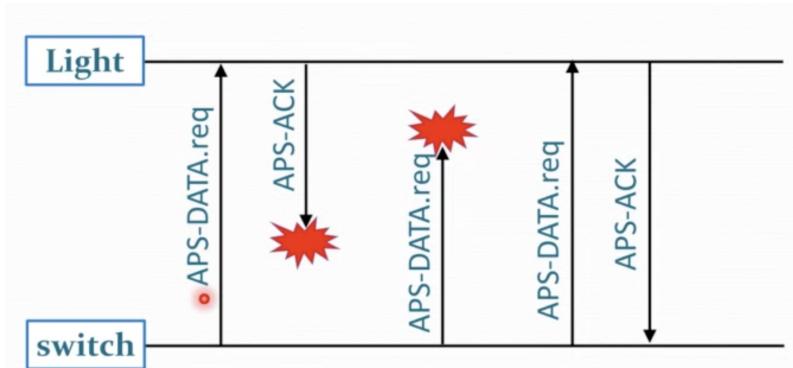
Profile ID	Profile name
0101	Industrial Plant Monitoring
0104	Home Automation
0105	Commercial Building Automation
0107	Telecom Applications
0108	Personal Home & Hospital Care
0109	Advanced Metering Initiative

L'ultimo elemento definito dall'APS è il Device ID. Il device ID non è del tutto necessario se la rete è autonoma perchè ogni dispositivo è specificato interamente dall'application profile e dal cluster ID. In ogni caso il cluster ID e l'Application profile ci indicano cosa il dispositivo può fare ma non chi è quel dispositivo. Ad esempio un dispositivo può essere una luce o un forno ed entrambi rispondono all'on-off cluster. L'informazione riguardo a quale è il dispositivo che stiamo utilizzando è necessaria per gli umani per supportare la configurazione della rete o per analizzare il contenuto della rete. Per questo motivo c'è il Device ID che è un identificatore di 16 bit che permette agli umani di capire che dispositivo è quello che stanno usando. Allo stesso tempo questa informazione non viene utilizzata per la gestione della rete, se sappiamo che un dispositivo ha un certo ID allora questo non mi dice a quali cluster e comandi risponderà, quindi dal punto di vista della rete quello che è importante è il profile ID e il cluster ID, dal punto di vista dell'installazione e della gestione fatta da un umano è importante anche il device ID.

Per quanto riguarda i servizi del layer dell'Application Support, i principali sono quelli che riguardano il data service e il binding. Per quanto riguarda il data service abbiamo il supporto alla trasmissione di pacchetti da un endpoint ad un altro, questo può essere implementato con il direct addressing o con l'indirect addressing. Il direct addressing è quello più comune, il mittente specifica il network address dell'APO che deve ricevere il messaggio, nell'indirect invece viene utilizzato il binding

service. In ogni caso il data service viene implementato utilizzando le primitive request, confirm e indication.

Qua è rappresentato il funzionamento:



Uno switch può inviare una richiesta per capire lo status della luce per capire se è accesa o meno, la luce manda un ACK che potrebbe essere perso, lo switch ci riprova ma comunque l'ACK potrebbe di nuovo andare perso, poi ci riprova di nuovo e la luce manda indietro l'ACK, a questo punto smette di inviare richieste. Si tratta di un comportamento classico.

Invece per quanto riguarda il binding abbiamo che un endpoint può connettersi a uno o più endpoint e questo è chiamato binding e supporta una comunicazione unidirezionale da un dispositivo ad un set di endpoint in un altro dispositivo. Questo set può essere configurato solamente dal ZDO del coordinatore o del router utilizzando due primitive del binding.

Il binding fornisce un modo di specificare la destination address, questo è chiamato indirect addressing. Questo è perchè i messaggi normalmente sono inviati specificando l'APL e la destination address ma se i dispositivi non sono molto potenti potrebbero non essere in grado di determinare l'address della destinazione, questa explicit address quindi non è possibile e quindi viene utilizzato l'indirect addressing. Con l'indirect addressing sarà invece il parent router che capirà dove deve essere inviato quel pacchetto.

Il binding e l'unbinding utilizza due primitive, Bind.Request e Unbind.Request. Il binding con l'indirect addressing funziona in questo modo:

- Viene utilizzata la binding table in cui manteniamo le associazioni tra source address e cluster identifier.
- Vengono creati dei pair in cui mettiamo l'endpoint destinazione e l'address di destinazione. Sarà il router che determinerà queste due informazioni.
- La binding table viene memorizzata nell'application sublayer e solitamente viene inizializzata quando deployamo la rete.

Nel caso della luce e dello switch, è complicato per lo switch sapere a quale luce deve collegarsi, questa informazione è conosciuta però all'installatore. Invece di operare direttamente sull'end device l'installatore può settare la binding table entry nel coordinatore e in questo modo quello switch e quella luce verranno collegati tra loro e tutti i messaggi dello switch verranno automaticamente inviati a quella luce.

Vediamo un esempio di binding table:

Src Addr (64 bits)	Src EP	Cluster ID	Dest Addr (16/64 bits)	Addr/Grp	Dest EP
0x3232...	5	0x0006	0x1234...	A	12
0x3232...	6	0x0006	0x796F...	A	240
0x3232...	5	0x0006	0x9999	G	—
0x3232...	5	0x0006	0x5678...	A	44

Qua abbiamo la destination e la source address che sono in forma di Mac address, facciamo questo perchè in una rete i dispositivi possono unirsi e disconnettersi per varie ragioni, quando si riconnettono prendono un indirizzo di rete differente, questo forzerebbe quindi una riconfigurazione della binding table ogni volta che qualcuno si connette.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Questo non permette ai router di implementare il routing direttamente, dobbiamo avere una tabella in cui abbiamo la traduzione in indirizzi di rete per implementare la comunicazione.

Possiamo vedere dall'esempio che dalla stessa fonte possiamo avere pacchetti che sono prodotti dallo stesso endpoint. Ad esempio se il primo trasmette dall'endpoint 5, questo sarà inoltrato ad un certo indirizzo e ad un certo endpoint. C'è anche la possibilità di settare un gruppo come destinazione, siamo nella terza riga in questo caso, qua abbiamo il flag G e l'endpoint di destinazione non rappresenterà un singolo address ma un group address quindi il messaggio verrà inviato ad un set di dispositivi.

Se consideriamo le prime due righe abbiamo due endpoint differenti, questa informazione è rilevante per capire a quale indirizzo di destinazione e a quale endpoint deve essere inoltrato il pacchetto.

Se consideriamo la prima riga e l'ultima vediamo anche un altro caso particolare, abbiamo che il mittente, l'endpoint mittente e il cluster ID sono uguali ma la destinazione è differente così come anche l'endpoint di destinazione. Questo perchè il router decide di inoltrare il pacchetto verso due destinazioni differenti.

Per mantenere queste associazioni tra Mac Address e Network address abbiamo a disposizione una Address Map. Questa Address Map mantiene l'associazione tra il MAC address di 64 bit e il network address di 16 bit.

IEEE Addr	NWK Addr
0x0030D237B0230102	0x0000
0x0030B237B0235CA3	0x0001
0x0031C237b023A291	0x895B

Perchè separarle? Abbiamo detto che è importante per evitare di modificare sempre la binding table in modo che rimanga sempre valida, quando un nodo si disconnette, l'informazione della disconnessione viene

invia in broadcast a tutti i nodi della rete che quindi aggiorneranno la Address Map. Quando un nodo si inserisce nella rete avviene la stessa cosa e l'informazione viene inviata in broadcast nella rete. Questa Address Map quindi viene mantenuta aggiornata ad ogni nuova join della rete e ad ogni abbandono e quindi permette di avere un binding valido per lunghi periodi senza necessità di fare altri cambiamenti nella binding table. L'inizializzazione della Binding table dovrebbe essere implementata dal tecnico che installa il dispositivo perché sa quale dispositivo deve comunicare con un altro, ad esempio la luce con lo switch per l'accensione.

11.3.4 Zigbee Device Object

Lo Zigbee Device Object è una speciale applicazione collegata all'endpoint 0, implementa l'end-device o il router o il coordinatore in base alla specifica situazione e configurazione. Lo ZDO è definito da uno speciale profilo chiamato Zigbee Device Profile, in particolare specifica tutti i servizi relativi al binding (servizio implementato dall'Application support sublayer), la decisione sull'utilizzo del binding parte dal ZDO.

Lo ZDO gestisce i servizi relativi a:

- Scoperta di dispositivi e servizi
- Gestione del binding e dell'application level
- Gestione della rete
- Gestione dei nodi

Per quanto riguarda il servizio di scoperta di dispositivi e servizi, questo viene specificato dallo Zigbee Device Profile. Dobbiamo considerare che in una rete Zigbee abbiamo molta dinamicità e tutti i dispositivi sono autonomi. Un dispositivo connesso alla rete che ha appena scoperto non sa niente di questa rete e dei dispositivi che sono all'interno,

conosce solamente i servizi che lui fornisce, per diventare operativo deve connettersi ai servizi degli altri dispositivi e gli altri dispositivi devono connettersi ai suoi servizi. Tutto questo è implementato dal Zigbee Device Profile. I servizi di cui stiamo parlando non sono altro che cluster che rispondono. Questi meccanismi sono utilizzati anche da un installatore o da una persona che deve mantenere la rete per capire quali sono i dispositivi nella rete e cosa fanno. Il processo di scoperta dei dispositivi all'interno della rete viene svolto dal Device Discovery. Questo protocollo è complesso perché la rete è distribuita e ci sono molti dispositivi, il problema è a quale dispositivi inviare le query riguardo al device discovery. La query viene inviata al coordinatore che poi può rispondere e può anche richiedere informazioni ai suoi router, ognuno dei router restituisce informazioni relative agli indirizzi dei suoi dispositivi associati. Ricordiamo che il coordinatore e il router mantengono delle address map, se i dati che sono in queste tabelle sono consistenti allora il coordinatore può rispondere direttamente alla richiesta. Potrebbe essere necessario inoltrare una query ad un router perché la tabella potrebbe non essere ancora aggiornata ad un certo punto. Alla fine il coordinatore restituisce al richiedente una lista con tutte le informazioni.

Il secondo step che segue il device discovery è il service discovery, questo segue lo stesso processo, abbiamo una richiesta al coordinatore che potrebbe poi inoltrare la richiesta ai vari router e risponde alla query che permette di identificare uno specifico cluster ID o un device descriptor. Questo protocollo è, anche in questo caso, mantenuto dal coordinatore.

Un altro servizio del ZDO è la gestione dei binding, questo è un meccanismo implementato dall'Application Sublayer e vengono utilizzate delle tabelle che devono essere riempite, è responsabilità dello ZDO di riempire queste tabelle. Lo ZDO invoca le primitive dell'EPS per riempire le binding table ma questa azione solitamente viene coordinata dall'installatore che lavora su una interfaccia per creare un binding tra due dispositivi specifici.

Lo ZDO fornisce anche servizi per gestire la rete, eseguire il setup, in-

vocare primitive e avviare la rete richiedendo la trasmissione dei beacon. Per quanto riguarda il node management lo ZDO inoltre risponde anche alle network request implementando tutta la politica per la gestione della rete.

ZDO fornisce anche supporto per la sicurezza, l'obiettivo è quello di proteggere i singoli dispositivi della rete ma non le applicazioni individuali sullo stesso dispositivo. Questo vuol dire che all'interno dello stesso dispositivo le varie applicazioni non sono protette l'una dall'altra, questa non è una assunzione drammatica perchè tutte le varie applicazioni all'interno di un dispositivo sono scritte dallo stesso programmatore o dalla stessa azienda quindi non ha senso proteggerle una dall'altra. Questo permette un utilizzo delle stesse chiavi tra differenti livelli dello stesso dispositivo e questo fa sì che il tutto sia più semplice. Vengono utilizzate varie chiavi, una è quella che viene usata per garantire la sicurezza a livello di rete e viene condivisa tra tutti i dispositivi della rete, poi abbiamo una Single Key per link per la sicurezza del singolo dispositivo.

I security services sono gli stessi garantiti dal Mac Layer:

- Abbiamo la sicurezza dell'invio dei messaggi (Message Integrity).
- Autenticazione dei dispositivi.
- Criptazione dei messaggi.
- Message Freshness per evitare duplicati.

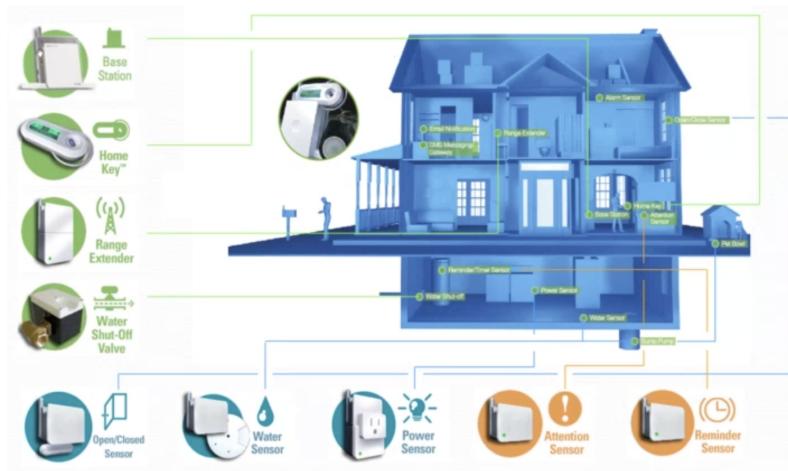
La sicurezza viene garantita da un Trust Center che è un modulo dell'application Layer, tipicamente è allocato nello ZDO e questo è il componente che fornisce le chiavi ai dispositivi che sono all'interno della rete, tramite il Trust Center possiamo distribuire le chiavi ai vari dispositivi. Come funziona? I dispositivi che stanno all'interno della rete utilizzano una master key per connettersi al trust center, la master key può

essere pre-assegnata al dispositivo quando il dispositivo viene prodotto (può essere nel firmware) oppure può essere inserita all'interno del dispositivo utilizzando un mechanism.

Tramite questa Master Key viene stabilita una connessione e poi possiamo richiedere le network keys o le link keys. L'encryption utilizzata da Zigbee fornisce un buon livello di sicurezza ma non eccezionale (AES-128), è comunque abbastanza per i dispositivi che stiamo considerando (low power device). Il fatto che i dispositivi che stiamo utilizzando sono poco potenti fa che venga utilizzata una Symmetric Encryption e decryption usando chiavi che vengono fornite dall'application layer. In ogni caso ci sono dei commandi che non possono essere criptati, in particolare l'association methods perchè in quel momento non abbiamo ancora la network key e la link key.

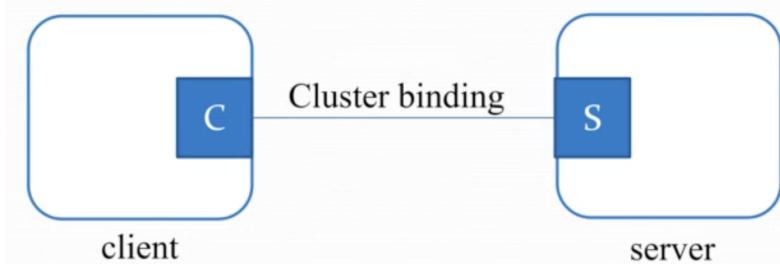
11.3.5 Zigbee Cluster Library

È una specifica dei cluster che possono essere implementati da ogni applicazione Zigbee. In questi cluster ci sono i possibili comportamenti dei vari dispositivi. Vediamo un esempio di cluster nel caso dell'home automation:



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

In un sistema di home automation abbiamo molte applicazioni che vengono eseguite allo stesso momento. Abbiamo anche sensori, switch per le porte, sensori di energia. Potremmo avere dei sensori per la sicurezza, in generale possiamo avere un grande numero di sensori e di attuatori. La Zigbee Cluster Library specifica il comportamento di questi dispositivi basandosi su un client-server model.



Il dispositivo che memorizza i dati è il server del cluster mentre invece il dispositivo che richiede l'accesso e la modifica dei dati è il client. Se pensiamo ad una applicazione che controlla una luce, la luce avrà un attributo che è ON/OFF, il client sarà lo switch che accende o spegne il server che invece sarà la luce.

I cluster sono raggruppati in Functional Domains:

General: to access and control attributes of any device, irrespective of their functional domain

Closures: for shade controllers, door locks etc

HVAC: for pumps (fan, heating, dehumidification etc.)

Lighting: to control lights

Measurement and sensing: illuminance, presence, flow, humidity etc.

Security and safety: security zone devices etc.

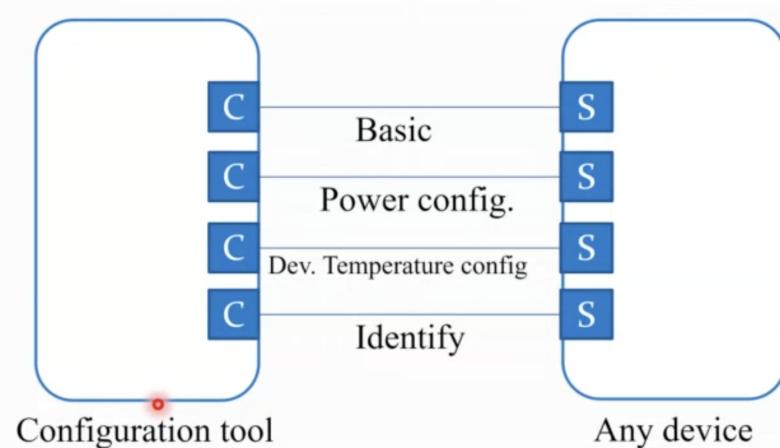
Protocol interfaces: to interconnect with other protocols

I comandi sono messaggi con un formato specificato dalla Zigbee

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

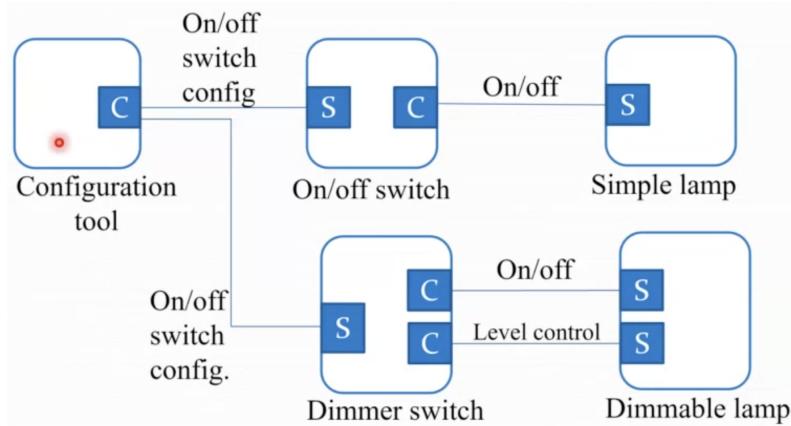
Cluster Library, i comandi possono essere degli attributi di lettura/scrittura o possono configurare un report che sarà molto utile se vogliamo avere una lettura periodica di un certo sensore. Altri comandi invece riguardano la scoperta di dispositivi o di cluster.

Vediamo una configurazione tipica tra un configuration tool e un qualsiasi dispositivo:



L'installatore solitamente ha un configuration tool con cui può applicare una certa configurazione nei vari dispositivi. Questo configuration tool deve potersi collegare ai vari dispositivi per applicare le varie impostazioni nei vari dispositivi o anche per leggere. Nell'esempio sopra abbiamo che il dispositivo risponde a varie richieste del configuration tool, prima otteniamo le informazioni del dispositivo, poi possiamo ottenere informazioni riguardo alla configurazione energetica del dispositivo, poi possiamo settare alcune impostazioni del dispositivo in termini di temperatura, poi con Identify possiamo far lampeggiare il dispositivo (è utile se stiamo installando tanti dispositivi, sono tutti identici quindi quello che li differenzia è dove sono posizionati e quali hanno già un binding, se ho un sensore e lo metto in una certa stanza devo capire quale è il dispositivo e quindi se questo lampeggia capisco che è proprio quello che sto configurando in quel momento).

Vediamo un altro esempio di come possiamo configurare un set di APO:



In questo caso vogliamo configurare un set di APO, abbiamo uno switch ON/OFF che è collegato con una "Simple Lamp", dobbiamo creare l'associazione tra la lampada e lo switch, allo stesso modo dobbiamo anche creare l'associazione tra il dimmer switch e la dimmable lamp. Questa associazione viene creata dal Configuration Tool, lavorando come un client setta la configurazione del dispositivo On/Off switch in modo tale che quando questo vuole accendere o spegnere la lampada andrà a trovare il dispositivo corrispondente e a fare la richiesta di accensione o spegnimento. Lo stesso vale per la configurazione del Dimmer Switch. Il configuration Tool deve anche configurare la Simple Lamp. Tipicamente quello che succede è che l'installer ha un tablet e questo si connette al Zigbee Network, usando un'applicazione puoi accedere alle informazioni riguardanti la rete. Quando eseguo questa configurazione ogni dispositivo sa cosa è e cosa può fare, una lampada per esempio sa di essere una lampada ma è necessario eseguire una procedura di commissioning in modo che la lampada sia connessa ad un on/Off switch che permetta l'accensione. È necessaria eseguire questa procedura di commissioning, è simile a quello che succede quando abbiamo una connessione tra le cuffie e lo smartphone, in questo caso la cuffia sa che è una cuffia e lo

smartphone sa di essere lo smartphone.

Pensiamo al caso in cui dobbiamo installare centinaia di dispositivi in una smart home, la complessità è tanta perchè ogni dispositivo deve essere installato in un posto specifico. La configurazione è complessa e quindi serve un modo per renderla automatica e renderla più semplice. Una volta che abbiamo completato questa procedura poi la rete è completamente autonoma.

Ogni dispositivo ha una serie di informazioni che gli vengono assegnate:

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	<i>ZCLVersion</i>	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x01	M
0x0001	<i>ApplicationVersion</i>	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0002	<i>StackVersion</i>	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0003	<i>HWVersion</i>	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0004	<i>ManufacturerName</i>	Character string	0 – 32 bytes	Read only	Empty string	O
0x0005	<i>ModelIdentifier</i>	Character string	0 – 32 bytes	Read only	Empty string	O
0x0006	<i>DateCode</i>	Character string	0 – 16 bytes	Read only	Empty string	O
0x0007	<i>PowerSource</i>	8-bit enumeration	0x00 – 0xff	Read only	0x00	M

Tra tutte queste in particolare abbiamo informazioni che riguardano la Power Source che è un numero di 8 bit, il significato è il seguente:

Attribute Value	Description
0x00	Unknown
0x01	Mains (single phase)
0x02	Mains (3 phase)
0x03	Battery
0x04	DC source
0x05	Emergency mains constantly powered
0x06	Emergency mains and transfer switch
0x07 – 0x7f	Reserved

Questo ci indica anche il tipo di batteria a cui stiamo connessi, se è una situazione di emergenza o meno ad esempio.

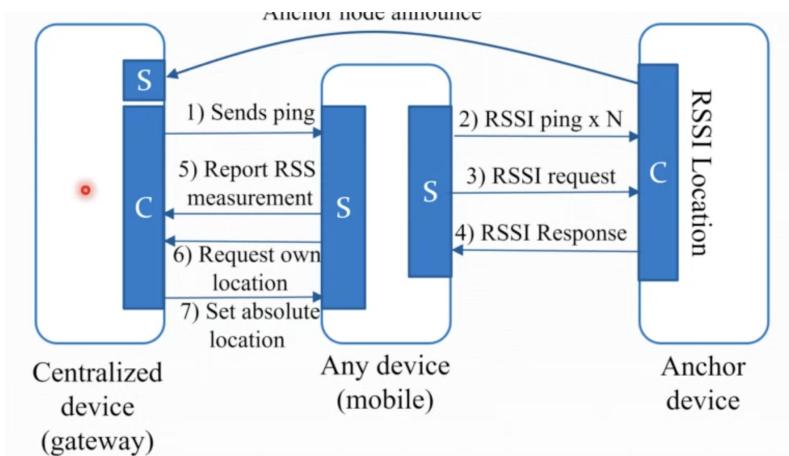
Questi dispositivi hanno anche l'attributo temperature measurement cluster che mi serve per capire la temperatura all'interno del dispositivo.

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	<i>MeasuredValue</i>	Signed 16-bit integer	<i>MinMeasuredValue</i> to <i>MaxMeasuredValue</i>	Read only	0	M
0x0001	<i>MinMeasuredValue</i>	Signed 16-bit integer	0x954d – 0x7ffe	Read only	-	M
0x0002	<i>MaxMeasuredValue</i>	Signed 16-bit integer	0x954e – 0x7fff	Read only	-	M
0x0003	<i>Tolerance</i>	Unsigned 16-bit integer	0x0000 – 0x0800	Read only	-	O

Questo è per capire che questa libreria è molto ricca, sicuramente ci sono dei dispositivi che non implementano tutti questi attributi, ad esempio queste informazioni sulla temperatura potrebbero non essere interessanti se abbiamo un sistema di allarme, se magari invece abbiamo un dispositivo per controllare la temperatura di un forno potrebbe essere più utile ed interessante avere un attributo di questo genere.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Vediamo ora un esempio di un cluster più complesso che è quello della localizzazione interna alla casa. È interessante vedere come l'indoor localization possa essere implementata in Zigbee, Zigbee non indica dove si trova un dispositivo ma ci permette di implementare comunque qualcosa che ci fa capire dove si trova il dispositivo. C'è un cluster per questa feature ed è l'RSSI Location Cluster. Utilizzando questo cluster possiamo scambiarci informazioni riguardo alla localizzazione, possiamo configurare dei canali per scambiare informazioni tra i dispositivi. Abbiamo vari setting differenti, quello tipico è quello in cui il nodo mobile si può muovere liberamente ed è localizzato dal server, ovvero dall'ambiente stesso. In questo contesto abbiamo bisogno di un numero di dispositivi che sono chiamati Anchor che producono le informazioni basilari per implementare gli algoritmi di localizzazioni.



In questo setting abbiamo un centralized device che chiede ai vari dispositivi di localizzare se stesso, il dispositivo invierà una serie di dummy messages ai vari Anchor, gli Anchor cercano di capire la forza di questo segnale, poi il dispositivo richiede la RSSI all'Anchor. L'Anchor invia quindi il report al dispositivo che l'ha richiesto, questo poi lo inoltra al dispositivo centrale. Qua vediamo solamente come si muovono le informazioni all'interno di questa comunicazione, non vediamo come funziona

l'algoritmo di localizzazione.

Anche il location Cluster ha vari attributi:

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	<i>LocationType</i>	8-bit Data	0000xxxx	Read / Write	-	M
0x0001	<i>LocationMethod</i>	8-bit enumeration	0x00 – 0xff	Read / Write	-	M
0x0002	<i>LocationAge</i>	Unsigned 16-bit integer	0x0000 – 0xffff	Read only	-	O
0x0003	<i>QualityMeasure</i>	Unsigned 8-bit integer	0x00 – 0x64	Read only	-	O
0x0004	<i>NumberOfDevices</i>	Unsigned 8-bit integer	0x00 – 0xff	Read only	-	O

Abbiamo informazioni che riguardano la qualità della localizzazione ad esempio. La localizzazione possiamo esprimere in vari modi differenti, abbiamo ad esempio la possibilità di esprimere la localizzazione in termini di coordinate (la terza è opzionale):

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0010	<i>Coordinate1</i>	Signed 16-bit integer	0x8000 – 0x7fff	Read / Write	-	M
0x0011	<i>Coordinate2</i>	Signed 16-bit integer	0x8000 – 0x7fff	Read / Write	-	M
0x0012	<i>Coordinate3</i>	Signed 16-bit integer	0x8000 – 0x7fff	Read / Write	-	O
0x0013	<i>Power</i>	Signed 16-bit integer	0x8000 – 0x7fff	Read / Write	-	M

Altri attributi di questa localizzazione riguardano anche il Calculation Period ovvero il tempo che passa tra una localizzazione e l'altra.

Chapter 12

Bluetooth

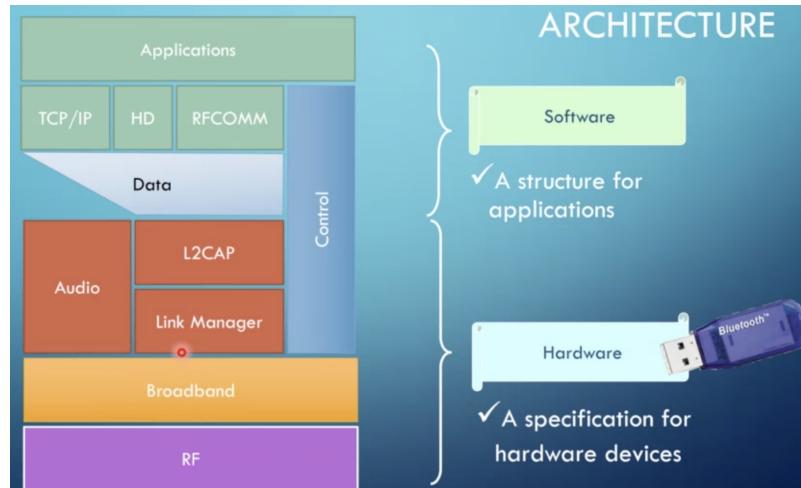
Il secondo protocollo rilevante nel campo dell'IOT è Bluetooth, questa tecnologia è nata come una sostituzione dei cavi e per questo era stata pensata con una copertura limitata (massimo 10 metri) poi nel corso degli anni si è evoluto ed ora è un protocollo molto complicato. Possiamo dire che attualmente Bluetooth implementa una Personal Area Network con un costo minore e con consumi minori dell'802.11.

Le applicazioni di Bluetooth sono tantissime, possiamo utilizzarlo per sincronizzare dispositivi, per implementare applicazioni Multimedia e tanto altro. La prima versione dello standard risale al 1999, la seconda al 2001, nel 2003 poi è stata migliorata ed è stata rilasciata una nuova versione. Nel corso degli anni sono state rilasciate altre versioni fino ad arrivare al 2010 quando è stata rilasciata la versione 4.0 che ha introdotto una distinzione tra le modalità di utilizzo:

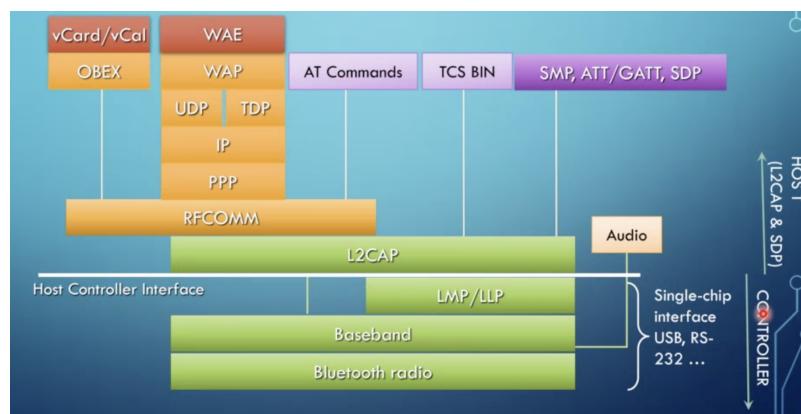
- Classic Bluetooth
- Bluetooth High Speed
- Bluetooth Low Energy (BLE), questo in particolare è lo standard che è stato pensato per essere in conflitto con Zigbee. Viene utilizzato su dispositivi che hanno una potenza molto bassa e quindi

è ottimo per gestire sensori e semplici attuatori.

Vediamo ora l'architettura di Bluetooth:



Lo stack del protocollo segue quello tipico, abbiamo alla base una parte che viene implementata in "Hardware" utilizzando ad esempio un Dongle e poi abbiamo in alto la parte implementata via "Software". All'interno della parte software molto è importato. Il livello più in alto di tutti è il livello applicazione e qua possiamo vederlo esploso in modo più dettagliato:



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

In particolare abbiamo una separazione tra il controller e l'host, il controller solitamente è implementato ad hardware in un chip e anche parte dell'host può essere implementata in hardware, ad esempio l'L2CAP, possiamo anche usare lo stesso chip. Sopra ad L2CAP possiamo poi mettere tutto il resto del livello applicazione. Altri protocolli che sono supportati da Bluetooth sono protocolli che riguardano AT Command e TCS Bin, questi sono protocolli per "Telephony" perchè possiamo ad esempio controllare tramite Bluetooth un modem e possiamo usare il nostro smartphone per fare chiamate in automatico.

C'è una cosa che è specifica di Bluetooth, l'audio support è connesso direttamente alla Baseband, per la comunicazione audio non abbiamo infatti bisogno di tutto il resto ma ci basta solamente un canale audio, quindi per ragioni di efficienza possiamo collegare l'audio alla Baseband.

L'architettura "Core Bluetooth" quindi include:

- Un host
- Uno o più controller che implementano le varie differenti versioni dello standard

Il core protocol riguarda:

- La radio che definisce la specifica delle frequenze radio
- Baseband: che definisce le procedure a basso livello del livello fisico
- LMP e LLP che sono due link layer
- HCI: il link tra hardware e software
- L2CAP che fa il ruolo di "trasport layer" anche se in realtà non lo è, questo fa sì che si abbia una interfaccia per i protocolli e le applicazioni ad un livello più alto per trasmettere e ricevere pacchetti.

- SDP: Security Discovery Protocol

Tutti gli altri protocolli sono invece adottati.

12.1 Bluetooth Basic Rate/Enhanced Data Rate

Bluetooth Basic Rate/Enhanced Data Rate è una delle parti più interessanti dello standard, questo opera a 2.4 GHz, garantisce un rate di 1Mbps e fino a 3 nel caso dell'Enhanced Data Rate. Con questo tipo di Bluetooth viene creata una piconet che è un gruppo di dispositivi coordinati da uno di loro che contiene la sincronizzazione, un Co comune e informazioni su frequency hopping pattern. La piconet è molto piccola, possiamo avere al massimo 8 dispositivi all'interno.

L'idea del Bluetooth è che partendo con un singolo physical link possiamo creare uno o più logical link. I link logici sono utilizzati per trasportare dati sincroni, asincroni e isynchronous ovvero un qualcosa che è a metà tra la comunicazione sincrona e la comunicazione asincrona.

Alcuni dei link logici che vengono utilizzati da Bluetooth sono utilizzati per protocolli di controllo. Sopra alla baseband l'L2CAP fornisce una astrazione del canale, rende i servizi affidabili implementando ACK, ritrasmissione e sincronizzazione.

Bluetooth è connection oriented e questo vuol dire che prima di comunicare è necessario stabilire la connessione e poi si può comunicare. Quando un dispositivo è connesso, la connessione viene mantenuta anche se non c'è passaggio di dati nella connessione.

In Bluetooth BR/EDR abbiamo una Sniff modes che permette di andare in uno sleep state per ridurre il consumo di energia, un dispositivo Bluetooth infatti può vivere vari mesi. La trasmissione avviene a 25mA e questo vuol dire che è sufficiente per gestire la classica comunicazione radio ma potrebbe non essere sufficiente in alcuni casi specifici.

12.2 Bluetooth Low Energy

Bluetooth Low Energy è ottimizzato per consumi molti bassi, simili a Zigbee. Ci sono tante cose che cambiano rispetto al classico Bluetooth, abbiamo dei pacchetti che sono più corti in modo da ridurre il consumo di energia, abbiamo anche meno canali da poter utilizzare e questo permette di migliorare la ricerca e il tempo di connessione. All'interno abbiamo anche una state machine più semplice rispetto a quella che troviamo in BR/EDR. Con il Bluetooth Low Energy possiamo utilizzare dispositivi con batterie a bottone perchè le richieste del protocollo sono molto semplici.

Anche il Bluetooth Low Energy lavora a 2.4GHz, abbiamo un data rate massimo di 1 Mbps, questo non è pensato per sostenere una comunicazione per molto tempo, la metrica che ci interessa non è il throughput ma il consumo di energia. Viene utilizzata una FDMA Frequency Division Multiple Access con 40 canali fisici e una TDMA ovvero una Time Division Multiple Access, su ognuno di questi canali comunica un dispositivo alla volta. Ogni canale fisico viene diviso in unità di tempo che sono chiamati eventi, ogni physical channel può trasmettere o un advertisement o un evento di connessione e si alternano.

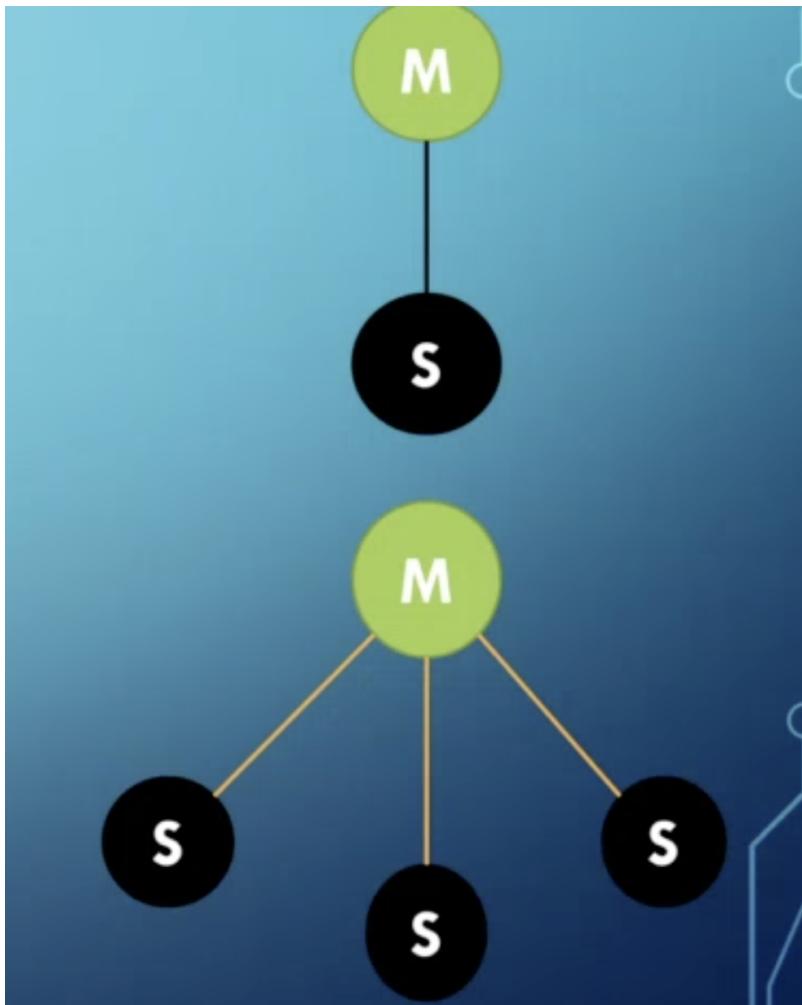
BLE è ottimizzato per inviare piccoli pacchetti di dati, il sensore trasmette ripetutamente i valori interni e basta, espone semplicemente lo stato interno che potrebbe arrivare da un sensore.

Il range di BLE è circa 150 metri ma solitamente è molto meno, la topologia è a stella e non supporta il multi hop. Possiamo avere anche più di 2 miliardi di dispositivi connessi e questo perchè non è connection-oriented. Inoltre fornisce anche sicurezza.

Range:	~ 150 meters open field
Output Power:	~ 10 mW (10dBm)
Max Current:	~ 15 mA
Latency:	3 ms
Topology:	Star
Connections:	> 2 billion
Modulation:	GFSK @ 2.4 GHz
Robustness:	Adaptive Frequency Hopping, 24 bit CRC
Security:	128bit AES CCM
Sleep current:	~ 1µA

Quindi BLE è adatto se vogliamo spedire dati piccoli. Abbiamo tanti use cases e sono simili agli use case di Zigbee, abbiamo sensori, prossimità, semplici meccanismi di localizzazione e tanti altri. Per quanto riguarda la topologia della rete, abbiamo una topologia a stella.

12.3 Topologia della rete



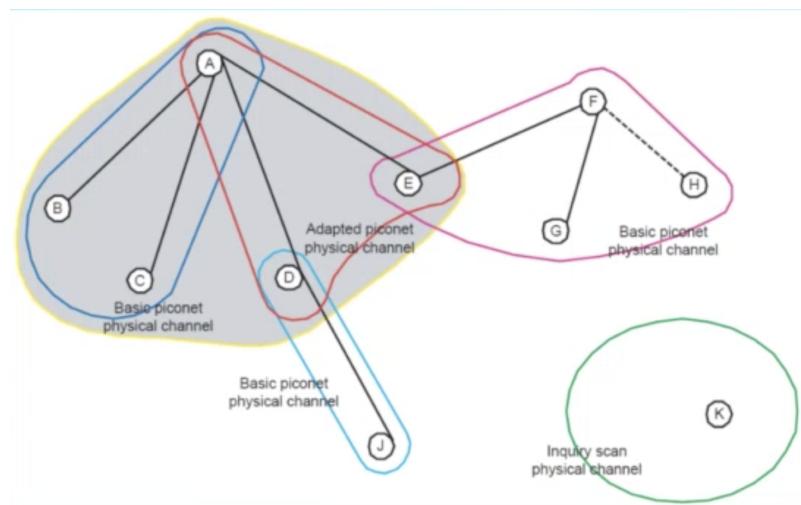
I dispositivi non hanno tutti lo stesso ruolo, sono Master e Slave, abbiamo due topologie, una è point to point con una connessione tra master e slave e una è point to multipoint con un master che è connesso a molti slave.

All'interno delle reti creiamo delle piconet che sono dei gruppi di dispositivi che sono connessi tra loro, in particolare uno è il master e gli altri sono slave e ogni comunicazione avviene con la star topology. In

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Bluetooth la maggior parte delle reti sono piconet, solamente in BLE possiamo non utilizzare la piconet e quindi comunicare direttamente.

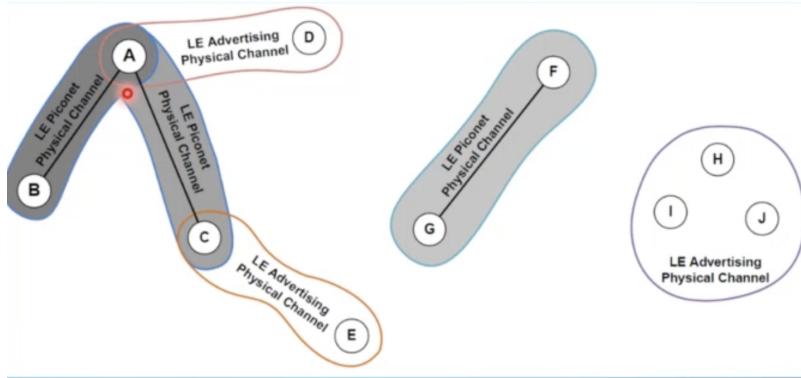
12.3.1 Esempio di topologia della rete nel caso BR/EDR



Abbiamo varie piconet che vivono tutte insieme nella stessa area, ad esempio abbiamo A che è il master per due piconet differenti. Il nodo E ad esempio è slave sia per A sia per F. Abbiamo poi un'altra piconet in cui D è il master e J è lo slave, in questo caso la D è master in una piconet e slave in un'altra. In generale possiamo connettere le varie piconet in questo modo con un dispositivo che fa da gate nelle due piconet. Abbiamo poi un dispositivo che è K che non è in nessuna piconet e che sta solamente usando il canale per trovare una piconet.

12.3.2 Piconet con BLE

Anche qua abbiamo il concetto di Piconet:



Abbiamo una piconet con il master A e due slave che sono B e C, poi abbiamo una piconet con master F e G come slave.

Abbiamo poi anche dei nodi che possono comunicare senza però formare una piconet, è il caso di D che invia ogni tanto il suo internal state e non forma una piconet, A però riceve lo stato di D e quindi fa una sorta di scanner. La stessa cosa avviene anche nel caso di E e di C con C che fa da scanner e E che fa da advertiser.

Un'altra cosa interessante è che qua abbiamo dei dispositivi che non partecipano ad una piconet, sono H, J e I che però sono in grado di ascoltarsi tra loro.

In pratica in Bluetooth BR/EDR tutti i vari nodi che comunicano devono essere all'interno della piconet o devono essere in cerca di una piconet, in BLE invece i dispositivi possono essere in una piconet ma è opzionale, tipicamente in BLE lavorano fuori dalla piconet e si ascoltano solamente a vicenda, alcuni sono advertiser e altri sono scanner. Non abbiamo quindi bisogno di formare la piconet, di avere un master e di avere uno slave.

In generale se abbiamo una piconet abbiamo un singolo master in ogni piconet che controlla l'accesso al canale dei vari slave. Nel caso di un BR/EDR in una piconet possiamo avere un massimo di 7 slave che sono attivi e comunicanti e un master, è un numero che è più che sufficiente per molte applicazioni. Possiamo anche avere altre stazioni

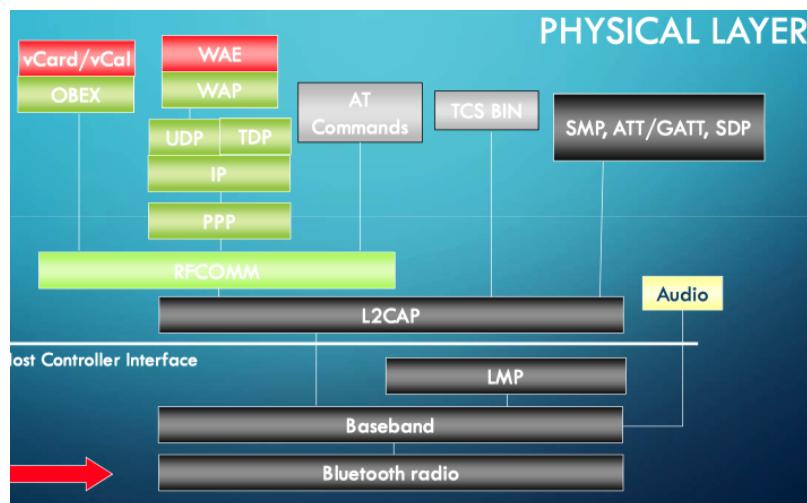
che sono all'interno della piconet ma che sono in parked mode. Nel caso di Low Energy non c'è limite al numero di active slaves nella piconet, gli active slave qua comunicano di meno rispetto al caso della BR, quindi nel LE l'unico limite riguarda le capacità del dispositivo e non il numero di dispositivi presenti nella rete.

Il fatto che con la BR/EDR possiamo avere un dispositivo in più di una piconet fa sì che si possano avere più piconet nella stessa area, queste vengono chiamate scatterness. Il protocollo Bluetooth non prevede delle regole per le scatterness, quindi possono esistere ma c'è bisogno di implementare qualcosa in questa soluzione, ad esempio anche il routing non è implementato di base quindi è compito del programmatore implementarlo. Se abbiamo bisogno di una rete scatterness probabilmente abbiamo bisogno di usare Zigbee e non Bluetooth.

12.4 Livelli di funzionamento di Bluetooth

12.4.1 Livello Fisico

Vediamo ora i livelli di funzionamento di Bluetooth:



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Bluetooth lavora con una banda di frequenza 802.11/802.11b



Reti differenti Wifi, Bluetooth e Zigbee possono convivere nella stessa area senza troppi problemi.

Nel caso di Bluetooth BR/EDR abbiamo una divisione in 79 canali fisici da 1MHz ciascuno. I dispositivi che sono all'interno della piconet svolgono 1600 salti al secondo, ad ogni salto si spostano da un canale fisico all'altro, ciascun canale fisico quindi è occupato per 0.625ms. Ognuno di questi periodi da 0.625 è detto slot ed è numerato in modo sequenziale, la frequenza di salto è condivisa tra tutti i dispositivi della rete, sono consentiti pacchetti in grado di riempire 1, 3 o 5 slots, quando finiamo l'invio del pacchetto allora la trasmissione radio torna alla frequenza richiesta dalla sequenza di salto.



Nel caso di Low Energy lavoriamo nella stessa Frequency Band ma la banda è divisa in 40 canali da 2MHz e gli ultimi tre sono destinati ai manager della rete.



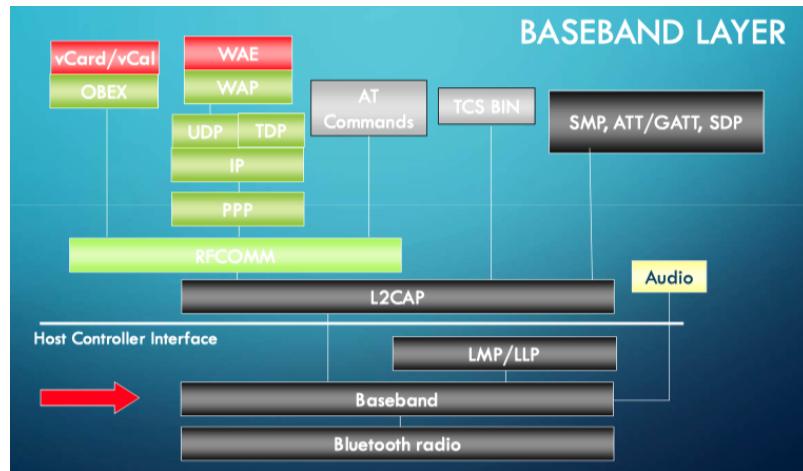
Low Energy lavora con il frequency Hopping, abbiamo una sequenza pseudo random di canali, questo vuol dire che ogni pacchetto viene mandato in un canale corrispondente al numero della Pseudo Random Sequence.

Riguardo al power level ovvero per la potenza della trasmissione, è possibile configurare tre possibili livelli:

- Class 1: possiamo estendere il range a circa 100 metri.
- Class 2: tipicamente viene utilizzato questo, abbiamo un range di circa 10 metri.
- Class 3: in questo caso abbiamo un range molto corto di 10cm

12.4.2 Baseband Layer

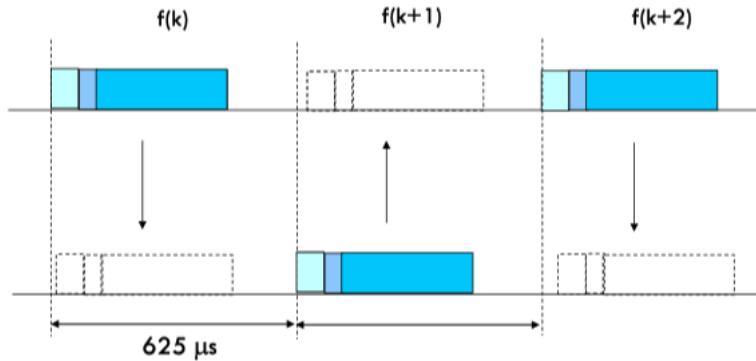
Un altro livello che vediamo è il baseband layer che implementa il frequency hopping, il power control, l'accesso al canale, il link control e fornisce anche meccanismi di controllo degli errori.



La baseband fa una astrazione del livello fisico

Il livello fisico viene utilizzato dal baseband layer per creare le funzionalità. Il canale fisico è una sequenza di 79 frequenze disponibili.

All'interno del livello fisico abbiamo una comunicazione fatta con Time Division, il master e lo slave utilizzano lo slot in modo alternato, prima trasmette il master poi trasmette uno slave, poi di nuovo il master e poi lo slave e così via. La responsabilità della sincronizzazione e del coordinamento è del master. In alcuni casi se lo slave o il master devono trasmettere allora è possibile estendere lo slot per trasmettere tutto. Il master esegue un poll dei vari slave per controllare se hanno qualcosa da inviare, il master invia dati negli slot dispari mentre gli slave trasmettono negli slot pari. Le informazioni vengono trasmesse tramite dei pacchetti, ogni pacchetto viene trasmesso in un frequency hop differente che però può essere esteso da 3 a 5 slots.



La comunicazione avviene nello slot $f(k)$ e il master trasmette, poi abbiamo uno slot successivo che è il $f(k + 1)$ e qui trasmette lo slave. I canali Bluetooth usano lo schema FH/TDD (Frequency Hopping Time Division Duplex). Il canale è diviso in slot consecutivi, contenenti un solo pacchetto di dati per slot, e ogni slot dura 625us. Per ognuno di essi è usata una sequenza di salto diversa e generalmente si ottiene un hop-rate di 1600 hps. Sulla stessa frequenza, slot consecutivi sono usati per la trasmissione e per la ricezione.

Il baseband layer implementa varie astrazioni:

- Abbiamo due logical transport tramite cui i messaggi possono essere mandati in broadcast agli slaves. Apparte questi due, tutti questi implementano una comunicazione point to point.

Per quanto riguarda il logical transport abbiamo 5 differenti tipologie:

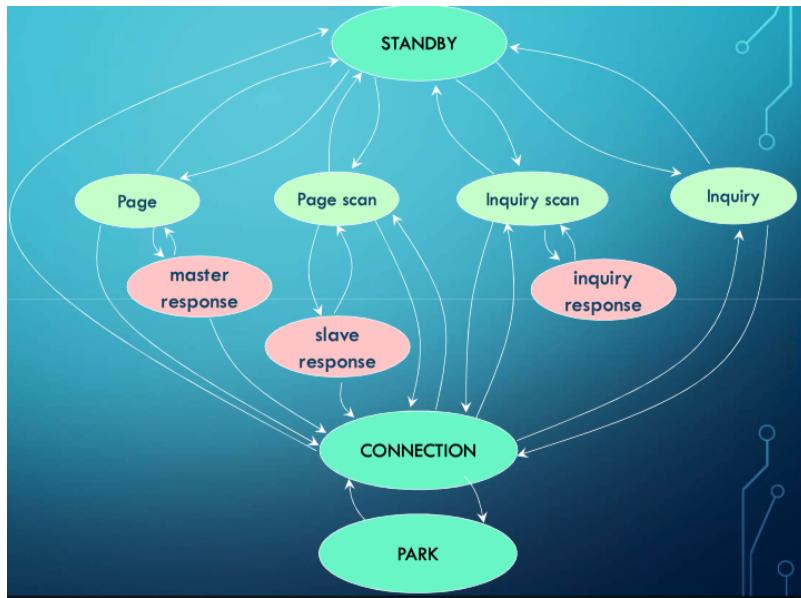
- Synchronous Connection-Oriented: in questo caso abbiamo una comunicazione point to point simmetrica tra il master e lo slave e abbiamo uno slot riservato per la comunicazione. Non abbiamo la ritrasmissione del pacchetto.
- Extended Synchronous Connection Oriented: Rispetto alla precedente abbiamo la comunicazione simmetrica e la ritrasmissione dei pacchetti.

- Asynchronous Connection Less: Con l'ACL lo scambio di pacchetti avviene in slot, uno slave può trasmettere solamente dopo e supporta l'integrità dei dati tramite ritrasmissione dei pacchetti.
- Advertising broadcast: Per quanto riguarda l'advertising broadcast, questo è unico nel Bluetooth LE. In questo caso la comunicazione è unidirezionale e non affidabile.
- Active Slave Broadcast: in questo caso abbiamo una comunicazione broadcast unidirezionale tra master e slave che sono attivi, la comunicazione non è affidabile.
- Parked Slave Broadcast: la comunicazione in questo caso è tra il master e lo slave che si trova in stato parked.

12.4.3 Stati del Bluetooth BR/EDR

Riguardo agli stati del BR/EDR, sono diversi rispetto al caso LE perché in LE sono semplificati in modo da poter essere implementati in un dispositivo semplice:

- Standby: il dispositivo non è ancora connesso ad una piconet, questo è lo stato iniziale, da questo punto in poi il dispositivo prova a connettersi ad una piconet esistente.
- Connection: il dispositivo è connesso e può comunicare.
- Park: da connesso può spostarsi nello stato Park, in questo caso non partecipa attivamente alla piconet ma rimane comunque sincronizzato con il canale e continua a ricevere messaggi broadcast, per tornare attivo di nuovo deve chiedere al master di tornare nello stato connection.

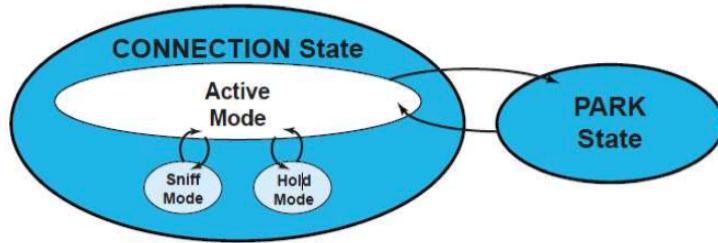


Quelli della immagine sopra sono i possibili stati in cui si trova un nodo che vuole accedere ad una certa piconet. All'inizio il nodo si trova nello stato STANDBY e dopo essersi connesso passa nello stato CONNECTION, dallo stato CONNECTION poi il nodo può passare allo stato PARK e da quello può tornare al CONNECTION. Per stabilire la connessione all'interno della rete e passare quindi da uno stato all'altro abbiamo le seguenti possibili operazioni intermedie:

- Page: il dispositivo che si connette alla rete non sa quando il master invierà un messaggio e viceversa. Questo stato di page viene utilizzato dal master che lo sfrutta per connettersi ad uno slave. Master e slave a questo punto non sono ancora sincronizzati, il master continua ad inviare messaggi di page scan fino a quando non riceve una risposta dallo slave.
- Page Scan: questo stato viene utilizzato dallo slave per trovare una piconet a cui connettersi. Il dispositivo slave deve ogni tanto entrare nello stato page scan per controllare se c'è un master che

vuole connettersi a lui e in tal caso può inviare una risposta a passare nello stato CONNECTION. Lo stato in cui mandiamo la risposta al master non è necessario.

- Page Response (Master Response e Slave Response): questa viene utilizzata dal master e dallo slave che arrivano in questo stato quando un page message viene ricevuto dallo slave. Il master in questo modo può inviare allo slave i parametri della rete come ad esempio il clock e la frequenza di hopping.
- Inquiry: l'inquiry message viene utilizzato per trovare altri dispositivi presenti all'interno della piconet. Il messaggio di inquiry viene trasmesso ripetutamente.
- Inquiry Scan: viene utilizzato dallo slave che vuole essere trovato, quindi ci mettiamo in attesa di ricevere un inquiry message. Il master che si accorge della presenza dello slave in stato inquiry lo contatta e lo slave può passare allo stato CONNECTION.
- Inquiry Response: questo non è obbligatorio, serve solamente se un dispositivo vuole rispondere ad un inquiry message.

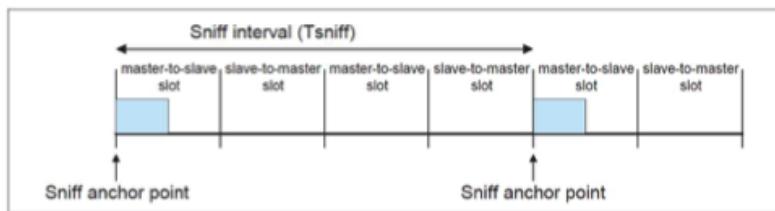


12.4.4 Baseband Layer BR/EDR: Connection State

Un substate dello stato CONNECTION è l'Active Mode, poi abbiamo due altri modi che sono lo Sniff Mode e l'Hold Mode, in particolare:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Active Mode: la conosciamo già, è quello in cui il dispositivo comunica con il master. Possiamo avere all'interno della piconet al massimo 7 dispositivi che si trovano in questo stato CONNECTION. Il master in questo caso schedula la trasmissione e mantiene la sincronizzazione.
- Sniff Mode: lo slave qua può ridurre il duty cycle per cercare di risparmiare energia, master e slave in questo caso possono comunicare solamente in alcuni specifici slots. Questo è come funziona:



Per lo sniff state, un dispositivo può essere d'accordo con il master, da un tempo iniziale blu all'altro punto blu il dispositivo può spegnere la radio e poi accenderla, poi l'accende di nuovo dopo il punto blu. Quando siamo in sniff mode il master adotta un meccanismo simile a quello di Zigbee. Una volta che un dispositivo Bluetooth è sincronizzato allora può entrare in uno stato di risparmio energetico che viene chiamato Sniff Mode, questo permette il dispositivo di ascoltare la piconet ad un rate inferiore.

Il periodo di tempo che passa tra due sniff può essere configurato in base all'applicazione che stiamo utilizzando e in base al dispositivo. L'Anchor point è lo slot in cui lo slave ascolterà la trasmissione del suo master. Il Tsniff è il numero di slots che troviamo tra due anchor points adiacenti. La sniff mode influenza solamente l'ACL logical transport, questo vuol dire che quando viene attivata questa modalità la trasmissione sull'ACL logical transport viene ridotta.

Nella sniff mode il dispositivo quindi può “assentarsi” dalla piconet per un certo tempo, il duty cycle quindi verrà definito in base al tempo in cui il dispositivo sarà presente nella rete e sarà attivo rispetto al tempo in cui sarà assente. Un possibile utilizzo dello sniff mode è tra un computer e una tastiera bluetooth, in questo caso l’utente potrebbe non scrivere qualcosa per un certo periodo di tempo e quindi sarà possibile passare in sniff mode. Quando l’utente poi cliccherà un tasto la connessione viene riportata nello stato attivo iniziale. La sniff mode può tornare utile anche quando il dispositivo deve entrare in una scatterness, per un certo periodo infatti sarà assente da una certa piconet e poi entrerà nell’altra.

- La Hold Mode è simile alla Sniff Mode.
- La park mode è quella in cui gli slave non partecipano alla piconet ma comunque rimangono sincronizzati.

12.4.5 Architettura di Low Energy

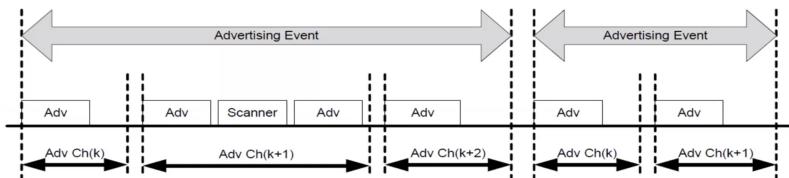
Il concetto di Low Energy è che tutto avviene tramite un advertising event, una trasmissione in broadcast di un evento che viene chiamato advertise.

Il Bluetooth Low Energy lavora con una banda a 2.4GHz con un frequency hopping tra 40 canali per raggiungere un data rate di 1Mbps. Il BLE utilizza due schemi per l’accesso ai dati, il frequency division multiple access (FDMA) e il time division multiple access (TDMA).

Nello schema FDMA il BLE salta tra 40 canali separati da 2MHz, 3 di questi sono usati come advertising channels e i restanti 37 come data channels. Nello schema TDMA un dispositivo trasmette un pacchetto ad un tempo predeterminato e lo scanning device risponde con un altro pacchetto dopo un certo intervallo di tempo predeterminato. I dati sono trasmessi tra dispositivi BLE durante due unità di tempo conosciuti come

eventi, advertising event e connection event.

L'advertising channel abilita la ricerca di dispositivi disponibili nelle vicinanze. Un advertiser invia dei pacchetti che indicano solamente che lui ha dei dati da comunicare, lo scanning device poi potrebbe richiedere all'advertising device di inviare altri pacchetti abilitando una comunicazione in broadcast o utilizzando solamente quell'advertising channel oppure può anche abilitare una comunicazione bidirezionale.



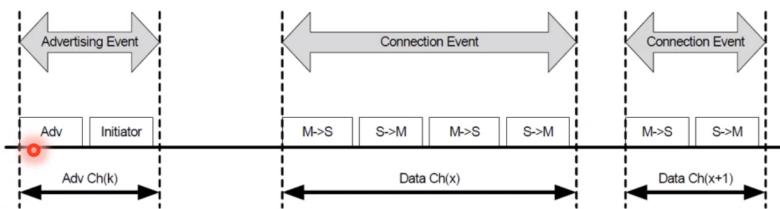
Nella immagine sopra vediamo come funziona l'advertising event:

- Gli advertisement sono inviati in frequency hopping, questi advertisement sono ricevuti da altri dispositivi che operano nelle vicinanze e sono chiamati scanners che ricevono gli advertisement.
- In alcuni casi uno scanner può rispondere all'advertiser e può ricevere una risposta.
- Nello specifico sappiamo che gli advertiser trasmettono pacchetti in al più 3 advertising channels.
- Uno scanner riceve gli advertising senza però avere l'intenzione di connettersi all'advertiser.
- Lo scanner potrebbe chiedere all'advertiser di inviare più informazioni, qua nell'immagine lo possiamo vedere nel secondo invio che avviene nel canale k+1.
- Periodicamente l'advertiser riavvia l'advertising Event (che abbiamo visto che comprende 3 invii su 3 canali differenti).

- L'advertiser può fermare l'advertising event quando vuole mentre l'evento è in corso.

È comunque possibile creare una piconet andando quindi a stabilire una comunicazione bidirezionale. In questo caso gli advertiser trasmettono connection Events.

Qua possiamo vedere che l'advertiser inizia questo processo di connessione utilizzando un advertising event, durante questo evento in particolare prima viene inviato un advertising packet di tipo connectable. L'initiator poi risponde inviando una richiesta di connessione all'advertiser che quindi risponde accettando la connessione. La connessione viene stabilita, per la comunicazione tra il master e lo slave vengono poi utilizzati dei Connection Event, all'interno di questi abbiamo una alternanza di comunicazione tra il master e lo slave. Il master è quello che avvia la comunicazione all'inizio e che può fermare la comunicazione in ogni momento. Per questa comunicazione vengono usati i 37 canali dati e quindi saltiamo tra questi canali. Quando viene avviata la comunicazione, l'advertiser diventa slave nella piconet in cui è entrato mentre invece l'initiator diventa master.



In BT LE un dispositivo può operare in 4 ruoli differenti:

- Connection Based:

- Peripheral Device: si tratta di un dispositivo che funziona come un advertiser, quindi lo possiamo connettere e funziona come uno slave all'interno di una connessione. Tipicamente è il ruolo di un sensore

- Initiator: si tratta di un dispositivo che fa la scansione degli advertisers e può iniziare la comunicazione funzionando come un master, questo tipicamente è il ruolo di un computer.
- One directional Communication:
 - Broadcaster: è un dispositivo che non si connette ma che comunque invia degli advertisement, ad esempio può essere un sensore di temperatura.
 - Scanner: è uno dispositivo che può ricevere degli advertisements senza però connettersi al dispositivo che li invia, non inizia la comunicazione, un esempio è uno schermo remoto che riceve i dati e li visualizza.

Se il dispositivo deve funzionare con una piconet allora abbiamo i dispositivi che sono connessi tra loro, abbiamo dei link fisici tra master e slave ma non tra gli slaves. Se invece non abbiamo la connessione siamo senza la piconet e per comunicare utilizziamo degli advertising event.

In BLE, sia se emettiamo un advertisement sia se siamo in una piconet e funzioniamo come slaves, quello che inviamo è un beacon. L'informazione contenuta all'interno di questo beacon può essere qualsiasi, può contenere dati di temperatura, dati di luminosità o altro. In generale l'utilizzo di questi beacon permette di avere dispositivi che consumano pochissimo e quindi di funzionare per anni e anni. Per le beacon application abbiamo solamente due ruoli:

- Peripheral: con la piconet
- Broadcaster: senza la piconet

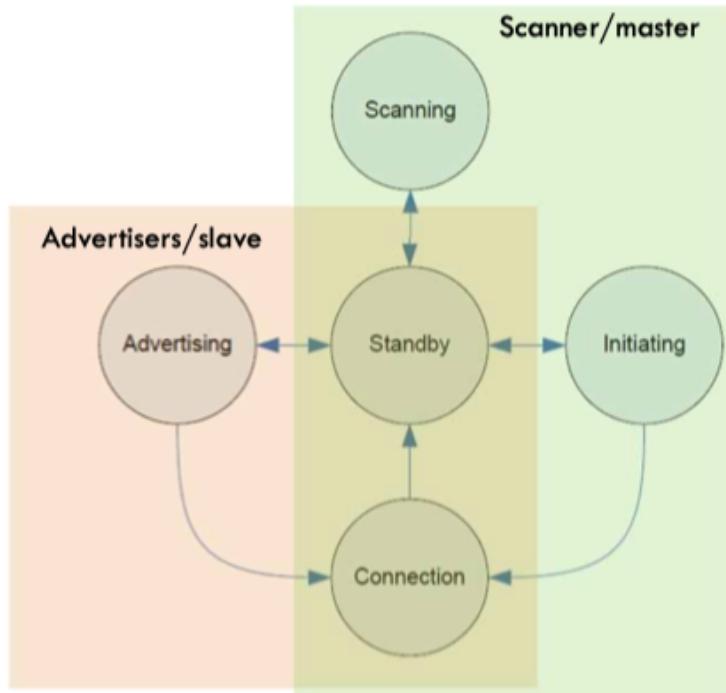
La differenza tra i due ruoli consiste solamente nel fatto che nel primo caso abbiamo una connessione mentre nel secondo no, quindi quello che succede è che per comunicare questa cosa dobbiamo inserire un flag

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

all'interno del beacon. Questi beacon sono pacchetti molto semplici e utilizzano GATT (Generic attribute profile). Gli use cases di questi beacon sono molto simili agli use case di Zigbee.

12.4.6 Baseband Layer states nel caso del BT Low Energy

Vediamo i possibili stati in cui può trovarsi un dispositivo quando si utilizza Bluetooth Low Energy.



La state machine è molto semplice:

- Standby: il dispositivo in questo caso non trasmette e non riceve nessun pacchetto.

- Advertising: quando il dispositivo è in standby può passare allo stato di Advertising, in questo modo trasmettiamo un advertising packet e poi attendiamo una risposta che può arrivare da un altro dispositivo. Un pacchetto advertising può essere utilizzato per due motivazioni, possiamo indicare ad un altro dispositivo (sia se sappiamo chi è il destinatario sia se non lo sappiamo) che il nostro dispositivo è pronto ad accettare richieste di connessioni in alternativa possiamo inviare informazioni a tutti i nodi che sono nelle vicinanze. Un pacchetto di questo genere è piccolo e quindi può contenere una quantità limitata di informazioni, per superare questa limitazione, è supportata una risposta che ci permette di inviare un secondo pacchetto che contiene dati addizionali che possono essere richiesti da un client utilizzando una scan request senza stabilire una connessione permanente con il dispositivo. In un advertising packet sono contenuti un certo numero di campi che tipicamente includono il nome del dispositivo e l'elenco dei servizi supportati dal dispositivo.
- Scanning: in questo stato il dispositivo attende un advertiser nell'advertising channel e si muove poi nello standby state per selezionare un advertiser in modo da iniziare la connessione.
- Initiating: il dispositivo che si trova in questo stato è chiamato initiator e possiamo arrivare in questo stato dallo standby. In questo stato il dispositivo attende un pacchetto da uno specifico dispositivo BLE e risponde anche a questi pacchetti per iniziare la connessione, quando inizia la connessione ci muoviamo nello stato connection.
- Connection: un dispositivo nello stato Connection può assumere due ruoli differenti, può essere il master o uno slave, i dispositivi che sono nello stato connection comunicano in base al tempo che viene definito dal master.

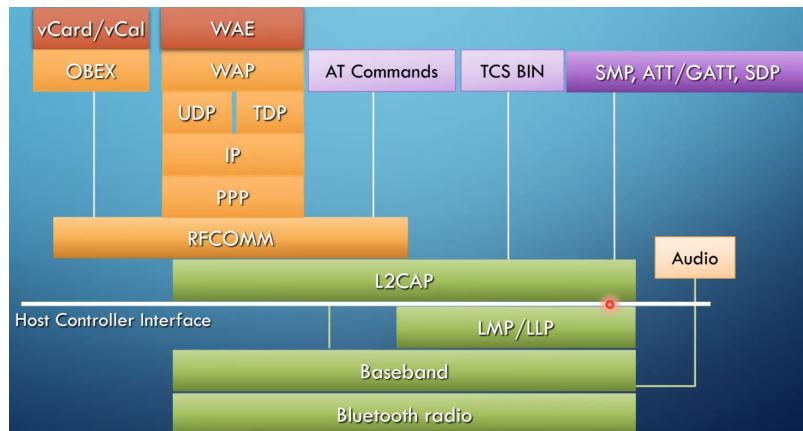
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Gli indirizzi sono indirizzi MAC di 48 bit, nel caso di Low Energy possono essere generati random o possono essere assegnati prima, ad esempio se il dispositivo è un advertiser abbiamo 24 bit relativi alla company name e 24 bit relativi al company ID.

12.4.7 Link Manager Layer

Implementa la connessione e la negoziazione della connessione tra i dispositivi, si occupa anche di gestire la piconet andando quindi ad eseguire una sincronizzazione del clock, a gestire l'energia consumata e a modificare i ruoli dei dispositivi nella rete. IN BR/EDR il ruolo del master non è definito, possiamo cambiare il ruolo del dispositivo all'interno della rete.

Per BR/EDR a questo livello si utilizza il Link Management protocol, nel caso LE invece usiamo il Link Layer Protocol LLP.



12.4.8 Livelli superiori

L2CAP implementa il multiplexing/demultiplexing.

L2CAP non fornisce supporto per la trasmissione di audio, non supporta la ritrasmissione e il multicast affidabile.

Riguardo RFCOMM, questo viene utilizzato per la comunicazione audio, non implementa protocolli di ritrasmissione ma implementa dei sistemi per rendere la trasmissione della voce più affidabile. Per quanto riguarda l'AT command, questo viene definito per controllare un modem o un telefono. I protocolli che vengono adottati sono quelli standard.

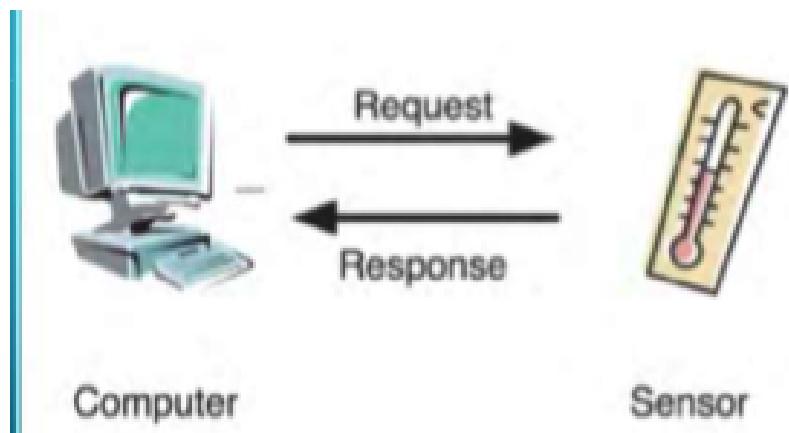
12.4.9 Gatt Services

Il Generic Attribute Profile (GATT) è obbligatorio per il BLE e può essere utilizzato anche in BR/EDR. Viene utilizzato per definire in che modo un dispositivo deve inviare e ricevere messaggi. Vengono definite due figure, il server e il client, vediamo un esempio di configurazione:

- Il computer è il client che invia comandi al server, che in questo caso è il sensore e offre dei servizi offerti dal Gatt. I servizi che offre il GATT sono una collezione di dati e comportamenti che mi permettono di svolgere una particolare funzione di un dispositivo.
- la risposta viene fornita tramite l'attribute profile.

Vediamo un esempio di servizio con GATT:

- Si tratta di un servizio di temperatura che fornisce informazioni riguardo alle caratteristiche del dispositivo che esponiamo come parte del nostro servizio di temperatura.
- Può permettere di inviare alcune caratteristiche e risponde alle richieste di lettura.
- Il computer in questo caso è il client che avvia la procedura di configurazione del sensore e legge i valori del sensore.



Un dispositivo che implementa il GATT Serveer possiamo chiamarlo Bluetooth Smart Device.

12.4.10 Attribute protocol (ATT)

L'attribute protocol è obbligatorio in Low Energy, in particolare gli attributi che sono esposti da un server possono essere scoperti, letti e scritti da un client. Abbiamo due ruoli, server e client, il server espone degli attributi con dei valori associati mentre il client vi accede, In ATT un attributo è associato con tre metadata:

- Attribute Type UUID che è un numero che può essere definito dal BT o dall'utente.
- Attribute Handle: un numero tramite cui un dispositivo viene identificato nel server.
- Un attributo viene associato al set di permessi, tipo lettura, scrittura.

Un attributo espone dei dati su un dispositivo remoto, le operazioni che sono supportate sono:

- Push: aggiunta di un valore. È una sorta di publish/subscribe mechanism, il client potrebbe chiedere al server di indicargli quando un attributo va sotto un determinato valore.
- Pull: accediamo ad un valore prendendolo dal server. Il client accede ai dati quando ne ha bisogno, non è un metodo efficiente se vogliamo monitorare dei dati che cambiano frequentemente.
- Get: per leggere la configurazione o un attributo. Altra cosa, per implementare una Get dobbiamo specificare cosa vogliamo leggere, ad esempio utilizziamo l'UUID di quello che vogliamo ottenere.
- Broadcast: viene inviato per inviare dati ad ogni dispositivo che è in ascolto.
- Set: viene utilizzato per configurare un server, ad esempio per controllare un attuatore.

12.4.11 Sicurezza di BT

La sicurezza di Bluetooth riguarda i vari layer del protocollo. Bluetooth è pensato per low power devices quindi non possiamo pensare dei meccanismi di sicurezza troppo complessi. Gli obiettivi sono

- Protezione verso intercettazione dei dati che vengono scambiati, in questo caso viene fornito un meccanismo di encryption dei pacchetti.
- Protezione contro attacchi Man in the middle, ad esempio vogliamo evitare che una persona fa finta di essere un altro dispositivo per connettersi al tuo smartphone (facendo finta ad esempio di essere le nostre cuffie).

La protezione si basa su un protocollo di pairing sicuro che utilizza la crittografia con chiave pubblica perchè qua i dispositivi sono più potenti rispetto a quelli di Zigbee e quindi si può fare (in Zigbee veniva

utilizzata la crittografia simmetrica che è più semplice e adeguata per dispositivi semplici e poco costosi). Nel caso di BT LE abbiamo un livello di sicurezza minore per via della potenza del dispositivo.

Bluettoth fornisce vari (4) modelli di accesso, il tipo di soluzione che viene utilizzata dipende dalla situazione e anche dalle capacità del dispositivo.

- Just Work: questo modello funziona senza chiavi crittografiche facendo solamente un controllo che il dispositivo che si vuole connettere è veramente quello che ci aspettiamo. In questo caso non ci sono da inserire PIN e non vengono fatti controlli da questo punto di vista. Quindi questo per esempio viene usato quando il dispositivo che connetto non ha un display, come ad esempio con delle cuffie. L'utente preme un pulsante, attiva il protocollo di pairing e poi i due dispositivi si connettono. La sicurezza è garantita dal fatto che i due dispositivi sono vicini tra loro e dal fatto che Bluetooth funziona con un range abbastanza breve.
- Numeric Comparison: l'utente controlla che i due dispositivi mostrano lo stesso numero e poi accetta la connessione. In questo caso i due dispositivi devono essere in grado di mostrare il pin e di farlo inserire.
- Out of Band: in questo caso le informazioni del pairing vengono trasferite utilizzando canali alternativi (quindi non le radio del bluetooth) per scoprire i dispositivi e trasferire codici crittografici.
- Passkey entry: un dispositivo ha un display ma non ha possibilità di inserire i dati, l'altro invece può far inserire i PIN ma non mostrarlo. È il caso del collegamento di una tastiera ad un computer.

Per quanto riguarda la crittografia tutto si basa su chiavi crittografiche e in particolare su 4 elementi:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

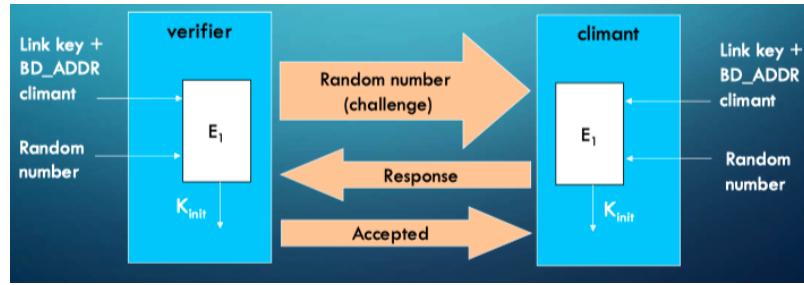
- Mac Level Address, è il Mac address del dispositivo che si connette e di quello a cui ci connettiamo. Questa informazione è pubblica e unica per ogni dispositivo possiamo semplicemente dire al dispositivo di prendere il suo MAC Address. È usato in un security mechanism ma non serve per gestire la crittografia.
- Private Key per autenticazione, questa è chiamata link key ed è di 128 bit. Non la possiamo usare in Zigbee per esempio.
- Oltre al link key c'è una private key che possiamo usare per l'encryption durante la comunicazione, queste hanno lunghezza variabile, è derivata dall'autentication key e dovrebbe essere rinnovata ad ogni sessione.
- Inoltre dovremmo essere in grado di produrre numeri pseudo-random di 128 bit.

Le transazioni tra i dispositivi sono gestite con una link key (128 bit), la link key viene utilizzata per l'autenticazione. La link key può essere prodotta in due modi:

- Semi permanentemente, viene installata nel dispositivo quando lo produciamo o durante il deployment, rimane poi dentro al dispositivo e può essere utilizzata in varie sezioni. Può essere cambiata e ogni tanto è una cosa sensata cambiarla.
- Temporary Key, queste sono chiamate master key e hanno una durata limitata per una sessione. Durante questi periodi i dispositivi usano questa master key e questo è una protezione anche per la link key perché meno la utilizziamo e più aumentano le possibilità che questa non venga scoperta da un attaccante.

Vediamo il meccanismo che viene utilizzato per il pairing di un dispositivo Bluetooth:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Nella notazione BT il dispositivo che si connette ad un network esistente viene chiamato climant, dall’altro lato abbiamo il verifier. Il ruolo del climant è dimostrare che conosce il segreto che gli permette di connettersi alla rete, questo segreto è la link key. Il verifier è tipicamente il master della piconet e il suo ruolo è quello di verificare che il climant conosca davvero il segreto.

Segue uno standard response mechanism e funziona in questo modo:

- Quando il climand richiede la connessione il verifier produce un numero random e lo invia al climant.
- Il climant produce una risposta che viene mandata al verifier, la risposta è prodotta partendo da questo random number, dal Mac Address del climant e dalla Link Key.
- Il verifier fa la stessa cosa perchè conosce il random number e conosce la link key e il MAC address del climant (perchè il climant aveva richiesto la connessione). Quindi il verifier controlla che la risposta che arriva dal climant sia corretta e in quel caso accetta la connessione. Da questo punto in poi il climant è nella connessione.

Un side effect di questa comunicazione è che verifier e climant devono essere d’accordo su una chiave K_{init} che viene utilizzata da questo punto in poi per criptare tutta la comunicazione.

Per quanto riguarda l’encryption possiamo utilizzare una semi-permanent key o una temporary key. La semi permanent key può essere utilizzata

solamente all'interno di pacchetti unicast mentre la temporary key possiamo utilizzarla anche per pacchetti broadcast.

<i>Semi-permanent key</i>	
Broadcast packets	Unicast packets
Not encrypted	Not encrypted
Not encrypted	Encrypted, semi-permanent key

<i>Temporary key K_{master}</i>	
Broadcast packets	Unicast packets
Not encrypted	Not encrypted
Not encrypted	Encrypted, K_{master} key
Encrypted, K_{master} key	Encrypted, K_{master} key

All'application Level la sicurezza è gestita dal Security Management Protocol. SMP è uno dei componenti in cima allo stack di Bluetooth, questo gestisce la sicurezza a livello applicazione che vuol dire che gestisce le Link keys e produce la Master key e fornisce la master key ai livelli più bassi in modo da avere l'encryption dei pacchetti. Per implementare la distribuzione delle chiavi SMP deve comunicare con l'SMP dell'altro dispositivo e questo lo fa tramite L2CAP che fornisce un security manager channel.

Overview finale delle varie versioni BT rispetto agli altri protocolli:

	Voice	Data	Audio	Video	State
Bluetooth ACL/HS	x	y	y	x	x
Bluetooth SCO/eSCO	y	x	x	x	x
Bluetooth low energy	x	x	x	x	y
Wi-Fi	(VoIP)	y	y	y	x
Wi-Fi Direct	y	y	y	x	x
ZigBee	x	x	x	x	y
ANT	x	x	x	x	y

Legend:
 State = low bandwidth, low latency data
 Low Power

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Wi-Fi è più potente di Bluetooth, può sostenere voce, data, audio e video, ad esempio il BTLE permette solamente di esporre uno stato ed è il diretto concorrente di Zigbee e di ANT (che è un altro standard per l'IoT).

Chapter 13

Signal Sampling con Arduino

All'interno del kit di arduino abbiamo i seguenti componenti:

- Arduino IDE
- Arduino (va bene anche la versione Uno) e cavo di connessione
- Una breadboard

13.0.1 Cosa si intende con sampling?

Abbiamo una variabile time-dependente e vogliamo registrare dei valori durante il tempo, il tempo che passa tra i vari sample deve essere molto preciso. Il signal sampling è un esempio di real-time processing perchè dobbiamo scrivere il nostro programma per essere precisi riguardo agli intervalli di tempo che passano tra due operazioni. Le misurazioni devono avvenire all'interno di un certo periodo di tempo, l'intero processo non funziona se la misurazione va fuori da questa finestra temporale, questa finestra temporale può essere più o meno estesa (minuti o microsecondi)

ma deve essere definita. Se siamo nella situazione in cui dobbiamo fare real-time computing dobbiamo essere sicuri perchè altrimenti non possiamo essere sicuri di rispondere entro il tempo che abbiamo definito. È anche fondamentale, quando si eseguono delle operazioni in real time, la definizione di un time scale, è utile per scansionare l'esecuzione e per fornire delle referenze temporali per gli eventi.

13.0.2 Codice Arduino

Consideriamo un esempio molto semplice:

```
void setup() { Serial.begin(57600); pinMode(13,OUTPUT); }
void blink() { digitalWrite(13, digitalRead(13) ^ 1); }
void loop() {
    Serial.println("Hallo");
    blink();
    delay(1000);
}
```

Si tratta di un programma che ogni secondo accende un led presente su Arduino e scrive Hallo. In questo caso nel nostro problema real time vogliamo stampare qualcosa ogni secondo e quindi vogliamo che per completare il loop sia necessario un tempo nell'intervallo [999.9, 1000.1] millisecondi. L'errore che otteniamo in questo caso è variabile:

```
10:38:51.162 -> Hallo  
10:38:52.158 -> Hallo  
10:38:53.155 -> Hallo  
10:38:54.151 -> Hallo  
10:38:55.180 -> Hallo  
10:38:56.177 -> Hallo  
10:38:57.174 -> Hallo  
10:38:58.170 -> Hallo  
10:38:59.166 -> Hallo
```

...

Il tempo che viene preso in questo caso è quello del computer, non quello dell'Arduino e questo comporta che l'errore è nel range di 10ms e non 0.1ms come invece avevamo richiesto.

Per evitare di stampare l'orario del computer possiamo risolvere il problema andando a stampare non il messaggio di Hello ma un messaggio che mi indica da quanto tempo (in millisecondi) l'Arduino è stato acceso.

```
void setup() { Serial.begin(57600); pinMode(13,OUTPUT);}
void blink() { digitalWrite(13, digitalRead(13) ^ 1); }
void loop() {
    delay(1000);
    unsigned long int t1=millis();
    Serial.println(t1);
    blink();
}
```

In questo caso noi accumuliamo dopo 20 secondi, un errore di 3 milisecondi che alcuni casi potrebbe essere troppo. Dato che vogliamo avere un errore massimo di 2 milisecondi allora vuol dire che possiamo avere al massimo un errore di 0.1 milisecondi in ogni misurazione, dato che $0.1ms * 20 = 2ms$. Il problema qua è che la scrittura sulla serial line prende del tempo e questo aumenta il tempo di durata del loop.

Per ottenere una visualizzazione migliore del tempo che passa potremmo utilizzare una versione differente del nostro codice, in realtà qua l'errore rimane perchè andiamo solamente a stampare la differenza tra il precedente tempo e il successivo.

```
void setup() { Serial.begin(57600); pinMode(13,OUTPUT); }
void blink() { digitalWrite(13, digitalRead(13) ^ 1); }
unsigned long int t0 = millis();
void loop() {
    delay(1000);
    unsigned long int t1=millis();
    Serial.println(t1-t0);
    blink();
    t0=t1;
}
```

Con questo codice vediamo che (più o meno) ad ogni step abbiamo un errore di 1 millisecondo ma noi lo vorremmo di 0.1 milisecondi e quindi dobbiamo trovare una alternativa.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

La strategia alternativa consiste nel non utilizzare il delay ma misurare il tempo dall'ultima attivazione.

```
void setup() { Serial.begin(57600);pinMode(13,OUTPUT); }
void blink() { digitalWrite(13, digitalRead(13) ^ 1); }
unsigned long int t0 = millis();
void loop() {
    unsigned long int t1=millis();
    if (t1-t0>=1000) {
        Serial.println(t1);
        blink();
        t0=t1;
    }
}
```

Questo loop viene eseguito in continuazione e non c'è ritardo, ogni volta che eseguo il loop eseguo calcolo il tempo attuale in millisecondi ed eseguo un test. La differenza tra il tempo attuale e quello della precedente attivazione deve essere maggiore di 1000 per eseguire il codice. Se è minore non faccio niente, se invece è maggiore o uguale allora eseguo il codice ovvero stampo, accendo il led e resetto (nel senso che ora al tempo della precedente attivazione assegno il valore attuale del tempo). Questa operazione all'interno dell'if prende un po' di tempo ma io comunque controllo all'interno del loop quanto tempo è passato da quando ho fatto l'ultima operazione e se è passato il tempo che volevo eseguo nuovamente il codice. Questo è il risultato che otteniamo, possiamo vedere che è ottimo:



Non siamo ancora sicuri che questo risultato vada bene per le nostre richieste, possiamo provare un altro sketch leggermente modificato in cui ora il PERIODO viene memorizzato in una costante.

```
#define PERIODO 15
void setup() { Serial.begin(57600); pinMode(13, OUTPUT);}
void blink() { digitalWrite(13, digitalRead(13) ^ 1); }
unsigned long int t0 = millis();
void loop() {
    unsigned long int t1=millis();
    if (t1-t0>=PERIODO) {
        Serial.println(t1-t0);
        blink();
        t0=t1;
    }
}
```

All'interno dello sketch abbiamo:

- Una setup function che viene eseguita solamente una volta quando accendiamo l'arduino. Prima mi dice su quale serial line stiamo lavorando e poi setta come output il Pin 13 che è direttamente connesso alla board.

- Poi abbiamo una funzione generica `blink()` che accende il led corrispondente a quel pin
- Abbiamo poi una variabile globale `t0` interna allo sketch, in questa variabile memorizziamo i millisecondi che passano dal momento del boot dell'arduino.
- Poi abbiamo bisogno di una funzione `loop()` che non si ferma mai e viene eseguita di continuo. All'interno di questa funzione abbiamo varie operazioni, in questo caso aggiorniamo il tempo `t1` e calcoliamo una differenza tra l'ultimo tempo che è stato registrato e il tempo corrente. Qua la differenza rispetto a prima è che non usiamo 1000 ma usiamo la costante periodo all'interno dell'`if`.

Al posto del serial monitor possiamo utilizzare il serial plotter che mostra al suo interno 500 sample e possiamo notare che ci sono delle spike che corrispondono ad alcuni punti in cui avviene l'esecuzione del nostro codice. In generale quindi abbiamo sempre un valore di 15 con alcune spike che arrivano a 16 che rappresentano l'errore. Quando stampiamo 16 vuol dire che la differenza $t_1 - t_0$ è 16 e non 15 come noi vorremmo.



Per fare una valutazione di questi risultati ci stiamo spostando verso il signal processing, eseguiamo il match della produzione dei dati con il

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

segna che sappiamo che ha una sequenza conosciuta e questo matching è fatto con il sampling.

13.1 Sampling

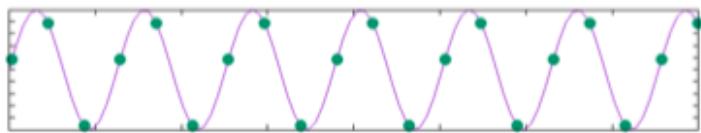
Il sampling consiste nella lettura del valore di una variabile time dependent (quello che chiamiamo segnale) ad una frequenza definita.

Eseguiamo il sampling utilizzando l'arduino A/D converter. Il sampling avviene con una accuracy di 10 bit che è abbastanza accurato per la maggior parte delle applicazioni. Dobbiamo essere molto precisi nel sampling per ottenere un risultato che sia significativo.

Qua viene seguito un approccio opposto, abbiamo un segnale di cui conosciamo un certo numero di cose e noi possiamo vedere come si comporta il nostro programma quando eseguiamo il sampling. Non stiamo provando a stimare il segnale usando il sampling ma proviamo a definire la precisione del sampling misurando un segnale di cui noi conosciamo un certo numero di informazioni.

Vediamo cosa succede con il sampling.

Se facciamo il sample di un segnale con una frequenza che è più alta rispetto alla frequenza caratteristica del segnale abbiamo un numero di valori che seguono la variazione del nostro segnale. In questo caso abbiamo 6 sample in ogni onda.



Cosa succede quando la frequenza del segnale è comparabile o anche minore rispetto alla frequenza del segnale?

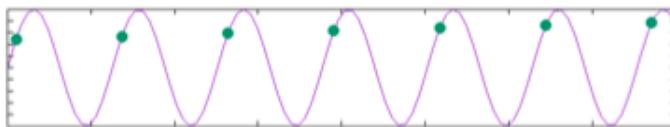
Se il sampling frequency è esattamente lo stesso vuol dire che abbiamo un modo per valutare la precisione del nostro tempo perchè se conosciamo la frequenza del segnale sappiamo che dovremmo ottenere questo risultato.

Dovremmo avere un valore del segnale esattamente negli stessi punti della curva. Se abbiamo un sampling period che è uguale al signal period abbiamo valori che sono sempre nello stesso punto, quindi quello che vediamo qua è una linea.



Quindi questo è un modo per valutare la precisione del nostro dispositivo di sampling.

Se invece il sampling period è minore della frequenza del segnale allora abbiamo un'altra forma che è simile a quella originale (abbiamo un alias). Dato che la frequenza del sampling è minore rispetto alla frequenza del segnale abbiamo che il punto si muove e alla fine otteniamo un'onda con una frequenza che corrisponde alla differenza delle frequenze (?).



Abbiamo quindi tre casi e li possiamo utilizzare per capire quanto è preciso il sampling.

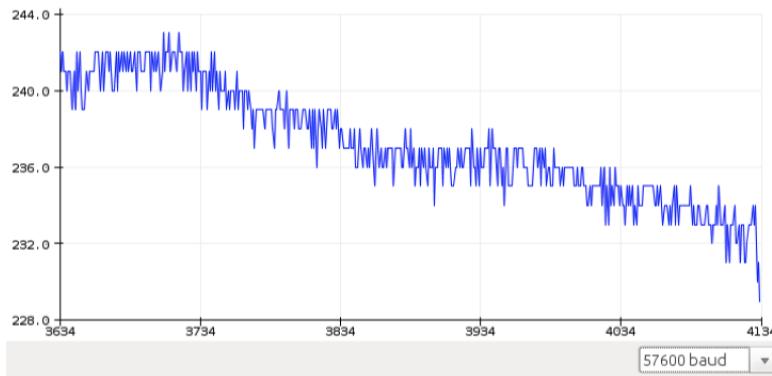
Consideriamo il cui vogliamo eseguire il sample di un segnale a 50Hz (la frequenza del segnale si misura in Hz e indica il numero di volte che il segnale periodico si ripete in un secondo).

Vogliamo fare il sample con un periodo che è 20 millisecondi, il segnale da 50Hz ha un periodo di 20 millisecondi quindi ci aspettiamo una frequenza identica sul sampling device e sul segnale, quindi siamo nel primo dei casi indicati in precedenza.

```
#define PERIODO 20
void setup() { Serial.begin(57600); pinMode(13,OUTPUT); }
void blink() { digitalWrite(13, digitalRead(13) ^ 1); }
unsigned long int t0 = millis();
void loop() {
    unsigned long int t1=millis();
    if (t1-t0>=PERIODO) {
        int adc=analogRead(A0);
        Serial.println(adc);
        blink();
        t0=t1;
    }
}
```

Il programma è uguale, cambia solamente il fatto che in questo caso facciamo il sampling e leggiamo l'analog input con la funzione analogRead e lo memorizziamo in adc e poi lo stampiamo. In questo caso non stampiamo più il valore del tempo, non ne abbiamo bisogno perché abbiamo un riferimento preciso di 50Hz.

In teoria eseguendo il sampling in questo modo dovremmo ottenere una linea orizzontale ma in realtà otteniamo questo:



Qua ci sono due fenomeni da tenere in considerazione:

- Abbiamo tanta variazione nei dati ed è anche veloce (qua ci aspettavamo una linea orizzontale nei dati) e quindi otteniamo qualcosa che non è costante. Questo problema è probabilmente connesso con il mio programma che non riesce a stabilire un tempo preciso per il collezionamento dei dati.
- Aliasing: possiamo fare poco per risolvere questo problema di aliasing perchè è dovuto a problemi con la sincronizzazione dell'orologio dell'arduino. Quando chiedo di misurare ogni 20ms l'intervallo sarà sistematicamente maggiore o minore e questo è l'effetto dell'aliasing e del fatto che abbiamo il segnale che si abbassa e poi ricrescerà più tardi.

13.1.1 Come risolviamo?

Vorremmo rendere l'azione del sampling completamente indipendente dal loop perchè nel programma precedente c'erano delle interferenze tra le due operazioni.

Per questa ragione vogliamo utilizzare un Timer che è presente all'interno di arduino, questo è un dispositivo hardware che funziona in modo molto più preciso rispetto all'implementazione che avevamo utilizzato prima. È

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

importante avere un sistema di questo genere che deve essere completamente indipendente rispetto al loop perchè nel loop potrebbero arrivare delle interruzioni che potrebbero disturbare il sampling.

Questo timer presente in arduino è indipendente dalla CPU di Arduino, è un dispositivo hardware che quando scatta genera una interruzione. Il timer è in pratica un generatore di interruzioni ad intervalli precisi.

L'handler dell'errore non dovrebbe contenere del codice troppo complesso da eseguire e in generale non deve eseguire l'handler di altri segnali.

Questo tipo di operazioni produce un side effect in modo semplice e il mio programma può osservare questo side effect. Quando succede il mio programma non è in grado di capire se l'handler sta eseguendo il codice o no ma se il mio programma può osservare il side effect dell'handler può esserci una operazione di sincronizzazione dal mio programma all'handler.

Prima di andare a produrre l'handler dobbiamo capire come controllare il Timer.

13.1.2 Timer

L'arduino ha il suo clock a 16MHz e genera un impulso ogni 63ns, questo impulso è utilizzato per incrementare 3 timer. Ogni timer ha una capacità limitata e quindi ad un certo punto abbiamo un periodo di overflow, la capacità è di 8 bit per il timer0 e il timer2 e 16 per il timer1, questo vuol dire che alla frequenza di 16MHz abbiamo un tempo di overflow di 16 microsecondi per il timer0 e il timer2 e di 4 millisecondi per il timer1.

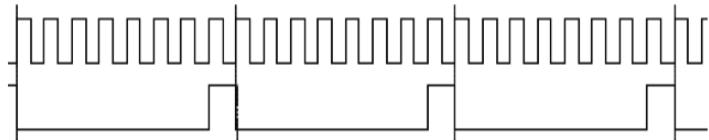
È possibile controllare un timer in modo che esegua il conto ad un rate più basso, in questo caso si parla di pre-scaling e questo potrebbe contare non fino all'overflow ma potrebbe fermarsi fino a che non arriva ad un threshold. Quando raggiunge il threshold allora viene generata una interruzione e viene resettato.

In questo modo siamo in grado di produrre una interruzione a intervalli predefiniti, producendo l'interruzione a intervalli definiti andiamo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

ad eseguire il nostro handler a intervalli definiti.

Il tempo di un singolo ciclo di overflow potrebbe essere troppo breve, quindi possiamo utilizzare il prescaling. In questo caso ad esempio abbiamo che, data la frequenza in input in cui abbiamo 8 impulsi, ne consideriamo solamente uno ogni 8. Con il prescaling quindi dividiamo la frequenza di input in modo da rallentare il conteggio del tempo e di estendere il tempo di overflow. In questo modo ad esempio riusciamo ad ottenere un tempo di 500ms al posto dei tempi di 16 microsecondi o 4 millisecondi che abbiamo con i timer indicati in precedenza.



Nel prescaling abbiamo potenze di 2 e quindi passiamo da 8 a 64 a 256 fino a 1024 che sono potenze di 2 ma non successive.

Ad esempio se abbiamo un prescaling di 64 ho una frequenza generata di 250KHz (al posto dei 16MHz che avevamo inizialmente), ho un peak ogni 4 microsecondi.

Se vogliamo una interruzione ogni millisecondo devo fissare il threshold a 250 perchè consideriamo 1ms e lo dividiamo per 4 microsecondi. Quello che facciamo è contare con un periodo di 4 microsecondi e quando arrivo a 250 allora sono sicuro che è passato un millisecondo, a questo punto produco la mia interruzione resetto e ricomincio il conteggio fino ad arrivare di nuovo a 250.

Se vogliamo avere una interruzione ogni 20 millisecondi (è il valore che ci interessava a noi perchè è relativo al periodo di 50Hz) possiamo utilizzare un timer con un prescalar di 256. Questo vuol dire che il periodo è di 16 microsecondi perchè abbiamo $16MHz/256 = 62.5KHz$ e il threshold è di 1250 perchè $20ms/16microsecondi = 1250$

Abbiamo tre timer, 2 con capacità 8 bit e uno con capacità 16 bit, dobbiamo saper contare fino a 1250 quindi mi servono più di 8 bit quindi devo usare il timer1 (che ha 16 bit).

Ogni timer viene configurato utilizzando un registro interno del processore, abbiamo tre timer e ognuno ha un set di registri di configurazione, sono variabili che compaiono all'interno del programma e hanno nomi predefiniti. Per il timer1 abbiamo:

- Due register TCCR1A e TCCR1B che controllano la funzionalità del timer, incluso il valore del prescaling.
- OCR1A è la variabile del threshold per resettare il timer.
- TCNT è il valore iniziale del timer, in caso ce ne sia bisogno.
- TIMSK1 serve per abilitare o no le mask interruptions.

La cosa fondamentale sono i primi due punti e questi vanno configurati.

Questo è il contenuto del regisro TCCR1B:

7	6	5	4	3	2	1	0
ICNC1	ICES1		WGM13	WGM12	CS12	CS11	CS10

Il registro è da 8 bit e all'interno abbiamo gli ultimi 5 bit che servono per controllare il timer, in particolare gli ultimi 3 servono per specificare il rescaling. A seconda del valore di questi ultimi 3 bit abbiamo differenti valori per il prescaling:

CS12	CS11	CS10	pre-scaling
0	0	1	1
0	1	0	8
0	1	1	64
1	0	0	256
1	0	1	1024

13.1.3 Risposta ad una interruzione

Siamo arrivati al punto in cui dobbiamo considerare il funzionamento della handler function. Abbiamo visto fino ad ora che la Handler function viene chiamata nel momento in cui avviene una interruzione. Il nome della funzione è pre definito come ISR e prende come parametro l'interruption vector che descrive l'interruzione in termini di parametri. Come già detto deve essere una funzione molto veloce e nella maggior parte dei casi può produrre dei side effect che le altre funzioni devono poter osservare.

Dalla seguente tabella, considerando che stiamo usando un prescaler=256, possiamo vedere alcuni parametri utili:

```
/*
 * OCR1A | Frequenza (Hz) | Plot (50Hz)
 *
 * -----
 * 62500 |      1 | aliasing
 * 6250 |     10 | aliasing
 * 1250 |     50 | aliasing
 * 125 |    500 | forma d'onda
 * 50 |   1250 | forma d'onda
 */

```

Consideriamo ora del codice, questo è il setup e qua possiamo vedere che nel setup si scrive nel control register. Una delle prime cose che

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

facciamo nel setup è bloccare le interruzioni perchè non vogliamo che le operazioni sensibili del setup vengano in qualche modo bloccate. Alla fine della procedura poi abilito di nuovo le interruzioni e nel mezzo assegno i vari valori, in questo caso assegnamo 125 a OCR1A e questo vuol dire che stiamo facendo il sample con frequenza 500Hz.

```

volatile boolean flag=0;
void blink() {digitalWrite(13, digitalRead(13) ^ 1);}
void setup() {
    Serial.begin(57600); pinMode(13,OUTPUT);
    noInterrupts();           //Inibisco le interruzioni
    TCCR1A = 0;               //Registro di controllo
    TCCR1B = 0;               //Registro di controllo
    TCNT1 = 0;                //Azzero il contatore
    OCR1A = 125;              //La soglia per l'azzeramento del contatore
    TCCR1B |= (1 << WGM12); //Abilita conteggio/interruzione/azzeramento
    TCCR1B |= (1 << CS12);  //Imposta il prescaler (a 256)
    TIMSK1 |= (1 << OCIE2A); //Generazione dell'interruzione
    interrupts();
}

```

Questo è quello che succede nel resto del codice, ISR è il nostro interrupt handler che riceve un interrupt vector ma non lo usa direttamente, setta solamente una variabile flag, questa è una cosa che è molto veloce da fare. Ogni volta che eseguiamo il loop viene controllato il flag e poi viene eseguita una certa azione se il flag è stato settato, una volta che finisco di eseguire il codice dentro all'if posso nuovamente settare a 0 il flag.

```

ISR(TIMER1_COMPA_vect) { flag=1; }

void loop() {
    if (flag) {
        unsigned long int t1=millis();
        int adc=analogRead(A0);
        Serial.println(adc);
        flag=0;
    }
}

```

Il loop e la funzione ISR funzionano in modo indipendente però quando poi la funzione loop vede il flag cambia il codice che viene eseguito.

Se utilizziamo il codice sopra con OCR1A=62500 otteniamo un sampling 1Hz. Il risultato che otteniamo non è realmente troppo interessante perchè abbiamo delle variazioni e i valori che vengono stampati sono molto irregolari.



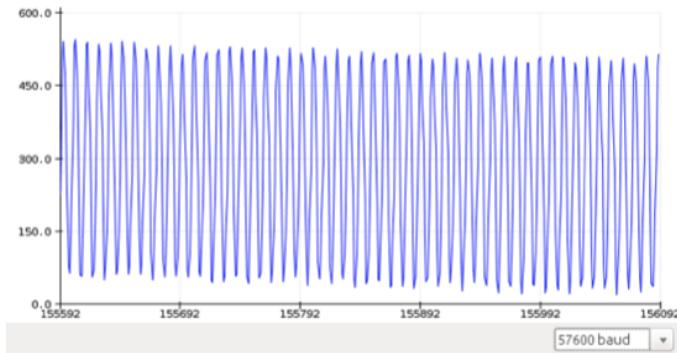
Ora vogliamo provare con OCR1A=1250, questo è il tipo di segnale che abbiamo visto corrisponde a 50Hz:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Nel grafico abbiamo una curva che abbiamo già visto con la precedente versione del nostro programma, è molto simile ad un alias della curva del seno, ricordare che qua noi vorremmo avere un valore costante e qui non ce l'abbiamo perchè c'è l'alias, però vediamo anche che non abbiamo più delle variazioni veloci e quindi la curva è più regolare, questo vuol dire che le misurazioni sono più precise.

Se abbiamo $OCR1A=125$ allora vuol dire che abbiamo una frequenza di 500Hz e questo vuol dire che abbiamo 10 sample ogni periodo della frequenza originale che è 50Hz.



Questo è più interessante, abbiamo una curva che ha ancora delle penedenze ed è probabilmente dovuto all'imprecisione dell'orologio. All'interno

del grafico abbiamo 500 punti ma non possiamo vedere il dettaglio della singola curva.

Nella maggior parte dei casi quello che succede è ogni 500 sample interrompiamo il trasferimento per qualche secondo e poi procediamo con un altro chunk di dati. Dato che qua abbiamo nel plot 500 data point è più semplice per me andarli a controllare e rappresentarli.

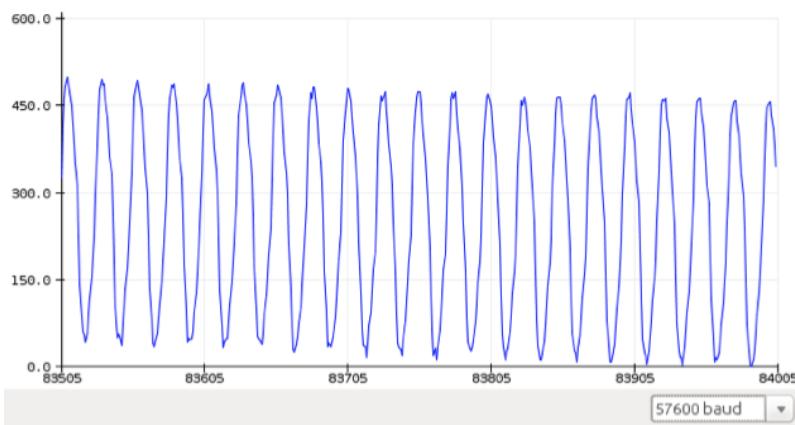
Consideriamo adesso un altro sketch, qui abbiamo la parte iniziale con il setup che è uguale a quella di prima e l'unica differenza è che abbiamo una threshold più bassa. La parte rilevante del codice in questo caso è la seguente:

```
ISR(TIMER1_COMPA_vect) { flag=1; }
int n=0;
void loop() {
    if (flag) {
        int adc=analogRead(A0);
        Serial.println(adc);
        flag=0;
        n++;
    }
    if (n==500) {
        delay(5000);
        n=0;
    }
}
```

Il programma qua colleziona 500 dati e quando li ha presi tutti e 500 si ferma per 5 secondi e poi va avanti nuovamente settando a 0 il numero di dati collezionati.

All'interno del loop abbiamo una serie di operazioni che non richiedono troppo tempo per essere eseguite ma abbiamo comunque una printline che prende tempo. Indipendentemente dal valore del timer non possiamo andare sotto al valore del tempo che è richiesto dalla printline e sappiamo che siamo nell'ordine di millisecondi.

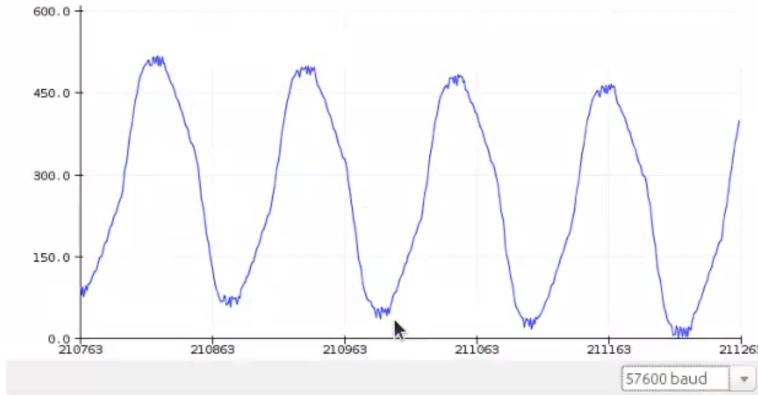
Questo è quello che succede, abbiamo una sampling frequency di 50 che vuol dire che abbiamo una frequenza più alta di 1KHz e qua possiamo vedere i dati che vengono raccolti.



Non possiamo però andare oltre 1KHz perchè? Consideriamo il loop, abbiamo delle operazioni che non richiedono troppo tempo ma comunque abbiamo una printline che richiede tempo, qualunque sia il valore della frequenza del timer e delle interruzioni prodotte dal timer, non possiamo andare sotto al valore del tempo richiesto dalla printline. Ogni printline prende circa 1 millisecondo, eseguendo l'esecuzione a 1KHz vuol dire che abbiamo solamente il tempo di fare la printline, quando vogliamo fare un sampling più serio dovremmo però andare oltre.

Un'altra opzione consiste nel raccogliere i dati in un buffer senza utilizzare la Serial operation. Una volta che abbiamo raccolto 500 dati all'interno del buffer (ho abbastanza spazio) posso mostrarli a schermo. Una volta che ho finito con i 500 dati posso usare un altro chunk e così

via, in questo modo otteniamo una sampling frequency di 6KHz. 6KHz è abbastanza per le frequenze audio e per la Nyquist Law possiamo eseguire il sample di un segnale con frequenza 3KHz.



Vediamo il codice, abbiamo per prima cosa la definizione del setup con tutte le variabili necessarie all'interno e qua in particolare definiamo anche BMAX=500 ovvero definiamo la dimensione del nostro buffer.

```
#define BMAX 50

void setup() {
    Serial.begin(57600);
    noInterrupts();           // Inibisco le interruzioni
    TCCR1A = 0;               // Registro di controllo
    TCCR1B = 0;               // Registro di controllo
    TCNT1 = 0;                // Azzero il contatore
    OCR1A = 10;              // La soglia per l'azzeramento del contatore
    TCCR1B |= (1 << WGM12); // Abilita conteggio/interruzione/azzeramento
    TCCR1B |= (1 << CS12); // Imposta il prescaler (a 256)
    TIMSK1 |= (1 << OCIE2A); // Generazione dell'interruzione
    interrupts();
}
```

Poi il nostro buffer lo dobbiamo dichiararare e poi lo possiamo andare ad utilizzare all'interno del loop. Notare che quando scatta l'interruzione

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

all'interno dell'handler andiamo a fare la `analogRead` e memorizziamo il contenuto all'interno del buffer. All'interno dell'handler abbiamo una operazione che comunque non è velocissima, richiede almeno 13 microsecondi.

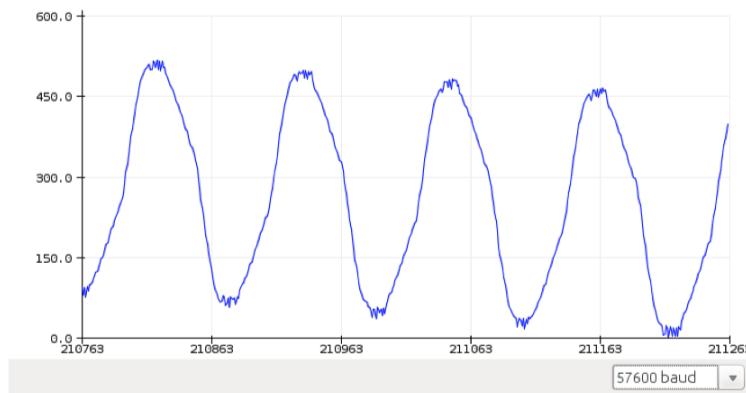
```

int n=0;      //il cursore
int A[BMAX]; //il buffer
ISR(TIMER1_COMPA_vect) {
    if (n<BMAX) { A[n++]=analogRead(A0); }
}
void loop() {
    if (n==BMAX) {
        for (int i=0; i<BMAX; i++) { Serial.println(A[i]); }
        delay(5000);
        n=0;
    }
}

```

All'interno del loop controlliamo se ho raggiunto la massima quantità di elementi nel buffer e poi se l'ho raggiunta mostriamo il contenuto del buffer.

Questo è quello che otteniamo:



non è eccezionale ma comunque è accettabile.

Il punto importante è che abbiamo imparato un modo per prendere un chunk di dati e poi memorizzarli nel buffer. Al momento siamo contenti con questa possibilità di raccogliere dati senza fare la printline ogni volta. La prossima cosa che vogliamo vedere è come calcolare il massimo, il minimo e la media di un certo segnale. Possiamo definire due variabili per memorizzare massimo e minimo, poi all'interno dell'ISR dobbiamo andare a considerare che oltre alla lettura che abbiamo fatto in precedenza dobbiamo anche assegnare i valori di massimo e minimo. Notare che non c'è una printline e abbiamo solamente l'operazione di lettura dei dati.

```

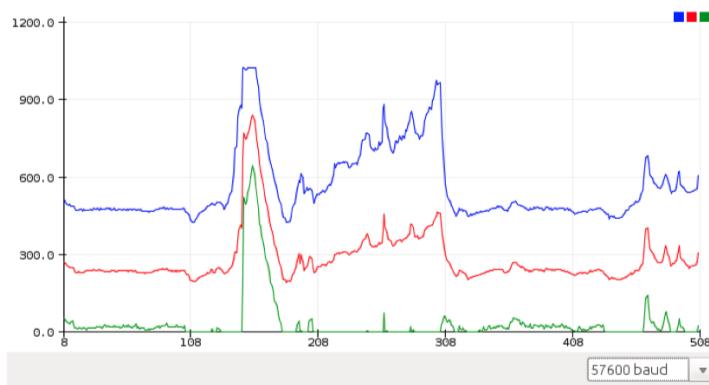
volatile int n=0;
volatile int massimo, minimo;
ISR(TIMER1_COMPA_vect) {
    if (n<BMAX) {
        int adc=analogRead(A0);
        if (n==0) {
            massimo=adc;
            minimo=adc;
        } else {
            massimo = (adc>massimo) ? adc : massimo;
            minimo = (adc<minimo) ? adc : minimo;
        }
        n++;
    }
}

```

Il loop è simile a quello che abbiamo avuto in precedenza, qua il delay è 100.

```
void loop() {  
    if (n==BMAX) {  
        Serial.print(massimo);  
        Serial.print(",");  
        Serial.println(minimo);  
        n=0;  
        delay(100);  
    }  
}
```

Questo è il grafico che otteniamo con i tre segnali:



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

13.2 Spectrum Analysis

Ora che abbiamo i nostri dati all'interno del buffer possiamo usare la trasformazione di Fourier che converte il segnale in un set di frequenze. La Fourier transform prende un segnale e lo quantizza in un set di frequenze in modo da avere una "firma" del segnale. Fortunatamente c'è una libreria.

Abbiamo una libreria per fare tutto ciò con Arduino, abbiamo dei limiti per la capacità di processare questi dati di Arduino, quindi non possiamo andare oltre 128 elementi contenuti nel buffer nel caso di Arduino 1.

Questo è il programma, definiamo l'oggetto che utilizziamo per eseguire la trasformazione che sarebbe l'arduinoFFT.

```
#define BMAX 128
#define FREQ 500L
#include <arduinoFFT.h>
arduinoFFT FFT = arduinoFFT();

void setup() {
    Serial.begin(57600);
    pinMode(13,OUTPUT);
    noInterrupts();           // Inibisco le interruzioni
    TCCR1A = 0;               // Registro di controllo
    TCCR1B = 0;               // Registro di controllo
    TCNT1 = 0;                // Azzero il contatore
    OCR1A = 1000000/(FREQ*16); // Soglia valida SOLO con prescaler a 256
    Serial.println(OCR1A);
    TCCR1B |= (1 << WGM12);   // Abilita conteggio/interruzione/azzeramento
    TCCR1B |= (1 << CS12);    // Imposta il prescaler (a 256)
    TIMSK1 |= (1 << OCIE2A);  // Generazione dell'interruzione
    interrupts();
}
```

Dobbiamo anche definire un semplice interrupt handler che prende il valore dell'input signal e setta questo valore all'interno dell'array che contiene le parti reali della nostra trasformazione mentre la parte immaginaria viene inizializzata a 0.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```

int n=0;
volatile double R[BMAX]; // Parte reale
volatile double I[BMAX]; // Parte immaginaria

ISR(TIMER1_COMPA_vect) {
    if (n<BMAX) {
        R[n]=analogRead(A0); // Parte reale con il segnale
        I[n]=0;           // Parte immaginaria inizializzata a 0
        n++;
    }
}

```

E questo è il programma che calcola la Fast Fourier Transform, ogni volta consideriamo se n è uguale a BMAX, poi vengono chiamate delle funzioni

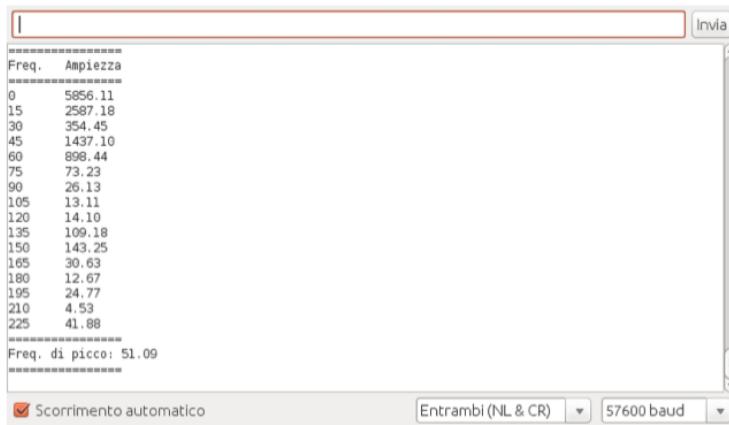
```

void loop() {
    if (n==BMAX) {
        FFT.Windowing(R, BMAX, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
        FFT.Compute(R, I, BMAX, FFT_FORWARD); // Calcolo della FFT
        FFT.ComplexToMagnitude(R, I, BMAX); // Generazione dello spettro
        double x = FFT.MajorPeak(R, BMAX, FREQ); // Frequenza di picco
        double k= FREQ/BMAX; // Conversione indice -> frequenza
        Serial.println("=====");
        Serial.println("Freq. |tAmpiezza");
        Serial.println("=====");
        for (int i=0; i<BMAX/2; i++) {
            Serial.print(i*k, 0);Serial.print("\t");Serial.println(R[i]);
        }
        Serial.println("=====");
        Serial.print("Freq. di picco: "); Serial.println(x,2);
        Serial.println("=====");
        delay(1000);
        n=0;
    }
}

```

L'output che otteniamo da un sampling a 500Hz è il seguente, tra 45Hz e 6 c'è un picco, ce ne sta un altro tra 135 e 150 e uno sta 195 e 210.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Il risultato che otteniamo è accettabile, non perfetto.

Chapter 14

MQTT

È un protocollo Application Level che funziona sopra allo stack generale di internet ma è pensato per la comunicazione machine to machine e per i dispositivi low power. MQTT possiamo anche embeddarlo sopra alla comunicazione di Bluetooth o di Zigbee volendo.

Conventional Internet protocol suite		
Layer	Protocol	Features
Application	HTTP	Application-level access to information, client/server
Transport	TCP/UDP	Communication channels with guarantees (TCP); just an interface to IP (UDP)
Network	IP	Addressing & routing; best-effort

In Internet la protocol suite è formata da tre layer del protocollo:

- Network layer dove il protocollo è IP, le feature di IP sono addressing e routing. IP non è affidabile.

- L'affidabilità l'otteniamo a livello trasporto dove abbiamo i canali di comunicazione con TCP, l'altro protocollo che è UDP è solo una interfaccia per IP e non garantisce la sicurezza.
- All'Application Layer abbiamo un application level access alle informazioni, il client per esempio fa una get query e il server risponde con la risposta.

Questo non riflette le vere richieste dei dispositivi IoT perchè i protocolli sono stati pensati per computer quando invece ora questi protocolli sono usati da IOT devices. Per essere connessi ad internet i dispositivi IOT devono implementare questi protocolli come IP e TCP che però non sono pensati per questi dispositivi.

Come possiamo fare a rendere i dispositivi IOT adatti per internet?
I dispositivi IOT hanno vari requisiti, a differenti livelli:

IoT devices requirements	
Network requirements	Impact on networking
Scalability / redundancy	Multi-hop, mesh networking
Security	Configurable, with different security levels for different devices capabilities
Addressing	Scalability of address space, low overhead on network protocols
Device requirements	Impact on application-level
Low power / battery powered	Low duty-cycle communications
Limited capacity (memory/processor)	Small footprint, low complexity protocols
Low cost	Reliability of the device, further constraints on the devices

- Se ci limitiamo ai requisiti di rete abbiamo necessità di scalabilità e ridondanza, considerare che ci aspettiamo tanti dispositivi nella rete perchè non devono essere usati direttamente dall'uomo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

ma sono indipendenti ed autonomi. Dovrebbero anche scalare in spazi dove non sono necessariamente presenti dei router. Questo si riflette sulla necessità di avere il multi hop o delle mesh network per estendere la rete.

- Sempre dal punto di vista della rete dobbiamo avere dei meccanismi di sicurezza. Internet fornisce dei meccanismi di sicurezza ma ci sono anche delle richieste per la sicurezza dei dispositivi IoT. La sicurezza di questi dispositivi deve essere configurabile con differenti livelli di sicurezza a seconda della differenti capacità del dispositivo.
- Per quanto riguarda l'addressing, abbiamo degli address space scalabili a causa del fatto che stiamo utilizzando dei dispositivi IoT.
- Per quanto riguarda il device IoT in se per se abbiamo un dispositivo che ha una bassa capacità energetica ed è alimentato a batteria.
- Abbiamo un dispositivo low cost.
- Abbiamo un dispositivo che è limitato in termini di processore e di memoria.

Gli ultimi tre punti hanno un impatto anche sull'application layer.

Tutti questi requisiti hanno differenti impatti sulla rete e sull'application layer, un esempio di come tutti questi requisiti vengono rispettati è MQTT.

14.1 MQTT

MQTT sta per Message Queuing Telemetry Transport è stato pensato come un protocollo di trasporto per mandare dati da un sensore ad un

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

computer, funziona come un pub sub protocol. MQTT è stato pensato come un protocollo pub/sub ed è veramente Lightweight, possiamo implementarlo con poco codice, richiede una bandwith veramente bassa e questo garantisce performances migliori di HTTP.

Il fatto di avere poco codice per implementare MQTT non è una cosa universale che vale per tutti i dispositivi che utilizzano il sistema, MQTT definisce ruoli asimmetrici per i dispositivi che stanno nel dispositivo. Alcuni dispositivi hanno effettivamente poco codice, pensiamo ad un Arduino Uno che ha solamente 4KB di memoria, allo stesso tempo per funzionare abbiamo bisogno di un dispositivo più potente che esegue l'altra parte del protocollo ed esegue le operazioni più impegnative.

Il modello generale di MQTT segue il paradigma pub sub ma a low level abbiamo un paradigma client server. Sembra una contraddizione ma poi diventa chiaro.

Il client server è ad un livello molto basso quindi il risultato che viene offerto all'utente poi è un meccanismo pub sub. La chiave è che è semplice da implementare sul client perchè la complessità è lasciata tutta sul server che fa anche il lavoro del client. MQTT può garantire differenti livelli di Quality of Service (che sono configurabili), è data agnostic, possiamo trasferire qualsiasi cosa tramite MQTT perchè dal punto di vista di MQTT stiamo trasferendo dei dati che sono solamente un po' di bit, niente di più, non dobbiamo guardare quali sono i tipi di dati che passiamo da una parte all'altra. È anche completamente automatico e possiamo usarlo per Mobile to Mobile communication e per applicazioni IOT.

MQTT è nato nel 1999 e poi è entrato nello standard OASIS (che ora è responsabile di tenerlo aggiornato e di decidere il suo funzionamento) e quindi poi ha preso anche una porta in TCP/IP che è la 1883 normalmente e 8883 se usiamo anche SSL. Se vogliamo usare la sicurezza con SSL però aggiungiamo overhead e questo potrebbe non essere adeguato per dispositivi low power, ad esempio con Arduino è praticamente impossibile.

MQTT è molto popolare in tanti campi e in tante applicazioni, anche Facebook Messenger tanto tempo fa usava MQTT. Quello che vediamo noi è MQTT 3.1.1 e in particolare vediamo una sintesi del protocollo, la versione che vediamo è del Novembre 2014.

14.1.1 PUB SUB Paradigm

Il paradigma Pub sub è stato introdotto in un paper nel 2003 come alternativa al classico paradigma client-server. In client-server abbiamo due ruoli, il client che invia richieste e il server che risponde. Nel caso di Pub-Sub definiamo tre attori, due attori sono il publisher e il subscriber che sono entrambi client e la cosa bella è che publisher e subscriber non si conoscono a vicenda perchè tutta la comunicazione avviene tramite un service che è il terzo attore, questo viene solitamente chiamato broker. Quindi il protocollo è Pub-Sub se consideriamo la comunicazione tra publisher e subscriber ma se ci riferiamo alla comunicazione subscriber broker e publisher broker allora abbiamo una comunicazione client-server.

Il broker è un componente che viene già fornita mentre il publisher e il subscriber sono quelli che dobbiamo andare ad implementare.

Il broker è quello che deve essere conosciuto sia dal subscriber che dal publisher, per creare una rete MQTT dobbiamo fare in modo che sia il publisher che il subscriber si connettano al broker. Tutta la comunicazione avverrà solamente tramite il broker, quindi:

- Dal publisher al broker.
- Dal broker al subscriber.

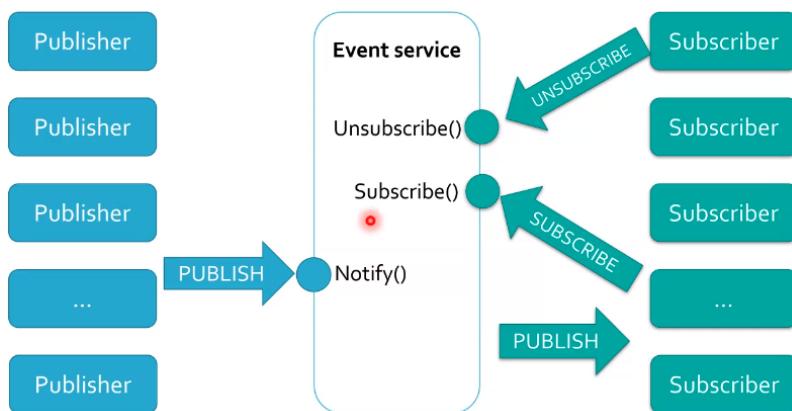
Publisher e subscriber non si conoscono tra loro e non mandano direttamente una comunicazione tra di loro.

L'idea è che il publisher produce eventi che sono i dati che produce, ad esempio un sensore può essere un publisher e trasmette queste informazioni producendo un evento, facendo questo interagisce solamente con

un broker. Il subscriber invece esprime l'interesse per un evento o per un pattern di eventi e riceve quindi una notifica dal broker ogni volta che il publisher produce un evento che matcha le sue richieste. Questo meccanismo fa sì che il publisher e il subscriber siano completamente separati in termini di tempo, di spazio e di sincronizzazione.

Il broker, chiamato anche Event Service quindi è nel mezzo, riceve tutti i messaggi che vengono pubblicati e poi li inoltra in base all'espressione di interesse dei nodi inoltre gestisce anche le richieste di subscription e di unsubscription.

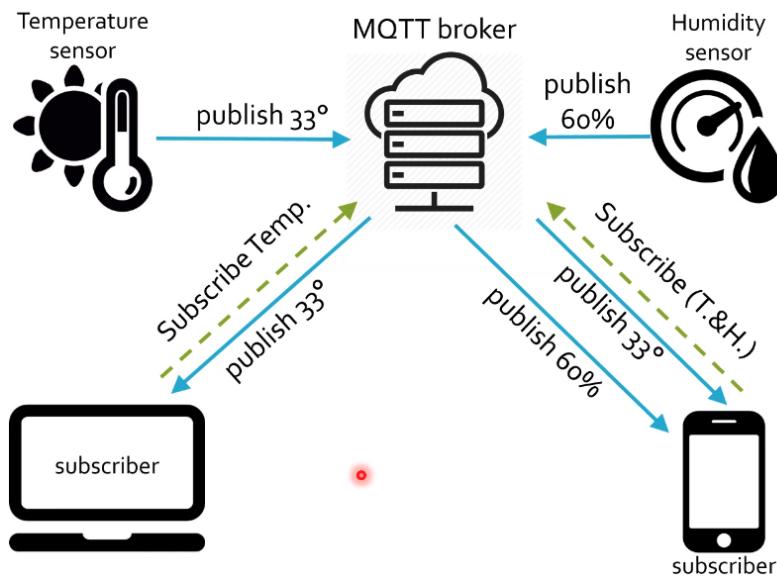
Il meccanismo di Pub/Sub può essere sviluppato in molti modi differenti, tipicamente abbiamo un broker che è un agente indipendente ovvero un servizio installato in un certo computer che accetta le operazioni di publish, subscribe, notify e unsubscribe.



La situazione è quella della immagine qua sopra, abbiamo un certo numero di publisher che pubblicano il loro messaggio all'event service. Quando arrivano questi messaggi dai publisher viene sollevato un evento `notify()`. I subscriber invece possono mandare un messaggio di `subscribe` o un messaggio di `unsubscribe` al broker e a seconda del tipo di messaggio si avvia un evento `subscribe` o un evento `unsubscribe`. Il broker poi pubblicherà i dati ai vari subscriber ovvero i messaggi che sono arrivati

dal publisher verranno inoltrati al subscriber che aveva espresso interesse per quel tipo di messaggio.

Vediamo un esempio:



Qui abbiamo due sensori, uno per la temperatura e uno per l’umidità. Abbiamo due subscriber e un broker. Uno dei due sensori fa il subscribe per la temperatura mentre l’altro sia per la temperatura che per l’umidità. Ogni volta che il sensore di temperatura pubblica un dato, questo viene inviato direttamente ai due subscriber mentre se pubblica qualcosa il sensore di umidità allora l’informazione verrà inoltrata solamente ad uno dei due subscriber.

Il meccanismo di pub/sub implementa tre tipi di decoupling nel publisher e nel subscriber:

- Il primo è in termini di spazio perchè publisher e subscriber non hanno bisogno di conoscersi tra loro e non condividono nulla, ad esempio non conoscono gli IP l’uno dell’altro perchè non devono stabilire una connessione TCP.

- Inoltre forniscono time decoupling perchè non abbiamo bisogno che il publisher e il subscriber siano attivi nello stesso momento per funzionare. Un publisher potrebbe inviare dei dati quando il subscriber non è ancora attivo, poi quando il subscriber si collega allora il broker gli invierà i dati successivi e volendo anche quelli che sono stati inviati in precedenza.
- Implementano inoltre una synchronization decoupling perchè le operazioni sono indipendenti tra publisher e subscriber, non sono sincrone.

Questo meccanismo solitamente permette di avere una migliore scalabilità dell'approccio client server, è tutto nelle mani del broker, dipende molto da come scala il broker. Dato che le operazioni sono semplici allora il broker può essere facilmente parallelizzato, infatti in generale nelle situazioni più complesse se vogliamo scalare abbiamo necessità di parallelizzare il broker. Allo stesso tempo la parallelizzazione del broker e la sua implementazione non è qualcosa che riguarda l'utilizzatore dei metodi publisher e subscriber.

Per fare in modo che il broker possa inoltrare i messaggi al subscriber, il broker deve sapere come matchare i messaggi con la subscription e questo avviene tramite un filtro che deve essere utilizzato. Abbiamo vari tipi di filtri che possiamo utilizzare:

- Il primo si basa sul topic del messaggio che abbiamo ricevuto, ogni messaggio infatti è associato ad una descrizione del contenuto del messaggio, quindi il subscriber potrebbe fare il subscribe per un topic specifico e solamente se abbiamo un match allora il messaggio può essere inoltrato al subscriber.
- In alternativa può dipendere dal contenuto, si tratta di una sorta di query. Per esempio possiamo fare il subscribe per valori della temperatura che sono maggiori di 30 gradi e in tal caso il broker

manderà solamente quel tipo di messaggio al subscriber. Notare che in questo caso i messaggi che vengono pubblicati non possono essere criptati o almeno il broker deve avere la chiave per decriptarli perchè deve leggere il contenuto.

- Una terza opzione è basata sul tipo, è tipica dei linguaggi OO e in questo caso possiamo filtrare in base al tipo del messaggio.

Per implementare il filtering il publisher e il subscriber devono essere d'accordo prima dell'inizio della comunicazione sui vari topic e anche sui tipi, allo stesso modo anche il broker deve essere al corrente. Se utilizziamo solamente i topic invece dal punto di vista del broker in realtà non ha molto importanza perchè fa solamente un match tra i topic subscribed e i topic published, quindi non ha bisogno di conoscere questi topic. Se usiamo i topic, il broker non fa analisi sul contenuto del messaggio e in questo caso il messaggio può essere criptato, in questo caso il broker può trasferire qualsiasi tipo di messaggio senza pensare a cosa sta trasferendo. Il publisher non può assumere che qualcuno sta ascoltando il messaggio, c'è la possibilità che il messaggio non sia ricevuto da nessuno.

Nel caso di MQTT si utilizza un meccanismo Pub/Sub e abbiamo bisogno che publisher e subscriber siano connessi con il broker dove il broker è un server su un altro computer, quindi hanno bisogno di sapere l'ip e la porta dove è connesso il broker. In MQTT la consegna del messaggio dal publisher al subscriber avviene quasi in real time. In ogni caso c'è da tenere a mente che il meccanismo è completamente asincrono. Il subscriber potrebbe essere offline quando il messaggio viene inviato, quando si connette di nuovo al broker però poi può ricevere i vecchi messaggi. Il meccanismo Invece è in near-real-time se publisher e subscriber sono sempre connessi entrambi, questo in generale non è garantito.

MQTT esegue il decoupling della sincronizzazione come nei vari meccanismi di publisher subscriber, publisher e subscriber sono asincroni, è

possibile comunque implementare la sincronizzazione se la vogliamo e possiamo utilizzare delle api sincrone.

MQTT è anche facile da utilizzare da client side perchè la complessità è completamente nel broker, per questo motivo MQTT è destinato in particolare ai low power devices. Il broker è installato in un server che non è un low power device, è qualcosa di più potente.

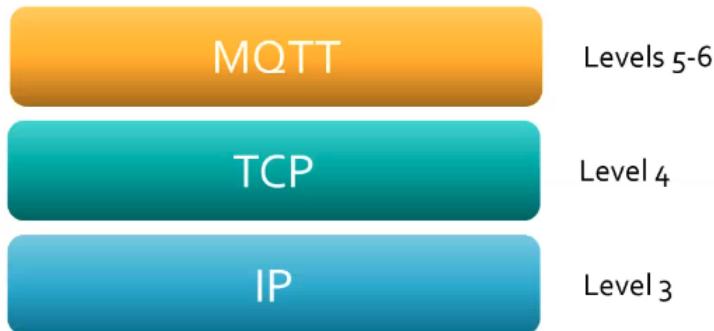
MQTT basa il filtraggio sui topic, questo vuol dire che ogni messaggio dovrebbe contenere un topic a cui viene associato un contenuto. Non ci sono regole su questi topic ma abbiamo delle best practice che vanno seguite.

Si suggerisce di organizzare i topic in una gerarchia in modo che possiamo essere in grado di gestire i topic ed in caso anche di estenderli.

MQTT garantisce anche QoS levels per i Messaggi che vuol dire che garantisce affidabilità nell'invio dei messaggi. Viene utilizzato TCP per la trasmissione dei messaggi e questo già mi garantisce affidabilità nell'invio dei dati perchè implementa l'ACK e la ritrasmissione ma se la connessione è rotta però non riusciamo a inviare nulla e perderemo anche i dati che verranno inviati dopo che la connessione non funziona. Dato che MQTT lavora ad un livello più alto allora è in grado di recuperare anche da situazioni in cui TCP diventa non funzionante.

Abbiamo tre livelli di QOS in MQTT, 0, 1 e 2. Il livello 0 indica che non c'è un QoS e che quindi utilizziamo solamente la QoS derivante da TCP mentre i livelli 1 e 2 garantiscono una maggiore affidabilità.

Nello stack abbiamo che MQTT si trova sopra a TCP e ovviamente sopra ad IP, possiamo vederlo come un protocollo di livello applicazione perchè si trova sopra a TCP ma se consideriamo lo standard ISO OSI avremmo altri due livelli sopra al livello trasporto quindi in questo caso MQTT garantirebbe delle funzionalità da session layer. Va bene sia chiamarlo middleware level sia protocollo a livello applicazione.



Prima di pubblicare o di ricevere un messaggio un client dovrebbe connettersi ad un broker e questo lo facciamo quando inviamo un connect message. Il connect message contiene:

- Un client ID che è solamente un identificatore numerico. È una stringa che identifica in modo univoco il client quando si connette al broker. Non possono esserci due client che fanno finta di essere lo stesso client. Questo client ID può essere vuoto, in questo caso è il broker che alloca un nome per quel client. Notare che se non inviamo un client ID allora il flag Clean Session deve essere TRUE.
- Abbiamo un flag per la Clean Session, questo è TRUE se vogliamo cancellare tutte le precedenti informazioni che abbiamo riguardo ad un certo client e che abbiamo registrato nelle precedenti sessioni. Se invece il flag è FALSE allora richiediamo che il broker mantenga lo status della sessione e verranno memorizzati i dettagli del client come ad esempio le subscription, i messaggi che non sono stati inviati a quel client (se abbiamo un QoS di primo o secondo livello) in questo modo se abbiamo una precedente sessione la ripristiniamo. Dopo la disconnessione se il flag è a false lo stato viene mantenuto.
- Username e Password, attenzione perchè vengono inviate in Clear Text e comunque sono opzionali

- Poi abbiamo un Will Flag che è opzionale. Questo flag lo utilizziamo quando un client si disconnette senza comunicarlo. Quando il broker se ne accorge notificherà gli altri client della disconnessione.
- Abbiamo il Keep Alive che è opzionale. Questo serve perchè in questo modo il client informa il broker che invierà dei periodici control packet per fare in modo che se il client o il link dovessero andare offline, il broker non riceverà il messaggio e quindi capirà che quel client è andato offline. Se vogliamo spegnere il Keep Alive ci basta settarlo uguale a 0.

Il broker quando riceve una richiesta di connessione da parte di un client risponde sempre con un ACK che nello specifico è il CONNEC-TACK, se invece la connessione non va a buon fine viene inviata una notifica per notificare che è avvenuto un errore.

Quando inizia la connessione inoltre il broker manda al client anche una informazione per far capire se erano presenti dei dati di una precedente connessione e se sono stati recuperati oppure no.

Subito dopo la connessione, il client che vuole pubblicare un messaggio lo può pubblicare, il messaggio in MQTT è molto semplice e contiene un topic e il payload. Il contenuto del payload in MQTT è un array di byte, il topic invece è una stringa. MQTT è completamente agnostico riguardo al contenuto del payload, il broker non deve controllare e leggere il contenuto del payload, lo prende così come è e poi lo passa al subscriber, il contenuto può essere una qualsiasi cosa.

Contenuto del messaggio publish:

- Contiene un identificatore del pacchetto che in pratica è un sequence number. Se il QOS è 0 allora possiamo utilizzare 0 come packetId.
- topicName: è una stringa, indica il topic di questo messaggio che stiamo inviando.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- QoS level per il messaggio che stiamo inviando, i possibili valori sono 0, 1 o 2.
- retainFlag: ci indica quando il messaggio deve essere mantenuto dal broker come ultimo valore possibile per questo topic, questo può essere applicato a tutti i messaggi che vengono pubblicati, non ha niente a che fare con le persistent sections. In questo modo quando un subscriber si connette otterrà questo messaggio, se nel frattempo un altro publisher invia un messaggio con lo stesso topic allora quello del broker verrà sovrascritto. Questo è utile se abbiamo un publisher che invia pochi dati, in questo modo il subscriber che fa il subscribe per un topic aspetterà troppo tempo per ricevere qualcosa, usando il retainFlag invece il subscriber riceve subito qualcosa e questo gli permette di capire che quel flag è realmente esistente e che il client pubblica effettivamente dei messaggi.
- duplicateFlag: serve per indicare che un certo messaggio è un du-
plicato di un altro messaggio per cui non abbiamo ricevuto l'ack.
Ha senso avere questo flag solamente nel caso in cui abbiamo il
livello del QoS maggiore di 0.

Quando il broker riceve il messaggio di publisher manda l'ack se $\text{QoS} > 0$, processa il messaggio, poi invia il messaggio ai vari subscriber.

Il publisher invece una volta che ha inviato il messaggio al broker se ne dimentica completamente, solo nel caso in cui abbiamo un $\text{QoS} > 0$ allora deve pensare a ricevere l'ACK e poi in caso a ritrasmettere il messaggio.

Dal punto di vista dei subscriber, esiste un messaggio SUBSCRIBE per eseguire il subscribe ad un certo topic. All'interno del messaggio di SUBSCRIBE abbiamo i seguenti componenti:

- Abbiamo il packetId che indica un sequence number del subscriber, è obbligatorio in questo caso.

- Poi abbiamo il campo topic in cui possiamo inserire una stringa per fare in modo di ricevere i messaggi che vengono inviati con quel topic. Se ne abbiamo più di uno ne scriviamo più di uno.
- Abbiamo il campo desired QoS perchè indichiamo la QoS che vorremmo avere ma non siamo sicuri che effettivamente verrà utilizzata dal broker, dipende da molti fattori. Anche questa se abbiamo vari topic viene ripetuta più volte.

Il broker quando riceve un messaggio di subscribe risponde al subscriber con un messaggio SUBACK che contiene:

- packetId: è lo stesso ID che viene utilizzato nel messaggio SUBSCRIBE.
- returnCode: abbiamo un valore numerico che viene restituito per ognuno dei vari topic per cui facciamo il SUBSCRIBE. Se ci sono stati errori viene restituito 128, se non ci sono stati errori invece restituiamo 0, 1 o 2 in base al tipo di QoS che viene scelto.

Il broker deve mandare un SUBACK perchè devo informare il client che ho ricevuto il messaggio di SUBSCRIPTION e devo anche informarlo se ci sono stati degli errori. Altrimenti, se non usiamo l'ACK, se ci sono stati errori o se il messaggio non è stato recapitato al broker il client che ha fatto la richiesta è convinto di aver fatto il subscribe e di ricevere poi dei messaggi ma in realtà non riceverà niente.

Un client può anche fare l'UNSUBSCRIBE per bloccare l'invio di nuovi messaggi da parte del broker, in questo caso il messaggio di UNSUBSCRIBE contiene più o meno gli stessi campi del subscribe:

- packetId
- Lista dei topic che voglio non seguire più

Anche in questo caso il broker poi risponde con un ACK in cui viene inserito il packetId in modo da essere sicuri di aver eseguito l'UNSUBSCRIBE.

I topic sono stringhe che sono organizzati in una gerarchia e sono separati da ”/”. All'interno di queste stringhe dei topic possiamo anche utilizzare delle wildcards:

- *home/firstfloor/ + /presence*: questo indica che vogliamo fare il subscribe a tutti i messaggi con topic presence in tutte le stanze del primo piano, qua utilizziamo il + per indicare tutti.
- *home/firstfloor/#* qua invece indichiamo tutti i sensori del primo piano.
- Abbiamo anche dei topic che cominciano con la \$ per ottenere statistiche interne di MQTT e nello specifico del broker. Questo tipo di topics non sono standardizzati quindi dipendono dal tipo di broker che stiamo utilizzando. Alcuni esempi presi da HiveMQ: *\$SYS/broker/clients/connected* indica il numero di clients che sono connessi.

È suggerito organizzare i topic in una gerarchia che abbia senso e che sia estensibile perchè potremmo sempre dover tornare al nostro codice per mantenerlo, specialmente se siamo in uno scenario IoT diventa difficile fare cambiamenti se installiamo nuovi sensori o cambiamo la loro posizione, quindi abbiamo bisogno di una situazione abbastanza flessibile. Se abbiamo una rappresentazione flessibile non abbiamo bisogno di fare troppe modifiche.

In generale non ci sono altre regole specifiche, le best practice dicono di:

- Non iniziare la definizione del topic con uno slash.
- Mettere nomi in cui non ci sono spazi.

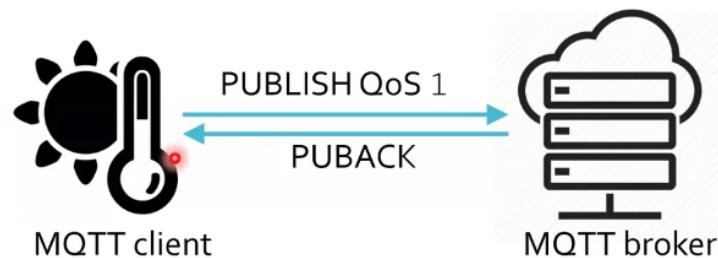
- Utilizzare topic con stringhe abbastanza brevi, li stiamo utilizzando su dispositivi IoT che hanno una capacità molto bassa, quindi se mettiamo delle stringhe molto lunghe avremmo molto overhead.
- Usare solamente UTF-8.
- Si deve utilizzare una gerarchia perchè in questo modo riusciamo ad estendere più facilmente i topics.
- È preferibile utilizzare dei topic specifici invece di topic troppo generici
- In alcuni casi possiamo embeddare nel topic il client ID, in modo da fare in modo che si possa identificare il sensore e in modo che posso fare il subscribe per quello specifico sensore.
- Evitare di fare il subscribe a tutti i messaggi utilizzando `#` perchè riceviamo troppi dati. È possibile ma è meglio evitare, può essere utile solamente se vogliamo fare un log di tutto lo stato del broker ma normalmente non viene mai utilizzato.

Per quanto riguarda il QOS in MQTT, solitamente è un accordo tra due peer, nel caso di MQTT è un accordo tra il publisher e il broker oppure tra il subscriber e il broker. Ci sono tre livello di QoS:

- QoS Level 0: questo vuol dire al più uno, si basa solamente sull'affidabilità del TCP e non implementa altri metodi per essere sicuro che i messaggi vengano consegnati correttamente. È un metodo best effort, non riceviamo un ACK dal ricevente e questo ci fornisce le stesse garanzie del TCP.



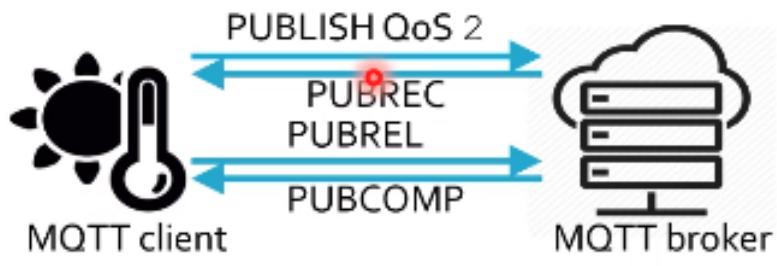
- QOS Level 1: qui abbiamo "at least one" vogliamo essere sicuri che il destinatario riceva il messaggio almeno una volta. Il broker in questo caso manda un ACK quando riceve il messaggio, lo stesso si applica tra broker e subscriber. In questo caso se il mittente non riceve l'ACK poi il messaggio viene inviato di nuovo.



- QOS Level 2: qui abbiamo "exactly once" e serve per evitare duplicati. È il livello più alto e richiede un doppio handshake tra il client e il broker.

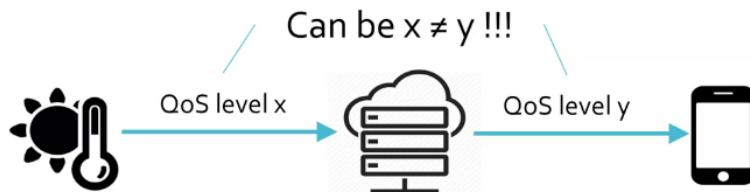
Il client in questo caso pubblica il messaggio e il broker lo riceve mandando indietro un ACK PUBREC al client. Il messaggio però viene mantenuto nel broker in attesa che il client invii un PUBREL, quando il broker riceve il PUBREL può inviare un PUBCOMP indietro al client. Tutto questo serve per essere sicuri di ricevere il messaggio solamente una volta ed evitare quindi di avere dei duplicati e in caso di eliminarli. Se ad esempio si dovesse perdere

il PUBREC il client andrebbe a inviare nuovamente il messaggio al broker ma dato che il broker mantiene lo stato del messaggio capisce che si tratta di un duplciato ed elimina il messaggio. Lo stato del messaggio deve essere mantenuto dal broker fino a quando non si riceve il PUBREL. Se il PUBCOMP si perde, il client invierà di nuovo il PUBREL e poi il broker risponderà nuovamente con PUBCOMP.



Altre cose da tenere a mente di QoS.

In generale la comunicazione tra publisher e broker è differente dalla comunicazione tra broker e subscriber perché il broker sta nel mezzo, la negoziazione del QoS avviene in modo separato tra le due comunicazioni, in generale questo vuol dire che il QoS tra publisher e broker può essere differente di quello tra broker e subscriber. Notare anche che per un singolo publisher possiamo avere vari subscriber che ricevono messaggi e in generale questi possono avere ognuno un QoS differente.



Se abbiamo per esempio una QoS 2 richiesta dal publisher, anche se nessuno dei subscriber fa il subscribe con il QoS uguale a 1 o a 2, il

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

broker comunque manterrà in memoria lo stato dei messaggi inviati dal publisher perchè non può sapere se un client farà il subscribe più tardi con una QoS 1 o 2 o anche se un client che si era disconnesso tornerà online più tardi.

Possiamo anche evidenziare delle best practice per la scelta del QoS:

- Scegliamo un QoS di livello 0 quando la connessione è affidabile e stabile, c'è sempre da ricordarsi che comunque sotto abbiamo TCP, quindi comunque abbiamo affidabilità. Possiamo usare QoS = 0 quando il singolo messaggio non è così importante perchè magari i messaggi vengono aggiornati molto frequentemente.
- Possiamo usare il QoS = 1 quando vogliamo essere sicuri che tutti i messaggi siano ricevuti e quando vogliamo gestire i duplicati.
- Scegliamo QoS = 2 se non vogliamo gestire i duplicati, ovviamente abbiamo un overhead maggiore.

14.1.2 Persistent Sections

Con le persistent sessions il broker memorizza lo stato della comunicazione con il client.

- Se un subscriber si disconnette, quando poi si riconnetterà non avrà bisogno di definire nuovamente i topic a cui è interessato perchè il broker se li ricorderà. Inoltre potrà ricevere anche tutti i messaggi che avrebbe dovuto ricevere mentre era offline, in particolare si tratta di messaggi per cui il broker non ha ricevuto una conferma, questi messaggi vengono inviati di nuovo nel momento in cui abbiamo una QoS pari a 1 o a 2.
- Il client può richiedere una persistent session nel momento in cui si connette e lo fa andando a settare il flag cleanSession a false.

- Anche se decidiamo di avere una persistent section non è detto che potremo poi memorizzare questi dati all'infinito, i dati verranno infatti memorizzati fino a quando il sistema lo permette.
- Se abbiamo un QoS pari a 2 anche il client dovrà mantenere uno stato con il messaggio che aveva inviato al broker in attesa della conferma della ricezione.

Possiamo evitare delle sessioni persistenti per client che sono solamente publisher che utilizzano QoS uguale a 0 o se i vecchi messaggi non sono importanti.

14.1.3 Retained Messages

Altre note riguardo ai retained messages, sono utilizzati perchè il publisher non ha alcuna garanzia che il messaggio sia stato effettivamente inviato al subscriber, possiamo solamente garantire che il messaggio arrivi al broker. Allo stesso modo se un client si connette al broker e fa il subscribe non può sapere quando arriveranno nuovi messaggi, dipende solamente da quando il publisher pubblicherà nuovi messaggi e può passare del tempo. Consideriamo ad esempio un dispositivo che pubblica il suo stato ON o OFF, magari lo accendiamo, passa ad ON e ci rimane per molto tempo, un subscriber che fa il subscribe per lo stato del dispositivo non potrà sapere niente fino a quando il dispositivo non verrà spento. Se invece utilizziamo il retained message invece possiamo riuscire a capire direttamente lo stato del dispositivo. Un altro esempio è quello in cui vogliamo conoscere lo stato di una porta, se siamo un subscriber che vuole ricevere lo stato di questa porta allora sarà corretto utilizzare il retainFlag a true. Per richiedere un retain message dobbiamo utilizzare retainFlag = True, in questo modo il broker andrà a memorizzare l'ultimo messaggio che è stato inviato dal publisher, solo l'ultimo, gli altri li cancellerà perchè non è una persistent storage. Se vogliamo cancellare

un retained message possiamo inviare un messaggio di retained vuoto sullo stesso topic.

14.1.4 Last Will

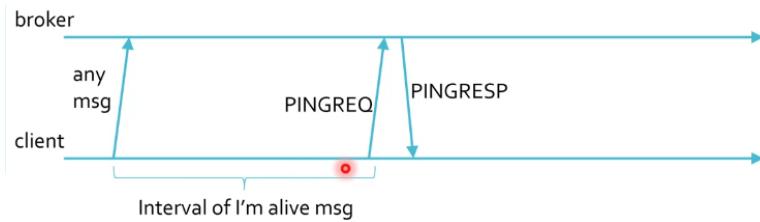
Il meccanismo Last will è utilizzato per informare i client che un certo client si è disconnesso senza prima annunciarlo. Al momento della connessione se richiediamo al broker di utilizzare Last Will dovremo anche indicare un messaggio con un topic e tutte le altre informazioni solite di un messaggio che il broker dovrà mantenere e dovrà poi inviare ai subscriber nel momento in cui il client si disconnette senza annunciarlo.

Se invece il client si disconnette con una procedura standard, il last Will Message non viene inviato. Il Last will quindi potrà essere utile se avviene un errore di rete, se il client non invia il KeepAlive, se si disconnette senza preavviso e se il broker chiude la connessione con il client a causa di un errore nel protocollo.

Il last will è specificato nel messaggio CONNECT e contiene 4 campi opzionali, possiamo indicare il topic del lastWill, il QoS del last will, il messaggio e il retain. I messaggi di Last Will sono spesso utilizzati con i retain message, ad esempio se un dispositivo pubblica il suo status ON e OFF e si spegne all'improvviso ai subscriber (quando si connetteranno) verrà inviato un messaggio in cui si indicherà che il dispositivo ora è offline.

14.1.5 Keep Alive

È un meccanismo per rendersi conto se il client e la sua connessione con il broker, sono ancora attive o no. Il client invia un periodico messaggio al broker per indicare che è ancora vivo. La frequenza di questi messaggi viene indicata nel messaggio CONNECT. Tutti i messaggi che inviamo per il Keep Alive devono anche prevedere l'invio dell'ACK da parte del broker.



Il messaggio di ping dal client non viene inviato sempre ma solamente se passa troppo tempo dal momento in cui il client invia l'ultimo messaggio, se in quel periodo non viene inviato nessun messaggio allora il client invia un PINGREQ e il broker risponderà con un ACK. Se il client non invierà il suo Keep alive in tempo allora il broker andrà a chiudere la connessione con quel client.

14.1.6 Struttura dei pacchetti di MQTT

Questa è la struttura di un pacchetto MQTT, ci sono dei header fissati, degli header variabili e il payload.

Fixed header, present in all MQTT control packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

- Il fixed header è formato da due byte ma può anche essere esteso se un certo bit viene settato.

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT control packet type				Flags specific to each MQTT control packet type			
byte 2...	extra length		Remaining length					

Il primo byte codifica il tipo di control packet di MQTT, ovvero ci dice quale è il tipo di pacchetto che stiamo mandando e alcuni flag

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

associati con il pacchetto. Per quanto riguarda il control packet abbiamo i seguenti tipi:

Name	value	direction of flow
reserved	0	forbidden
CONNECT	1	client to server
CONNACK	2	server to client
PUBLISH	3	client to server or server to client
PUBACK	4	client to server or server to client
PUBREC	5	client to server or server to client
PUBREL	6	client to server or server to client
PUBCOMP	7	client to server or server to client
SUBSCRIBE	8	client to server
SUBACK	9	server to client
UNSUBSCRIBE	10	client to server
UNSUBACK	11	server to client
PINGREQ	12	client to server
PINGRESP	13	server to client
DISCONNECT	14	client to server
reserved	15	forbidden

- Il variable header contiene il packetId, solamente l'operazione di CONNECT e di CONNACK non contengono il packetId. Inoltre sono contenute delle informazioni che dipendono dal control packet.
- Riguardo al payload, in molti casi non lo abbiamo, nel caso del connect ad esempio è necessario e contiene i dati riguardanti il client, nel caso del PUBLISH ad esempio è opzionale e contiene il messaggio.

14.2 MQTT e Arduino

Esiste una libreria per MQTT che funziona anche su Arduino e che in particolare implementa solamente una parte delle funzioni che sono nec-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

essarie. In particolare non viene implementata la sicurezza con SSL e TLS, non viene implementato il QoS di livello 2 e il payload ha una lunghezza limitata di 128 bytes.

Alcune informazioni riguardo all'interfaccia della libreria MQTT per arduino:

- Abbiamo un costruttore PubSubClient per creare il client che prende come parametro il server, la porta, una callback che viene invocata quando arriva il messaggio dal broker
- C'è poi la funzione connect() per connettersi al broker.
- Ci sono anche altre funzioni, in caso vedere la documentazione.

Ci sono vari broker disponibili, ad esempio Mosquitto, Mosca o HiveMQ.

14.3 Alternative a MQTT

14.3.1 Limiti di MQTT

Uno dei limiti di MQTT è anche il suo aspetto più importante è il fatto che utilizza un broker centralizzato perchè abbiamo un single point of failure che può essere un problema quando utilizziamo un sistema IoT distribuito. Inoltre utilizziamo il TCP questa è una grossa limitazione perchè non è la cosa meno costosa che possiamo utilizzare in una rete con dispositivi IoT, questo perchè le connessioni TCP devono essere mantenute e stabilite. Sappiamo ad esempio che TCP richiede più risorse rispetto a quell che invece sono richieste da UDP.

14.3.2 Competitor

HTTP

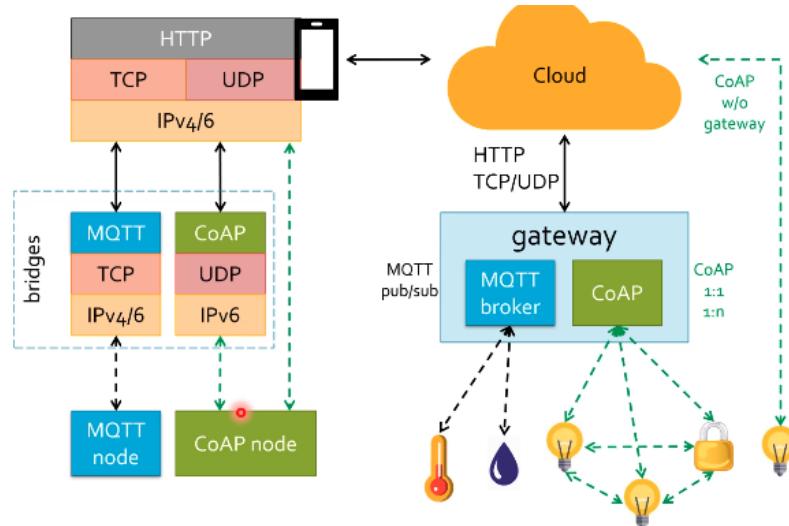
Un competitor è sicuramente HTTP, qua sotto nella immagine ci sono le differenze.

	MQTT	HTTP
pattern	publish/subscribe	client/server
focus of communication	single data (byte array)	single document
size of messages	small and small header	large, details encoded in text form
service levels	3 QoS levels	same service level for all messages
kind of interaction	1 to 0; 1 to 1; 1 to N	1 to 1

Notare che HTTP non differenzia il QoS.

CoAP

Un altro competitor è CoAP:



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Un nodo di CoAP può connettersi direttamente ad un gateway (nel caso di MQTT il gateway è il broker) inoltre abbiamo anche la possibilità di connetterci direttamente ad un qualsiasi altro client CoAP o direttamente al Cloud.

I client in CoAP si possono connettere tutti l'uno con l'altro direttamente mentre in MQTT la comunicazione è sempre da un client al broker e la comunicazione 1 to N è implementata sempre tramite il broker.

Altre informazioni riguardo a CoAP. È un protocollo standard in RFC 7252 ed è ottimizzato per dispositivi low power e per l'IoT.

In generale è un protocollo per la Machine-Machine communication perchè non richiede un intervento umano per essere configurato. È supportato anche da Eclipse.

È basato su client server, a differenza di MQTT, questo non vuol dire che la complessità sta nel server, solo che il server ci risponde. Il client accede le risorse che stanno sul server utilizzando le richieste GET, PUT, POST, DELETE. Funziona in modo simile a HTTP ma funziona tramite UDP/IP (HTTP con TCP) e vengono utilizzati degli header molto piccoli per inviare le informazioni.

Inoltre ci permette di avere dei meccanismi di sicurezza molto forti e utilizza Uniform Resource Identifiers per descrivere i nodi della rete.

CoAP è pensato per nodi con micro controllori a 8 bit che hanno una quantità minima di RAM e di ROM, è anche compatibile con gli standard più recenti come IPv6 e in particolare 6LoWPANs che possiamo pensarli come una sorta di compressione degli headers di IPv6. Attualmente ci sono tentativi di unire 6LoWPAN con Zigbee e quindi probabilmente in futuro 6LoWPAN sostituirà il network level di Zigbee.

Pro e contro:

- Pro: funziona con UDP.
- Pro: fornisce sicurezza.
- Pro: supporta il multicast.

- Pro: permette di avere una comunicazione asincrona come in MQTT.
- Contro: non è uno standard ancora maturo.
- Contro: l'affidabilità dell'invio dei messaggi non è troppo sofisticata.

Entrambe sono adeguate per applicazioni nel campo dell'IoT, MQTT è più adatta per dispositivi che sono veramente tanto low power perchè ci permette di avere il decoupling tra producer e consumer. Allo stesso tempo possiamo dire che la sicurezza di CoAP non la imitiamo in MQTT.

Chapter 15

Synchronization in Wireless Sensor Network

Quando abbiamo parlato del sampling è stato utilizzato il clock, oggi vediamo come il clock viene sincronizzato tra i dispositivi IoT e i computer, questo è fondamentale perché i computer per come sono oggi sono discreti, operano in step e il tempo di questi step viene calcolati con un clock. Il clock di vari dispositivi che si trovano nella stessa rete può essere mantenuto sincronizzato l'uno con l'altro. La sincronizzazione del clock dei dispositivi IoT è rilevante per varie ragioni:

- Perchè ogni dispositivo è controllato dal clock ma il valore è utilizzato per eseguire una operazione di data fusion ovvero per unire vari flussi di dati che vengono inviati dai sensori e che poi vengono messi nello stesso stream. Per avere uno stream che sia significativo dal punto di vista del tempo ho necessità di avere dei dispositivi che devono essere sincronizzati.
- Per una ragione simile agli eventi vengono aggiunti dei timestamp.
- Vogliamo gli stessi riferimenti in termini di tempo per eventi che avvengono su dispositivi differenti.

- Se riusciamo a dare dei tempi precisi ad ogni evento possiamo almeno riuscire a ordinare le varie misurazioni che vengono eseguite.
- Una cosa completamente differente da quella che abbiamo appena visto è la possibilità di localizzare i dispositivi in base alle informazioni del clock, questo solitamente è chiamato sincronizzazione. Se sono in grado di misurare il tempo posso anche localizzarlo.
- La sincronizzazione, in modo simile, è utile anche per tracciare un certo target, questo può essere utile per esempio in applicazioni astronomiche dove dobbiamo controllare una antenna o un emettitore per mantenerlo all'interno di un'area, quindi ho bisogno di un certo tipo di clock synchronization.

La sincronizzazione è importante anche nel campo del networking abbiamo applicazioni simili a quelle che abbiamo visto fino ad ora, la sincronizzazione è capace di schedulare degli eventi per mantenere l'attività dei dispositivi sotto controllo. Inoltre abbiamo dei protocolli di sicurezza che si basano sul tempo, alcune porte sono aperte sono aperte solamente in certi intervalli di tempo e inviano o ricevono solamente in quegli intervalli. Per far sapere agli altri quando una certa porta è aperta è necessario che tutti abbiano un tempo condiviso per sapere esattamente quando una porta sarà aperta o chiusa. Sempre per le reti è importante la sincronizzazione per lo scheduling, nello specifico nella situazione di TDMA (Time Division Multiple Access), la divisione in tempo è basata su un tempo condiviso per tutti i dispositivi.

15.1 Synchronization Algorithms

Possiamo partire considerando quali sono gli algoritmi di sincronizzazione basilari, in generale partiamo considerando le basi e abbiamo alcune assunzioni:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- In generale troviamo che l'overhead di comunicazione non è critico, possiamo considerarlo come un parametro dell'algoritmo oppure no, dipende dal tipo di ipotesi che facciamo.
- La computazione che svolgiamo non è particolarmente critica, possiamo svolgere computazioni più complesse per ottenere una migliore sincronizzazioni ma non è molto importante.
- Assumiamo che i dispositivi sono connessi ad internet se questi devono sincronizzarsi, la configurazione della rete deve essere stabile.
- Se vogliamo aggiornare il clock non devono esserci delle limitazioni specifiche per quel che riguarda la manutenzione periodica

Queste ipotesi sono difficili da estendere in una sensor network perché l'overhead di comunicazione è critico così come la computazione perché abbiamo dei dispositivi che sono limitati e abbiamo limiti legati al consumo di energia. Solitamente assumiamo di lavorare a batteria, lo storage anche è limitato e la capacità di comunicazione è limitata. Anche la rete è instabile, la qualità dell'infrastruttura di comunicazione non è così buona, in ogni momento può diventare non disponibile, inoltre la stabilità della rete non è garantita anche perché i dispositivi possono disconnettersi in ogni momento causando cambiamenti nella topologia della rete. Per quanto riguarda il mantenimento della rete, non sempre è vero che i dispositivi sono sempre accessibili, non è facile che i dispositivi siano stabili per lunghi periodi senza mantenimento.

Quali ipotesi dobbiamo fare quando consideriamo il caso delle sensor network?

Quindi abbiamo capito perché il clock è importante e quali sono le ipotesi che dobbiamo assumere

Cosa è il clock?

È formato da due componenti, uno è la fonte che produce impulsi con una frequenza che è costante. Solitamente le basi sono formate da quartz crystal ma abbiamo anche la possibilità di utilizzare altri risonatori. Abbiamo anche un contatore che viene usato per contare gli impulsi prodotti dall'oscillatore. Clocks sono creati da physical devices. I loro parametri hanno una precisione limitata in questo modo siamo sicuri che se abbiamo due clock con lo stesso label saranno differenti. Non possiamo affidarci al fatto che saranno identici.

Le caratteristiche di questi dispositivi cambiano durante le operazioni e i loro cambiamenti dipendono dalla temperatura, dalla pressione dell'aria, dai campi magnetici, dalla energia fornita, dall'età poichè più il tempo passa più abbiamo influenza sulla frequenza del clock.

Tipici parametri:

- Le tipiche frequenze di clock per un dispositivo IoT sono da 1 a 100MHz. Questo è il range che solitamente viene utilizzato, magari in futuro potremo trovare frequenze più alte.
- La frequenza di tolleranza da 10 a centinaia di parts per million che è preciso ma non troppo.
- I quartz based clock hanno una accuracy e una stabilità migliore.
- I ceramic based clock costano di meno, sono meno accurati e dipendono anche dalla temperatura.
- La capacità del contatore va da 8 a 64 bits.

15.1.1 Clock Model

Il clock consiste in due componenti:

- Una è una fonte di frequenza che produce impulsi ad una frequenza che è regolare e costante. Molto spesso questo tipo di dispositivo è prodotto con il quarzo o con altri tipi di risonatori.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Abbiamo poi un contatore degli impulsi che vengono prodotti dall'oscillatore.

I clock sono dispositivi fisici con dei parametri operativi che hanno una precisione limitata. Le caratteristiche possono cambiare durante il funzionamento e in particolare dipendono da:

- Temperatura dell'ambiente
- Pressione dell'aria
- Campi magnetici
- Alimentazione elettrica
- Età perchè a mano a mano che il tempo passa, questo influenza la frequenza del clock.

I tipici parametri operativi del clock sono:

- La frequenza va da 1 a 100MHz, questo è il range che tipicamente viene utilizzato ora, in futuro si potrebbero trovare dispositivi che hanno un range maggiore.
- La frequency tolerance è da decine a centinaia di parts per million che è preciso ma non tantissimo. In quartz hanno una migliore accuracy e una migliore stabilità, solitamente sono migliori degli altri, quelli in ceramica sono invece meno costosi ma hanno una accuracy minore e una frequenza che dipende molto dalla temperatura.
- Capacità del contatore che va da 8 a 64 bit. Ad esempio nell'Arduino originale abbiamo un clock di 16MHz e i contatori hanno una capacità di 8 o 16 bit.

Date le caratteristiche che conosciamo, questa è la regola che utilizziamo per il calcolo del clock in un certo preciso momento dati i parametri che abbiamo introdotto in precedenza:

$$C_i(t) = t_0 + \epsilon_0 + (1 + \rho_0)(t - t_0) + \frac{1 + \delta}{2}(t - t_0)^2$$

$C_i(t)$ è il valore del clock che chiamiamo i (relativo al dispositivo i , ne abbiamo vari quindi dobbiamo identifierli) al tempo t , qualsiasi tempo t , non sappiamo quale tempo, conosciamo solamente il valore delm tempo. Come intepretiamo la formula? Sappiamo il clock ad un certo reference time nel passato che è t_0 e la differenza tra il tempo e il reference time è ϵ_0 (ϵ_0 è il clock offset al tempo di riferimento). ρ_0 è il clock drift al tempo di riferimento che vuol dire che quando passa un secondo in realtà non passa proprio un secondo ma passa un secondo più o meno un valore ϵ_0 , dobbiamo moltiplicare questa costante che contiene il drift per il tempo che è passato dall'origine. Dobbiamo anche tenere in considerazione un'altra quantità che è l'aging ed è costante. L'aging è il drift che cambia in base ad alcuni parametri e questa è la δ (indica la variazione della frequenza di clock nel corso del tempo, qua la δ viene considerata costante nel tempo t). Questo però è una variazione di secondo ordine quindi consideriamo la differenza tra il tempo di ora e il tempo precedente che però viene elevata al quadrato.

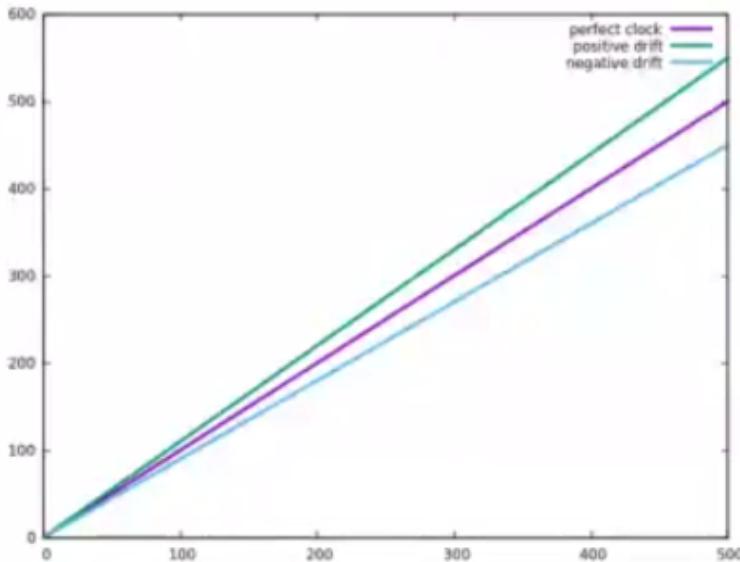
Quello che vediamo nella formula non è niente di nuovo, è un modello approssimato per il nostro clock, è una Taylor Expansion, abbiamo un termine di grado 1 e un termine di grado 2.

Esaminiamo i due parametri che abbiamo introdotto, ρ e δ e vediamo cosa succede. Consideriamo solamente l'impatto dei parametri che è il Clock Drift e solitamente viene espresso in part per million.

Nel caso in cui abbiamo un positive drift osserviamo un clock che si muove più velocemente rispetto ad uno corretto. Nella figura abbiamo una dimostrazione di cosa succede al valore del clock in tre casi:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Il clock è perfetto.
- Il clock ha un positive drift, in questo caso abbiamo che il valore del clock è sopra a quello perfetto.
- Il clock ha un negative drift, in questo caso il valore è sotto a quello perfetto.



Quello che vorremmo ottenere è una sincronizzazione di questo clock, nella figura abbiamo il caso specifico e semplificato in cui non abbiamo un offset iniziale. Per il nostro esempio va bene perchè vogliamo evidenziare il modo in cui le linee divergono.

Vediamo più nel dettaglio il clock drift e vediamo qualcosa di più analitico.

$$\rho(t) = \frac{\partial(C(t))}{\partial(t)} - 1$$

Il clock drift rate è legato alla prima derivata di $C(t)$ ovvero del clock value al tempo $\rho(t)$.

- Quando $\rho = 0$ gli intervalli di tempo sono misurati esattamente perchè ρ influenza solamente la misurazione del time interval. Con un singolo valore non possiamo osservare errori perchè abbiamo solamente un singolo valore, possibilmente ha anche un errore ma è sempre e solo un valore. Se invece abbiamo almeno due valori possiamo osservare il drift perchè l'errore del primo e del secondo sono differenti a causa del drift.
- Nei clock reali la ρ cambia in base a vari fattori, la temperatura è uno dei fattori più rilevanti. Possiamo cercare di regolare questo posizionando i clocks che sonon più critici all'interno di box termo regolati.
- La ρ cambia velocemente durante il tempo a causa della crystal aging e non abbiamo modo di controllarla.

Dato che abbiamo questa quantità $\rho(t)$ che varia nel tempo dobbiamo considerare questo e una cosa possibile è considerare la media di questi valori nel corso del tempo.

$$\bar{\rho} = \frac{C_1 - C_0}{t_1 - t_0} - 1 = \frac{(C_1 - C_0) - (t_1 - t_0)}{t_1 - t_0} = \frac{(C_1 - t_1) - (C_0 - t_0)}{t_1 - t_0} \quad (1)$$

Questo perchè per esempio la variazione di temperatura sarà sempre all'incirca di un certo valore, qualche volta sarà maggiore, qualche volta sarà minore ma in generale avremo una media della temperatura che sarà costante. Quindi è ragionevole cercare di avere un valore medio per il drift. Abbiamo $C_1 - C_0$ che indica la variazione del clock e se dividiamo questa quantità per $t_1 - t_0$ e poi sottraiamo 1 otteniamo la media.

Possiamo fare dei calcoli basilari per ottenere questo che indica l'errore del nostro clock, la distanza del nostro clock dal tempo reale è proporzionale al tempo che è passato dal tempo iniziale. Nel caso in cui vogliamo una prova, l'errore cresce con il tempo, più grande è la ρ e più grande è l'errore che otteniamo.

$$(C_1 - t_1) = (C_0 - t_0) + \bar{\rho}(t_1 - t_0)$$

Possiamo anche dire che questo cresce in modo indefinito perchè quando il tempo va avanti e a mano a mano che il tempo va avanti la differenza aumenta e questo è un problema.

Per introdurre un ragionamento riguardo al drift possiamo introdurre il bounded drift rate model principalmente per risolvere problemi in modo reale con il clock. Assumiamo che il drift non va mai sopra ad un valore massimo che è ρ_{MAX}

$$\forall t, |\rho(t)| \leq \rho_{max}$$

Questa è una assunzione importante perchè assumiamo che siamo in grado di evitare che le condizioni come temperatura o campi magnetici,

ad esempio, facciano sì che la ρ superi il valore ρ_{max} . È una assunzione forte che però è utile per risolvere i problemi, in questo modo riusciamo però a limitare l'errore che il clock può avere.

A questo punto possiamo riscrivere le espressioni precedenti in questo modo dove invece di ρ introduciamo ρ_{max} . In pratica ρ_{max} è una approssimazione al caso pessimo della nostra clock operation.

$$(C_1 - t_1) \in [(C_0 - t_0) - \rho_{max}(t_1 - t_0), (C_0 - t_0) + \rho_{max}(t_1 - t_0)] \quad (3)$$

Questa è una formula che dimostra che il valore dell'errore al tempo t_1 è in un certo intervallo.

vediamo qualche esempio:

lapse	in seconds	max error
1 sec	1	± 0.1 msec
1 hour	3.600	± 360 msecs
1 day	86.400	$\pm 8,64$ secs

Abbiamo una tabella in cui mettiamo in relazione il lapse $t_1 - t_0$, se abbiamo un clock "normale" che possiamo trovare sul mercato con un ρ_{max} di 100ppm allora in 1 secondo avremo 0.1msec, in un giorno avremo 10 secondi di differenza. Quindi dopo 10 giorni abbiamo più di 1 minuto di differenza e questa è la ragione per cui siamo obbligati a lavorare sulla clock synchronization.

La conclusione è che per cercare di mantenere il drift error sotto controllo dobbiamo cercare di sincronizzare periodicamente il nostro clock.

15.1.2 Period between synchronization operations

Introduciamo un'altra quantità che è la maximum drift ovvero il massimo errore ϵ_{max} che vogliamo avere. Questo ϵ_{max} è relago con ρ_{max} che è il drift massimo durante il tempo perchè in ogni unità mi allontano dal tempo reale che ho misurato a t_0 di una quantità che dipende dalla distanza dal tempo originale e dal drift.

$$|(C_1 - t_1) - (C_0 - t_0)| < \rho_{max}(t_1 - t_0) < \epsilon_{max}$$

Otteniamo quindi un upperbound:

$$t_1 - t_0 < \frac{\epsilon_{max}}{\rho_{max}}$$

Abbiamo fatto una assunzione per il drift massimo e possiamo vedere che abbiamo bisogno di risincronizzare. Se partiamo con il tempo perfettamente sincronizzato a t_0 , dopo un tempo di $\frac{\epsilon_{max}}{\rho_{max}}$ abbiamo bisogno di sincronizzare di nuovo se vogliamo tenere il nostro errore sotto a ϵ_{max} .

Se la nostra sincronizzazione non è precisa, avremo bisogno di fare sincronizzazioni più frequentemente. Assumiamo che la sincronizzazione del clock abbia inizialmente un errore ϵ_0 :

$$|(C_0 - t_0) - o_0| < \epsilon_0$$

Allora questo è quello che otteniamo.

$$|(C_0 - t_0) - o_0| < \epsilon_0$$

Questa qua sopra è l'operazione con cui sincronizziamo il nostro clock con un tempo di riferimento, noi non sappiamo mai quale è il tempo di riferimento ma possiamo considerare che abbiamo un clock di riferimento che ha una precisione che è migliore della nostra. Una volta che ho il reference clock, la sincronizzazione che mi permette di avere lo stesso valore del clock nel mio secondo dispositivo, quanto è precisa? Quanto è preciso il clock che ottengo dopo la nostra clock synchronization?

Il valore dell'offset O_0 è quello che introduciamo durante questa operazione, in pratica quello che facciamo è:

- Chiediamo all'altro dispositivo che ore sono.
- Poi ce lo dice, ma prima che io fisso il mio clock a quel tempo che mi è stato indicato il tempo passa e questo tempo è l'errore O_0 .

Questa cosa la dobbiamo tenere in considerazione ed è quello che facciamo con le formule qua sopra, la nostra operazione iniziale ha un errore che è ϵ_0 .

Abbiamo bisogno quindi di essere meno ottimisti riguardo all'intervallo di tempo tra due sincronizzazioni perchè l'errore massimo che vogliamo necessita di considerare che non deve partire da un errore 0 ma deve partire da ϵ_0 . Quindi quello che abbiamo nella formula è:

$$t_1 - t_0 < \frac{\epsilon_{max} - \epsilon_0}{\rho_{max}}$$

Quindi il parametro ϵ_0 è rilevante e dobbiamo essere in grado di stimarlo altrimenti non riusciamo a fare il calcolo.

15.1.3 Esempio

Consideriamo il seguente problema:

A clock has a maximum drift rate of 100ppm ($=10^{-4}$). We want to keep a maximum drift of 0.1s. What is the maximum resynchronization period assuming that the error in the clock synchronization operation is negligible ($\epsilon_0=0$)?

Utilizzando l'equazione indicata in precedenza vogliamo calcolare $t_1 - t_0$ e otteniamo questo:

$$t_1 - t_0 < \frac{\epsilon_{max}}{\rho_{max}} = \frac{0.1s}{10^{-4}} = 1000s$$

Questo rappresenta il periodo massimo.

15.1.4 Clock Aging

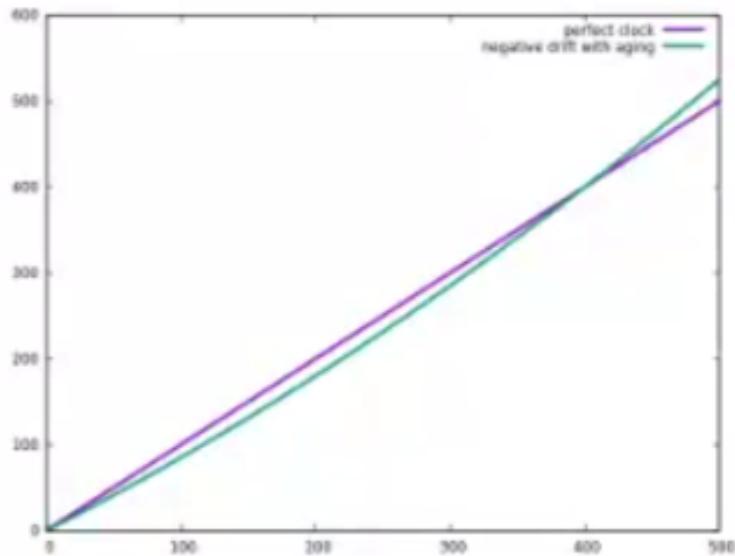
Al momento abbiamo limitato la nostra analisi al clock drift, ora vediamo la clock aging che è la second order variation del clock.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Il clock aging corrisponde ad un cambiamento graduale nella frequenza, l'aging vuol dire che il clock va a diminuire, questo solitamente è espresso in ppm/year e sfortunatamente questo cambia nel corso del tempo.

Normalmente questo è relativo al fenomeno dell'aging che è associato con il crystal o con il capacitor che progressivamente degradano.

In questa figura vediamo cosa succede ad un clock che inizialmente è lento con un $\rho_0 = -0.2\text{ppm}$ e un aging positivo. La linea più scusa è quella del perfect clock, l'altra linea inizialmente è più lenta rispetto al clock perfetto ma a mano a mano che andiamo avanti diventa più veloce.



Il clock aging è rappresentato dalla seconda derivata di $C(t)$.

$$\delta(t) = \frac{\partial^2(C(t))}{\partial t^2} - 1$$

Rimane costante in un lungo periodo di tempo ma tende a descrescere nel corso del tempo.

15.1.5 Esercizi

Esercizio 1

Vediamo un primo esercizio

The clock of a MCU ticks every approx. 1ms, with an unknown drift.
We observe the following:

- ① At 9:30:00AM UTC has a value of 60000.
- ② When the clock has value 43260000, the UTC time is 21:30:01.
- ③ A: Compute the value of the drift in ppm, assuming that the reference clock reading is exact.

In questo caso vogliamo calcolare il valore del drift, la soluzione è simile al precedente esercizio:

- we use equation 1
- the difference between the two clock values is 43200000.
- the difference between the two UTC times is 12 hours+1second, which means $(12 * 60 * 60s + 1s) * 1000 = 43201000ms$
- therefore the clock delayed 1000ms during 43201000ms
- the drift is $-1000ms/43201000ms = -0,000023148 \approx -23ppm$

Esercizio 2

- A: Compute the drift range, in case the UTC reading has an error of $\pm 500ms$
- 500ms correspond to 50% of the measured delay (1000ms): therefore the drift range lays in the interval $23ppm \pm 50\%$, i.e. $[12ppm, 35ppm]$

↑

15.1.6 Clock Accuracy

La clock accuracy viene definita rispetto ad un tempo di riferimento, questo vuol dire che il tempo di riferimento non è un tempo reale. Abbiamo una clock accuracy rispetto a qualcosa che sappiamo essere un tempo di riferimento. Abbiamo una formula che è simile a quella che abbiamo visto per il caso in cui il tempo è reale e non è un tempo di riferimento:

$$\epsilon(t) = |C(t) - T_{ref}(t)|$$

Dalla formula sopra possiamo poi capire l'accuracy:

$$\epsilon(t) = \max_{i \in N} |C_i(t) - T_{ref}(t)|$$

Il clock drift degrada la clock accuracy come nel caso del real time, se il drift è conosciuto dato che è costante possiamo provare a stimarlo. Se siamo in grado di fare una approssimazione del drift ad un certo punto e calcoliamo nuovamente la stessa quantità dopo un certo punto possiamo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

provare a capire quale è il valore per calcolare una approssimazione del vero valore del drift e possiamo capire come cambia il drift.

Possiamo provare ad approssimare il drift e possiamo compensare questo valore, una volta che ho una stima possiamo compensare il clock value utilizzando questa approssimazione. Se leggo il clock uno o due volte durante il giorno e vedo che alla prima lettura ho un secondo di distanza rispetto al clock reale e alla seconda misurazioni abbiamo 2 secondi di differenza allora possiamo stimare che il drift durante il tempo è 1 secondo tra le due misurazioni. Una volta che ho capito quale è il drift possiamo utilizzarlo all'interno della nostra formula e compensare la sua presenza.

Se riusciamo a fare questa approssimazione del drift ottenuta come calcolo e approssimazione del drift allora otteniamo questa che è la valutazione dell'errore ad un certo punto:

$$\epsilon(t) \leq \epsilon_0 + (t - t_0) * \rho_{max}$$

Questo mi permette di dire che ad un certo punto t l'errore del mio clock verrà limitato dalla quantità sopra. Questo è importante all'interno di una applicazione perchè vogliamo sapere quale è la clock precision su cui possiamo fare affidamento e se non è abbastanza allora devo sincronizzare di nuovo il clock. Questa formula sopra dipende quindi da ϵ_0 che è l'errore quando leggiamo il clock al tempo t_0 più il drift che abbiamo accumulato durante il tempo. Questo quindi è un upperbound dell'errore.

15.2 Clock Distance

La clock distance è un'altra caratterizzazione (completamente differente) di un sistema di clock. La clock distance non dipende da un tempo di

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

riferimento, quindi qua non consideriamo il tempo reale ma facciamo solamente vedere in che modo sincronizziamo il clock all'interno del sistema e facciamo vedere quanto sono distanti due clock all'interno di un sistema. La distanza al tempo t di questi due clock di A e di B la definiamo come:

$$\epsilon(t) = |C_A(t) - C_B(t)|$$

Se invece abbiamo un set di N nodi la distanza al tempo t la possiamo calcolare come:

$$\epsilon(t) = \max_{i,j \in N} |C_i(t) - C_j(t)|$$

In questa formula noi prendiamo la massima distanza tra qualsiasi possibile coppia di clock, specifichiamo anche come parametro una t perché vogliamo fare il calcolo ad un certo punto specifico (ad ogni punto). In alcuni casi definiamo questa distanza in modo statistico come $X - th$ percentile della distanza (noi non la vediamo).

15.3 Problema della sincronizzazione nelle WSN

Passiamo ora alle Wireless Sensor Network, abbiamo il nostro sistema che ha una serie di clock. Vogliamo considerare questo come nostro target, ad ogni tempo vogliamo la distanza dal clock sia minore di ϵ .

$$\forall t, \forall_{i,j \in [1..N]}, |C_i(t) - C_j(t)| < \epsilon$$

Questa è una conseguenza dell'implementazione del nostro sistema, la ϵ è una sorta di trade off tra vari parametri che usiamo nel design. Che tipo di trade off vogliamo? Minore è la ϵ e migliore è la precisione della sincronizzazione che otteniamo, però abbiamo anche dei limiti fisici come ad esempio la distanza, la variazione di temperatura perchè possiamo avere dei dispositivi che sono all'interno di box termostatici, il tempo di comunicazione che ha impatto sulla precisione della sincronizzazione. Inoltre dobbiamo anche considerare la quantità di risorse disponibili per la sincronizzazione, in particolare per quanto riguarda l'energia.

15.3.1 Metodi per la sincronizzazione

Abbiamo vari metodi differenti per la sincronizzazione.

- Interna vs Esterna: voglio sincronizzarmi con qualcosa che è esterno al sistema o voglio mantenere la distanza il più vicino possibile internamente?
- Poi abbiamo differenze per quanto riguarda il communication models, abbiamo una comunicazione che può essere simmetrica o asimmetrica nel senso che possiamo ricevere senza inviare oppure fare entrambi, unicast o multicast, implicita o esplicita nel senso che abbiamo una operazione implicita o no per l'operazione di clock oppure sono in grado di mandare questi dati con piggyback durante altre interazioni?
- Clock synchronization vs timescale Synchronization? Posso tenere il clock come è ora ma trasformo il clock value in modo da avere un tempo adeguato

- L'operazione di sincronizzazione deve essere un qualcosa di continuo o deve essere on-demand?
- Oppure possiamo avere una sincronizzazione come one shot operation nel senso che decido di sincronizzare, quindi invio il mio messaggio e poi sono sincronizzato oppure come multi shot nel senso che invio un messaggio, ricevo una risposta e vado avanti, alla fine abbiamo il clock sincronizzato.
- One Hop vs multi hop synchronization. Arrivo al reference clock con una operazione con un singolo hop oppure faccio un multi hop per arrivare?

Andiamo a vederli più nel dettaglio.

External vs Internal Synchronization

External vuol dire che la fonte è esterna alla nostra rete, vengono utilizzati dei protocolli che permettono di sincronizzare con una reference time source come ad esempio DCF 77, GPS o NTP. Qui in questo caso abbiamo bisogno di un reference time esterno. Per l'external possiamo utilizzare un clock signal che solitamente è affidabile e che arriva da un atomic clock, questo arriva con una trasmissione radio a onda lunga. La frequenza di trasmissione in Germania per esempio è 77.5 kHz con un range di 2000km per l'mitter DCF77, in Francia per esempio è diverso. L'accuracy tipicamente è di 2ms. Questo tipo di sincronizzazione viene utilizzato anche per gli orologi da parete. Abbiamo una buona accuracy, un costo basso e il consumo di energia è abbastanza buono.

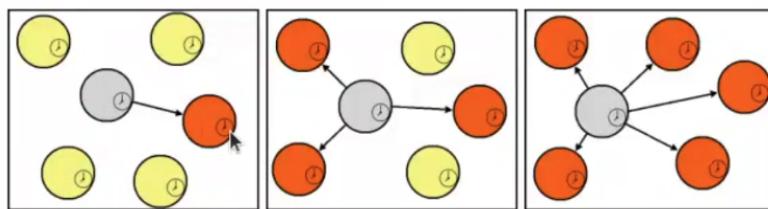
Nel caso della internal synchronization invece abbiamo un protocollo che funziona all'interno del nostro sistema, ad esempio potrebbe essere limitato all'interno della nostra WSN, in questo caso non abbiamo bisogno di un reference time esterno. In questo caso vengono utilizzati satelliti GPS. In questo caso abbiamo una accuracy e un drift che sono

eccellenti però abbiamo la necessità di Line of Sight tra il satellite e il ricevente, per ricevere tra l'altro abbiamo bisogno di un hardware specializzato che non è troppo economico e inoltre abbiamo anche bisogno di batterie adeguate per mantenere questa procedura. L'external synchronization utilizza anche il protocollo NTP che viene largamente utilizzato in internet, ci sono dei server a livello rete che inviano dei timestamp in modo che gli altri server o gli utenti finali possano ottenere la sincronizzazione. Abbiamo anche dei limiti per NTP perché richiede uno scambio periodico di messaggi e questo può essere un problema nelle WSN perché la connessione ad internet potrebbe non sempre essere disponibile e potrebbe essere mediata da gateway o da altri nodi e quindi non sarebbe la migliore soluzione possibile.

Abbiamo una evoluzione dell'NTP con il PTP che è migliore rispetto alle altre soluzioni, abbiamo una accuracy migliore ma è orientata specialmente alle wire network e non alle wireless. Di base è un three way protocol.

Unicast vs Multicast

Vediamo i tre casi che possiamo avere con questi protocolli:



- Unicast: un nodo per sincronizzarsi legge il clock dell'altro nodo.
- Multicast: da un nodo sincronizziamo vari altri nodi.
- Broadcast: un nodo sincronizza tutti gli altri.

Explicit vs Implicit

L'explicit synchronization richiede uno scambio di messaggi solamente per la sincronizzazione, abbiamo un overhead maggiore perchè non sfruttiamo il traffico già esistente per fare la sincronizzazione ma in generale otteniamo una migliore qualità della sincronizzazione.

Con la sincronizzazione implicita cerchiamo di sfruttare le altre comunicazioni per eseguire il piggyback delle informazioni di sincronizzazione. Questo comporta un overhead più basso ma allo stesso tempo limita anche la qualità della sincronizzazione.

Clock Synchronization vs Timescale

Abbiamo una System Clock Synchronization che è simile a quella che abbiamo con NTP, in questo caso aggiorniamo il valore del clock a livello hardware (il livello più basso).

Con il Timescale transformation manteniamo il valore dell'hardware come è già ma sappiamo che non è corretto quindi applichiamo una al livello di software clock per ottenere un clock che è più simile al reference clock. Ogni applicazione potrebbe avere il suo clock senza la necessità di condividere necessariamente un clock a livello hardware.

Time instant vs time interval

Nel primo caso noi abbiamo solamente la necessità di avere la conoscenza del tempo in un certo istante, questo valore possiamo utilizzarlo immediatamente. Per esempio questo viene utilizzato per aggiungere il timestamp ad un evento.

In alternativa se abbiamo un time interval possiamo avere una serie di valori del clock in un certo intervallo, il periodo può essere fissato oppure variabile. La clock synchronization potrebbe essere controllata da un master o da uno slave. E possiamo anche avere clock value da differ-

enti master e questo aumenta l'accuracy del nostro protocollo. Questo permette anche di stimare la precisione del clock locale.

Continuous vs On Demand

Nel caso continuo vuol dire che possiamo schedulare la sincronizzazione in modo circolare continuamente. Questo potrebbe andare ad aumentare i costi perchè potremmo avere un aumento dei costi di comunicazione ma anche in termini di consumo di energia.

L'alternativa è implementare l'operazione solamente su richiesta, quando mi serve questa informazione la richiedo, questo è particolarmente utile nelle WSN.

One Shot vs Multi Shot

Nel caso one shot potremmo avere l'operazione di sincronizzazione che termina una volta che abbiamo scambiato una singola informazione. In alternativa possiamo scambiarci varie informazioni, questo cerca di ridurre l'effetto del ritardo nella comunicazione. Probabilmente questa soluzione riesce ad essere più preciso.

15.3.2 Direct vs Multi-Hop synchronization

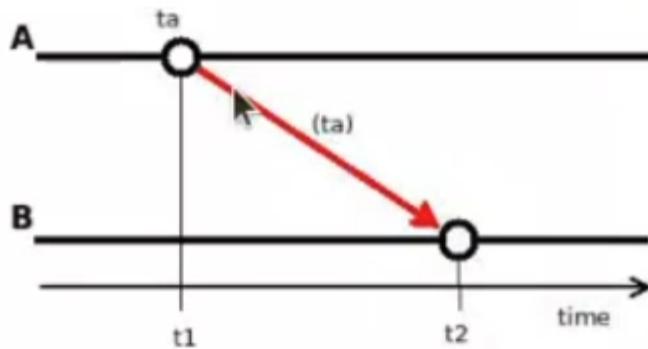
Nel caso della sincronizzazione diretta abbiamo che riusciamo a raggiungere il reference clock in un singolo hop.

Nel caso del multi-hop devo invece andare a chiedere informazioni a chi conosce già il clock del reference clock ma in questo caso abbiamo che ad ogni clock si degrada la clock precision. In generale se abbiamo la possibilità dobbiamo limitare ed evitare il più possibile la multi hop synchronization.

15.4 Synchronization Protocol

Vediamo in che modo funziona la sincronizzazione.

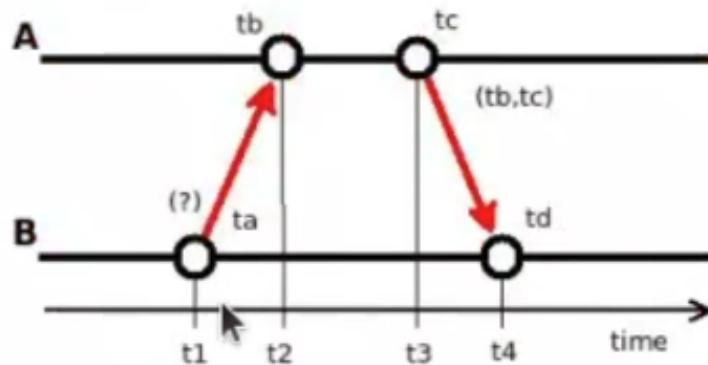
Abbiamo la one way synchronizationm in cui abbiamo un master che invia l'informazione del clock al dispositivo slave:



- A possiede il reference clock (è il master).
- Al tempo t_1 A invia a B (che è lo slave) il suo local clock t_a .
- Al tempo t_2 B riceve il messaggio.
- Assumiamo che il ritardo di questo messaggio sia limitato da $t_2 - t_1 < \epsilon$.
- B modifica il suo clock in $t_a + (\epsilon/2)$
- La massima differenza tra C_B e C_A al tempo t_2 è $\epsilon/2$

Questo è buono nel caso in cui abbiamo una connessione asimmetrica in cui abbiamo un nodo che riceve e uno che riceve e non viceversa

Se abbiamo una two way synchronization invece il funzionamento è differente.



Se abbiamo un collegamento simmetrico vuol dire che abbiamo uno che può sia inviare sia ricevere. In questo caso abbiamo lo slave B che invia la richiesta al master A e poi la A restituisce una risposta a B. In questo caso abbiamo un modo per calcolare una stima del tempo di trasferimento, abbiamo anche una stima dell' ϵ_0 che è l'errore nella operazione di sincronizzazione. Il tempo che spendiamo per la comunicazione è il seguente:

$$(t_d - t_a) - (t_c - t_b)$$

Se assumiamo che un tempo uguale è preso dall'invio e dalla ricezione allora possiamo calcolare questa stima per l'errore potenziale.

$$C_B(t_4) = t_c + \frac{(t_d - t_a) - (t_c - t_b)}{2}$$

15.4.1 Symmetric vs asymmetric synchronization

La symmetric synchronization introduce una assunzione che serve per limitare l'errore, la sincronizzazione asimmetrica ha un upper bound per questo errore perchè il roundtrip time è misurabile. L'upperbound che otteniamo è il seguente:

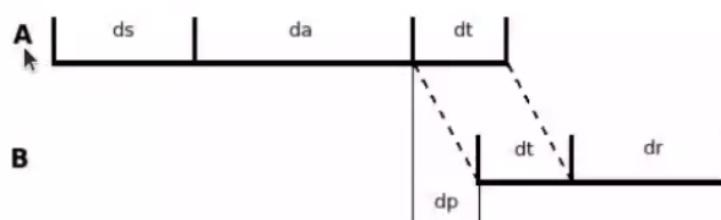
$$\epsilon < \frac{(t_d - t_a) - (t_c - t_b)}{2}$$

15.4.2 Problemi

Abbiamo una serie di problemi con tutti questi tipi di operazioni:

- Problemi con le richieste: abbiamo una incertezza per quanto riguarda il tempo che è necessario per il master B per estrarre il suo local time, spendiamo un tempo per costruire il messaggio, poi per propagarlo e poi anche per riceverlo e processarlo nel nodo A.
- Problemi con le clock operation (reading e writing)
- Problemi con le risposte

Questi problemi li possiamo vedere da qua:



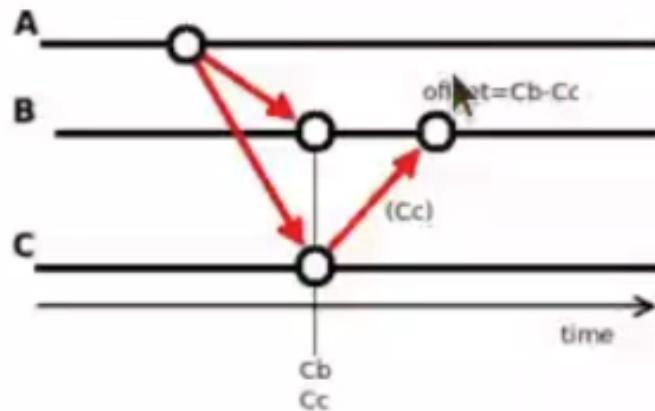
Qua abbiamo suddiviso l'operazione di invio in un numero di sotto operazioni, abbiamo:

- Send Time.
- Access Time.
- Il messaggio poi deve essere messo in coda per essere inviato.

Qui abbiamo un ritardo poi e poi il messaggio viene ricevuto da B. Questi tempi sono in alcuni casi variabili, in altri casi sono fissati. Quando devo pensare ad un protocollo di questo genere devo fare delle ipotesi riguardo alla durata di tutti questi tempi perchè qua io sto scrivendo una applicazione realmente esistente e funzionante e non posso avere delle quantità di tempo imprevedibili, devo avere comunque degli upper bound.

15.5 Reference Broadcast synchronization

È un caso molto rilevante che copre la situazione in cui vogliamo avere una sincronizzazione interna di vari dispositivi periferici. Tutto quello di cui ho bisogno è una unità che deve lavorare come beacon e che di tanto in tanto manda dei messaggi che vengono ricevuti dalle altre unità.



Potrebbero esserci dei ritardi tra quando il tempo viene inviato e tra quando viene ricevuto ma possiamo assumere che in questo protocollo tutte le unità ricevono il beacon allo stesso tempo. Questo ci fornisce le basi per la sincronizzazione perchè a questo tempo le peripheral units si scambiano il loro tempo e in questo modo tutte le unità sono in grado di calcolare la loro distanza reciproca e alla fine convergere per avere un clock sincronizzato.

Le operazioni che svolgiamo nella immagine precedente sono le seguenti:

- B e C ricevono un beacon da A.
- B e C registrano il valore del clock quando il beacon viene ricevuto.
- C invia a B il suo local time quando riceve il beacon.
- B calcola la sua distanza da C, sarà limitata superiormente ma non c'è modo di dire che questa distanza sia davvero vicina al tempo reale.

Chapter 16

Indoor Localization

16.1 Introduzione

Le motivazioni che ci portano ad studiare l'Indoor Localization sono varie:

- Nel corso degli ultimi anni abbiamo visto la nascita di tanti sistemi wireless che sono utilizzati in moltissimi tipi di applicazioni.
- La presenza di questi dispositivi wireless permette la creazione di servizi che sono basati sulla localizzazione all'interno di un certo ambiente.
- C'è stata nel corso del tempo anche una richiesta via via crescente di informazioni riguardanti la posizione di oggetti e persone sia in un ambiente esterno sia interno. Il problema della localizzazione all'esterno è stata bene o male risolto utilizzando il GPS o Galileo mentre invece quello della localizzazione all'interno rimane un problema ancora aperto da risolvere.

Tramite l'indoor localization è possibile studiare dove si trova l'utente all'interno di un certo ambiente, garantirgli la sicurezza e fornire dei

contenuti personalizzati in base alla posizione in cui si trova. Possiamo anche controllare se nelle vicinanze ci sono dei punti di interesse ma anche tracciare gli oggetti all'interno di una fabbrica ad esempio.

Partiamo col definire quali sono le informazioni riguardo alla localizzazione e alla posizione per noi, possiamo definire Physical e symbolic location:

- Physical: parliamo di localizzazione fisica quando utilizziamo delle coordinate 2D o 3D per indicare la posizione di una persona o di un oggetto, ad esempio possiamo usare latitudine e longitudine.
- Symbolic: parliamo di symbolic location quando parliamo tramite il linguaggio naturale, ad esempio possiamo dire che ci troviamo vicino a qualcosa.
- Absolute: parliamo di absolute location quando utilizziamo un sistema di riferimento condiviso.
- Relative: parliamo di relative location quando ogni oggetto ha un certo riferimento, ad esempio la distanza da un access point o la distanza dalla destinazione.

È sempre possibile convertire l'absolute location nella relative location ma non è possibile fare il contrario normalmente. Possiamo fare la conversione da relative ad Absolute location quando la absolute position del punto di riferimento per la relative location è conosciuta oppure se abbiamo dei relative reading multipli per la relative location.

16.2 Outdoor Localization

Prima di parlare dell'indoor localization parliamo della Outdoor localization. Abbiamo in questo caso uno standard chiamato GNSS (Global Navigation Satellite Systems) introdotto dagli USA negli anni 70. Questo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

tipo di sistema include un sistema di satelliti che coprono le varie parti del mondo e che spesso si sovrappongono anche:

- GPS: questo copre gli USA e in parte l'Europa
- GLONASS: per la Russia ma copre anche l'Europa
- Galileo: copre tutta l'Europa
- BeiDou: abbiamo anche un sistema usato in Cina che in alcuni casi si sovrappone con quello europeo

Ora come ora gli smartphone possono utilizzare tutti questi sistemi.

Vediamo l'architettura del GNSS, è composto da tre componenti principali che vengono chiamati segmenti:

- Space Segment: dove sono i satelliti
- Control Segment: è il segmento che si occupa di gestire il segnale trasmesso e ricevuto dal satellite GNSS.
- User Segment: è rappresentato dai dispositivi che eseguono l'applicazione GNSS. Quindi gli smartphone ad esempio.

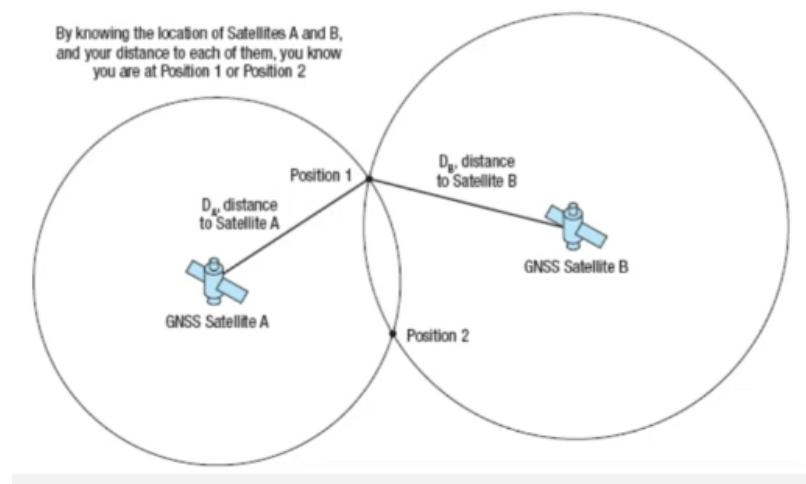
Per quanto riguarda il meccanismo operativo abbiamo 4 differenti fasi:

- Propagation: dal satellite al dispositivo
- Reception: effettuata dal dispositivo
- Computation: viene effettuata direttamente sul dispositivo
- Application: utilizza questa informazione

16.2.1 Computation Phase

Siamo particolarmente interessati alla computation phase. Come funziona? Abbiamo delle range measurement da almeno 4 satelliti che sono utilizzati per determinare la posizione dei dispositivi stessi. Per ogni satellite che viene tracciato il ricevente determina quanto tempo ci ha messo il segnale del satellite per arrivare a destinazione e questo determina quindi la distanza dal satellite.

Come funziona? Il dispositivo calcola il propagation time perchè sia quando il segnale ha lasciato il satellite sia quando il segnale è arrivato nel dispositivo. Quindi è possibile calcolare il Propagation Time come differenza di questi due tempi. La distanza dal satellite viene calcolata moltiplicando il propagation time con la velocità della luce. Se abbiamo un mondo bidimensionale abbiamo questa situazione, questo è il modo in cui la computazione funziona, se il dispositivo si aggancia a due satelliti possiamo trovare due possibili in cui posso trovarmi.



A causa del clock error del receiver (abbiamo sul dispositivo un clock che non potrà essere accurato come lo è ad esempio quello del satellite) non potremo avere una localizzazione precisa. In particolare quello che

succede è che quando moltiplichiamo la velocità della luce con il tempo che passa non potremo avere una misurazione precisa ma avremo un errore di circa 1500 metri che non è accettabile. Quindi vogliamo un terzo satellite per migliorare questa stima della posizione, il receiver può in questo modo ottenere una localizzazione che converge in un singolo punto e non due come in precedenza, in questo modo abbiamo una localizzazione più precisa con 3 satelliti in un mondo bidimensionale. Se siamo in un modo 3D invece mi servono almeno 4 satelliti per calcolare una posizione, solitamente nel mondo reale abbiamo più di 4 satelliti per ottenere la localizzazione.

16.3 Indoor Localization

Date le base del caso Outdoor possiamo vedere come funziona l'indoor localization. Questo è un problema chiave per molte applicazioni, come abbiamo visto nel caso Outdoor abbiamo uno standard che ormai è definito e viene utilizzato, in un'area in cui non arriva il GPS dove non possiamo ricevere informazioni dai satelliti c'è la necessità di soluzioni.

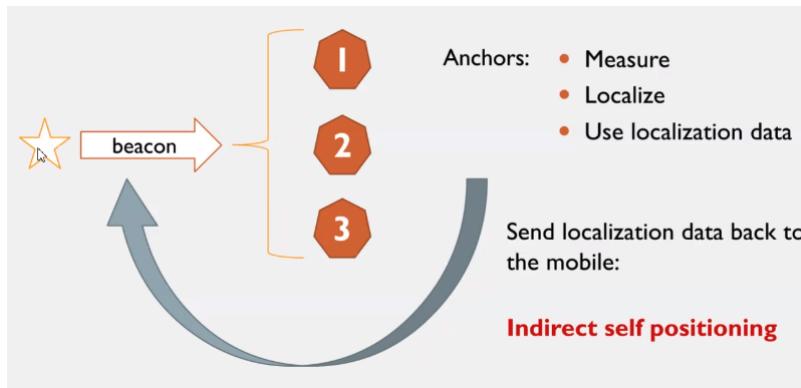
16.3.1 Topologies

Prima di vedere le tecnologie disponibili per l'indoor localization vediamo il glossario. Ad esempio possiamo cominciare a vedere i tipi di topologia per i meccanismi di localizzazione, abbiamo due topologie principali e due topologie derivate.

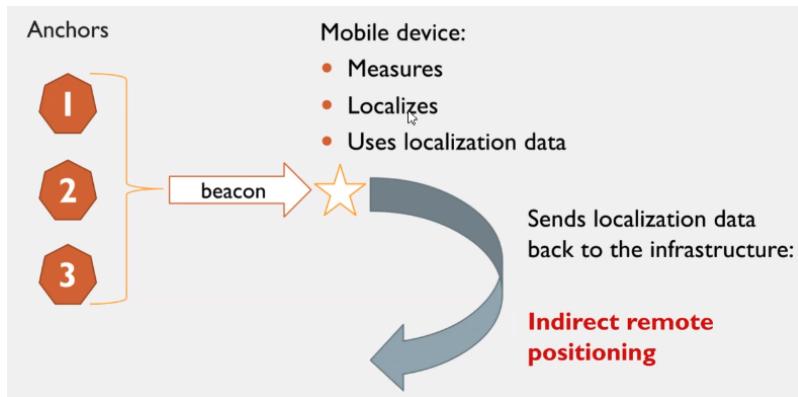
- Remote Positioning: in questo caso l'unità che deve essere localizzata è mobile e funziona come trasmettitore, l'unità di misurazione viene fissata, è un anchor e abbiamo un location manager che viene fissato (potrebbe essere lo stesso anchor) ed esegue l'algoritmo di localizzazione. Con l'immagine qua sotto possiamo avere una visione più chiara del Remote Positioning, abbiamo in questo caso tre

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

anchors che misurano il ranging dal dispositivo mobile, localizzano il nodo e utilizzano i dati di localizzazione per capire la posizione del dispositivo.

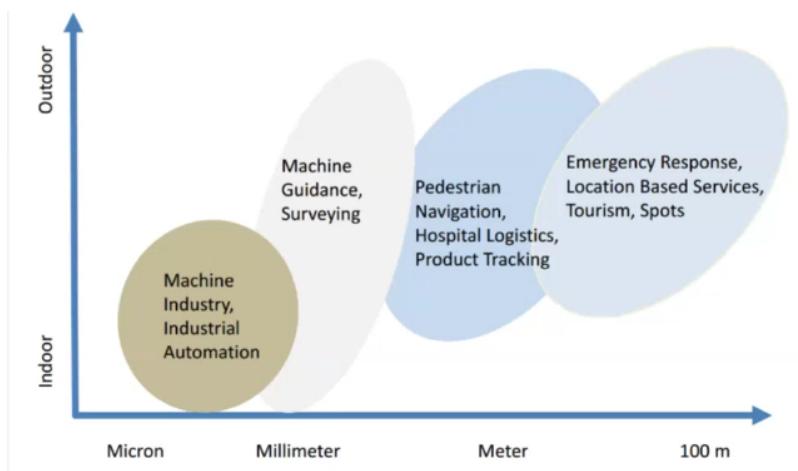


- Self Positioning: anche in questo caso l’unità che deve essere localizzata è mobile, qua in questo caso però è la stessa unità che deve essere localizzata che riceve il segnale da un anchor fissato e poi è la stessa unità che svolge l’algoritmo di localizzazione. L’anchor in questo caso funziona solamente come trasmettitore. Da questa immagine vediamo come funziona il self positioning, qua in questo caso abbiamo i tre anchors che trasmettono un beacon e i dispositivi mobili misurano questo segnale, lo localizzano e poi se il dispositivo manda indietro la posizione all’anchor allora si parla di Indirect Remote positioning.



- Indirect Remote positioning: questa è un self positioning ma in questo caso il dispositivo che dobbiamo localizzare invia la sua posizione ad un location manager remoto.
- Indirect Self-Positioning: simile alla remote positioning ma in questo caso è il location manager che manda la posizione al dispositivo mobile.

Basandoci sull'accuracy che otteniamo e sul tipo di localization che stiamo effettuando possiamo avere vari scenari possibili.



A seconda dello scenario che stiamo considerando abbiamo differenti richieste in termini di accuracy e di copertura. Ad esempio se siamo in una fabbrica vogliamo una accuracy altissima, in uno scenario più comune, come ad esempio quello umano siamo interessati a sistemi che sono in grado di misurare la posizione nel range di metri e possono coprire zone più grandi.

Come viene gestita l'indoor localization ora? Non c'è una soluzione unica che viene utilizzata, queste soluzioni sono varie in termini di segnali che vengono utilizzati, in termini di tecnologia e in termini di metriche così come di processing methods utilizzati.

Ognuna di queste soluzioni che consideriamo ha dei vantaggi e degli svantaggi, in molti casi abbiamo un trade off tra varie metriche (accuracy, costo, scalability)

16.3.2 Tipi di segnali e tecnologie

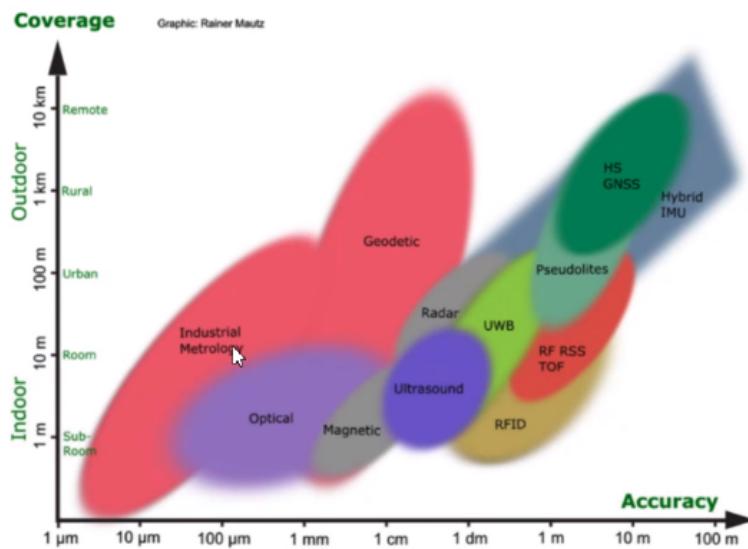
Partiamo con i tipi di segnali e con le tecnologie che possono essere utilizzate per realizzare un sistema di Indoor localization, abbiamo varie tecnologie che hanno differenti performances in termini di difficoltà di implementazione. Tutte queste tecnologie hanno performances differenti e costi differenti in termini di costi per l'implementazione. Vediamo i tipi di segnali che possono essere utilizzati:

- Segnali infrarossi: con questo tipo di segnali abbiamo che i dispositivi devono vedersi tra loro per comunicare.
- Per l'ultrasound abbiamo una buona accuracy ma abbiamo uno short range e inoltre abbiamo anche lo stesso problema degli infrarossi con i dispositivi che devono vedersi.
- Per l'UWB abbiamo bisogno di dispositivi specializzati e spesso il costo è anche abbastanza alto. Possiamo avere una

accuracy di 10-20 centimetri. Solitamente non abbiamo questo tipo di radio sui nostri smartphone.

- Con il Wifi abbiamo una accuracy minore ma comunque il costo è minore perché possiamo sfruttare il wifi che magari è già esistente e solitamente i dispositivi personali hanno il Wi-Fi embeddato all'interno.

In base al tipo di segnale che utilizziamo possiamo avere una differente accuracy e una differente copertura. Tutti questi segnali sono solitamente specializzati per uno scenario apposito.



Poi abbiamo differenti metriche per i segnali, l'hardware che viene implementato negli indoor localization Systems si basa su un approccio per derivare le distanze tra le entità utilizzando delle metriche che sono calcolate sullo stesso segnale. Abbiamo tre approcci principali:

- Un primo approccio si basa sul tempo: tempo di arrivo, RTT, Differenza di arrivo del tempo.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Un secondo approccio si basa sull'angolo di arrivo.
- Possiamo avere un approccio che si basa sulla forza del segnale che viene ricevuto.

Time of Arrival

Vediamo il primo approccio che si basa sul tempo, l'approccio più semplice si chiama Time Of Arrival. In questo caso per la distanza tra la measuring unit e il mobile target abbiamo una assunzione, la distanza è direttamente proporzionale alla propagation time.

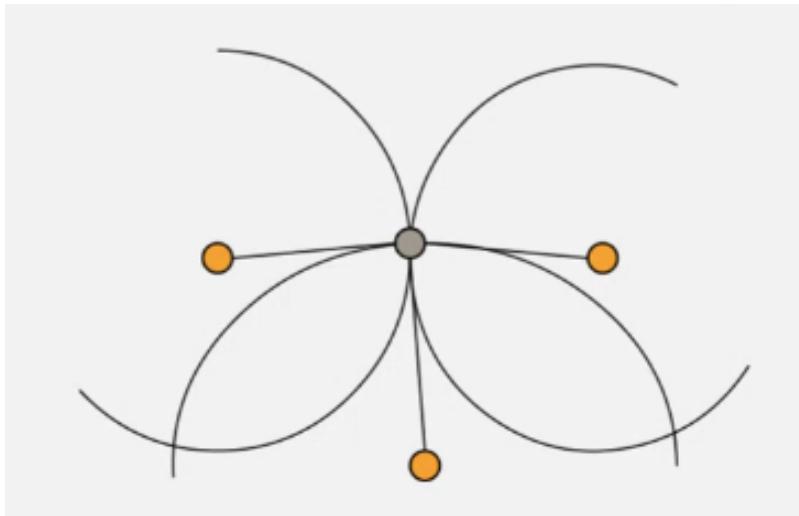
Come funziona?

- Abbiamo il dispositivo mobile target che emette un segnale radio al tempo t
- La measuring unit riceve un segnale radio al tempo t'
- La measuring unit stima la distanza come $(t' - t) * p$

Ci sono due possibili problemi con questo sistema:

- Deve esserci una sincronizzazione stretta tra chi trasmette e chi riceve
- Il segnale deve codificare il transmission time t

Per stimare questa distanza e per stimare la posizione del dispositivo mobile target in un sistema 2D sono richiesti almeno 3 differenti anchors, le posizioni possono essere calcolate con differenti metodi, ad esempio l'intersezione dei cerchi che possiamo creare e che sono centrati nell'anchor.



in 3D invece mi servono almeno 4 anchor differenti, possiamo pensare la posizione come il calcolo dell'intersezione di questi cerchi. Un'altra possibile soluzione per il calcolo della posizione consiste nella soluzione di un non linear optimization problem, come ad esempio least squares.

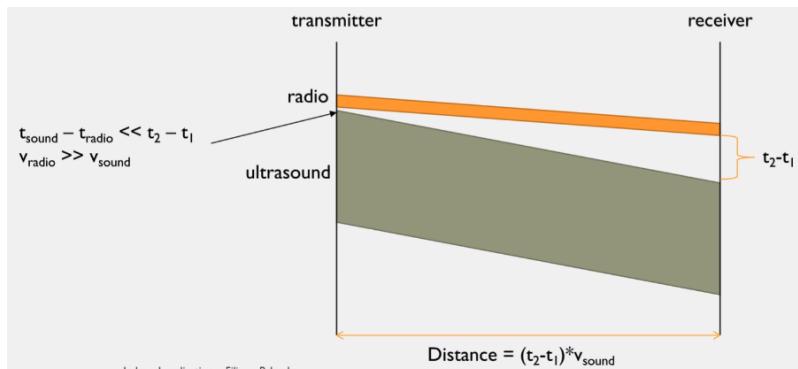
$$\min \sum_{i=1}^n \left| c \cdot (t_i - t) - \sqrt{(x_i - x)^2 + (y_i - y)^2} \right|$$

Qua la variabile sconosciuta sono il tempo in cui è stato emesso il segnale e le coordinate del dispositivo target. Conosciamo invece le coordinate dell'anchor e il tempo di arrivo del segnale nel target. Se minimizziamo questa distanza otteniamo le coordinate che ci interessano e anche il tempo che stavamo cercando.

In alcune applicazioni il Time of Arrival viene calcolato in modo più intelligente utilizzando segnali di differente natura, ad esempio segnali radio e segnali acustici, questo perchè il segnale radio è più vicino del segnale acustico, quindi il segnale radio possiamo utilizzarlo per la sincronizzazione delle measuring units. Poi possiamo dire che la differenza

di tempo tra l'arrivo di due segnali è praticamente proporzionale alla distanza perché il segnale radio è di un ordine di magnitudine più veloce rispetto al segnale acustico.

Questo è il caso che abbiamo menzionato ora:



Possiamo fare qualche computazione per confermare che la distanza è corretta e che è la distanza tra il momento in cui arriva il segnale ultrasound e il momento in cui arriva il segnale radio moltiplicata per la velocità del suono.

Vediamo un esempio, abbiamo il nodo che trasmette Tx e il nodo che riceve Rx e la distanza è la reale distanza tra i due nodi. Consideriamo che al tempo t_r viene inviato il segnale radio che viene ricevuto al tempo t_1 .

Dopo un tempo t_w inviamo un altro segnale ma in questo caso inviamo ad esempio l'ultrasound signal che viene inviato al tempo t_s e viene ricevuto al tempo t_2 . Vogliamo calcolare la distanza tra ricevente e mittente.

L'assunzione che facciamo qua è che abbiamo il $t_w = t_s - t_r$ che è la differenza tra il momento in cui viene trasmesso il t_s e il tempo in cui viene trasmesso t_r e sappiamo che $t_w \ll t_2 - t_1$. Questo è perchè la velocità del segnale radio è maggiore rispetto alla velocità del suono ovvero $v_r \gg v_s$.

Calcoliamo la distanza, per quello che abbiamo visto ora è uguale

al tempo in cui il segnale viene ricevuto meno il tempo in cui viene trasmesso moltiplicato per la velocità del segnale.

$$d = (t_1 - t_r) * v_r$$

Sappiamo che questo è valido anche per l'ultra sound signal quindi abbiamo:

$$d = (t_2 - t_s) * v_s$$

Possiamo partire da queste equazioni per estrarre il tempo, considerata la relazione che abbiamo scritto in precedenza $t_w = t_s - t_r$ possiamo riscrivere la formula della distanza come:

$$d = (t_2 - t_r - t_w) * v_s$$

Da qua possiamo estrarre il nostro tempo in cui viene trasmesso il primo segnale come:

$$t_r = t_2 - t_w - d/v_s$$

Ora possiamo sostituire t_r nella prima equazione e otteniamo:

$$d = (t_1 - t_2 + t_2 + d/v_s) * v_r$$

Possiamo spostare i vari termini e ottenere la seguente equazione:

$$d(v_s - v_r)/v_s * v_r = t_1 - t_2 + t_w$$

Ora possiamo fare una prima considerazione, abbiamo detto che la velocità della radio v_r è maggiore della velocità del suono v_s , quindi possiamo rimuovere v_r dall'ultima formula e otteniamo:

$$-d * v_r/v_s * v_r = t_1 - t_2 + t_w$$

Possiamo fare un'altra considerazione, abbiamo detto che $t_w \ll t_2 - t_1$ quindi possiamo scrivere:

$$d/v_s = t_2 - t_1$$

Se lo muoviamo dall'altra parte dell'equazione la distanza possiamo scriverla come:

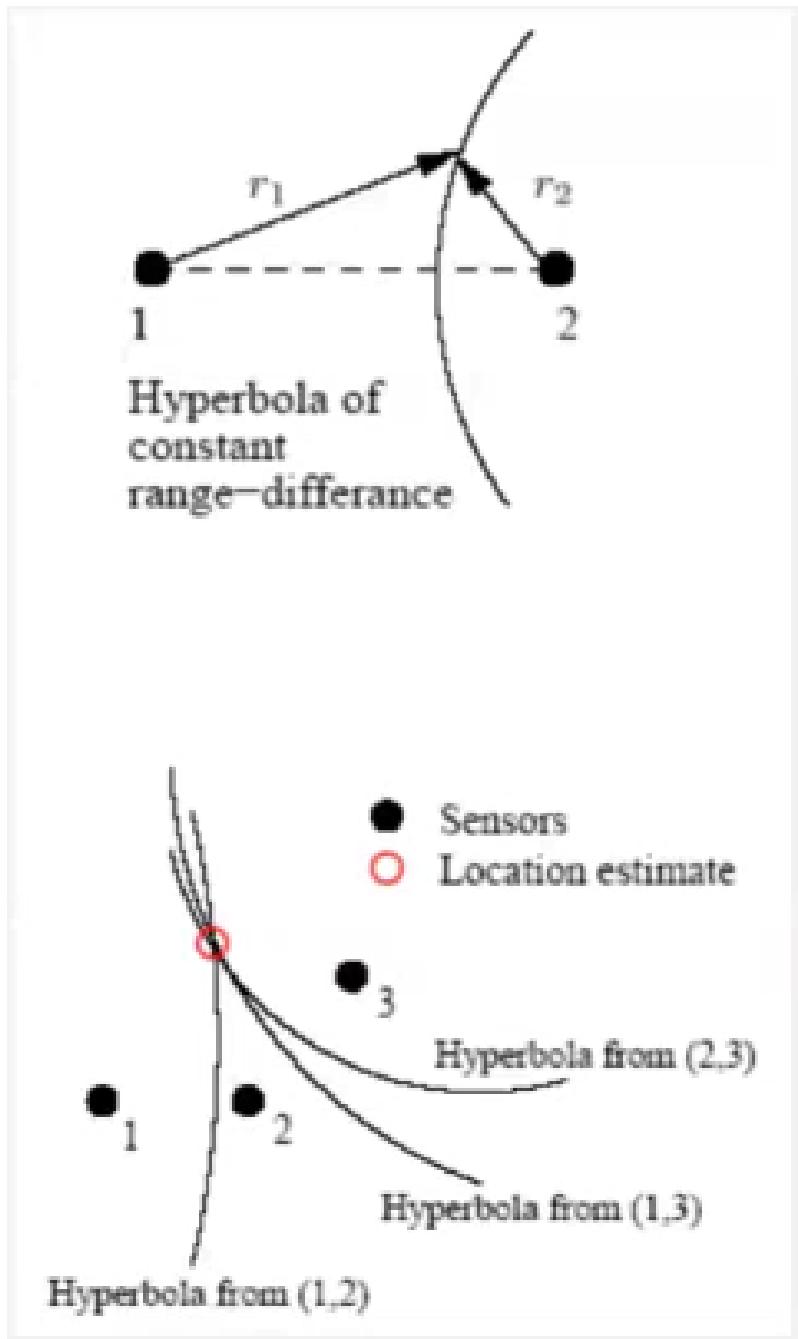
$$d = (t_2 - t_1) * v_s$$

Abbiamo dimostrato la formula della distanza $d = (t_2 - t_1) * v_s$ e in particolare che la distanza può essere calcolata conoscendo solamente la velocità del suono e la differenza del tempo di arrivo $t_2 - t_1$.

16.3.3 Time Difference of arrival

Possiamo anche essere più furbi e invece di avere solamente un trasmettitore possiamo pensare di avere più trasmettitori e in questo caso possiamo considerare i vari differenti tempi di interarrivo tra i vari dispositivi che trasmettono. In questo caso possiamo sfruttare l'Hyperbolic Location Theory che indica che un set di punti ad una distanza costante $c * \Delta$ da due dispositivi che trasmettono è un'iperbole e che ogni coppia di sensori fornisce due iperbole e nel punto in cui si intersecano si trova l'mitter.

Ad esempio se abbiamo M che è il dispositivo mobile (quello rosso)



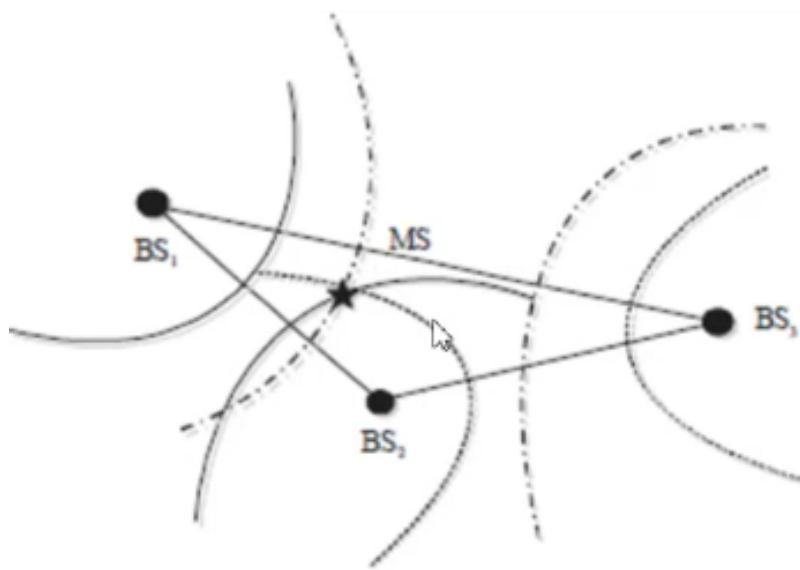
E abbiamo due tempi di arrivo all'anchor A1 e A2, possiamo calcolare

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

la distanza come la differenza tra le distanze:

$$d_{diff} = d(M, A_1) - d(M, A_2) = (t_1 - t_2) * c$$

Se abbiamo varie Base station abbiamo varie iperbole e queste si sovrappongono solamente in un punto e questo ci permette di capire quale è la localizzazione precisa del nodo mobile.



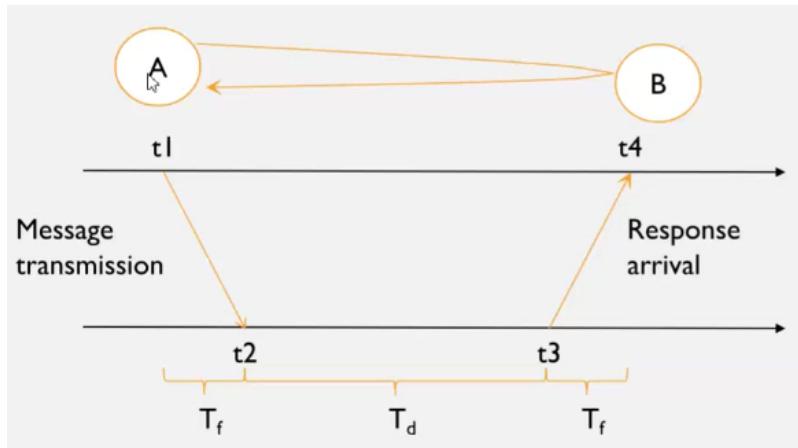
16.3.4 Round Trip Time of Flight

Un altro modo per utilizzare il tempo come signal metric è il Round Trip Time of Flight. In questo caso il transmitter e la measuring unit sono la stessa e il dispositivo che deve essere localizzato è solamente un transponder. In questo caso la measuring unit misura la differenza tra il tempo della trasmissione t_1 e il tempo della ricezione t_2 . La distanza quindi sarà:

$$d = c * (t_1 - t_2)/2$$

Questo riduce la necessità di sincronizzazione rispetto al Time of Arrival (ToA) ma se siamo in range piccoli il processing time del transpon-

der e della measuring unit non sono trascurabili e quindi vanno stimate nel modo corretto. Per esempio se abbiamo il nodo B di cui vogliamo stimare la posizione che riceve un segnale e poi lo manda indietro al nodo A.



Ora vediamo che il segnale parte da A e va a B al tempo t_1 , viene ricevuto al tempo t_2 poi passa del tempo per processare il segnale e inviarlo indietro, viene inviato da B ad A al tempo t_3 e poi viene ricevuto al tempo t_4 .

Conoscendo questi tempi possiamo tranquillamente andare a calcolare la distanza A-B.

$$d = c \cdot T_f = c \cdot \frac{1}{2}(T_4 - T_1 - T_d)$$

16.3.5 Angle of Arrival

Un'altra metrica può essere l'Angle of Arrival, in questo caso la location target la otteniamo con l'intersezione di varie coppie di linee di angoli. Nel caso 2D abbiamo bisogno di almeno due reference point e degli angoli corrispondenti, nel caso 3D almeno 3 punti con gli angoli corrispondenti.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

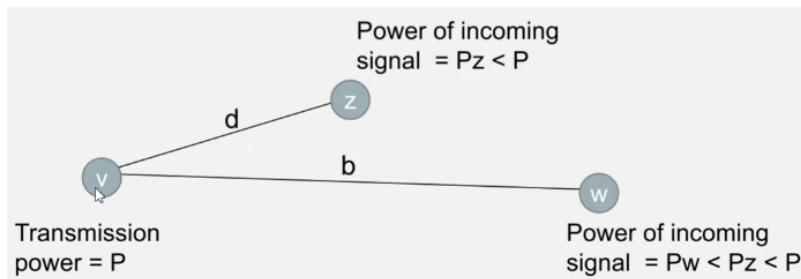
Per fare questo abbiamo bisogno di antenne direzionali che sono più costose e spesso non sono disponibili nei sensori. Recentemente queste sono state utilizzate perché grazie alla minuatuariazione possiamo avere varie antenne sullo stesso chip. La misura dell'angolo dovrebbe essere molto accurata.

16.3.6 TOA, TDOA, AOA

La grande assunzione che abbiamo in tutte queste metriche è che i dispositivi in queste metriche devono "vedersi" non può esserci qualcosa nel mondo, altrimenti il segnale viene "rovinato" dal multipath che mi modifica il tempo di arrivo e l'angolo.

16.3.7 Received Signal Strength

Un'altra metrica che può superare queste difficoltà è il Received Signal Strength. L'assunzione qua è che il segnale radio si attenua con la distanza, la potenza diminuisce in modo esponenziale, c'è una relazione tra la distanza e l'attenuazione del segnale. Se abbiamo al nodo V la potenza di trasmissione P , possiamo dire che quando il segnale arriva al nodo Z avremo una potenza del segnale P_z che sarà minore di P e quando il segnale arriva a W avremo una potenza P_w che sarà minore di P_z e di P .



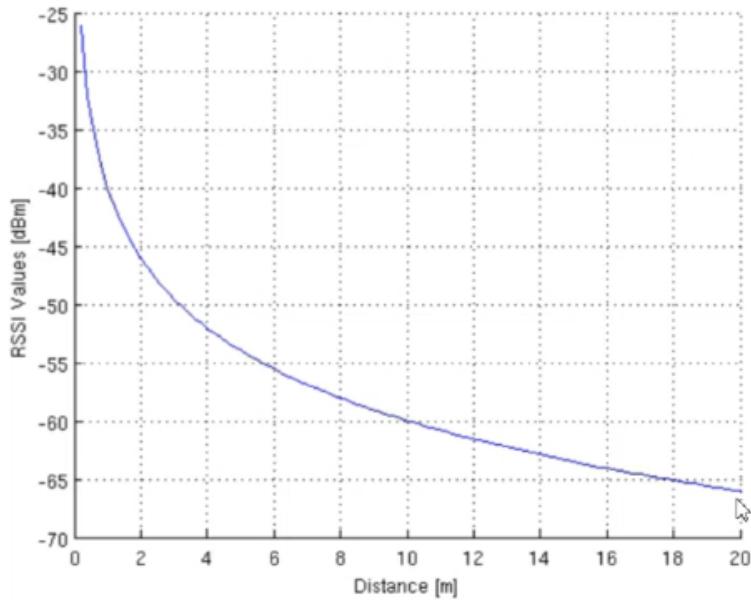
L'equazione che mette in relazione la potenza dell'invio del segnale

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

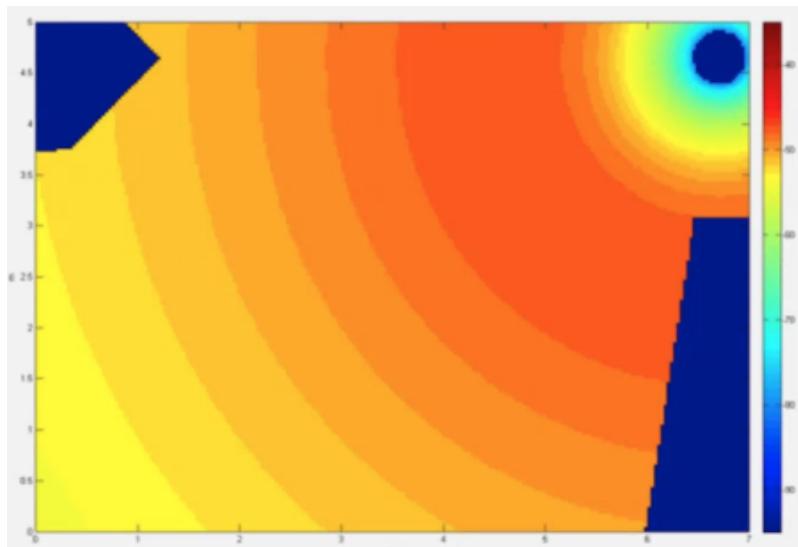
e la distanza tra mittente e destinatario è questa che funziona bene nel caso in cui stiamo all'esterno ma non funziona troppo bene all'interno.

$$P_R = P_T \frac{G_T G_R \lambda^2}{(4\pi)^2 d^n}$$

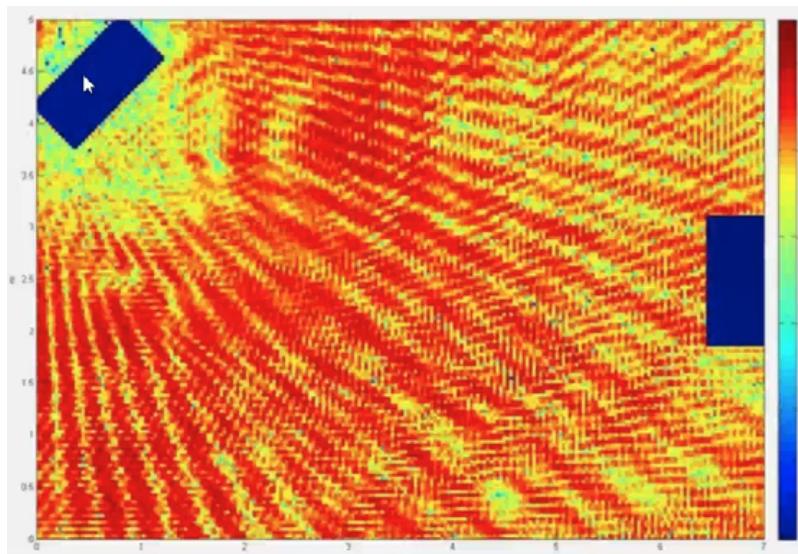
In uno scenario reale ci aspettiamo che la potenza del segnale vada a diminuire quando la distanza aumenta.



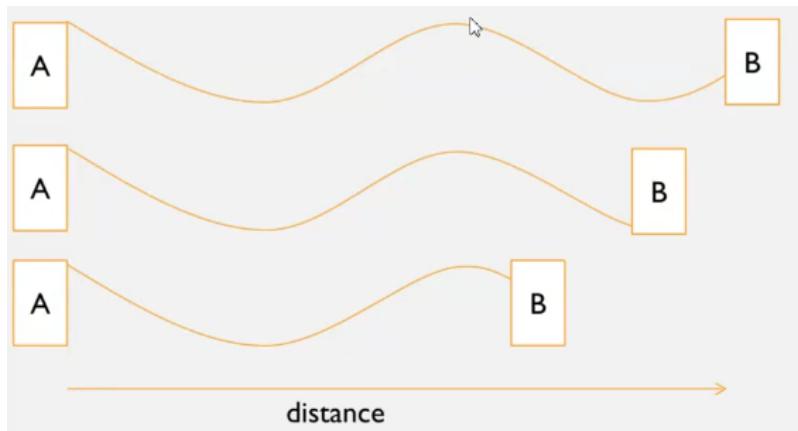
Questo non è vero negli ambienti chiusi perchè all'interno la potenza del segnale diminuisce molto quando il segnale riflette su qualcosa, idealmente la situazione dovrebbe essere questa:



Ma in realtà a causa degli ostacoli la situazione è questa:



Possiamo anche sfruttare la Received Signal Phase (RSP), in questo esempio assumiamo che il mittente A invia un segnale sinusoidale:



Se calcoliamo la fase possiamo determinare la reale distanza tra i due dispositivi, una volta che abbiamo calcolato la distanza possiamo andare ad usare lo stesso algoritmo di triangolazione che utilizziamo in ToA. Questo metodo però non funziona per distanze più lunghe di una lunghezza d'onda, possono essere utilizzati differenti segnali con differenti lunghezze d'onda. Anche in questo caso transmitter e receiver devono "vedersi" (Line of Sight).

Commenti sulle metriche che abbiamo visto:

- Per le time based abbiamo bisogno di un hardware/software speciale per la sincronizzazione che spesso non è disponibile nei prodotti commerciali. È anche importante l'accuracy del clock che dipende tra l'altro dalla temperatura. Abbiamo anche algoritmi di ottimizzazione complessi.
- Per Angle of Arrival e Received Signal Phase ci serve un hardware particolare per misurare l'angolazione.
- Per la Received Signal Strength non abbiamo bisogno di hardware apposito, ma non è molto accurata.

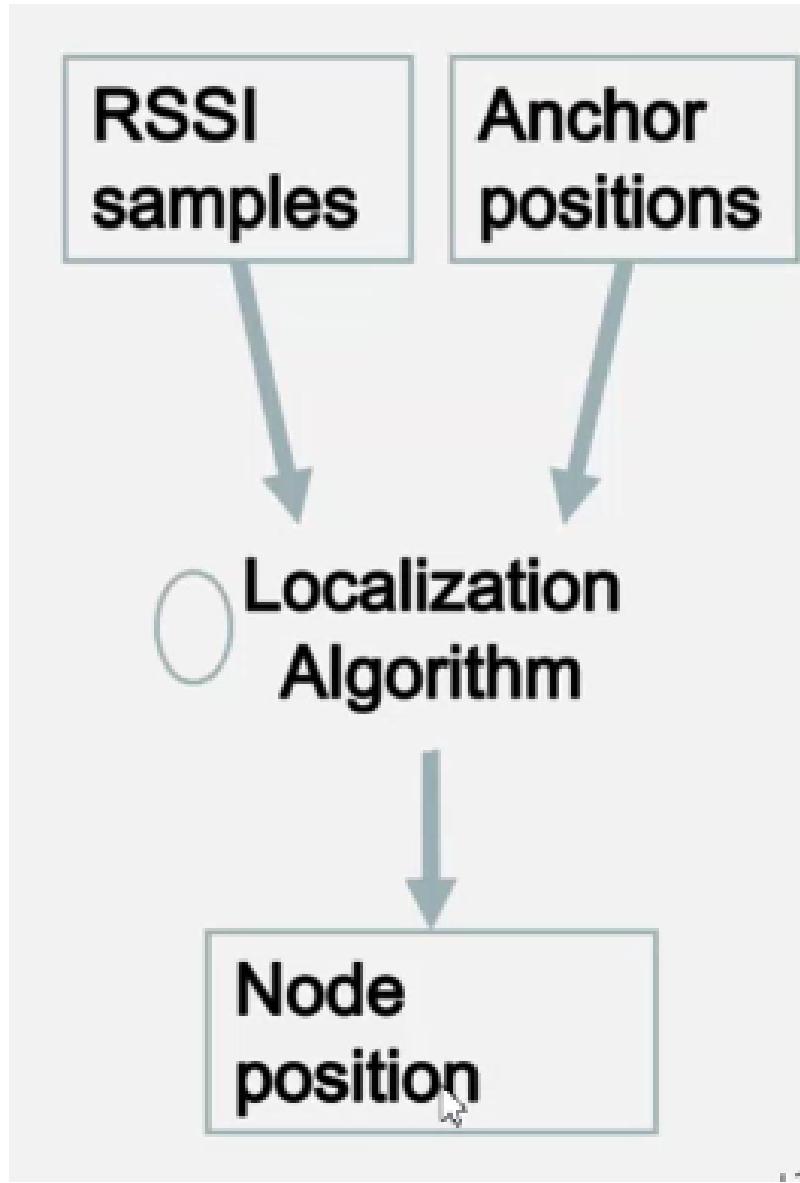
Tutte queste metriche sono affette dal problema del multipath. Dobbiamo sviluppare degli algoritmi per cercare di superare questi problemi.

16.4 Processing Methods

Una volta che abbiamo il segnale possiamo avere vari possibili approcci.

- Il primo è il range free approach è indipendente dai parametri del canale ma abbiamo una localizzazione imperfetta. Qua evitiamo il ranging eseguendo un confronto diretto dei sample RSSI.
- Con un range based approach abbiamo la localizzazione che si basa sull'RSSI ranging dai parametri del canale e potenzialmente potremmo avere una perfetta localizzazione nel momento in cui utilizziamo un canale ideale e questo solitamente non succede nel caso della localizzazione all'interno.

Essenzialmente abbiamo a disposizione l'RSSI Sample, la posizione dell'Anchor e tramite un Localization Algorithm proviamo a calcolare la posizione del nodo.



Abbiamo differenti processing Methods, possiamo dividerli tra Range Free (in generale hanno una bassa accuracy) e Range Based:

- Tra i range Free abbiamo:
 - Il più usato tra quelli Range Free è il metodo dei centroidi,

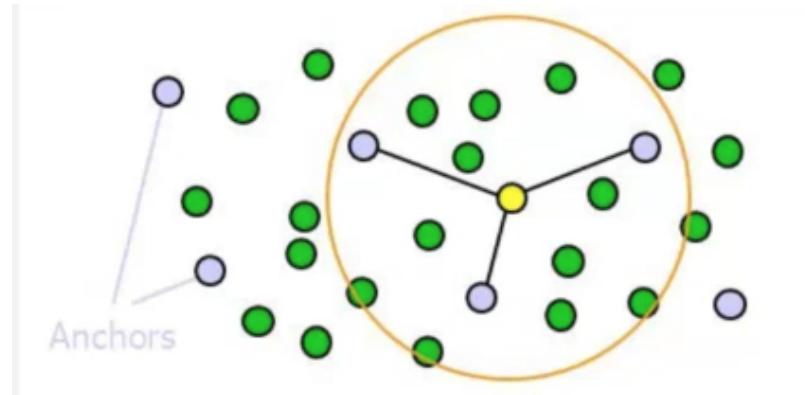
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

in questo caso è importante posizionare nel modo corretto l'anchor.

- Tra i Range Free abbiamo anche DV-HOP, che funziona male se i nodi si muovono o se sono distribuiti male.
 - Sempre tra i range free abbiamo APIT che necessita di una approssimazione per funzionare e ha problemi con le assunzioni
- Tra i range based abbiamo:
 - Il più usato è il Multilateration
 - Il mix max è molto semplice ma le performances non sono buonissime
 - Il maximum Likelihood che è complesso ma asintoticamente può essere ottimo.

16.4.1 Range Free: Centroid

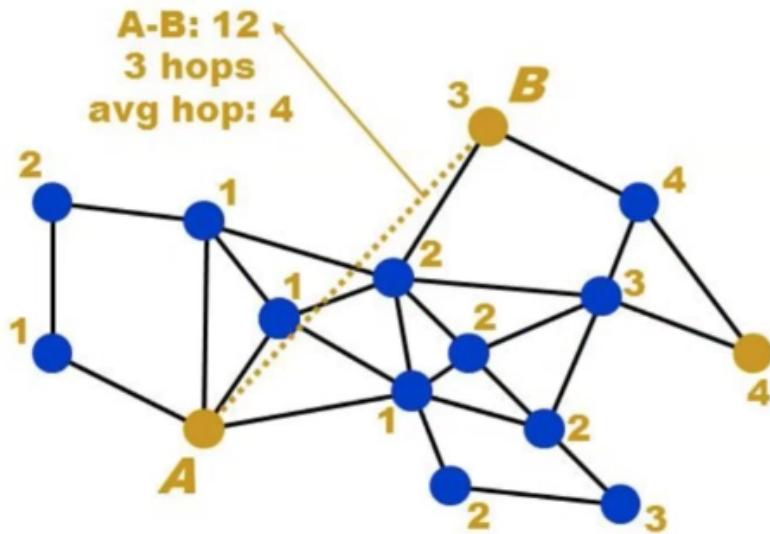
L'algoritmo più semplice tra i Range Free, l'idea è di non usare un range, si deployano molti anchor e periodicamente gli anchor inviano in broadcast la loro localizzazione, quello che fa l'algoritmo è ascoltare gli anchor che inviano in broadcast e poi fare la stima della posizione facendo una media.



Per il corretto funzionamento è fondamentale il posizionamento iniziale degli anchor.

16.4.2 Range Free: DV-HOP

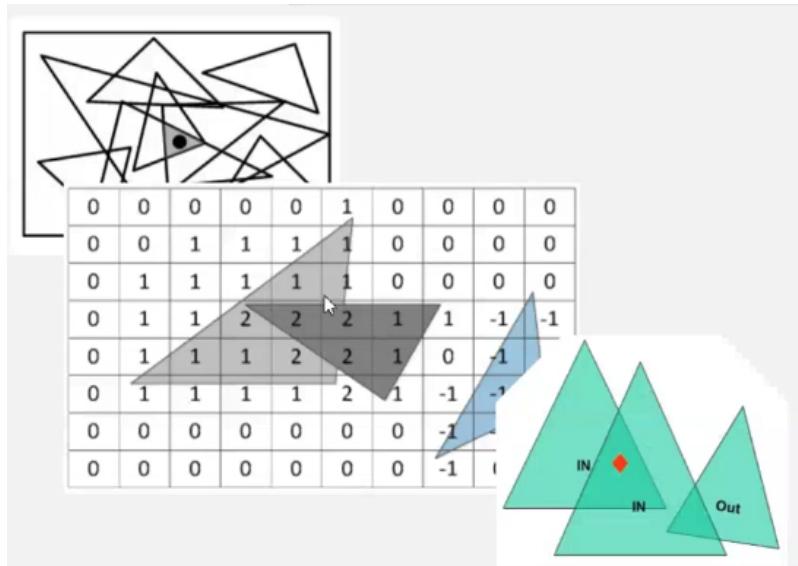
Un altro algoritmo che possiamo utilizzare è DV-HOP, qua gli anchor inviano all'interno della rete le posizioni che si conoscono e inviano la hop distance media tra tutti gli anchor.



Per esempio tra A e B abbiamo 3 hop ma gli hop medi sono 4, moltiplichiamo la media con il numero di hop. Quindi abbiamo $3*4=12$ metri tra A e B.

16.4.3 APIT

È simile al centroide ma qua abbiamo un ambiente composto da vari triangoli, quando un nodo riceve da un anchor crea una mappa sommando un 1 ogni volta che riceve un segnale da un angolo del triangolo.



16.4.4 Range-Based: Multilateration

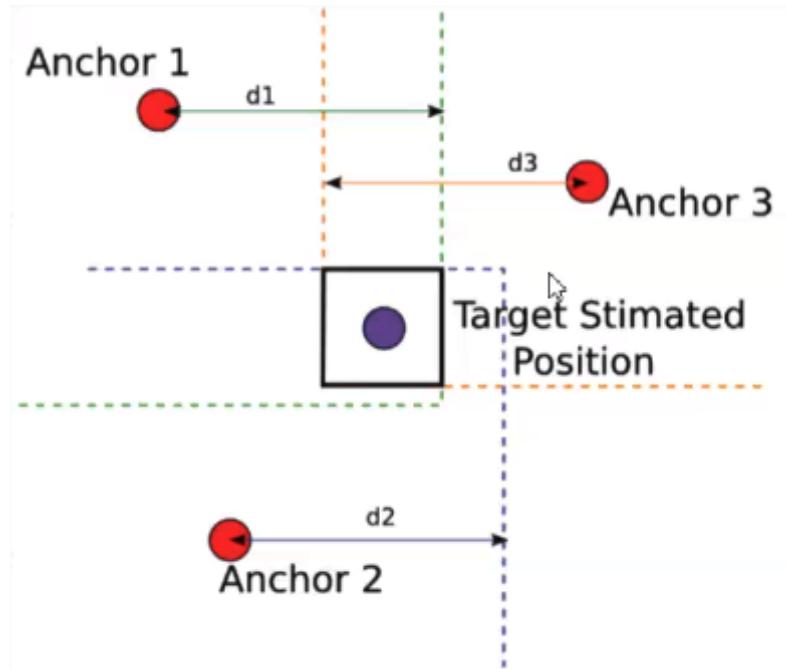
Per quanto riguarda gli algoritmi Range Based abbiamo il Multilateration che è il più semplice. Possiamo calcolare le distanze, ad esempio se abbiamo 3 nodi abbiamo 3 cerchi che sono stati creati calcolando r_1 , r_2 e r_3 per ciascuno di essi. Poi l'algoritmo calcola l'intersezione tra i tre cerchi, questi r_1 , r_2 e r_3 non sono accurati perchè dipendono dal multipath, quindi non funzionano in un ambiente reale.

16.4.5 Range-Based: Min-Max

Min-Max è un altro algoritmo interessante, è molto semplice, consiste nel calcolare un minimo e un massimo, cosa fa?

$$[\max(x_i - d_i), \max(y_i - d_i)] \times [\min(x_i + d_i), \min(y_i + d_i)]$$

Viene calcolato un minimo e un massimo dell'RSS, il nodo di cui vogliamo trovare la posizione crea una associazione tra la posizione di ogni anchor e la potenza dell'RSS ricevuto da quell'anchor. Poi invertendo la nominal distance-power loss law il nodo stima la distanza da quel beacon. Il nodo crea una coppia di linee orizzontali e una coppia di linee verticali, ognuna per ogni coppia di anchor.



Poi il nodo localizza se stesso all'interno del rettangolo considerando le linee che sono più interne, in pratica considera il centroide del rettangolo.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Esercizio: Abbiamo un indoor environment e abbiamo 3 anchor, A1, A2 e A3, i tre anchor ricevono un segnale radio da un nodo mobile (quello rosso).

La distanza tra m e A1 è di 3 metri, la distanza con A2 è 2 metri e con A3 è 3 metri.

Usando Min Max si vuole calcolare la posizione di m e il localization error?

Per trovare la soluzione possiamo partire con l'approccio analitico, abbiamo queste coordinate calcolate come differenza e calcolando il massimo e il minimo:

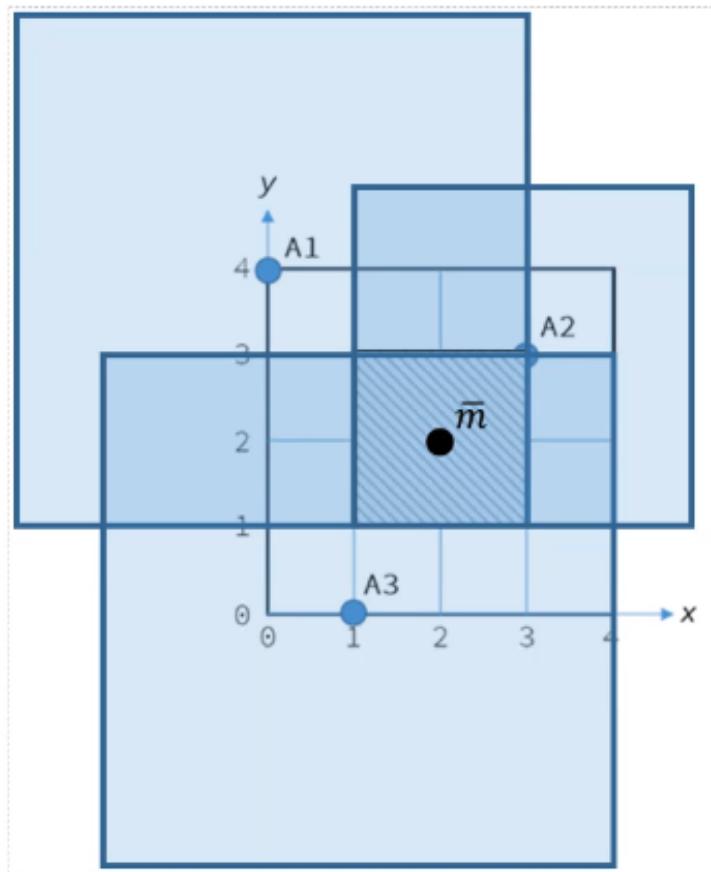
$$[\max(x_i - d_i), \max(y_i - d_i)] \times [\min(x_i + d_i), \min(y_i + d_i)]$$

In questo caso la distanza viene calcolata dall'anchor node (abbiamo le varie distanze) abbiamo questa formula:

$$\begin{aligned} &[\max(-3, 1, -2), \max(1, 1, -3)] \times [\min(3, 5, 4), \min(7, 5, 3)] \\ &\quad [1, 1] \times [3, 3] \end{aligned}$$

Otteniamo le coordinate dell'area che sono (1, 1) (3, 3), la nostra stima è al centro di questo rettangolo, quindi per noi la stima del centroide è in posizione (2, 2).

Possiamo ottenere lo stesso risultato utilizzando la soluzione grafica, in questo caso dobbiamo creare dei rettangoli attorno a A2, A3 e A1 calcoliamo i corrispondenti rettangoli:



Se prendiamo la linea verticale più interna della intersezione dei tre rettangoli otteniamo lo stesso rettangolo che avevamo ottenuto in modo analitico.

Quale è l'accuracy di questa tecnica?

Calcoliamo l'Euclidean Distance e otteniamo un errore di circa 71 centimetri.

$$d(m, \bar{m}) = \sqrt{(x - \bar{x})^2 + (y - \bar{y})^2} = \sqrt{0.25 + 0.25} \cong 0.71\text{m}$$

È abbastanza semplice da calcolare e da implementare ma anche graficamente non è complicato.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

16.4.6 Range-Based: Maximum Likelihood

Un altro approccio che si basa sul ranging è il Maximum Likelihood, in questo caso abbiamo un vettore di valori RSSI r ottenuti da n beacon con coordinate x_b e y_b . L'algoritmo calcola la probabilità a priori di ricevere il vettore r per ogni potenziale posizione del nodo sconosciuto. La posizione che massimizza questa probabilità viene poi selezionata come posizione stimata del nodo.

Possiamo scrivere in modo matematico tutto questo e vediamo che quello che vogliamo fare è solamente minimizzare il mean square error.

$$\begin{aligned} f_i(x_0, y_0) &= d_i - \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} \\ f_i(x_0, y_0) &= 0 \\ -x_i^2 - y_i^2 + d_i^2 &= (x_0^2 + y_0^2) + x_0(-2x_i) + y_0(-2y_i) \end{aligned}$$

In particolare imponiamo che la cost function $f_i(x_0, y_0)$ deve essere uguale a 0. In particolare andando avanti con i calcoli quello che otteniamo è:

$$-x_i^2 - y_i^2 + d_i^2 - (-x_n^2 - y_n^2 + d_n^2) = 2x_0(x_n - x_i) + 2y_0(y_n - y_i)$$

Alla fine otteniamo il sistema $y = Xb$ che viene poi risolto utilizzando $b = (X^T X)^{-1} X^T y$. Le matrici che abbiamo in questo calcolo sono le seguenti:

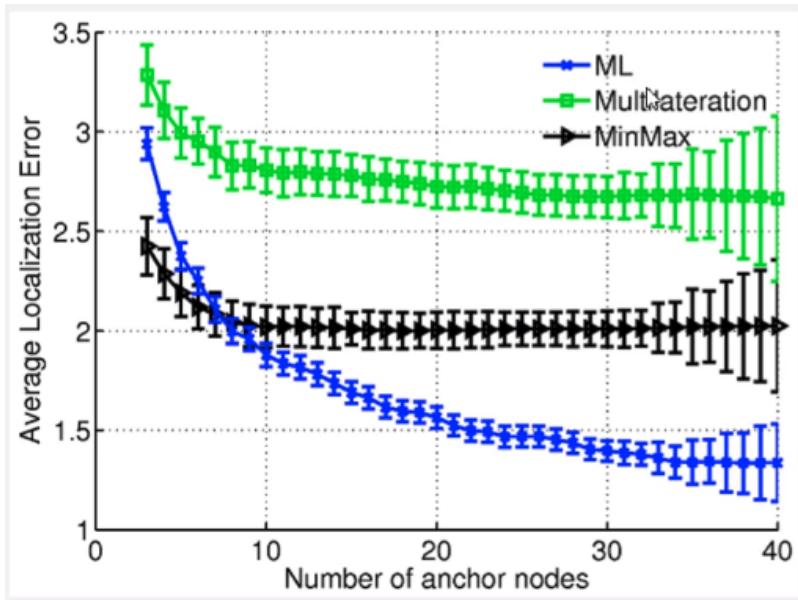
$$X = \begin{bmatrix} 2(x_n - x_1) & 2(y_n - y_1) \\ \vdots & \vdots \\ 2(x_n - x_{n-1}) & 2(y_n - y_{n-1}) \end{bmatrix}$$

$$y = \begin{bmatrix} -x_1^2 - y_1^2 + d_1^2 - (-x_n^2 - y_n^2 + d_n^2) \\ \vdots \\ -x_{n-1}^2 - y_{n-1}^2 + d_{n-1}^2 - (-x_n^2 - y_n^2 + d_n^2) \end{bmatrix}$$

$$b = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

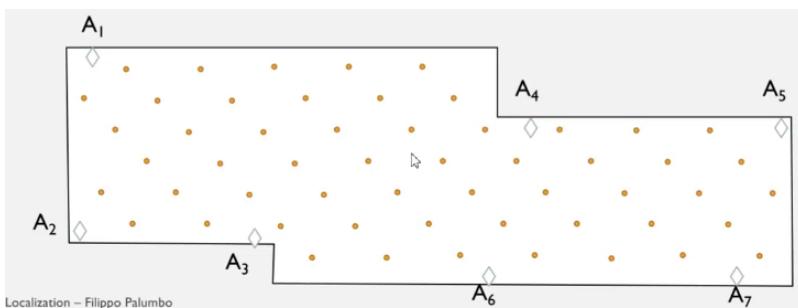
Questo metodo è più complicato dagli altri ma minimizza la varianza della stima dell'errore a mano a mano che il numero delle osservazioni sale ad infinito, nella maggior parte delle occasioni il numero delle osservazioni è basso quindi questo approccio mi fornisce una soluzione che non è ottima.

In questo grafico abbiamo in particolare un confronto tra Multilateration, Min Max e Maximum Likelihood. Con l'aumento del numero dei nodi la Maximum Likelihood diventa la scelta migliore, all'inizio invece, quando abbiamo pochi dati abbiamo che Min Max funziona meglio.

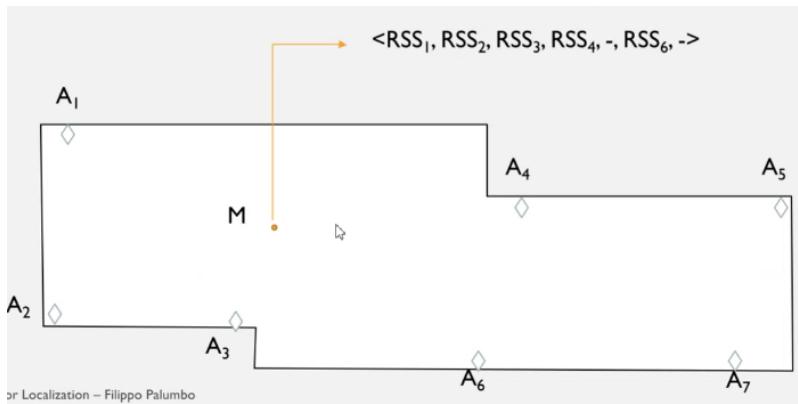


16.5 Scene Analysis

Un'altra tecnica crescente che sta diventando sempre più utilizzata è la Scene Analysis, è una sorta di ibrido perchè non utilizza esattamente le distanze che vengono calcolate come metrica ma va a fare una "fotografia" dell'ambiente ad un certo punto misurando l'RSSI rispetto ad un set di anchor nell'ambiente. Quindi abbiamo una griglia di misurazioni e per ogni punto abbiamo una tuple di misurazioni RSS per ogni anchor. In questo caso abbiamo 7 misurazioni per ogni posizione.



Poi in un altro momento successivo abbiamo una certa posizione x, y , calcoliamo in ogni punto la stessa RSSI e proviamo a fare il match con le tuple calcolate in precedenza e scegliamo quella che minimizza le distanze. La tuple che sceglieremo sarà quella che ci fornirà la stima del punto per la posizione che non conosciamo.



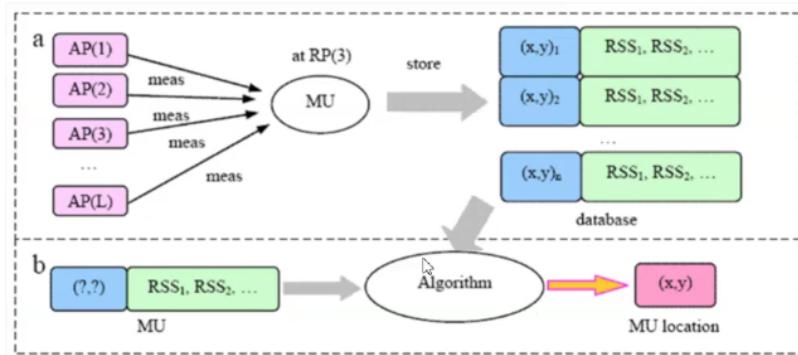
Vediamo il funzionamento più nel dettaglio:

- La posizione del target è determinata misurando l'RSS del target rispetto agli anchors.
- Abbiamo una nuova tuple R per i nuovi RSS
- R viene confrontata con tutte le tuple R_i che già avevamo memorizzato.
- La posizione del target viene approssimata con la posizione dei punti la cui tupla è più simile a R.

Per trovare le Tuple più simili possiamo usare dei metodi probabilistici, oppure la KNN oppure le neural network. Abbiamo una fase offline in cui prendiamo i nostri dati, che chiamiamo fingerprint, quindi otteniamo dei vettori di fingerprint. Nella online phase eseguiamo il match

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

di dell'RSS con un fingerprint esistente in modo probabilistico oppure usando delle metriche di distanza.



Per capire la tuple più simile, il metodo più usato negli anni passati era il KNN.

Avevamo una tuple sconosciuta e le tuple precedentemente raccolte e volevamo trovare i K punti per cui Mean Square è minimo. Poi la posizione del target viene stimata come media delle posizioni tra le coordinate dei K punti che corrispondono alle tuple che matchano meglio.

Un altro approccio, simile al KNN consiste nell'utilizzo del Machine Learning, in questo caso il database delle informazioni del fingerprint viene memorizzato in una Neural Network. Le informazioni del fingerprint vengono utilizzate per eseguire il training delle neural network e poi nella fase online per ogni tuple abbiamo una predizione data dalla neural network. L'accuracy dipende dalla qualità del training, nell'ambiente di casa sono stati notati errori di circa 2-3 metri, lo stesso discorso vale per KNN.

Con l'approccio del fingerprinting abbiamo un costo di configurazione per costruire il database, inoltre se l'ambiente cambia dobbiamo ricostruire il database quindi abbiamo anche un costo di mantenimento e di ricostruzione.

16.6 Device-Free Localization

Recentemente è nato un nuovo approccio che si basa sul Device-Free Localization, possiamo sfruttare la degradazione dell’RSSI a causa del corpo umano oppure possimo sfruttare l’effetto doppler. Ad esempio nelle nuove antenne del Wi-Fi abbiamo almeno 3 antenne, quindi abbiamo un sistema MIMO e possiamo utilizzare l’effetto doppler che ci permette di capire la posizione del target senza utilizzare dispositivi come smartphone o wearable devices.

Una di queste tecniche è la radio tomography, abbiamo vari anchor che trasmettono segnali in un ambiente e l’utente che interferisce con questo segnale può essere identificato controllando l’area in cui la degradazione è maggiore.

16.6.1 CSI Based

Un altro metodo che consiste nell’utilizzo del Device-Free Localization è il CSI. Il CSI è il Channel State Information che è una informazione che viene trasmessa dagli access point commerciali ed è embeddata nel protocollo 802.11AN per il WiFi. Possiamo utilizzare il multipath effect al livello fisico per ottenere questi Channel State information che portano la phase di ogni subcarrier. Abbiamo bisogno di avere accesso al livello fisico del dispositivo mentre invece con l’RSSI ci basta avere accesso al Mac Layer.

Una volta che abbiamo il CSI possiamo utilizzare Fingerprinting, Multilateration e il MIMO per capire la distanza dall’utente e quando gli utenti si muovono utilizzando l’effetto Doppler.

16.6.2 Proximity

Un altro possibile approccio è la proximity, possiamo stimare la posizione di un oggetto basandoci sulle sue informazioni di connettività. Se abbi-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

amo Bluetooth, se vediamo che il dispositivo trasmette ad una potenza molto bassa possiamo capire che siamo vicini al dispositivo.

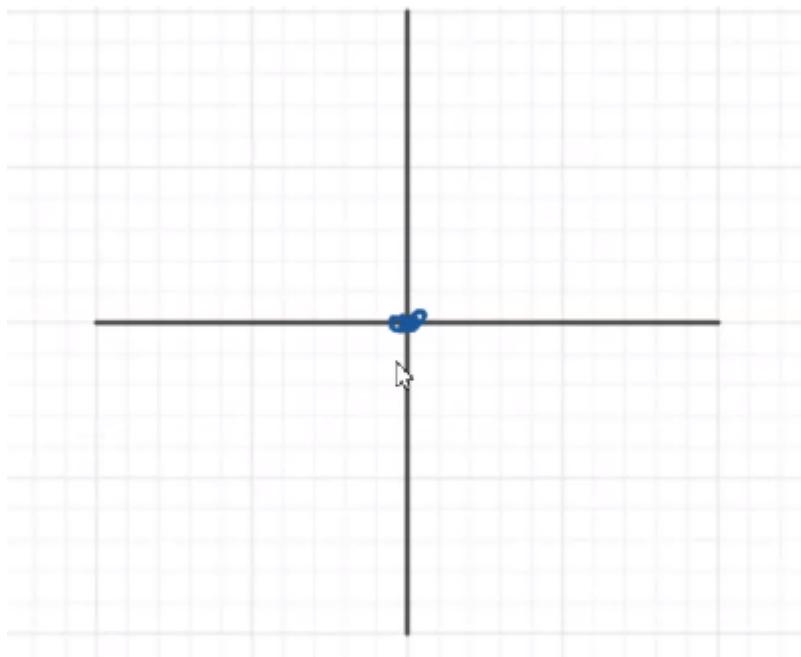
16.7 Performance Metrics

Con questi sistemi una cosa che vogliamo fare è capire l'errore che facciamo in termini di accuracy e di precision. Vogliamo misurare le performance del localization system. In particolare ci interessano:

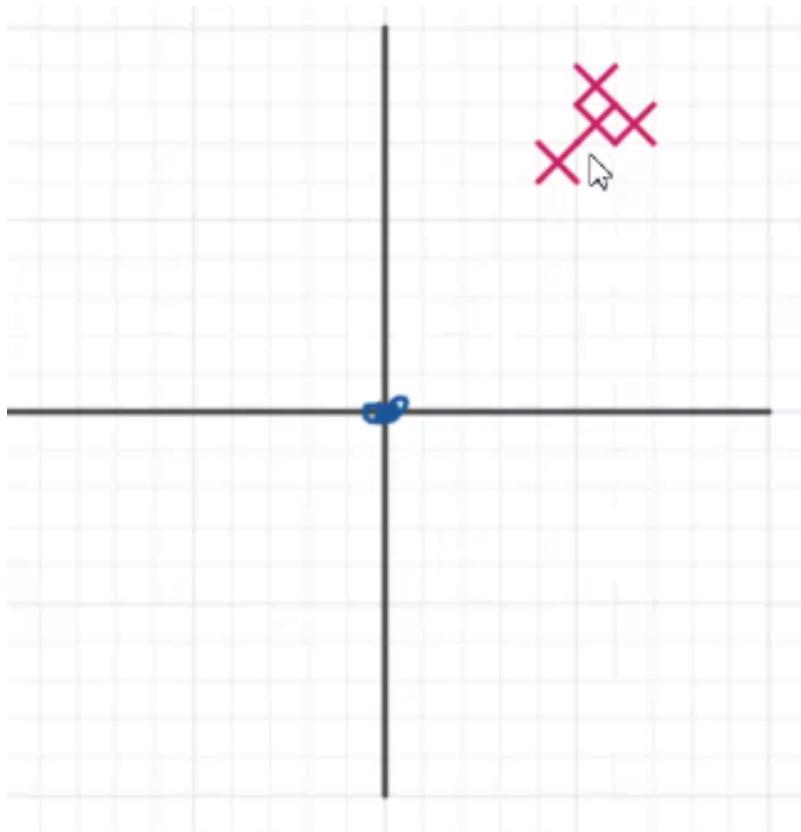
- Accuracy: indica la mean distance error tra la posizione reale e la posizione stimata.
- Precision: Misura la consistenza del sistema, in particolare se facciamo varie stime, come varia l'accuracy? La precision viene misurata con la distribuzione dell'accuracy della localizzazione. In pratica quanto sono sparsi i punti che stimo.

Basandoci sulle performances che vogliamo ottenere possiamo avere scenari differenti e possiamo utilizzare delle tecnologie differenti. Non abbiamo solamente queste due metriche, possiamo anche considerare Complexity, Robustness, Scalability e Costo che è cruciale se vogliamo avere un sistema di localizzazione interno.

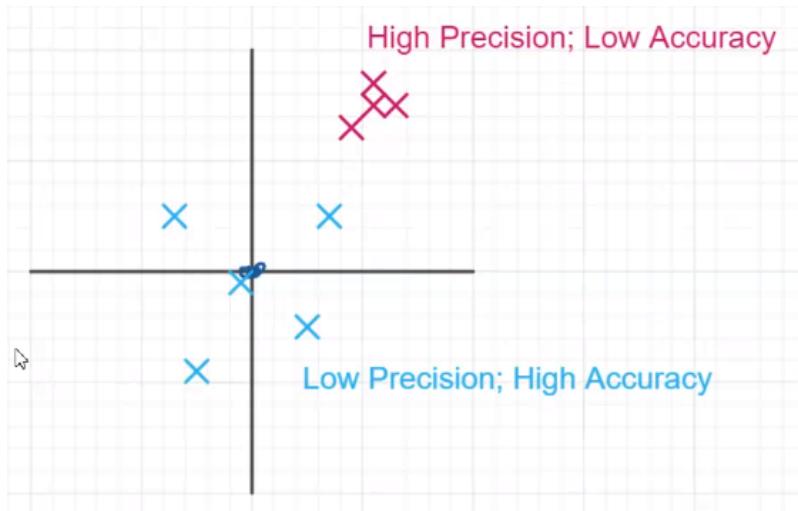
Per capire meglio l'idea di accuracy e precision vediamo un esempio, supponiamo di voler capire dove si trova il punto in blu



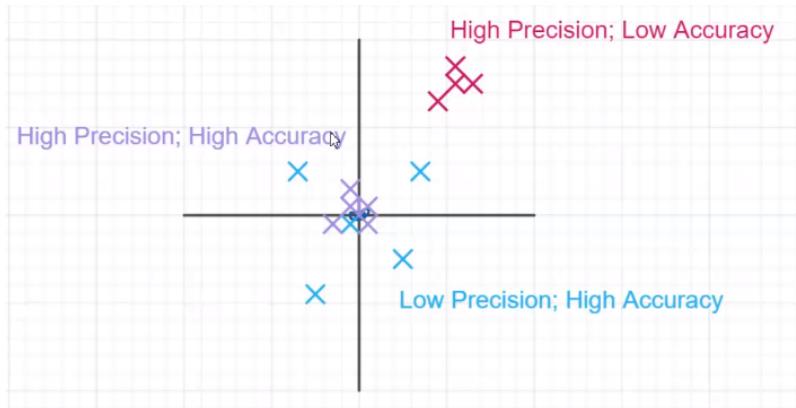
Un sistema che ci fornisce questa stima e che dice che la posizione si trova in alto a destra è alto in termini di precision (perchè i punti che ho stimato sono tutti vicini tra loro) ma è basso in termini di accuracy (perchè sono lontano dal punto blu).



Se invece abbiamo un sistema in cui le X stanno più vicine al punto che cerchiamo allora possiamo dire che il sistema è low in precision perchè è sparso ma ha una accuracy più alta dell'altro perchè i punti si trovano più vicini al punto che sto stimando.



Quello che vogliamo è un sistema in cui abbiamo tutti i punti vicini e che però devono anche essere vicini al target, ovvero un sistema che deve avere una precisione alta e una accuracy alta.



16.8 Futuro

Recentemente sono entrati sul mercato vari sensori e dispositivi e tutti questi segnali che vengono emessi possono essere letti da uno smartphone. Possiamo utilizzare ad esempio wearable devices per l'indoor localization

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni e delle slide. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

e per aumentare l'accuracy. In questo caso abbiamo un approccio data fusion, per aumentare l'accuracy possiamo sfruttare alcune informazioni aggiuntive:

- Possiamo utilizzare la mappa.
- Possiamo utilizzare le posizioni già conosciute e la prossimità a queste posizioni.
- Possiamo utilizzare gli Inertial Systems.
- Possiamo usare visual e images computing.