UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

ANALISI SPERIMENTALE DI CONNETTIVITÁ PER UNA RETE DI SENSORI WIRELESS

Relatore: Ch.mo prof. LUCA SCHENATO

Laureando: FRANCESCO ROVERON

TESI DI LAUREA

CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE

ANNO ACCADEMICO 2006-2007

Alla mia famiglia

A Veronica

INDICE

SOMMARIO

Nella tesi vengono discussi ed elaborati i dati relativi alla connettività di una rete wireless ponendo l'attenzione principalmente su alcuni parametri che caratterizzano la qualità della trasmissione e vengono descritti l'algoritmo e le modalità con cui sono stati raccolti.

Nell'elaborazione delle informazioni si è cercato un taglio statistico piuttosto che ingegneristico, mi spiego: in letteratura mancano informazioni sull'impatto che l'ambiente, qualsiasi esso sia, ha su di una WSN, senza distinzione del tipo di ostacoli e della loro disposizione. L'importanza di ciò nasce dal fatto che la maggior parte delle reti senza fili viene implementata in aree con disposizione ad elevata variabilità e diventa impossibile, per chi la realizza, tenere in considerazione uno schema statico dell'ambiente circostante e di tutti quegli imprevisti che non possono sempre essere calcolati a priori.

L'obiettivo che si vuole raggiungere al termine di questo studio risulta essere l'individuazione di intervalli, significativi dal punto di vista statistico, all'interno dei quali ha senso considerare possibile e affidabile la connessione in termini di distanza e qualità.

Ne uscirà un quadro complesso a causa del concatenarsi di tre problemi principali: assorbimento della potenza trasmessa a causa di fenomeni fisici, imprevedibilità degli ostacoli posti nel mezzo di una comunicazione e la scarsa affidabilità del valore misurato della potenza in ricezione.

INTRODUZIONE

L'utilizzo di dispositivi wireless è una realtà nel campo della ricerca, ma è diventato di quotidiano utilizzo anche all'interno delle nostre case. Si pensi al termostato del riscaldamento, ai sistemi di allarmi, al bluetooth, etc.

Figura 1: esempi di nodi wireless; al centro il Tmote-Sky

La cosa che più differenzia gli strumenti sopra citati e i dispositivi adoperati nella tesi, è il tipo di utilizzo. Infatti, a parte la dotazione più o meno accessoriata (sensori, calcolatori, memorie) la cosa più delicata da gestire è la durata di vita media e il posizionamento strategico per creare la rete desiderata. Ecco perché ci si pone così tanti problemi per quanto concerne la capacità e la qualità di trasmissione, che sono condizionate da:

- alimentazione fornita;
- potenza in trasmissione;
- distanza.

Queste diventano delle variabili fondamentali se pensiamo ai tempi di funzionamento dell'intera rete e alla copertura che vogliamo più estesa possibile.

Possiamo pensare, ad esempio, a una network interna ad una palazzina servizi: la presenza umana varia al variare delle ore del giorno con picchi negli orari d'ufficio, ore centrali della mattinata e del pomeriggio, e basse incidenze agli orari di pranzo e di notte; la temperatura e l'umidità non sono costanti, sebbene il sistema di condizionamento sia magari centralizzato; computer, sedie e oggettistica varia possono cambiare la loro disposizione passando da sopra a sotto la scrivania, o da dentro a fuori degli armadi. Altro esempio è una rete di sensori ambientali per il monitoraggio degli incendi d'estate: c'è la possibilità che i sensori possano essere trasportati o distrutti

dagli animali (perché ingeriti o calpestati); l'umidità può variare molto nell'arco della giornata (al mattino potrebbe esserci nebbia, per esempio).

Senza considerare le interferenze radio, problema ormai esteso in tutte le aree anche non eccessivamente urbanizzate: creano disagi persino i sensori di presenza delle porte automatiche dei supermercati o i rilevatori degli antifurto.

Mentre una parte di questi imprevisti può essere evitata con alcuni accorgimenti o gestita nel momento in cui si conosce il fenomeno che li generano, una gran parte di essi è totalmente imprevedibile. Il variare delle condizioni climatiche può essere gestito utilizzando parametrizzazioni diverse nell'arco della giornata, mentre altrettanto non si può fare nel momento in cui un'apparecchiatura, vicina al sensore, viene accesa generando un forte campo magnetico.

Nella tesi si propone e realizza un metodo per raccogliere quanti più dati possibili, ma risulta chiaro che ai fini statistici servirebbero quantità di informazioni molto più elevate rispetto a quelle raccolte e una differenziazione delle tipologie di ambiente esaminate. Proprio per questo si è cercato di implementare l'algoritmo affinché sia il più portabile possibile e riutilizzabile con numeri di sensori e per orari di funzionamento variabili.

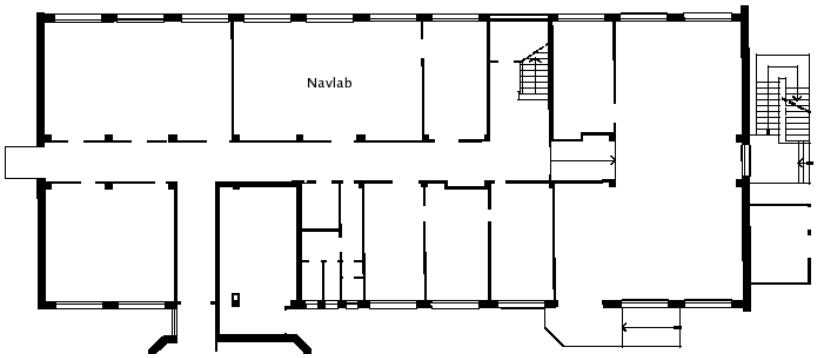


Figura 2: pianta del piano terra del DEI-A

Possono risultare comunque di aiuto le informazioni trovate poiché riguardano un ambiente abbastanza definito dove si conoscono i potenziali disturbi e, cosa più importante, identifica piuttosto concretamente le condizioni tipiche di utilizzo in ambiente lavorativo. L'esperimento è infatti avvenuto in un intero piano dell'edificio DEI-A nell'arco dell'intera giornata lavorativa.

capitolo 1

Principi generali sulla connettività

1.1 Le reti di sensori wireless

Il wireless, frenato inizialmente dai costi non proprio competitivi dei dispositivi, ha assunto oramai un ruolo chiave non solo nell'industria, ma anche nella vita comune del cittadino. Le reti che sfruttano questa tecnologia si suddividono in base alla potenza delle onde magnetiche utilizzate e, di conseguenza, in base alle distanze coperte.

In questa trattazione si considera una PAM (personal area network): una rete a bassa potenza capace di coprire lunghezze non superiori ai 50 metri indoor e ai 150 metri outdoor.

In generale, una WSN (wireless sensor network), è costituita da dispositivi programmabili, chiamati nodi, capaci di comunicare via radio tra loro e spesso dotati di sensori (come ad esempio per la temperatura) in grado di instaurare delle connessioni ben distribuite e capillari con l'obiettivo di monitorare attentamente un'area territoriale (industriale, faunistica, urbana..). Applicazioni basate su queste reti, oltre al monitoraggio ambientale, sono il controllo e manutenzione di macchine utensili, monitoraggio delle funzioni vitali di pazienti dimessi dall'ospedale, controllo di strutture meccaniche e molte altre.

La principale differenza che intercorre tra le reti ad-hoc e quelle di sensori è il numero di nodi che le compongono: in una WSN il numero di sensori può essere di gran lunga superiore a quello delle reti tradizionali, con una direzione del traffico dati non precisa; normalmente ogni nodo spedisce pacchetti a tutti i dispositivi in ascolto, e non ad alcuni soltanto.

Il punto fondamentale è comunque l'aspetto energetico: utilizzandoli come sensori e posizionandoli anche in luoghi non facilmente o frequentemente raggiungibili, vengono spesso alimentati a batterie; diventa necessario assicurare loro un'autonomia discreta in base al servizio richiesto, per evitare onerosi tempi di sostituzione (nel caso il numero di sensori sia elevato) o lunghi spostamenti per raggiungerli tutti.

Proprio per gestire la perdita di alcuni dispositivi in caso di scarsa alimentazione elettrica, le reti di sensori devono essere dinamiche: qualsiasi malfunzionamento, hardware o software, o danno subito non devono precludere la connettività globale e neppure richiedere l'intervento di operatori umani.

Affinché i dati spediti via etere siano ricevuti e arrivino privi di errori, sono necessari dei protocolli di trasmissione.

Per questo tipo di reti, diversamente che per gli standard utilizzati nell'informatica, i protocolli sono snelli perché pensati su misura per i dispositivi su cui si lavora. Questo comporta una completa gestione dell'hardware, fino ai livelli più bassi, con un incremento delle potenzialità e un miglior controllo.

1.2 Lo standard IEEE 802.15.4

Questo standard consente reti di comunicazioni senza fili a basso costo in applicazioni con potenza limitata e ha come obiettivi la semplicità nell'installazione, l'affidabilità, l'economicità in termini di tempo, durata (si pensi all'alimentazione) e costo; questi obiettivi sono stati raggiunti utilizzando sistemi di modulazione più lenti, schemi di codifica meno pesanti e pacchetti dati più corti.

I dispositivi che utilizzano questo protocollo sono di due tipi: full-function device (FFD) e reduced-function device (RFD). Gli FFD lavorano come sonda all'interno di una personal area network (PAN), come coordinatori (Base Station) o come apparecchi (servodispositivi) e possono scambiare informazioni tra di loro e con gli RFD. Questi ultimi, invece, sono molto semplici e possono solamente inoltrare richieste o allarmi agli FFD ma non tra loro; funzionano praticamente da rilevatori.

Se per definizione bastano due dispositivi per realizzare una rete (low-rate wireless personal area networks, LR-WPANs), non è altrettanto precisa la descrizione di "copertura della rete": per i dispositivi wireless le caratteristiche della propagazione sono dinamiche e aleatorie e dipendono fortemente dalla posizione, dall'ambiente circostante, dalle condizioni ambientali e fisiche.

Topologie

A seconda delle specifiche di partenza, è possibile l'implementazione di due diverse topologie: a stella e peer-to-peer. Nella prima, i nodi comunicano con un "coordinatore centrale" (FFD) che solitamente è programmato diversamente dagli altri per essere in grado di gestire o anche solo iniziare la comunicazione; è talmente importante che spesso è l'unico tra i dispositivi ad essere alimentato direttamente dalla rete per scongiurarne lo spegnimento.

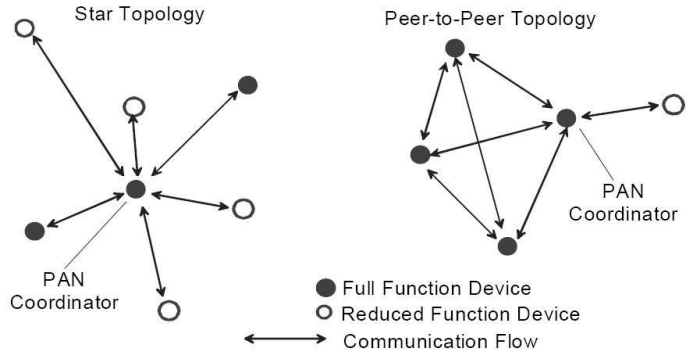


Figura 1.1: due topologie di rete

La pear-to-pear, che pure può disporre di nodi coordinatori e quindi di maggiore importanza rispetto agli altri, si caratterizza per il fatto che ogni dispositivo può instaurare un collegamento con ogni altro che rientri entro la portata della trasmissione. Questo permette la formazione di reti anche molto complesse da implementare ma è l'ideale per alcune applicazioni industriali di controllo, di monitoraggio e di tracking. Con queste potenzialità è possibile effettuare trasmissioni multi-hops per instradare messaggi anche ai nodi più remoti sebbene le regole necessarie per farlo non siano proprie del protocollo.

Architettura

Definiamo tre layer, ciascuno dei quali offre delle funzionalità a quelli più alti. Il più esterno è quello fisico (PHY) che è responsabile:

- dell'attivazione e disattivazione del trasmettitore radio
- del calcolo dell'LQI sui dati ricevuti
- della ricezione, la trasmissione e l'impostazione della radio.

Le frequenze d'utilizzo dipendono da paese a paese e sono licenziate tramite misure di legge: se da un lato può sembrare una garanzia, dall'altro può risultare un problema nel momento in cui un apparecchio prodotto o utilizzato in America, viene importato o utilizzato in Europa.

Sono possibili tre data-rate: 250 Kbps, 40 Kbps, 20 Kbps e si possono allocare 16 canali sulla banda ISM (Industrial Standard Medical) a 2450 MHz, 10 sulla 915MHz e uno a 868 MHz.

Frequency band	Geographical region	Maximum conductive power/ radiated field limit	Regulatory document
2400 MHz	Japan	10 mW/MHz	ARIB STD-T66 [B14]
	Europe (except Spain and France)	100 mW EIRP or 10 mW/MHz peak power density	ETSI EN 300 328
	United States	1000 mW	Section 15.247 of FCC CFR47 [B14]
	Canada	1000 mW (with some limitations on installation location)	GL-36 [B15]
902–928 MHz	United States	1000 mW	Section 15.247 of FCC CFR47 [B14]
868 MHz	Europe	25 mW	ETSI EN 300 220 [B10]

Tabella 1.1: Bande di frequenze libere (ISM) e relative potenze di trasmissione ammesse

Il secondo livello, chiamato MAC (medium access control), garantisce la ricezione delle unità di dati del protocollo MAC attraverso il servizio dei dati del PHY. Per questo arriva anch'esso a impostare e lavorare sui parametri della radio:

- gestisce la sincronizzazione e la sicurezza dei collegamenti
- utilizza il CSMA-CA per l'accesso ai canali.

L'ultimo strato è composto dal *network layer*, che fornisce la configurazione di rete e il routing dei messaggi, e dall'*application layer*, che gestisce le funzioni proprie del dispositivo.

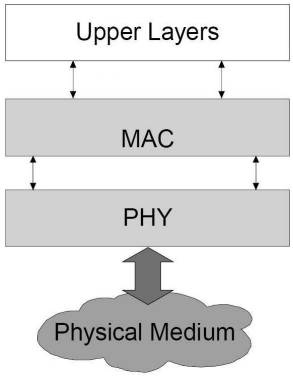


Figura 1.2: architetture dell'802.15.4

Capitolo 2

Analisi dell'apparato

2.1 Tmote Sky

I mote utilizzati nell'esperimento, a cui faremo riferimento nel testo, sono i Tmote Sky prodotti della MoteIv corporation (www.moteiv.com); sono dispositivi a bassissima potenza dotati di diversi sensori ambientali.

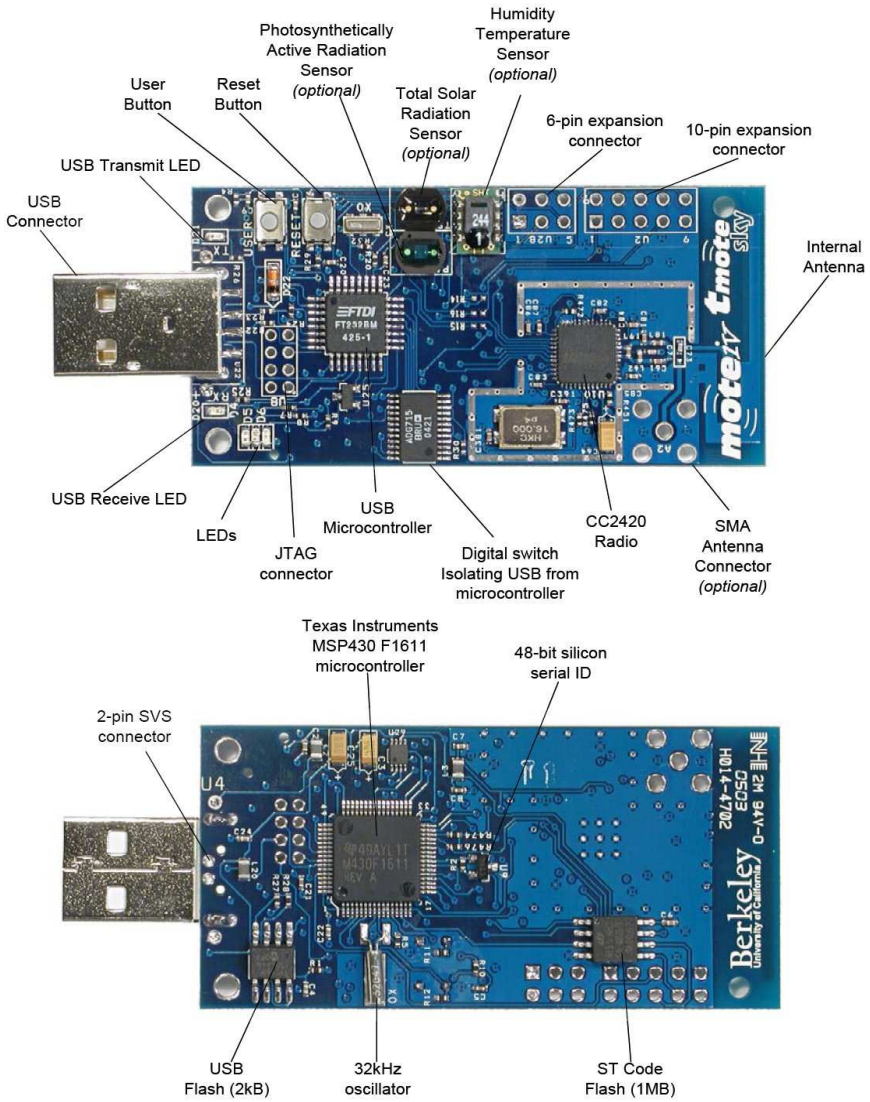


Figura 2.1: le due faccie del Tmote-Sky

Vediamo nei particolari di cosa è composto:

Microprocessore

Prodotto dalla Texas Instruments, l'MSP430 è caratterizzato da 10kB di RAM, 48kB di flash ROM e un'architettura 16bit RISC. Ha un oscillatore (DCO) che opera sopra gli 8 MHz e un oscillatore al quarzo utile alla calibrazione del primo. E' dotato ancora di 8 porte ADC interne e altrettante esterne, necessarie per leggere i valori dei sensori o dei livelli di batterie.

Radio

Tmote utilizza il Chipcon CC2420; fornisce tutto il necessario per comunicazioni wireless affidabili basate sullo standard IEEE 802.15.4. Viene controllata dall'MSP430 tramite il bus SPI e possono essere programmati i parametri fondamentali per la trasmissione: potenza, frequenza e gruppo. La radio lavora infatti attorno ai 2.4GHz con una modulazione O-QPSK, ma è possibile selezionare tra 16 canali differenti numerati da 11 a 26, con i quali ci si può spostare dalla frequenza base con step di 5 MHz per canale. Per quanto riguarda la potenza in trasmissione, sono disponibili 32 diversi livelli, quantizzati in 8 possibili valori elencati di seguito dal più elevato (31) al più debole (3).

PA_LEVEL	TXCTRL register	Output Power [dBm]	Current Consumption [mA]
31	0xA0FF	0	17.4
27	0xA0FB	-1	16.5
23	0xA0F7	-3	15.2
19	0xA0F3	-5	13.9
15	0xA0EF	-7	12.5
11	0xA0EB	-10	11.2
7	0xA0E7	-15	9.9
3	0xA0E3	-25	8.5

Tabella 2.1: livelli di potenza disponibili per la trasmissione e relativi consumi

Questo integrato fornisce ancora un indice di robustezza del segnale ricevuto (RSSI) e un indicatore di qualità della connessione (LQI).

Interfaccia USB

Tmote Sky utilizza un USB controller per gestire la comunicazione seriale col PC. Una volta inserito sulla relativa borchia verrà visualizzato come dispositivo nelle macchine Linux o OSX, e come COM in Windows; gli verrà inoltre assegnato un indirizzo identificativo univoco.
es. col comando "motelist" si verifica quali dispositivi sono collegati al computer

> motelist
Reference Device Description
---------- ---------- ----------
NAV00065 /dev/ttyUSB0 DEI NAVLAB mote
NAV00063 /dev/ttyUSB1 DEI NAVLAB mote

in questo caso siamo in ambiente Linux e abbiamo due Tmote collegati: al primo viene assegnato l'identificativo /dev/ttyUSB0, al secondo /dev/ttyUSB1; la prima colonna indica invece il codice che identifica il singolo nodo.

Antenna

E' interna, posta parallelamente alla superficie della piastra, ma esiste la possibilità di montarne di esterne tramite il connettore apposito. Ha un range di copertura di 50 metri indoor e oltre 125 metri outdoor ma non è perfettamente omnidirezionale e l'irragiamento può risentire della presenza o meno delle batterie e persino della posizione del mote (si ha un incremento delle prestazioni posizionandolo in verticale piuttosto che disteso).

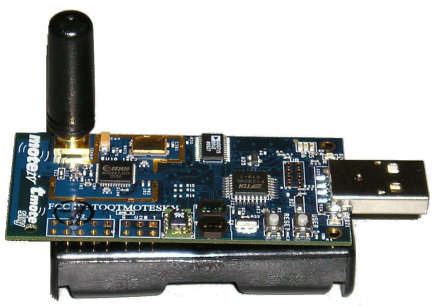


Figura 2.3: mote con antenna esterna

Flash esterna

L'ST M25P80 40MHz fornisce 1024kB per memorizzare dati: è composta di 16 segmenti di 64kB l'uno ed è collegata al microcontrollore tramite l'SPI, lo stesso bus che collegava calcolatore e radio. Questa condivisione delle risorse richiede un minimo di gestione; infatti lo stesso mezzo può essere utilizzato da un unico componente alla volta e non è possibile, per esempio, scrivere nella flash e ascoltare la radio contemporaneamente.

Sensori

Come già detto, c'è la possibilità di integrare questi mote con dei sensori ambientali: Sensirion AG SHT11 o SHT15; tra i due la differenza principale è l'accuratezza e, per entrambi la misura, ottenuta per conversione dall'ADC, è a 14 bit.

L'MSP430 dispone comunque, anche in assenza dei sensori suddetti, di un indicatore del livello delle batterie e uno per la temperatura interna del microcontrollore; entrambi non sono molto precisi e richiedono di essere calibrati.

Connettori per l'espansione

Sono utili per controllare, tramite Tmote, altri dispositivi (per esempio relè, display LCD e altre periferiche digitali); ce ne sono due, uno a 6 e uno a 10 pin.

Considerando l'alimentazione a batterie, è d'obbligo tenere in considerazione il livello minimo per il quale il dispositivo funziona correttamente. Tale livello non è lo stesso per tutti i componenti:

	MIN	MAX	Unità
Tensione al microcontrollore durante l'esecuzione del programma	1.8	3.6	V
Tensione al microcontrollore durante la programmazione della flash	2.7	3.6	V
Tensione di alimentazione radio	2.1	3.6	V

Tabella 2.2: range di tensione

2.2 sistema operativo: TintOS 2.2

La descrizione appena fatta dell'hardware dei Tmote Sky, giustifica la ricerca di un sistema operativo di piccole dimensioni, in grado di trovare posto nella piccola memoria a disposizione. Si tenga presente, infatti, che oltre al sistema operativo e al software con cui saranno programmati, verranno temporaneamente salvati in RAM anche i dati ascoltati via radio o reperiti dai sensori eventualmente fino alla loro memorizzazione in flash. La capacità di calcolo del MSP430 non è elevata; ciò è accettabile, poiché in una rete di sensori l'elaborazione dati spetta a un calcolatore remoto. Sarà necessario quindi che i mote siano precisi e veloci nella gestione degli eventi (è necessario per esempio gestire più rapidamente possibile l'arrivo di un pacchetto dalla radio prima di ricevere il successivo).

TinyOS è il sistema operativo che fa al caso nostro: appositamente realizzato per le reti di sensori, è caratterizzato da una architettura a moduli che sono caricabili singolarmente. Così il programmatore utilizza solo quelli realmente necessari alla sua applicazione, come per esempio quello per la radio o per la scrittura in Flash, snellendo la dimensione dell'applicativo da memorizzare in RAM. Ogni modulo fornito implementa o fornisce interfacce software in grado anche di gestire la risposta agli eventi. In questa trattazione si farà riferimento alla versione TinyOS 2.0 uscita nel settembre 2006, che rispetto alla precedente risulta più versatile nella gestione della memoria e migliorata per quanto concerne il Resource Arbitration (il protocollo per l’utilizzo dei componenti interni).

2.3 Linguaggio di programmazione: nesC

Non esiste una vera e propria distinzione tra sistema operativo e applicativo: nesC, di fatto, è una estensione del C basato sulla guida ad eventi, appositamente pensato per realizzare i concetti e le strutture di esecuzione del TinyOS. La completa concatenazione di sistema operativo e applicazioni rende impossibile caricare più programmi sullo stesso mote tanto meno modificarli in maniera dinamica.

Caratteristiche principali di questo linguaggio sono:

- Separazione del costrutto nesC e dei moduli propri del sistema operativo: un generico programma è costituito di un file chiamato configurazione e di uno detto modulo; il primo richiama i componenti che forniscono le interfacce e definisce la connessione dei vari componenti (wiring) mentre il secondo contiene il codice da eseguire.
- Ciascun componente, attraverso delle interfacce, esercita le sue funzionalità mettendo a disposizione dei *comandi* a ciascuno dei quali fa corrispondere degli *eventi*. L'utilizzatore lavora attraverso i primi e gestisce il flusso del programma tramite il verificarsi degli eventi. La chiamata a queste azioni e la gestione degli avvenimenti viene implementata nel file di modulo. Un semplice esempio è quello del componente che gestisce i timer: mette a disposizione dei comandi, coi quali si sceglie di far partire il clock (in maniera ciclica o meno e impostandone i tempi), e un evento che ne segnala le scadenze.

File di configurazione (fornisce l'interfaccia)

```
configuration TimerMilliP {
  provides interface Timer<TMilli>;
}
…
```

Definizione dell'interfaccia

```
interface Timer<precision_tag>
{
  command                    void
startPeriodic(uint32_t dt);
  command                    void
startOneShot(uint32_t dt);
  event void fired();
  …
}
```

capitolo 3

Studio del problema

Nodo cruciale in una trasmissione wireless è la distanza che questa deve coprire e le attenuazioni dovute all'ambiente circostante. Per rilevare le variazioni delle caratteristiche della rete si è pensato di disporre quanti più nodi possibili e interrogarli ciclicamente per un lungo periodo di tempo.

Come già accennato nel sommario, al termine di questo studio risulta interessante individuare alcuni intervalli, significativi dal punto di vista statistico, all'interno dei quali ha senso considerare possibile e robusta la connessione in termini di distanza e qualità.

3.1 Trasmissione delle onde radio

A seconda della frequenza d'utilizzo, l'onda elettromagnetica assume diversi nomi e caratteristiche:

- Onde Radio: raggiungono al massimo le centinaia di kHz; sono in grado di passare attraverso gli edifici percorrendo quindi lunghe distanza grazie alla ionosfera che le riflette quasi totalmente verso il basso;
- Dalle Microonde fino alle frequenze più elevate, le onde sono estremamente direzionali e col crescere della frequenza riescono sempre meno a oltrepassare gli ostacoli (anche quelli di piccole dimensioni; già la pioggia è in grado di assorbire frequenze dell'ordine di grandezza di 10^{11}).

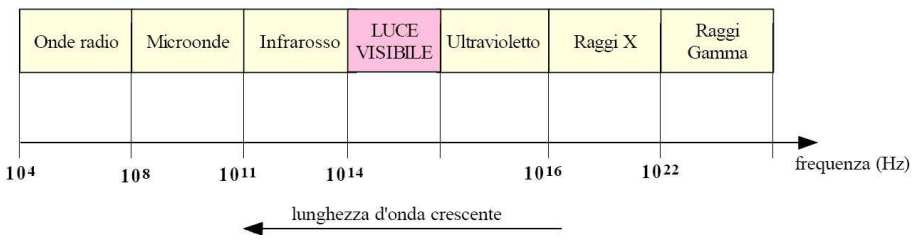


Figura 3.1: panoramica dello spettro elettromagnetico

In generale, la trasmissione delle onde magnetiche secondo direzioni prestabilite o uniformi è di difficile realizzazione a causa di tre fenomeni: riflessione**, diffrazione e scattering**. Ciascuno di questi si verifica in misura direttamente o inversamente proporzionale al rapporto tra la lunghezza d'onda del campo elettromagnetico e le dimensioni dell'oggetto su cui lo stesso collide. Purtroppo si lavora a frequenze vicine ai 2.4GHz che significano lunghezze d'onda dell'ordine dei decimi di metro, dimensioni compatibili con la maggior parte degli oggetti di uso comune comportando l'insorgere di tutti e tre i diversi disturbi introdotti precedentemente. Se l'ostacolo su cui urta il segnale ha *dimensioni maggiori* rispetto alla lunghezza d'onda, si verifica *riflessione*: viene rimbalzato con un angolo pari all'opposto di quello di incidenza.

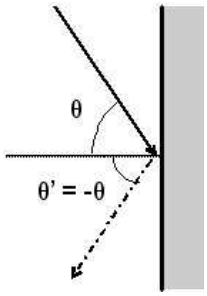


Figura 3.2: rappresentazione schematica della riflessione

La *diffrazione*, invece, si manifesta se un'onda incontra un ostacolo le cui *dimensioni sono comparabili o minori* rispetto alla propria lunghezza d'onda. Nel caso in cui questa colpisca una fenditura tra due corpi impenetrabili, dopo averla attraversata, si crea una superficie di onde semi-circolari, all'incirca aventi la stessa intensità della prima, che si propagano in ogni direzione.

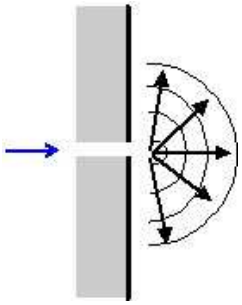


Figura 3.3: rappresentazione schematica della diffrazione

Lo *scattering* si verifica anch'esso in presenza di corpi di *dimensioni comparabili* con la lunghezza d'onda del campo. Sono fenomeni di questo tipo i cambi di direzione del campo: quando un'onda colpisce anche solo un corpuscolo, trasferisce parte della sua energia a questo che la diffonde in nuove direzioni, un po' come avviene nella teoria degli urti.

A causa dei tre disturbi descritti e in generale per l'inevitabile dispersione nello spazio, le onde, sebbene trasmesse dalla stessa sorgente, possono percorrere tragitti differenti e impiegare tempi significativamente diversi per arrivare allo stesso ricevitore introducendo degli sfasamenti. Se la differenza di fase tra le onde ricevute si avvicina ai 180°, la somma è distruttiva e i dati ricevuti vengono compromessi, ma proprio in virtù di questo sfasamento, la radio potrebbe interpretarli come due segnali diversi. Questo effetto è noto con il nome di **multi-path fading** (attenuazione da cammini multipli): il segnale originale e quello riflesso arrivano al ricevitore più o meno contemporaneamente poiché entrambi viaggiano alla velocità della luce, col risultato che il ricevitore non riesce a distinguerli e interpretarli e li considera differenti e incorrelati.

Il problema può essere attenuato utilizzando due antenne per lo stesso ricevitore posizionate opportunamente, vista la scarsa probabilità che il multi-path affligga entrambe nello stesso modo (tecnica della *diversità nello spazio*) oppure mantenere alcuni canali di riserva su cui spostare la comunicazione nel caso in cui il disturbo cancellasse la banda di utilizzo (*diversità di frequenza*). Sono entrambe soluzioni poco realizzabili quanto concerne questo esperimento: la prima risulta dispendiosa di mezzi visto che si desidera contenere le dimensioni dei dispositivi e la facilità di installazione, la seconda di difficile implementazione poiché si cercava un algoritmo veloce è rappresentativo: non tutte le applicazioni che utilizzeranno l'802.15.4 gestiranno questo tipo di disturbo.

È risultato comodo e veloce spostare l'intera comunicazione su un preciso canale, utilizzato nell'edificio solo per l'esperimento, così da eliminare l'eventualità di disturbi grossolani.

3.2 Ipotesi di lavoro

La maggior parte delle scelte implementative è stata condizionata dalla quantità di dati memorizzabili in flash. La dimensione della memoria è 1024 kB, ma non tutta è resa disponibile all'utilizzatore. L'ultimo segmento (64kB) contiene la "*golden image*", un insieme di dati forniti dal produttore che racchiudono tutto il necessario per riprogrammare il mote e riportarlo in "buono stato" in caso di malfunzionamento.

Sector	Address range	
15	F0000h	FFFFFh
14	E0000h	EFFFFh
13	D0000h	DFFFFh
12	C0000h	CFFFFh
11	B0000h	BFFFFh
10	A0000h	AFFFFh
9	90000h	9FFFFh
8	80000h	8FFFFh
7	70000h	7FFFFh
6	60000h	6FFFFh
5	50000h	5FFFFh
4	40000h	4FFFFh
3	30000h	3FFFFh
2	20000h	2FFFFh
1	10000h	1FFFFh
0	00000h	0FFFFh

Tabella 3.1: organizzazione della memoria

Questi 64kB possono essere modificati solo se il dispositivo è collegato all'USB, il che non risulta possibile durante la raccolta dati. L'indirizzo di partenza da cui l'applicativo inizia a scrivere in memoria corrisponde allo 00000 in esadecimale.
Si hanno a disposizione:

$$65536B/settore \cdot 15settore = 983040B .$$

L'idea di partenza è quella di far spedire ad ogni mote 80 pacchetti più volte nell'arco dell'intera giornata. Le informazioni da memorizzare sono

- Numero identificativo del ciclo di raccolta dati per risalire al momento della giornata,
- Livello della batteria del trasmettitore,
- Livello della batteria del ricevitore,
- Potenza della trasmissione,

- LQI,
- RSSI,
- Indirizzo del mittente,
- Indirizzo del destinatario.

Cercando di compattare i dati, si è riusciti a memorizzarli ognuno in un byte: il pacchetto completo si compone di 9byte, per 80 pacchetti da spedire alla massima potenza, significa 720byte a mote, per ogni rilevazione.

$$720B \cdot numeroNodi \cdot cicliRilevazione = 983040B$$

Se volessimo controllare il comportamento di 50 dispositivi, riusciremmo a fare 27 rilevazioni e ipotizzando un intervallo di 10 minuti tra l'una e l'altra si riuscirebbe a testare la rete per un periodo di 4 ore e 40 minuti circa.

Volendo sfruttare come prima prova il piano terra del DEI-A che presenta una superficie di 540 m^2 all'incirca, si è visto che una ventina di nodi potevano essere sufficienti, con un rapporto di quasi un mote per 25m^2.

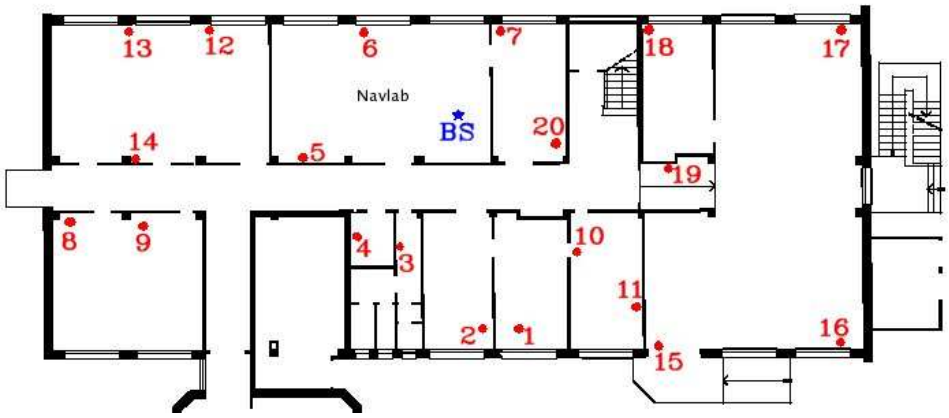


Figura 3.4: planimetria e posizione dei mote

Con questi dati si poteva assicurare un funzionamento ininterrotto di circa 11 ore con interrogazioni cicliche ripetute ogni 10 minuti.

Proprio per le dimensioni dell'ambiente considerato e per la sua struttura (muri, armadi e apparecchiature elettriche) si ipotizza che due mote posizionati su due laboratori differenti e opposti sul lato lungo del piano, non siano in grado di costruire un link tra loro sebbene usando il livello più elevato di potenza disponibile. Se ciascuna radio non riuscirà a captare le trasmissioni di tutte le altre, lo spazio necessario a memorizzare le informazioni sarà molto inferiore a quello ipotizzato, ma non conoscendo la misura in cui un nodo è "isolato" dagli altri, non possiamo sbilanciarci e

lasciare la rete connessa per più del tempo previsto nel caso peggiore (tutti ricevono tutto).

A questo punto nasce anche il problema della sincronizzazione: sebbene Tmote-Sky disponga di un oscillatore al quarzo e fornisca quindi degli strumenti di tipo “timer”, la loro stabilità non è del tutto affidabile, soprattutto se si pensa di lasciarli accesi per una decina di ore senza mai controllarne le derive, che a fine giornata potrebbero risultare discordanti soprattutto a causa dell’interferenza della temperatura.

Si è pensato di inserire un **flooding**: alcuni eventi vengono segnalati da un nodo principale tramite l’invio di pacchetti speciali che vengono ripetuti da ogni altro per assicurarsi che possano arrivare anche agli angoli più remoti del locale. Questa risulta essere l’unica trasmissione *multi-hops* dell’esperimento, necessaria affinché tutti i mote, all’incirca nello stesso istante, siano informati della situazione corrente.

L’idea generale è che ogni nodo inoltri i suoi 80 pacchetti senza indicarne il destinatario e chiunque li riceva possa salvarli in memoria così da ottenere la lista completa di tutti i potenziali link realizzabili tra i vari punti su cui abbiamo posizionato i sensori.

3.3 Alimentazione

Come ripetuto più volte, un aspetto fondamentale per il buon funzionamento del mote è il tipo di alimentazione: collegato all’USB, il dispositivo, indifferentemente dalla presenza o meno delle batterie, dispone di 3V che risultano costanti se non è prevista l’accensione dei LED o di altri integrati con consumi elevati, nel qual caso è inavitabile registrare temporanei cali di tensione. Nel caso si colleghi il battery pack (due stilo AA da 1.5 V cadauna), come spiegato in [2.1], tutto diventa dipendente dal livello di carica. Esistono vere e proprie teorie sui metodi di risparmio energetico: sapendo gestire in maniera ottimale l’accensione e lo spegnimento degli integrati e lasciando in funzione solo alcuni timer (low power mode), un Tmote-Sky può funzionare benissimo per alcune settimane.

Alcuni test preliminari hanno mostrato che senza portare mai il dispositivo in Power Down e caricandogli lo stesso software utilizzato poi per la raccolta dati, il livello di batteria dopo 12 ore era sceso di 0.2V.

Questo ha portato alla conclusione che utilizzare coppie di batterie nuove poteva essere poco significativo dato che, a fine esperimento, la loro tensione avrebbe raggiunto valori ancora distanti dalla soglia critica. Si è scelto così di utilizzare batterie "rodate" con valori di tensione compresi tra 1.3 e 1.4 volt.

3.4 RSSI

Come riportato in dettaglio in [2] e [3], l'RSSI (Received Signal Strength Indicator) è un valore a 8 bit restituito direttamente dal chip radio CC2420 tramite variabile a registro. È facile ricavare la potenza dell'onda radio in ricezione (RSS) dal valore di RSSI tramite la relazione:

$$RSS = RSSI + offset_{dBm}$$

dove offset viene calcolato in maniera empirica in fase di progettazione e normalmente è vicino al valore -45dBm.

Il valore restituito dalla radio è quantizzato in 100 valori e presenta un'accuratezza di 6 dBm; risulta mappato nel seguente modo:

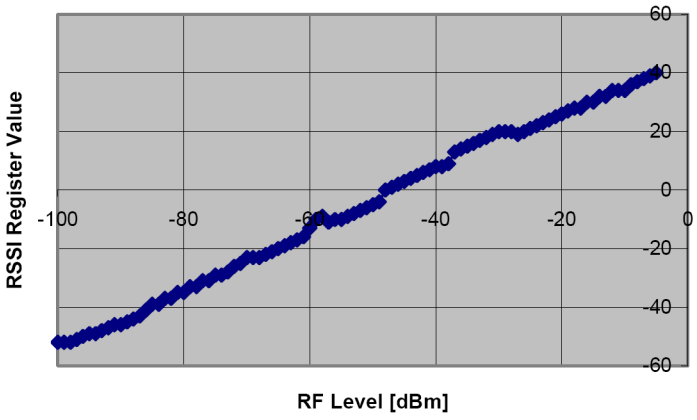


Figura 3.5 : valori tipici di RSSI vs potenza segnale radio

3.5 Modello del Canale

Come già spiegato, durante la propagazione dell'onda elettromagnetica il campo è sottoposto ad attenuazione; questa tende ad aumentare con la distanza per fenomeni

fisici di assorbimento ed in ambiente urbano è appesantita ulteriormente dai fenomeni di riflessione e diffrazione. L'entità dell'attenuazione subita dal segnale trasmesso è un parametro che deve essere attentamente valutato in fase di progetto di un sistema di comunicazione in quanto determina la massima distanza che può essere raggiunta con una data tecnica di trasmissione.

Si è già detto di non trovarsi nelle condizioni di esaminare uno spazio libero; al contrario si ha a che fare con un ambiente molto ostile e risulta necessario, per determinare la relazione tra RSSI e potenza ricevuta, procedere con metodi sperimentali e affidarsi a modelli già utilizzati in letteratura, comunque sviluppati a partire da relazioni empiriche; si rimanda per esempio a [1, 7, 8, 9] .

Normalmente viene utilizzato il seguente modello:

$$P_{RC}\big|_{dBm}(d) = P_{d_o}\big|_{dBm} - 10\gamma\log_{10}(\tfrac{d}{d_0}) \quad (1)$$

$$P_{RC}\big|_{dBm}(d) = P_{d_o}\big|_{dBm} + 10\gamma\log_{10}(d_0) - 10\gamma\log_{10}(d) \quad (2)$$

in cui compaiono:

- P_{RC} : potenza del campo elettromagnetico rilevato al ricevitore;
- P_{d_o} : potenza del campo elettromagnetico a una distanza d_0 dal trasmettitore;
- γ : esponente di perdita di propagazione ossia il fattore di decrescenza della potenza in funzione della distanza dal trasmettitore;
- d_0 : distanza di riferimento dal trasmettitore;
- d : distanza del ricevitore dal trasmettitore.

Utilizzando d_0 = 1 m, la (1) diviene

$$P_{RC}\big|_{dBm}(d) = P_{TX}\big|_{dBm} - P_{d_o}\big|_{dBm} - 10\gamma\log_{10}(d) - \chi_\sigma \quad (3)$$

in cui compare in maniera esplicita la potenza di trasmissione e una variabile aleatoria gussiana a media nulla e varianza σ che tiene conto delle fluttuazioni di potenza dovute all'oscuramento (shadowing) causato dalla presenza di ostacoli che attenuano il segnale radio. Le variazioni dovute a cause esterne al mezzo trasmissivo, come il movimento delle persone, restano escluse da questa modellizzazione e risulta possibile diminuire la loro incidenza solo utilizzando medie temporali della misura della potenza di campo.

Anche l’incidenza delle pareti o degli ostacoli “quasi trasparenti” vengono trascurate per non appesantire il modello: sarebbe necessario uno studio dei materiali trapassati (dal legno, al mattone, al cemento armato) e un’analisi degli ostacoli che ogni link può attraversare. Ci si aspetta quindi da questo modello un andamento indicativo e di confronto per i dati che andremo a raccogliere.

capitolo 4

Algoritmo sviluppato

Prima di tutto definiamo:

- *Ciclo di raccolta dati* (o semplicemente *ciclo*): intervallo all'interno del quale ciascun nodo spedisce le informazioni relative alla propria trasmissione; si ripete ogni 10 minuti fino al termine dell'esperimento;
- *Intervallo di chiamata*: sottoperiodo nel quale è suddiviso il ciclo sopra descritto; ve ne sono tanti quanti sono i mote utilizzati più uno indispensabile per la sincronizzazione.

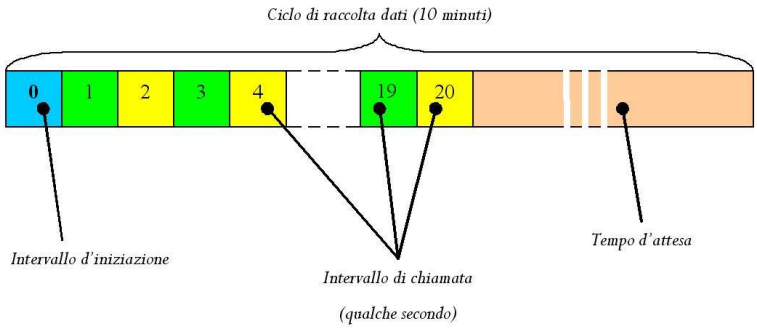


Figura 4.1: rappresentazione del ciclo di raccolta dati

Utilizzeremo 20 nodi disseminati per il piano e indicizzati con un numero compreso tra 1 e 20 a ciascuno dei quali corrisponderà l'intervallo identificato con lo stesso valore.

4.1 Chiamata

Risulta necessario l'utilizzo di un nodo supplementare chiamato *Base Station* e programmato per scandire il susseguirsi dell'*intervallo di chiamata* attraverso l'invio di un pacchetto detto appunto "*chiamata*".

```
typedef struct chiamata {
```

```
    nx_uint8_t serialNumber;
    nx_uint8_t code;
    nx_uint16_t idTx;
    …
} chiamata;
```

Il primo campo memorizza il numero sequenziale del ciclo in corso, "*code*" contiene un codice identificativo del tipo di pacchetto e "*idTx*" l'indirizzo del nodo che deve trasmettere in quello spazio temporale. Essendo queste informazioni indispensabili per la riuscita della raccolta dati, e non avendo una copertura completa della rete, è stato implementato il flooding di "*chiamata*": il mote riceve il pacchetto; se è la prima volta che ne riceve uno per il nodo *X* in quel determinato ciclo, lo reinoltra e tiene memoria del passaparola effettuato; altrimenti, se riceve altre chiamate a quell'indirizzo (a causa del flooding effettuato anche dagli altri sensori), scarta il pacchetto e ritorna ad ascoltare la radio. Nell'esempio in *Figura 4.2* si nota come la chiamata al *mote 2* spedita dalla Base Station può arrivare a destinazione solo se fatta rimbalzare dal più vicino *mote 1*.

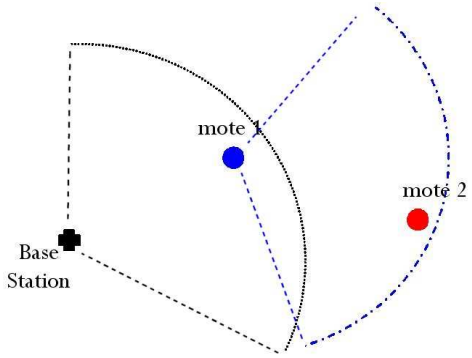


Figura 4.2: esempio di flooding

Ecco un esempio di dati spediti dalla Base Station:

	serialNumber	Code	idTx
1	1	38	0
2	1	38	1
3	1	38	2
4	…	…	…

Tabelle 4.1: esempio di chiamate effettuate dalla Base Station

quello con *idTX* a valore nullo serve a preparare la rete a un nuovo ciclo; quando arriva, ogni dispositivo inizializza alcune variabili e predispone la radio all'ascolto.

4.2 Trasmissione e ascolto

Quando a un mote arriva un pacchetto "*chiamata*", va a leggerne l'indirizzo contenuto così da identificare l'intervallo attivo in quell'istante e controlla se sia il suo turno. In caso affermativo, non effettua il flooding del pacchetto e inizia a trasmettere subito la serie di messaggi. Nel frattempo tutti gli altri mote hanno già concluso l'eventuale reinoltro della chiamata e si trovano pronti ad ascoltare la radio. Il pacchetto dei dati trasmessi è così costituito:

```
typedef struct BlinkToRadioMsg {
  nx_uint8_t serialNumber;
  nx_uint8_t code;
  nx_uint16_t battery;
  nx_uint8_t pTx;
} BlinkToRadioMsg;
```

Il primo campo è il numero sequenziale del ciclo in corso, "*code*" un codice che identifica il tipo di pacchetto, "*battery*" è il livello di batteria del trasmettitore, "*pTx*" la potenza a cui è settata la radio.

I destinatari, nella fattispecie chiunque riceva il segnare radio, memorizzano temporaneamente i dati in RAM fintantoché viene segnalata la fine della sequenza dal trasmettitore con un secondo pacchetto speciale detto "*txTerminata*"

```
typedef struct txTerminata {
  nx_uint8_t serialNumber;
  nx_uint8_t code;
  nx_uint16_t mitt;
  …
} txTerminata;
```

del tutto simile a “*chiamata*” con la differenza che ora viene segnalato, tramite il campo “*mitt*”, l’indirizzo del nodo che ha terminato la trasmissione. Quando un mote lo riceve inizia la procedura di memorizzazione in flash. Risulta più comodo salvare l’intero blocco di messaggi ricevuti piuttosto che uno alla volta poiché l’operazione di scrittura, divisa in accesso, scrittura e verifica, richiede tempi non trascurabili dell’ordine anche delle decine di millisecondi durante i quali il dispositivo non può ascoltare la radio aumentando il rischio di perdere dati.

Come per l’altro pacchetto speciale, anche in questo caso è necessario il flooding: se il generico nodo non dovesse riceverlo, al ciclo successivo andrebbe a sovrascrivere i dati a quelli precedenti senza prima averli salvati.

4.3 Lettura dei dati memorizzati

Come spiegato al paragrafo [2.3] risulta possibile caricare un solo applicativo alla volta sul Tmote-Sky. Per leggere la flash si è pensato di installare, al termine della rilevazione, un secondo software che leggesse riga per riga la memoria e la spedisse all’interfaccia seriale. Collegando il mote al PC e sfruttando una classe java implementata dai sviluppatori del TinyOS per l’ascolto del traffico dati sulla COM, si è potuto indirizzare su file il flusso di dati e, vista la formattazione e la scrittura in esadecimale, collezionarli in formato testo.

Questa realizzazione è stata possibile perché la flash, a meno di fortissimi campi elettromagnetici inesistenti in condizioni di normale utilizzo, viene formattata e cancellata solo con comandi precisi, proprio come l’hard-disk di un computer, mantenendo il proprio stato nel tempo.

capitolo 5

Analisi dei dati ottenuti

I dispositivi sono rimasti accesi, sparsi per il piano, per quasi nove ore consecutive, dalle 10 del mattino alle 18.40 circa.

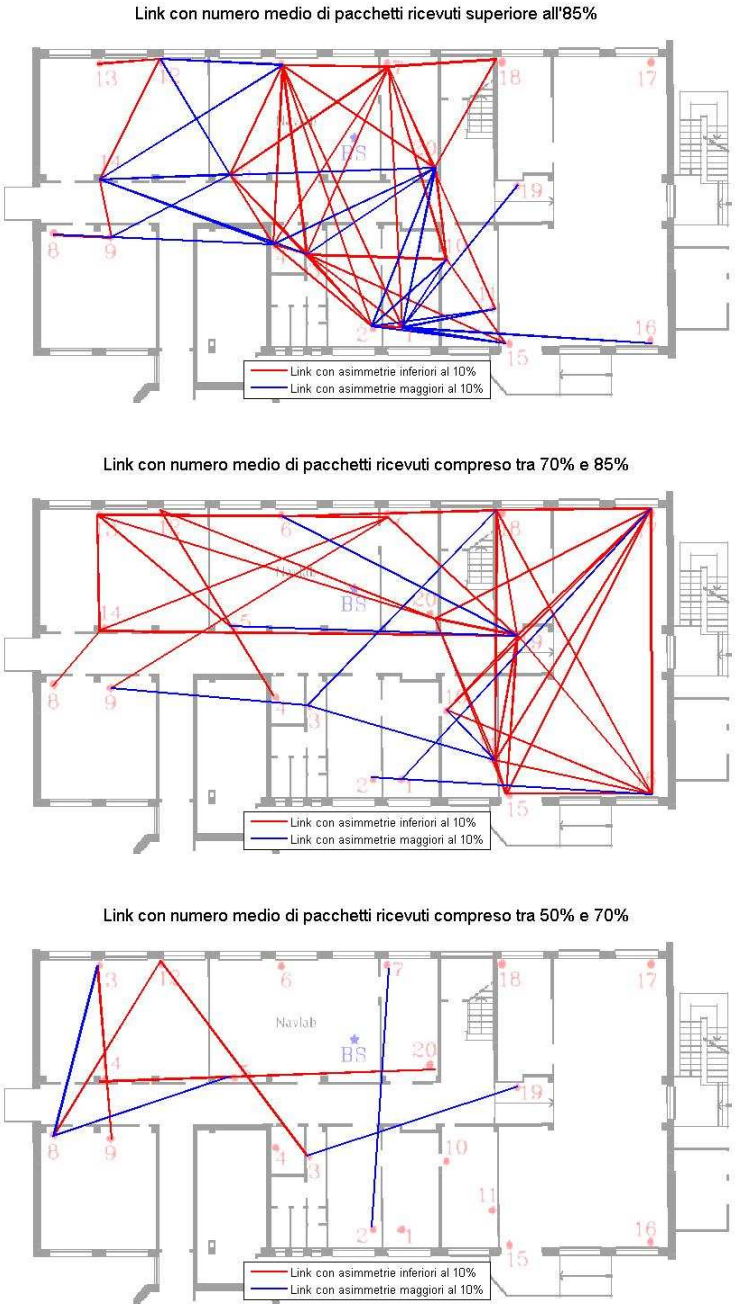


Figura 5.1: connettività media dell'intera giornata e misura delle asimmetrie dei link

Verificando i dati raccolti, si nota che i nodi che hanno ricevuto più del 50% dei messaggi trasmessi in etere, sono tutti posizionati nell'area centrale del piano che risulta essere anche densamente più ricca di dispositivi.

I grafi rappresentano tutti i collegamenti instauratisi all'interno di tre diversi intervalli di successo evidenziando con colori differenti le due fasce di tolleranza dell'asimmetria: in rosso i link maggiormente significativi e in blu quelli non del tutto biunivoci. Facciamo alcuni esempi: nel caso del grafo che considera percentuali di successo superiori all'85%, la comunicazione tra il mote 12 e il 14 risulta bilanciata, al contrario di quella tra il 6 e il 14: quest'ultimo ha inviato almeno 300 pacchetti in più rispetto al suo duale. Analizzando i dati di tutte le connessioni instauratesi, l'asimmetria media si aggira attorno a 250 pacchetti, con alcuni picchi che arrivano a 1200 pacchetti di differenza sullo stesso percorso. Le differenze maggiori si misurano sulle lunghe distanze e in presenza di diversi ostacoli, in particolare muri, mentre risultano più omogenee le connessioni all'interno della stessa stanza. La posizione del mote e quindi dell'antenna, sembrano incidere molto sotto questo profilo.

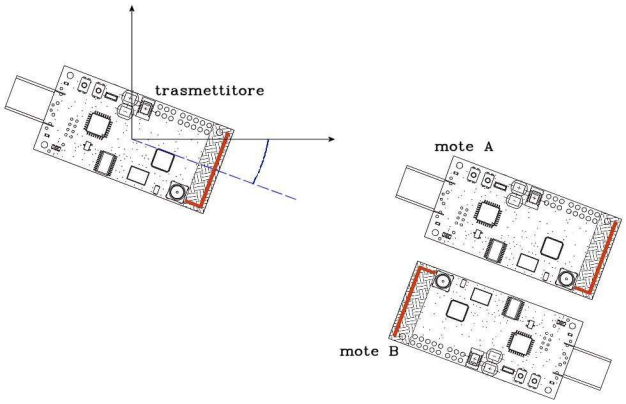


Figura 5.2: esempio di posizionamento di alcuni mote

Nell'esempio qui sopra, se si ipotizzano i dispositivi A e B a una certa distanza dal trasmettitore, questi non riceverebbero gli stessi pacchetti con la stessa qualità. Identica cosa se fossero ruotati di soli 90°. Per disturbare il meno possibile, i mote sono stati spesso posizionati negli angoli delle stanze, o posati su mensole poco utilizzate: le riflessioni e gli spigoli generano fenomeni di multi-path fading e degradano la connessione. Gli stessi problemi sono stati evidenziati da altri studi, si rimanda per esempio a [1].

Con la seguente tabella si vogliono sottolineare le tre aree in cui è possibile dividere il piano:

1. zona ad alta concentrazione di sensori e di connettività (a cui appartiene il mote 5);
2. zona intermedia con buone prestazioni (rappresentata dal mote 18 e dal 12);
3. zona scarsamente connessa e con cattivi risultati (tutta la zona periferica o molto ostruita da ostacoli, in cui rientra il mote 8).

pacchetti ricevuti al mote 5		
mittente	*media nella giornata*	σ
1	15,24	19,76
2	75,86	15,48
3	73,37	18,53
4	74,00	15,10
6	72,08	14,71
7	71,02	14,50
8	39,63	31,26
9	67,31	17,10
10	10,45	19,63
11	1,04	4,03
12	67,33	13,75
13	57,31	23,23
14	66,22	13,94
15	0,39	1,25
18	1,27	6,15
19	63,80	15,42
20	68,47	9,84

pacchetti ricevuti al mote 18		
mittente	*media nella giornata*	σ
1	4,24	11,49
3	71,61	11,21
4	15,14	17,14
5	2,98	7,99
6	66,86	14,03
7	71,16	1,57
10	8,76	18,27
11	67,80	9,75
12	27,71	25,79
15	23,02	26,27
16	66,67	9,64
17	69,67	1,09
19	68,51	9,79
20	69,90	0,30

pacchetti ricevuti al mote 12		
mittente	*media nella giornata*	σ
3	56,4	26,74
4	67,7	17,65
5	76,0	0,35
6	74,7	1,99
7	38,6	27,90
8	52,9	28,82
9	39,8	25,92
13	70,0	0,47
14	70,0	0,28
18	4,5	8,60
20	58,8	20,32

pacchetti ricevuti al mote 8		
mittente	*media nella giornata*	σ
3	2,67	8,22
4	76,82	0,65
5	47,76	28,08
6	1,04	4,49
9	70,63	10,09
12	53,53	24,36
13	46,90	24,09
14	69,80	0,72

Tabella 5.1: medie e deviazioni standard caratteristiche della connettività di alcuni nodi

Questi pochi dati presentati, sono d'aiuto per comprendere meglio l'asimmetria: i nodi considerati qui sopra sono tra loro connessi a due a due, ma non sempre le medie di collegamenti reciproci si avvicinano.

5.1 Continuità della connettività

Si è più volte detto che la connettività dipende da numerosi fattori legati all'ambiente su cui si trova la rete; per dare un'idea della variabilità dei link, si è calcolata la media

del numero di pacchetti ricevuti, non più sull'intera giornata, ma su quattro intervalli temporali di due ore ciascuno.

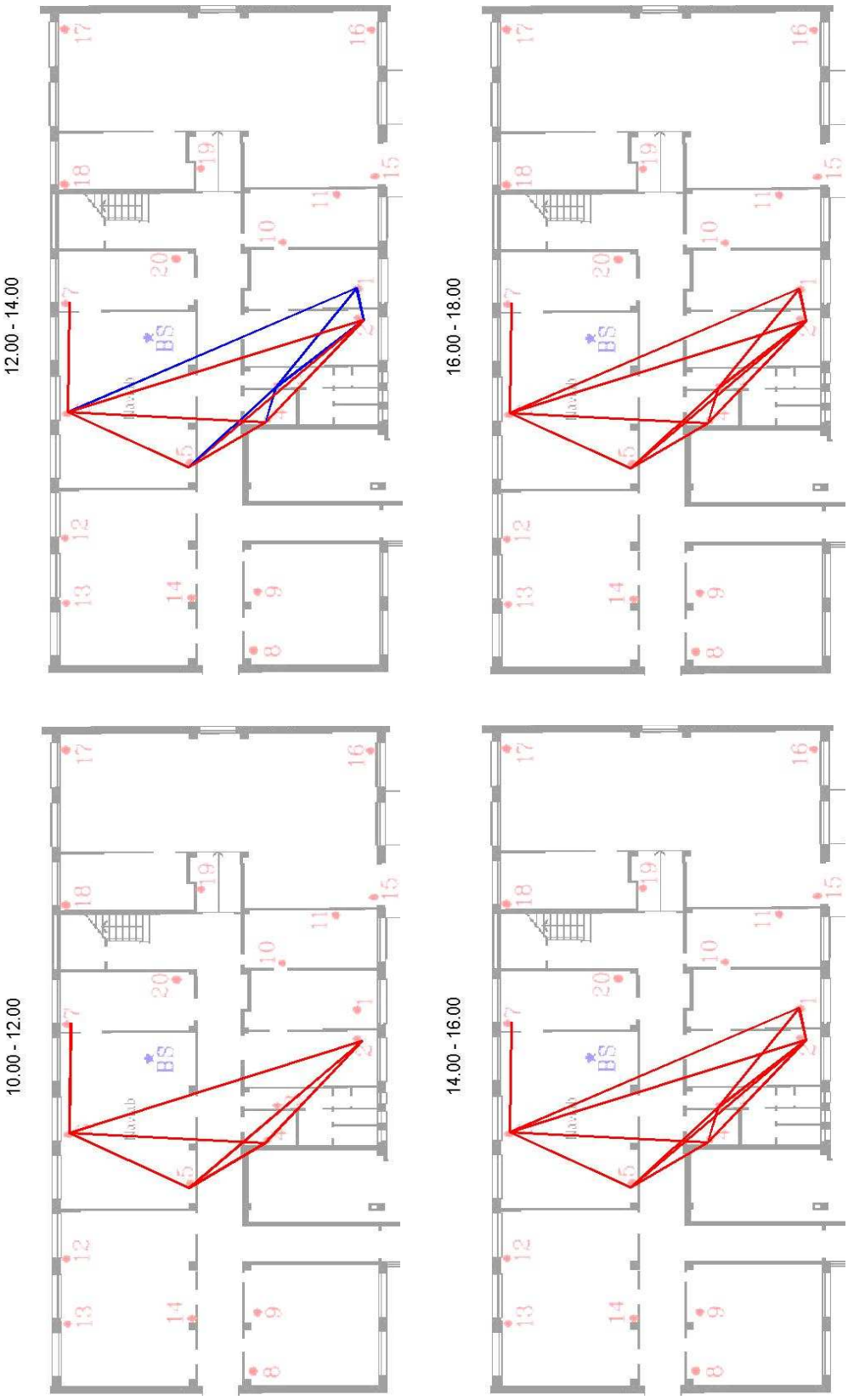
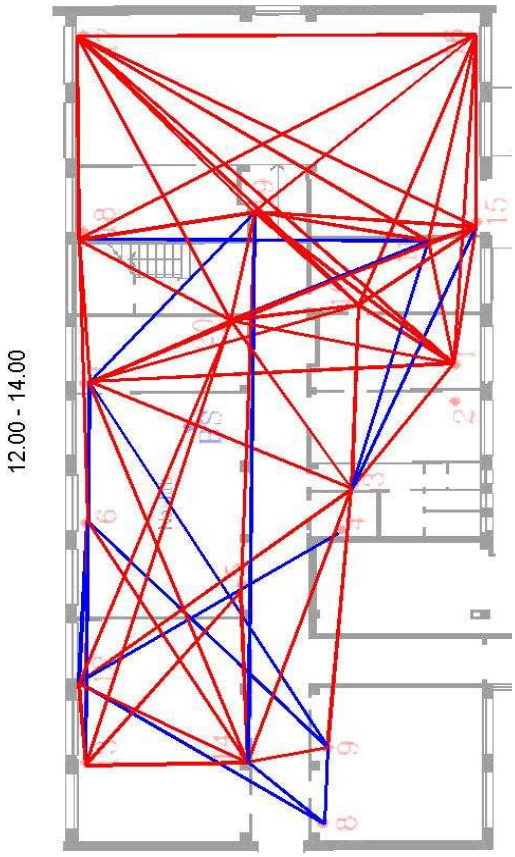
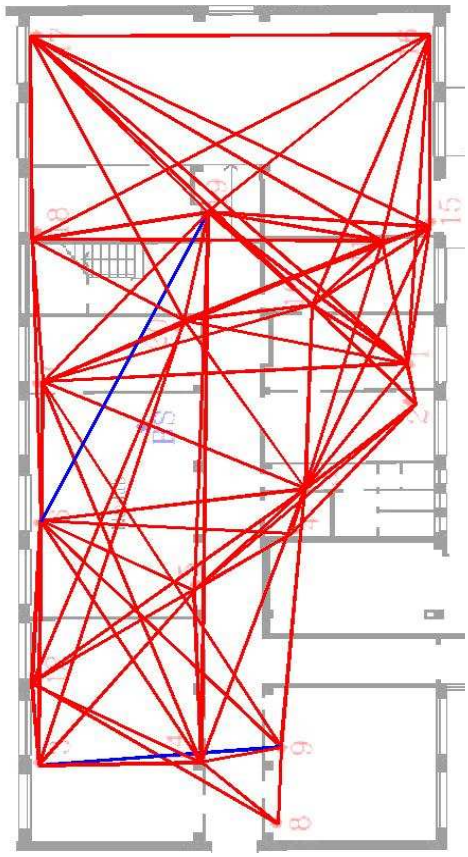


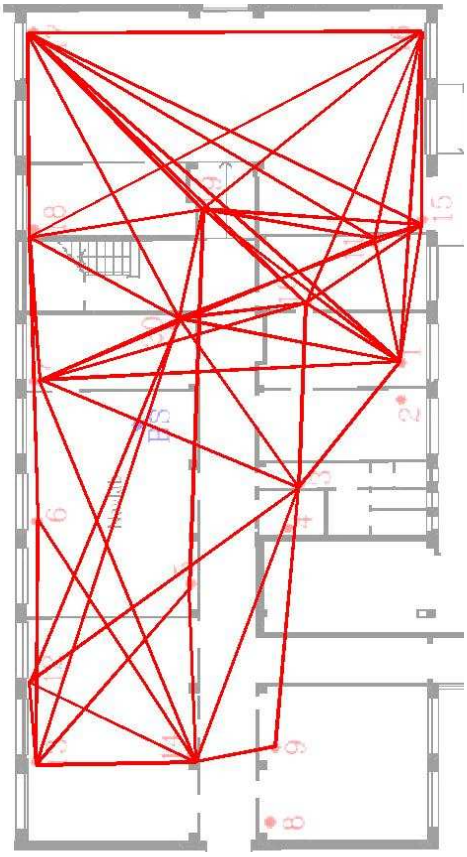
Figura 5.3: variabilità nel tempo dei collegamenti con più dell'85% di pacchetti ricevuti



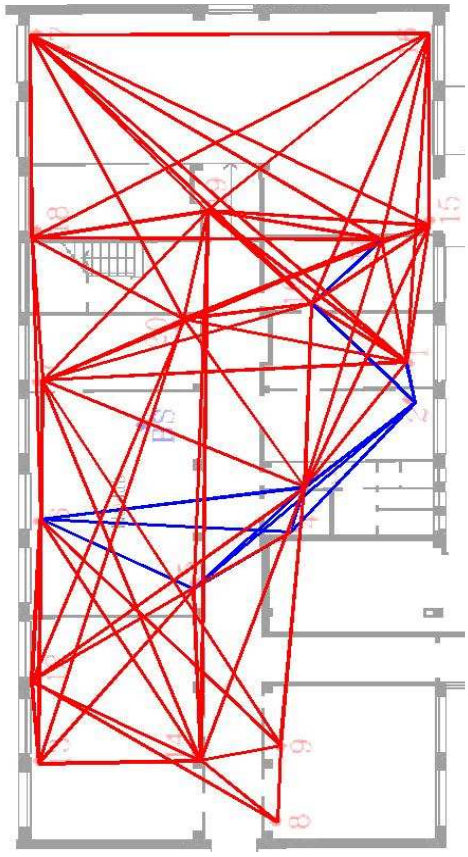
12.00 - 14.00



16.00 - 18.00



10.00 - 12.00



14.00 - 16.00

Figura 5.4: variabilità nel tempo dei collegamenti con pacchetti ricevuti tra 70% e 85%

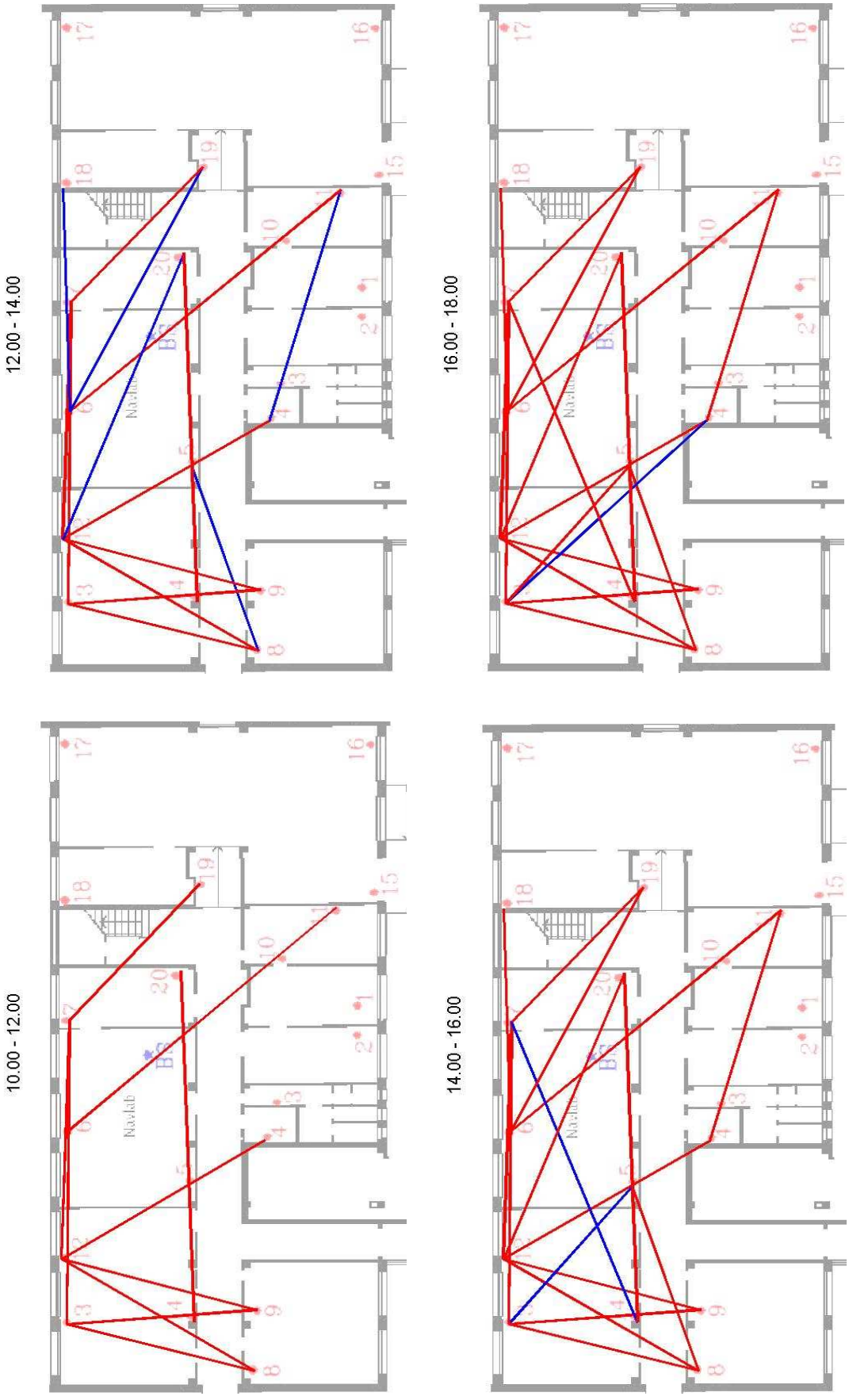


Figura 5.5: variabilità nel tempo dei collegamenti con pacchetti ricevuti tra 50% e 70%

In blu vengono rappresentati i link che nelle due ore precedenti non erano presenti, così da rendere immediata la lettura dei grafi.

In *Figura 5.3* si considerano link con percentuali di pacchetti ricevute superiore all'85%; è ben visibile come nel corso delle prime due ore, rispetto a tutto il resto della giornata, mancano tutti i collegamenti dei nodi 1 e 3. Questo non significa che inizialmente i sensori non funzionassero, ma semplicemente avevano una media di pacchetti ricevuti inferiore al limite preposto. Passato mezzogiorno, la situazione si stabilizza e non si hanno altre forti variazioni. Questa apparente (dico apparente poiché ci si basa su medie temporali) stabilità dipende senza dubbio dall'alta qualità dei collegamenti considerati. Gli stessi concetti non valgono per i grafi di *Figura 5.4* e *Figura 5.5*, caratterizzati da una percentuale di ricezioni corrette rispettivamente comprese tra il 70% e l'85% e tra il 70% e il 50%. Le variazioni sulla connettività sono riconoscibili: nel primo esempio, durante le seconde due ore, aumentando di qualità, divengono visibili tutti i link al nodo 8 e si intensificano quelli che coinvolgono i 9, 3, 15 e 11; nella prima parte del pomeriggio incrementano le connessioni ai nodi 2, 3, 4 e 6 e altre due arrivano nel corso delle ultime ore dell'esperimento. Anche per il terzo caso la variabilità perdura nell'arco dell'intera giornata.

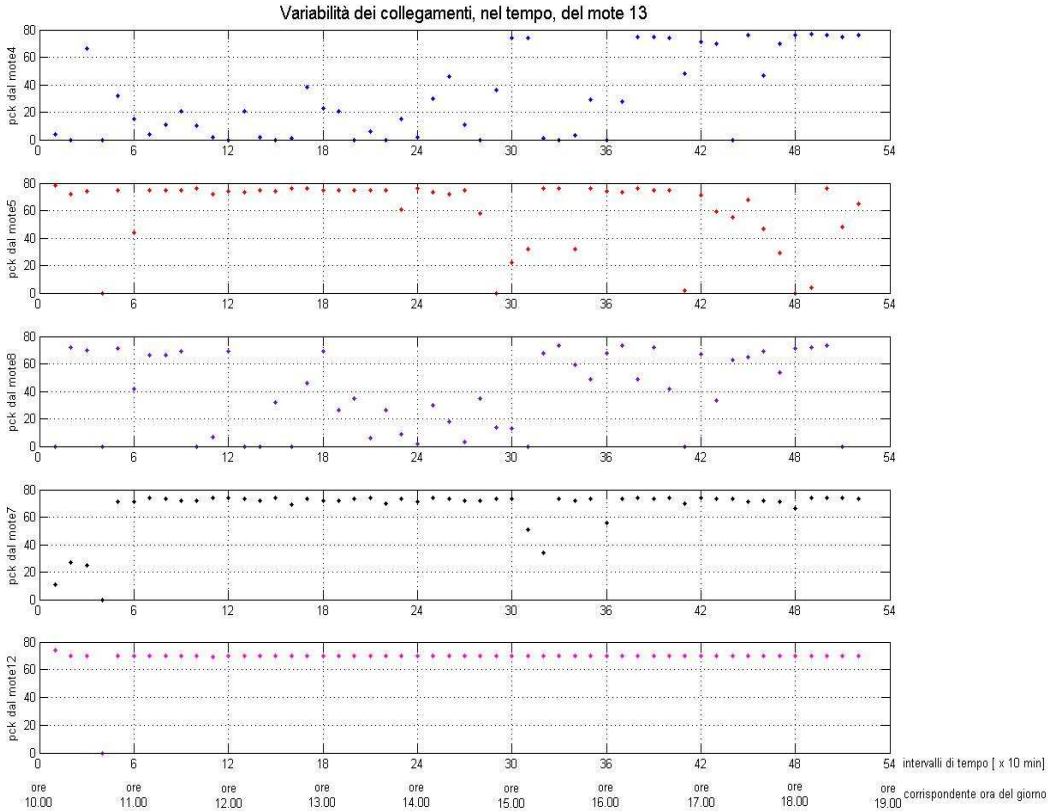


Figura 5.6 : continuità nel tempo di alcuni link

In *Figura 5.6* vengono rappresentati solo 5 tra i diversi link instaurati dal nodo 13 e sono sufficienti per verificarne le principali caratteristiche: disomogeneità e continuità, per esempio. Sono presenti, infatti, collegamenti stabilizzati a valori molto alti per tutta la giornata e altri che denotano qualche incertezza; link quasi del tutto stocastici, come quello con il mote 8, e altri che sono affidabili e costanti solo in alcune porzioni di tempo (al mattino o solo al pomeriggio).

Si vuole portare nuovamente un esempio: di seguito i dati raccolti dal mote 12, suddivisi nei soliti intervalli dalla durata di due ore, messi a confronto con quelli medi riferiti all'intero arco di raccolta. Si percepisce come i link con percentuali di pacchetti ricevuti molto alte tendano a restare costanti nel tempo con deviazioni standard sempre molto basse a volte coincidenti allo zero, spiegando la debole variabilità del primo tra i tre esempi considerati rispetto agli altri; per contro, i link con percentuali di successo inferiori sono meno stabili e hanno variazioni significative della dispersione dal valore medio, sintomo del diverso grado di difficoltà che la radio incontra a trasmettere o ricevere nei diversi momenti della giornata.

pacchetti ricevuti al mote 12 nell'intera giornata			***pacchetti ricevuti al mote 12 su intervalli di due ore***							
			media nella giornata				σ			
Mittente	*media nella giornata*	σ	10-12	12-14	14-16	16-18	10-12	12-14	14-16	16-18
3	55,3	6,11	60,1	57,3	52,4	53,6	24,76	25,64	29,88	26,83
4	66,4	4,39	52,9	68,9	71,6	76,0	22,86	18,00	11,71	1,29
5	74,5	2,34	76,1	76,0	76,0	75,9	0,64	0	0	0,28
6	73,3	2,34	73,9	74,9	75,0	74,9	3,95	0,28	0	0,28
7	37,9	6,23	54,6	28,1	28,1	47,4	22,09	26,31	25,10	27,31
8	51,3	6,53	49,6	65,5	41,5	56,1	29,88	15,16	30,46	28,49
9	39,0	5,81	45,8	18,4	47,3	49,6	24,09	18,98	24,91	22,79
13	68,7	2,15	70,1	70,0	70,0	70,0	0,95	0	0,00	0
14	68,7	2,15	70,2	70,0	70,0	70,0	0,55	0	0,00	0
18	4,4	1,89	0,3	0	5,9	10,8	0,62	0	7,05	11,96
20	57,7	4,81	69,2	45,2	55,7	62,5	0,90	25,05	22,78	15,01

Tabella 5.2: medie e deviazioni standard caratteristiche della connettività del nodo 12

5.2 Qualità della connettività

5.2.1 Distanza e RSSI

Una delle caratteristiche peculiari di questi dispositivi è il calcolo, sul messaggio ricevuto, dei fattori che identificano la qualità del segnale e la correttezza del pacchetto percepito. Tra questi, il valore più significativo è l'RSSI (*received signal strength indicator*) spesso utilizzato per la localizzazione, proprio per il legame esistente tra il suddetto indice e la distanza percorsa dal segnale.

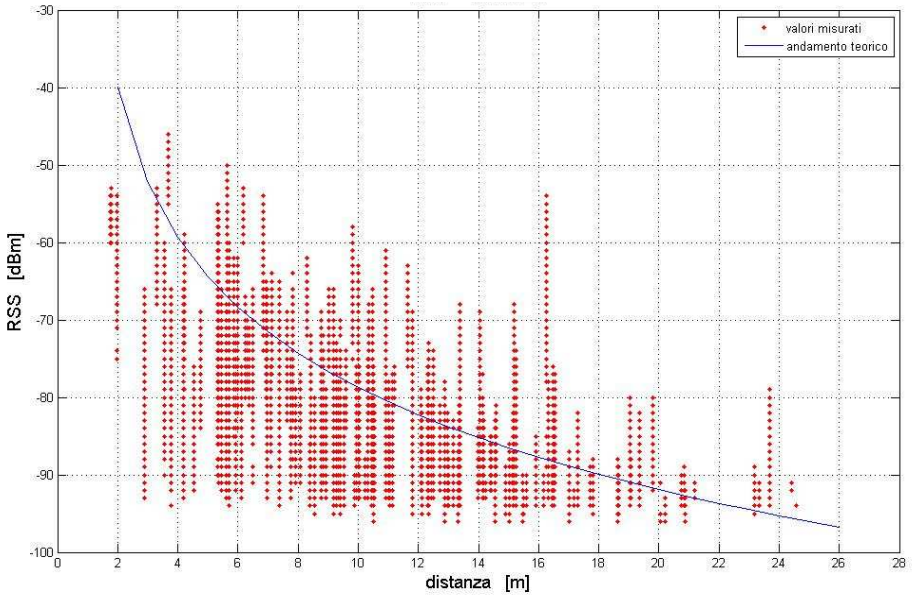


Figura 5.7 : legame RSS vs distanza

Per vedere meglio l'andamento del RSSI in funzione dello spazio, si è voluto riassumere in un unico grafico tutti i collegamenti avvenuti sovrapponendovi il modello del canale definito in (3), utilizzando i seguenti coefficienti stimati:

$$A = -42.5 \qquad \gamma = 1.8 \qquad \sigma = 4.9 \tag{4}$$

Si può osservare un'ampia oscillazione attorno al valore stimato che, a prima vista, sembra rendere impossibile la previsione della posizione precisa del trasmettitore sulla base dell'RSSI misurato. Per valori di potenza ricevuta molto bassi, che significa al di sotto di -80dBm, l'incertezza arriva a misurare anche 20-25 metri, quasi tre quarti della lunghezza del piano, mentre, per misure inferiori, l'errore si ridimensiona velocemente, arrivando a qualche metro.

Si delinea un andamento impreciso del rapporto RSS-distanza in linea con gli studi già presenti in letteratura, per esempio [8], che definiscono impraticabile in ambiente indoor l'utilizzo dell'RSSI per la localizzazione e, in generale, per la stima della distanza; si sperava in una migliore precisione del modello matematico poiché realizzato anche per aree urbane e non solo per il campo aperto e sebbene sia visibile sull'estremo superiore dell'insieme dei dati raccolti un andamento approssimabile a quello stimato, stessa cosa non si può dire per i valori vicini al limite inferiore.

Per comprendere concretamente come la distanza non sia l'unico fattore a caratterizzare la potenza in ricezione, confrontiamo tre link istauratisi tra coppie diverse di dispositivi, posizionate ciascuna all'interno di stanze diverse ma con lunghezze molto simili:

- ✓ mote 8 – mote 9 : 3.3 m
- ✓ mote 12 – mote 13: 3.54 m
- ✓ mote 10 – mote 11: 3.69 m

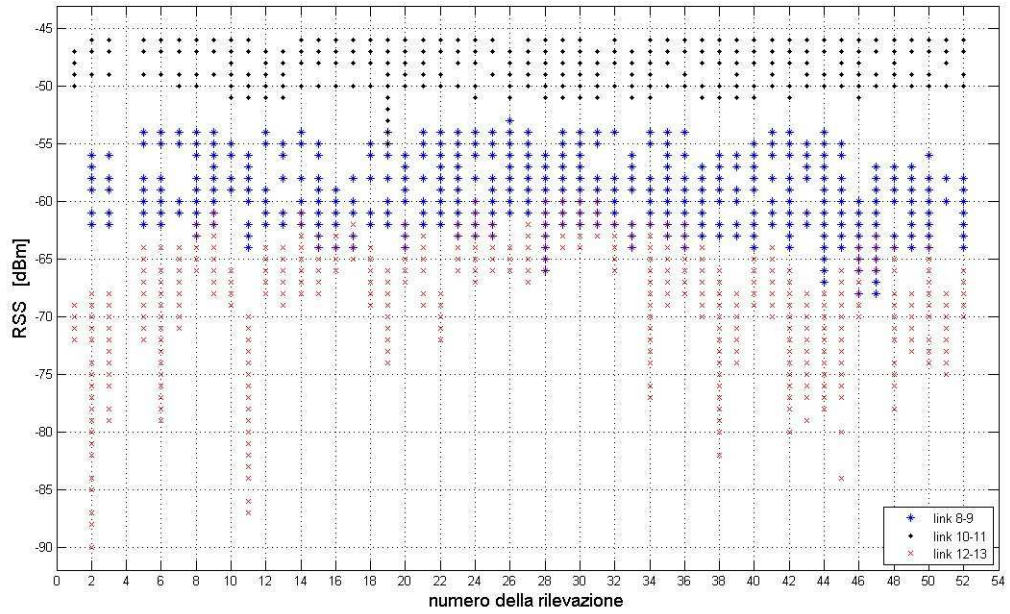


Figura 5.8 : RSS nel tempo

In linea teorica possiamo affermare, guardando il grafico, che il collegamento più lungo è quello con valori di RSS più elevati; questo è sicuramente in linea col fatto che tra i tre considerati, è anche quello che presenta meno ostacoli tra ricevitore e trasmettitore (una sorta di campo aperto); il link 12-13, che per distanza era il medio, presenta i valori più bassi di potenza ricevuta con una vistosa variabilità; questo

comportamento probabilmente è dovuto al fatto che erano posti all'interno di un laboratorio di elettronica in cui si stavano effettuando alcune misure per le quali venivano usate diverse macchine e per di più controllate da un alto numero di persone continuamente in movimento. Alla luce di questi dati, sembra non essere possibile trarre deduzioni concrete dal confronto di link con differenze inferiori ai 10dBm circa.

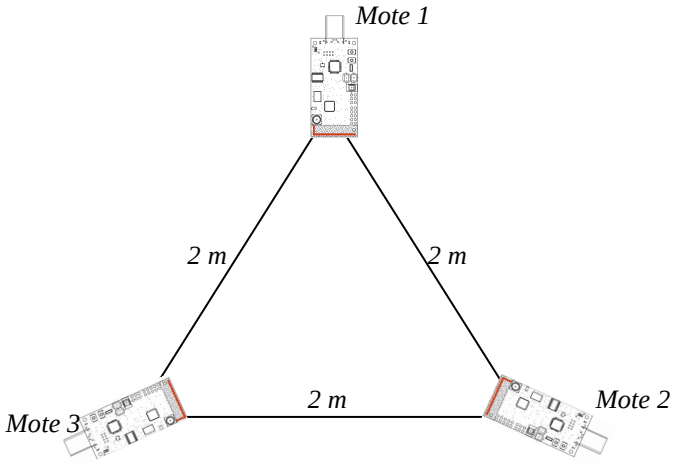


Figura 5.9 : schema primo test sul confronto dell'RSSI misurato da mote diversi

Un esame successivo ha dimostrato che TmoteSky diversi, posti alla stessa distanza dal trasmettitore, registrano valori differenti di RSSI. Si è utilizzato lo stesso software dell'esperimento sul piano su tre mote posti agli angoli di un triangolo equilatero di lato 2 m, tutti e tre con l'asse rivolto verso il centro della figura.

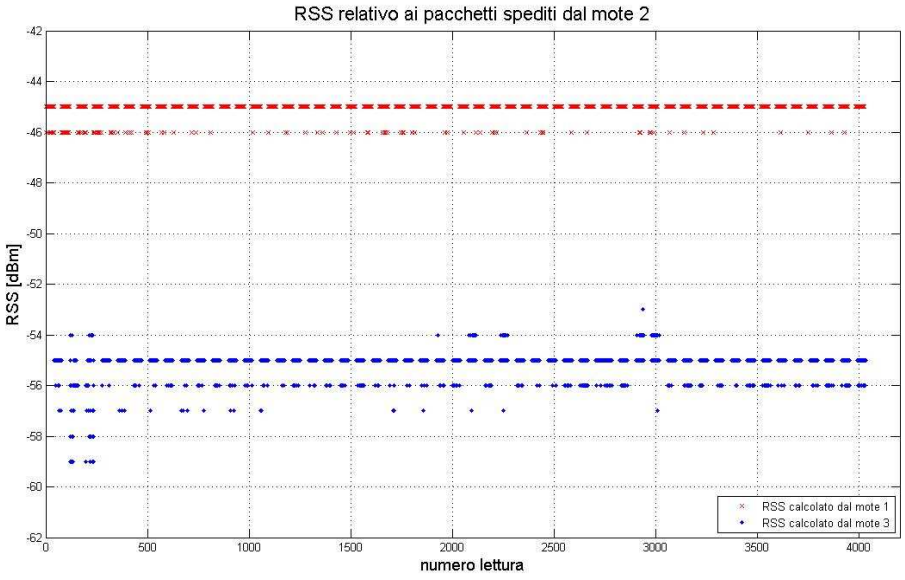


Figura 5.10: confronto tra misure di mote diversi dell'RSS di uno stesso link

Il caso dei pacchetti spediti dal mote 2 è quello con maggior incongruenza: i messaggi arrivano al dispositivo 1 e al 3 con una potenza che, in media, differisce di 10 dBm. Il test è avvenuto su di una piattaforma piana priva di ostacoli: sebbene tutt'intorno la stanza sia densa di oggetti spigolosi, supponiamo per un momento di poter trascurare come cause di una così cospicua disfunzione, l'incidenza di scattering, riflessioni e diffrazioni, oltre al fenomeno del multi-path fading. In particolare l'assenza di persone all'interno della zona interessata dall'esperimento è testimoniata dalla costanza dei valori dell'RSSI misurato che presentano varianza inferiore all'unità.

Si potrebbe allora giustificare l'asimmetria delle misure con la non precisa onnidirezionalità dell'antenna, che nei mote in ricezione ha posizioni assolute diverse rispetto al trasmettitore: stando al materiale presente in letteratura ([1]) la direzionalità dell'antenna non dovrebbe incidere fino a questi valori.

Per verificare l'estraneità o meno dell'antenna, si è eseguito un secondo test, simile al primo, ma questa volta i mote sono stati considerati uno alla volta e tutti con la stessa posizione nei confronti di un riferimento comune. Si è lasciato fisso il mote 1 e lo si è fatto colloquiare con un secondo dispositivo a due metri di distanza, entrambi disposti con i loro assi paralleli. Scambiate sette serie da 80 pacchetti l'una, il secondo mote veniva sostituito con il successivo, per cinque volte.

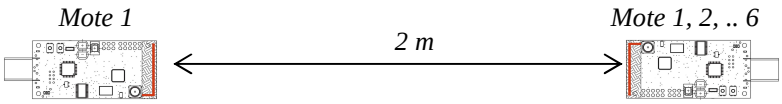


Figura 5.10 : schema secondo test sul confronto dell'RSSI misurato da mote diversi

La *Tabella 5.3* di sinistra riporta le medie dei valori misurati dal nodo di riferimento sui pacchetti ricevuti da tutti gli altri e quella di destra gli RSS calcolati da tutti gli altri nodi sui pacchetti ricevuti dal mote fisso; subito sotto si riportano in grafico le medie temporali (ogni 10 rilevazioni) delle stesse grandezze.

Si premette che dal punto di vista hardware, i mote 1, 2 e 3 del primo test sono gli stessi del secondo e ivi mantengono l'identificativo. Da qui una prima discrepanza: l'RSS dei pacchetti spediti dal mote 2 all'1, in *Figura 5.10* rappresentati in rosso, differiscono, nei due test, di 10 dBm, seppur calcolati su link di identica lunghezza. Appare alterato pure il link 1-3, anche se con un divario attorno all'unità, quindi molto inferiore.

La porzione di sinistra della *Tabella 5.3* dovrebbe comunque presentare valori molto simili tra loro; questi test sono stati fatti in tempi molto brevi e quindi le condizioni ambientali possono considerarsi uguali per tutti i nodi.

Pacchetti ricevuti dal mote1

	media	varianza
spediti dal 2	-55,92	0,56
spediti dal 3	-54,62	0,56
spediti dal 4	-52,93	1,00
spediti dal 5	-52,28	1,32
spediti dal 6	-52,19	1,50

Pacchetti spediti dal mote1

	media	varianza
ricevuti dal 2	-53,66	0,47
ricevuti dal 3	-51,46	0,75
ricevuti dal 4	-50,01	1,61
ricevuti dal 5	-49,67	1,28
ricevuti dal 6	-51,00	0,82

Tabella 5.3: RSS medi e loro varianze

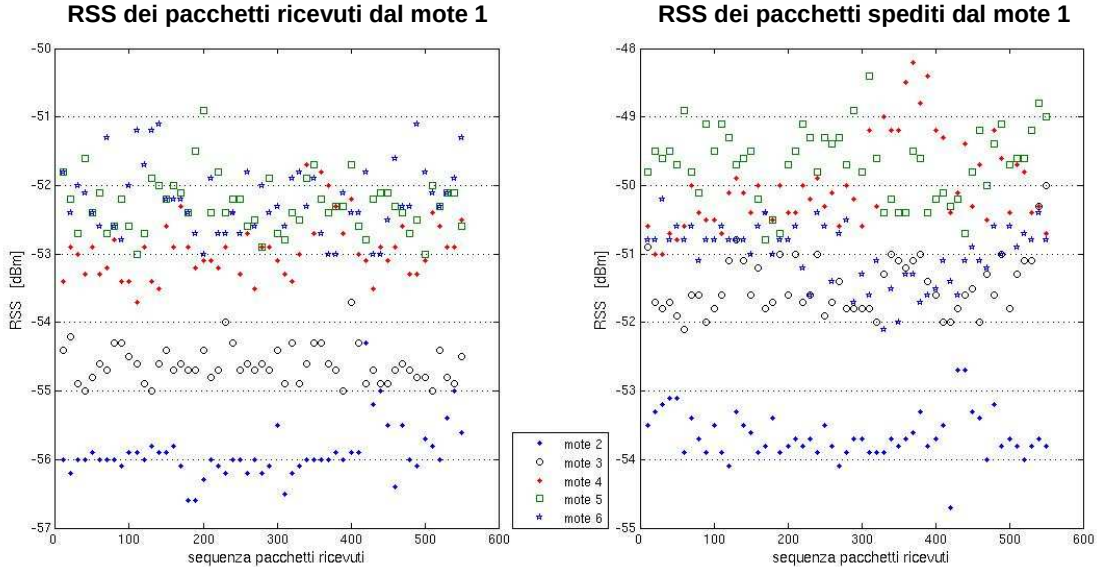


Figura 5.11: confronto tra RSS riferiti a un medesimo link, calcolati da differenti mote

Le rilevazioni dei cinque mote considerati presentano un gap comunque contenuto rispetto al test precedente, sintomo di una possibile influenza negativa dell'irregolarità nell'irraggiamento dell'antenna.

Per risolvere completamente il problema, oltre a prendere in considerazione l'utilizzo di antenne esterne, è necessaria l'attenta calibrazione di ogni singolo mote: dal valore di RSSI calcolato dal chip radio, come già esposto, si ricava la potenza in ricezione sommando un offset da misurarsi sperimentalmente facendo delle prove sul campo; si può ipotizzare che effettuare la calibrazione affidandosi ad un offset calcolato come media del valore misurato per un numero limitato di dispositivi sia estremamente approssimativo. Ai fini della precisione, l'operazione dovrebbe essere eseguita volta per

volta sul singolo mote con un esperimento simile a quello descritto sopra, per di più eseguendolo più volte su diverse distanze e non solo a campo aperto, calcolando l'offset come media dei valori misurati relativi allo stesso dispositivo; questo metodo è sicuramente molto laborioso ma assicura maggior accuratezza, necessaria in alcune applicazioni come localizzazione o tracking.

5.2.2 Affidabilità nella distanza

Risulta interessante vedere, a questo punto, la robustezza dei link in funzione della loro lunghezza, al di là degli ostacoli che ciascuno possa incontrare per avere a grandi linee un'idea, in fase di progettazione di una ipotetica rete wireless indoor, dei limiti spaziali da rispettare. Dai dati empirici raccolti, risulta che la distanza limite tra un mote e l'altro sembra essere di 14 metri, oltre i quali si registra una serie sempre più fitta di link con percentuali di pacchetti persi pari al 100%, di fatto collegamenti mai instauratisi; questo comportamento è ben visibile in *Figura 5.12.*

I link compresi nella fascia tra gli 8 e i 14 metri hanno una buona probabilità di poter essere utilizzati: le connessioni che percorrono uno spazio compreso tra 8 e 12 metri, infatti, ricevono in media il 74% dei messaggi mentre, quelli con lunghezze fino a 14 metri, hanno percentuali di pacchetti persi vicine al 30%, prestazioni ancora accettabili per diverse applicazioni.

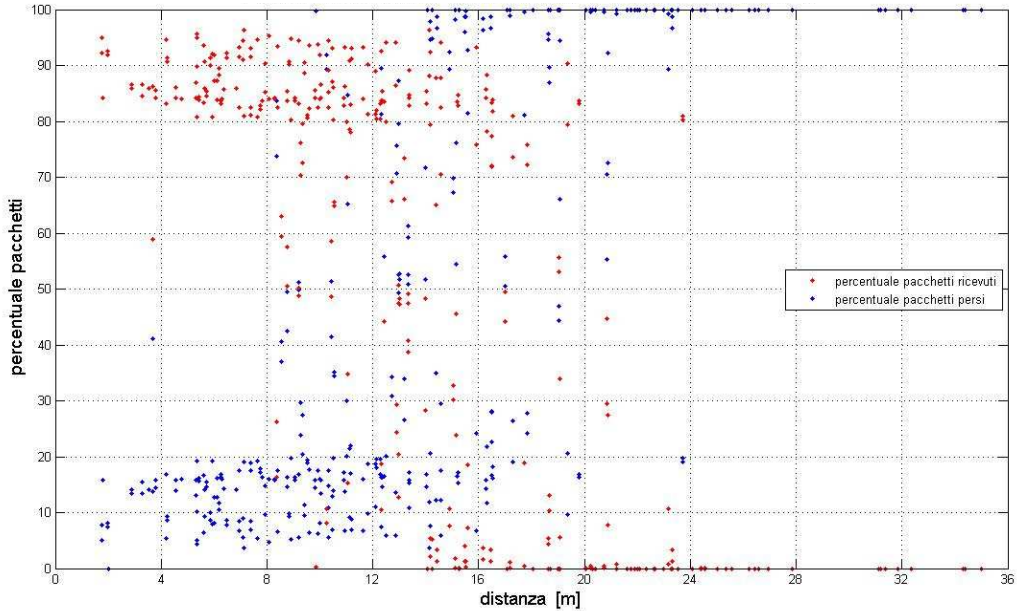


Figura 5.12 : percentuale di pacchetti persi e ricevuti in funzione della distanza, link per link

5.2.3 Affidabilità dell'alimentazione

Un altro punto fondamentale è l'affidabilità del battery pack. Come già spiegato si sono utilizzate batterie *AA* già adoperate per altri test, che presentavano quindi tensioni non sempre vicine a quelle nominali d'acquisto, e non si è mai gestito il risparmio energetico; tutto questo per apprezzare meglio le conseguenze dovute al variare della tensione. Al termine dell'esperimento si è potuto constatare che l'alimentazione non era mai scesa sotto i valori minimi indicati da data sheet e la differenza tra la tensione all'accensione e allo spegnimento non ha mai superato gli 0.23 V; ciò ha permesso solo di confermare il corretto comportamento dei dispositivi all'interno degli intervalli di funzionamento raccomandati senza dare la possibilità di vedere che tipo di malfunzionamento possa accadere nel caso in cui la tensione risulti insufficiente.

Un primo grafico mostra l'andamento lineare della potenza in ricezione in funzione del livello di carica sia del trasmettitore che del ricevitore

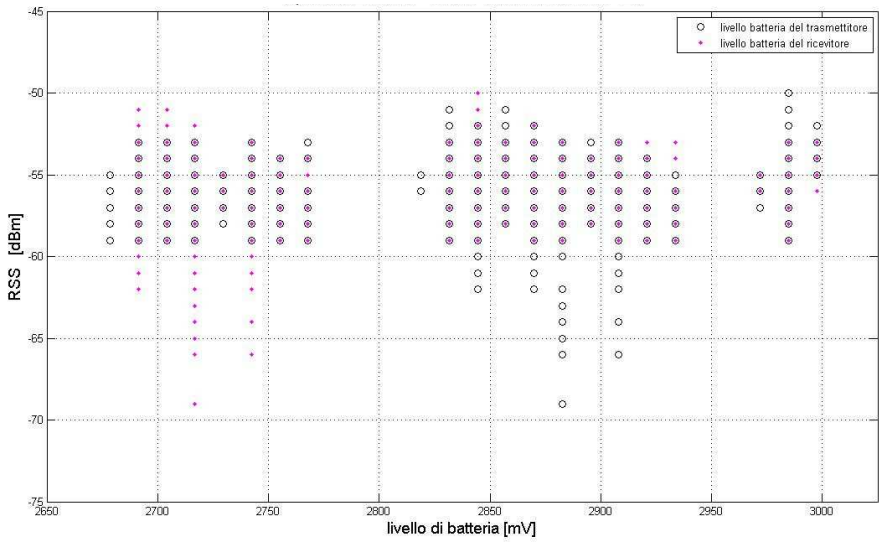


Figura 5.13: dipendenza RSS dal livello di carica delle matterie nel link 7-20

Anche nella panoramica di *Figura 5.14* è possibile constatare l'andamento uniforme del dispositivo all'interno dell'intero intervallo di tensione di funzionamento sia nel caso in cui funzioni da trasmettitore che da ricevitore.

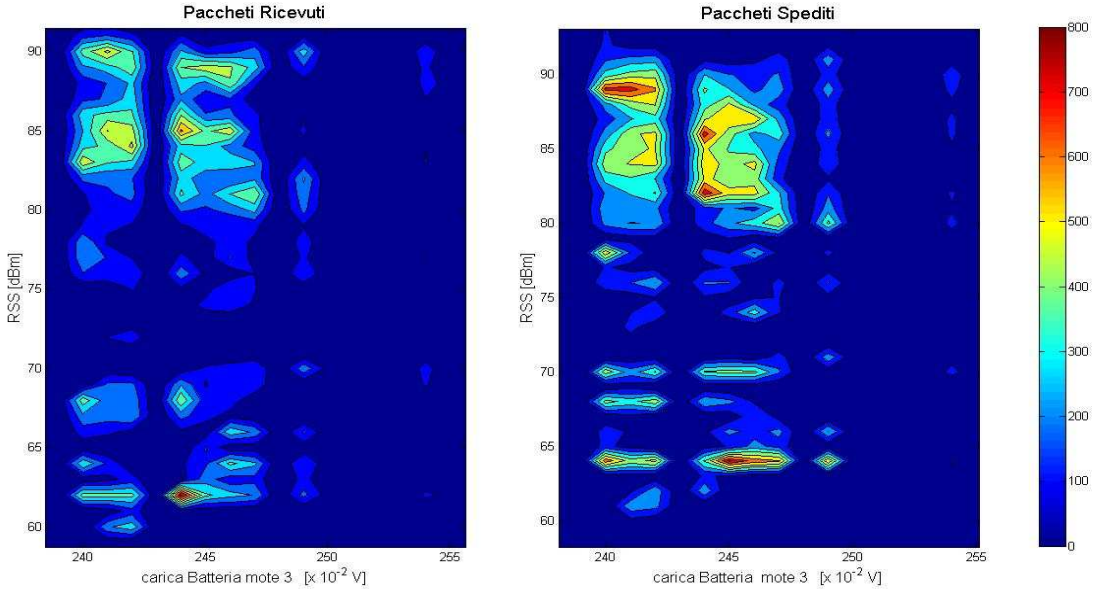


Figura 5.14: pacchetti ricevuti e spediti dal mote 3 in funzione dell'alimentazione e dell'RSS

In *Tabella 5.4* e *Tabella 5.5* si sono volute riassumere rispettivamente le tensioni dei singoli mote ordinate in maniera crescente in base al consumo registrato durante la prova e il numero di link sostenuti da ogni dispositivo.

mote	livello batteria [mV] iniziale	finale	differenza
1	2691	2589	102
12	2742	2640	102
5	2806	2679	128
8	2717	2577	140
9	2742	2602	140
11	2768	2628	140
6	2628	2487	140
18	2755	2615	140
2	2781	2628	153
3	2551	2398	153
4	2806	2653	153
10	2628	2449	179
13	2832	2653	179
14	2704	2526	179
19	2832	2653	179
20	2870	2691	179
15	2806	2602	204
17	2895	2679	217
7	3061	2832	230
16	2806	2577	230

Tabella 5.4 : livelli di carica dei singoli mote

mote	link sostenuti
8	17379
9	27080
13	27114
16	28590
18	29856
17	30735
12	31355
15	33309
14	36660
10	38296
11	40678
19	43004
4	43498
1	44796
2	45820
5	46064
6	49198
7	53101
3	55029
20	55402

Tabella 5.5 : link sostenuti

La posizione dei dispositivi nella lista di sinistra non rispecchia per niente quella che riassume il numero di collegamenti effettuati: sebbene la potenza di trasmissione sia sempre la stessa, il consumo non risulta proporzionale al numero di pacchetti spediti o ricevuti, le cui classifiche risultano abbastanza simmetriche, e di conseguenza non sembra legato neppure all’assorbimento di corrente dovuto alla scrittura in flash.

Conclusioni

Nella tesi si sono analizzati i comportamenti di ogni singola connessione che potrebbe costituire una ipotetica WSN basata sulla stessa disposizione dei nodi. La continuità nel tempo della connettività è un requisito fondamentale per poter utilizzare una network con la certezza di averla in perfette condizioni e senza rischiare di non arrivare in alcuni punti della rete. Si è visto, di fatto, che alcuni link sono soggetti a variazioni temporali dipendenti da molti fattori che si presentano senza una precisa temporizzazione e che spesso si sommano tra loro aumentando l'intensità del disturbo; la realizzazione della WSN non è comunque in pericolo: oltre a questi collegamenti non del tutto affidabili, ne esistono di più precisi e costanti, presenti dall'inizio alla fine dell'esperimento.

Si è mostrato purtroppo inadeguato il modello del canale in ambiente indoor, di gran lunga approssimativo soprattutto nei casi in cui i link presentino valori di potenza in ricezione molto bassi, con approssimazioni vicine alle dimensioni del piano: solo un campione ristretto di valori può effettivamente essere utilizzato per la stima della distanza. Collegato c'è poi il problema della veridicità della misura dell'RSS: non si è calcolata la differenza tra valore vero e misurato, ma si è dimostrato che anche per distanze ridotte, come per esempio un paio di metri, misure diverse differiscono anche di 10 dBm. Incide parecchio, in questo caso, l'antenna interna del TmoteSky: nel secondo test, mettendo uno alla volta i dispositivi nella stessa posizione rispetto al trasmettitore, l'intervallo di incertezza diminuisce almeno del 25% circa. Questa considerazione spinge a ritenere l'antenna la prima causa dell'asimmetria nei collegamenti.

Una volta constatato che entro certi livelli di tensione l'alimentazione non crea malfunzionamenti, rimangono da nominare gli ultimi due problemi: la distanza, con connessi ostacoli, e la calibratura. Quest'ultima, da eseguire su ogni mote, risulta essere condizione necessaria, anche se non sufficiente, per raggiungere un'accettabile accuratezza nel calcolo delle distanze in funzione dell'RSS misurato. Per quanto concerne la distanza, la *Figura 5.12* mostra chiaramente che sotto gli 8 – 9 metri, anche se attraverso un muro e qualche altra ostruzione, la connessione risulta, con ottima

probabilità, affidabile; delinea chiaramente anche il limite superiore: sopra i 14 m la probabilità di instaurare dei link utili è veramente molto bassa.

Per avere un quadro completo della connettività, risulterebbe necessaria la ripetizione della raccolta dati per un periodo molto più lungo, per esempio qualche giorno, per verificare la ripetitività degli eventi negli stessi istanti di giornate diverse. Fondamentale anche replicare l’esperimento utilizzando diversi livelli di potenza, facilmente realizzabile visto che il software utilizzato in questa tesi è riutilizzabile in qualsiasi ambiente e già predisposto per la gestione delle potenze in trasmissione.

APPENDICE

BlinkToRadio.h

```
#ifndef BLINKTORADIO_H
#define BLINKTORADIO_H

enum {
    channelRadio = 13,
    numNodi = 6,
    numPktSfilza = 80,
    numCicli = 60,
    AM_TEST_SERIAL_MSG = 9,
    AM_BLINKTORADIO = 6,
    TIMER_PERIOD_MILLI = 64,
    TIMER_PERIOD_LEGGO = 15,
    TIMER_PERIOD_SCORRO = TIMER_PERIOD_MILLI*120,
    TIMER_PERIOD_ESTERNO = TIMER_PERIOD_MILLI*120*numNodi+1,
    codeBlinkToRadioMessage = 58,
    codeChiamata =      38,
    codeSiComincia = 28,
    codeTxTerminata = 48,
    valueOfInitialization = 0,
};

typedef struct BlinkToRadioMsg {
    nx_uint8_t serialNumber;
    nx_uint8_t code;        // 58 = default (decimale)
    nx_uint16_t battery;
    nx_uint8_t pTx;
} BlinkToRadioMsg;

typedef struct chiamata {
    nx_uint8_t serialNumber;
    nx_uint8_t code;        // 38 = default (decimale)
    nx_uint16_t idTx;
} chiamata;

typedef struct txTerminata {
    nx_uint8_t serialNumber;
    nx_uint8_t code;        // 48 = default (decimale)
    bool sfilzaSpedita;
    nx_uint16_t mitt;
} txTerminata;

typedef struct totalMsg {
    nx_uint8_t serialNumber;
    nx_uint8_t temperature;
    nx_uint8_t batteryMitt;
    nx_uint8_t batteryDest;
    nx_uint8_t pTx;
    uint8_t lqi;    //      async command uint8_t getLqi( message_t* p_msg );
    int8_t rssi;    //      async command int8_t getRssi( message_t* p_msg );
    nx_uint8_t dest;
    nx_uint8_t mitt;
```

```
} totalMsg;

#endif
```

BlinkToRadioC.nc

```
#include <Timer.h>
#include "BlinkToRadio.h"
#include "StorageVolumes.h"

module BlinkToRadioC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as TimerEsterno;
  uses interface Packet;
  uses interface AMSend;
  uses interface BlockWrite as Write;
  uses interface SplitControl as AMControl;
  uses interface Receive;
  uses interface CC2420Config;
}

implementation {
    bool busy = FALSE;
    bool isFirst;    // controllo per la formattazione dalla flash
    message_t packet;
    uint16_t counter;        // id di chi deve trasmettere; assuem valori da 0 a numNodi
    uint8_t serialNumber;  // numero del ciclo attuale
    uint64_t c;

    enum {
        CONFIG_ADDR = 0;
    };

    event void Boot.booted() {
        counter = 0;
        serialNumber = 0;
        c = 0;
        isFirst = TRUE;
        call CC2420Config.setChannel(channelRadio);    // setto il canale della radio
        call CC2420Config.sync();                      // applico la modifica al canale radio
    }

    task void cc() {
        if (!busy) { // se radio libera
            chiamata* btrpkt = (chiamata*)(call Packet.getPayload(&packet, NULL));
            btrpkt->code=codeChiamata;          // codice identificativo del pkt
            btrpkt->idTx=counter;               // id di chi deve trasmettere la sfilza
            btrpkt->serialNumber=serialNumber;  // numero del ciclo
            btrpkt->kiRipete=TOS_NODE_ID;       // firmo il pkt
            counter++;                          // icremento l'id di chi deve trasmettere
            if (call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(chiamata)) == SUCCESS) {
                                                // spedisco chiamata
                busy = TRUE;                    // radio occupata
            }
        }
    }

    event void Timer0.fired() {
        if (!busy) { // se radio libera
            chiamata* btrpkt = (chiamata*)(call Packet.getPayload(&packet, NULL));
```

```
        btrpkt->code=codeChiamata;              // codice identificativo del pkt
        btrpkt->idTx=counter;                   // id di chi deve trasmettere la sfilza
        btrpkt->serialNumber=serialNumber;      // numero del ciclo
        btrpkt->kiRipete=TOS_NODE_ID;           // firmo il pkt
        counter++;                              // icremento l'id di chi deve trasmettere
        if (call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(chiamata)) == SUCCESS) {
                                                // spedisco chiamata
            busy = TRUE;                        // radio occupata
        }
    }
    if(counter == (numNodi+1)) {                //quando ho spedito l'ultima chiamata
        call Timer0.stop();                     // fermo il Timer0
        if (serialNumber >= numCicli)           // se ho effettuato il numero di cicli desiderato
            call TimerEsterno.stop();           // fermo TimerEsterno
    }
}

event void TimerEsterno.fired() {
    counter=0;                                  // inizializzo l'id di chi deve trasmettere
    serialNumber++;                             // incremento il numero del ciclo
    call Timer0.startPeriodic(TIMER_PERIOD_SCORRO); // start timer ogni TIMER_PERIOD_SCORRO
    post cc();                                  // per non perdere il primo ciclo
}

task void ccc(){
    counter=0;                                  // inizializzo l'id di chi deve trasmettere
    serialNumber++;                             // incremento il numero del ciclo
    call Timer0.startPeriodic(TIMER_PERIOD_SCORRO); // start timer ogni:TIMER_PERIOD_SCORRO
    post cc();                                  // per non perdere il primo ciclo di clock
}

event void AMControl.stopDone(error_t err) {
}

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
    if (len == sizeof(txTerminata)) {
        txTerminata* tellTerminata = ((txTerminata*)payload);
        if(tellTerminata->code == 48) {
            if(isFirst==TRUE) {
                call Write.erase();
                isFirst=FALSE;
            }
            call Write.write(CONFIG_ADDR+(c*sizeof(txTerminata)), tellTerminata, sizeof(txTerminata));
            c++;
        }
    }
    if (len == sizeof(chiamata)) {
        if(isFirst==TRUE) {
            call Write.erase();
            isFirst=FALSE;
        }
    }
    return msg;
}

event void AMSend.sendDone(message_t* msg, error_t error) {
    busy = FALSE;                               // radio libera
}

event void Write.writeDone(storage_addr_t addr, void* buf, storage_len_t len, error_t error) {
    call Write.sync();
}
```

```
    event void Write.syncDone(error_t error) {
    }

    event void Write.eraseDone(error_t error) {
    }

    event void CC2420Config.syncDone( error_t error ) {
        call AMControl.start();
    }

    event void AMControl.startDone(error_t err) {       // componentistica Mote accesa
        if (err == SUCCESS) {
            call TimerEsterno.startPeriodic(TIMER_PERIOD_ESTERNO);     // start ciclo ogni 10min
            post ccc();                                  // per non perdere il primo ciclo di clock
            }
        else {  // se non è andato a buon fine lo start
        call AMControl.start();                          // lo richiamo
            }
    }

}
```

Serial

```
#include <Timer.h>
#include "BlinkToRadio.h"
#include "StorageVolumes.h"
#include "message.h"

module BlinkConfigC {
    uses {
        interface Boot;
        interface Leds;
        interface SplitControl as AMControl;
        interface Timer<TMilli> as Timer0;                  // Timer per inoltro pkt dati
        interface BlockWrite as Write;                      // scrive in flash
        interface Packet;
        interface AMSend as AMSendSpedisco;                 // Send per inoltro sequenza dati
        interface AMSend as AMSendFlooding;                 // Send per flooding
        interface Receive;
        interface CC2420Packet as SetRadio;                 // per avere LQI e RSSI della trasmissione
        interface AMPacket;
        interface Read<uint16_t> as ReadVoltage;            // leggere livello batterie
        interface Read<uint16_t> as ReadTemperature;        // leggere temperatura
        interface Packet as PacketSpedisco;
        interface CC2420Config;                             // modifica canale radio
        }
    }

implementation {
    enum {
        CONFIG_ADDR = 0          // indirizzo di partenza per scrittura in flash
    };

    bool busy = FALSE;                      // segno l'utilizzo delle risorse (radio e flash)
    bool taskChiamataBusy = FALSE;          // segno l'utilizzo del task "taskChiamataBusy()"
    bool taskTxTerminataBusy = FALSE;       // segno l'utilizzo del task "taskTxTerminataBusy()"
    bool taskBlinkToRadioBusy = FALSE;      // segno l'utilizzo del task "taskBlinkToRadioBusy()"
    bool taskSfilzaFatta = FALSE;           // segno l'utilizzo del task "taskSfilzaFatta()"
    bool isFirst = TRUE;                    // controllo per l'inizializzazione della flash
    message_t pktSfilza;                    // utilizzato per l'invio dei pkt della sfilza
    message_t pktChiamata;                  // utilizzato per il flooding del pkt "chiamata"
```

```
chiamata tellChiamata;                  // salvo payload per renderlo disponibile al task
message_t pktTxTerminata;               // utilizzato per il flooding del pkt "txTerminata"
txTerminata tellTxTerminata;            // salvo payload per renderlo disponibile al task
totalMsg ricevuti[numPktSfilza];        // riverso qui la sfilza ricevuta prima di memorizzarla in flash
uint32_t c;// indice per scorrrere Ricevuti[]
BlinkToRadioMsg tmpBlinkToRadioMsg; // salvo payload
uint8_t valBat;                         // memorizzo val. livello batteria
uint8_t valTemperature;                 // memorizzo temperatura
uint8_t floodingChiamata[numNodi+1];// tengo memoria del flooding pkt "chiamata"
uint8_t floodingTxTerminata[numNodi+1];         // tengo memoria del flooding pkt "txTerminata"
uint8_t singoliPkt;
uint8_t livelli_potenze;
uint8_t potenza;
uint32_t copiaC;                        // indirizzo dove scrivere in flash
uint8_t m;                              // indice per inizializzare vettori
uint8_t SerialNumberTmp;                // copia del numero di ciclo

event void Boot.booted() {
    call CC2420Config.setChannel((uint8_t)channelRadio);    // setto il canale della radio
    call CC2420Config.sync();                               // applico la modifica al canale radio
    singoliPkt = 0;
    c = 0;
    copiaC = 0;
    livelli_potenze = 0;
    potenza = 31;
    valBat=0;
    for(m=0; m<(numNodi+1); m++) {
        floodingChiamata[m] = valueOfInitialization;
        floodingTxTerminata[m] = valueOfInitialization;
    }
    call ReadVoltage.read();                                // lettura livello batteria
    call ReadTemperature.read();                            // lettura temperatura
    }

event void Write.writeDone(storage_addr_t addr, void* buf, storage_len_t len, error_t error) {
    call Write.sync();                                      // assicuro la scrittura in flash
}

task void sfilzaFattaTask() {                               // avviso di fine invio sfilza
    if (!busy) { // se radio libera
        txTerminata* msgTxTerm = (txTerminata*)(call Packet.getPayload(&pktTxTerminata, NULL));
        msgTxTerm->sfilzaSpedita = TRUE;
        msgTxTerm->code = codeTxTerminata;
        msgTxTerm->serialNumber=SerialNumberTmp;
        msgTxTerm->mitt = TOS_NODE_ID;
        msgTxTerm->kiRipete = TOS_NODE_ID;                  // firmo il pkt
        if (call AMSendFlooding.send(AM_BROADCAST_ADDR, &pktTxTerminata, sizeof(txTerminata))
        == SUCCESS) {                                       // spedisco
            atomic{busy = TRUE;}                            // radio occupata
        }
    }
    taskSfilzaFatta=FALSE;                                  // task terminato
}

event void Timer0.fired() {             // invio un pkt dato ogni TIMER_PERIOD_MILLI millisecondi
    call Leds.led2Toggle();
    call Leds.led0Off();
    if(singoliPkt<10) {                 // invio 10 pkt per livello di potenza
        if (!busy) {                    // se radio libera
            BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)(call Packet.getPayload(&pktSfilza, NULL));
            btrpkt->pTx=potenza;
            btrpkt->code=codeBlinkToRadioMessage;
            btrpkt->kiSpedisce=TOS_NODE_ID;
```

```
            btrpkt->serialNumber=SerialNumberTmp;
            call ReadVoltage.read();
            call ReadTemperature.read();
            btrpkt->battery=valBat;
            call SetRadio.setPower( (message_t*)&btrpkt, potenza);      // imposto potenza radio
                if (call AMSendSpedisco.send(AM_BROADCAST_ADDR, &pktSfilza,
                sizeof(BlinkToRadioMsg)) == SUCCESS) {    // spedisco
            atomic{busy = TRUE;}          // radio occupata
        }
    }
  }
  if(livelli_potenze<8 && singoliPkt>9) {   // entro quando ho trasmesso 10 pkt di ugual potenza
      singoliPkt=0;         // azzero contatore pkt
      livelli_potenze++;  // cambio livello potenza
          switch (livelli_potenze) {
            case 0 :      potenza=31;
                          break;
            case 1 :      potenza=27;
                          break;
            case 2 :      potenza=23;
                          break;
            case 3 :      potenza=19;
                          break;
            case 4 :      potenza=15;
                          break;
            case 5 :      potenza=11;
                          break;
            case 6 :      potenza=7;
                          break;
            case 7 :      potenza=3;
                          break;
            case 8 :      call Timer0.stop();// sfilza trasmessa; spengo il timer
                          if(taskSfilzaFatta==FALSE) {          // se task non in esecuzione
                                taskSfilzaFatta=TRUE;       // lo marco
                                post sfilzaFattaTask();       // e lo richiamo
                          }
                          break;
          }
  }
}

task void ricChiamataTask() {
                                                              // check stato flooding per questo pkt
  if((floodingChiamata[tellChiamata.idTx] < tellChiamata.serialNumber)) {
      chiamata* chipkt = (chiamata*)(call Packet.getPayload(&pktChiamata, NULL));
      floodingChiamata[tellChiamata.idTx] = tellChiamata.serialNumber;
                                                              // segno il flooding corrente
      if(tellChiamata.idTx == 0) {                            // se è il primo pkt del ciclo (idTx == 0)
          if(isFirst==TRUE) {                                 // se è il primo ciclo
              call Write.erase();                             // viene inizializzata la flash
              atomic{ isFirst=FALSE;
          }                                                   // format flash avvenuto
      }
      tellChiamata.kiRipete = TOS_NODE_ID;                    // firmo il flooding
      *chipkt=tellChiamata;
      if (!busy) {                                            // se radio libera
      if(call AMSendFlooding.send(AM_BROADCAST_ADDR,&pktChiamata,sizeof(chiamata))=SUCCESS) {
                                                              // flooding
              atomic{busy = TRUE;}                            // radio occupata
          }
      }
      singoliPkt = 0;                                         // all'inizio del ciclo inizializzo le variabili
      livelli_potenze = 0;
```

```
        potenza = 31;
        }
        else{                                                   // se non è il primo pkt del ciclo
            if(tellChiamata.idTx == TOS_NODE_ID) {              // se tocca a me spedire la sfilza
                call Timer0.startPeriodic(TIMER_PERIOD_MILLI);  //parte timer per spedirla
            }
            else {                                              // se non tocca a me
                tellChiamata.kiRipete = TOS_NODE_ID;
                *chipkt=tellChiamata;
                if (!busy) {                                    // se la radio è libera
                if(call AMSendFlooding.send(AM_BROADCAST_ADDR, &pktChiamata, sizeof(chiamata)) ==
                SUCCESS) {                                      // flooding
                    atomic{busy = TRUE;}                        // radio occupata
                }
                }
            }
        }
    }
    atomic{taskChiamataBusy=FALSE;}                             // task terminato
    }

    task void ricTxTerminataTask() {
        if(floodingTxTerminata[tellTxTerminata.mitt] < tellTxTerminata.serialNumber) {
                                                                // check stato flooding per questo pkt
            txTerminata* txTermpkt = (txTerminata*)(call Packet.getPayload(&pktTxTerminata, NULL));
            floodingTxTerminata[tellTxTerminata.mitt] = tellTxTerminata.serialNumber;
                                                                // segno il flooding corrente
            tellTxTerminata.kiRipete = TOS_NODE_ID; // firmo il pkt
            *txTermpkt=tellTxTerminata;
                if (!busy) {                                    // se radio libera
                    if (call AMSendFlooding.send(AM_BROADCAST_ADDR, &pktTxTerminata,
                    sizeof(txTerminata)) == SUCCESS) {          // flooding
                    atomic{busy = TRUE;}                        //radio occupata
                    }
                }
            call  Write.write(CONFIG_ADDR+copiaC*sizeof(totalMsg),  &(ricevuti[0]),  c*sizeof(totalMsg));
                                                                // metto sfilza in flash
            copiaC =copiaC + c;                                 // aggiorno l'indirizzo di scrittura in flash
            c = 0; // azzero l'indice per scorrere il vettore dei pkt dati ricevuti
        }
        atomic{taskTxTerminataBusy=FALSE;}                      // task concluso
    }

/** RICEZIONE : il mote, in standby, ascolta la radio. Quando arriva un messaggi olo selezione prima in
base alla dimensione, poi controlla il codice che lo contraddistingue e se il task corrispondente non è in
esecuzione, lo richiama */
    event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
        call Leds.led2Off();
        call Leds.led0Toggle();
        if (len == sizeof(chiamata)) {                          // arriva un pkt di tipo "chiamata"
            tellChiamata = *((chiamata*) payload);
            if ((tellChiamata.code == codeChiamata) && (!taskChiamataBusy) &&
            (tellChiamata.idTx <= numNodi)) {
                atomic{taskChiamataBusy=TRUE;}
                SerialNumberTmp=tellChiamata.serialNumber;      // tengo copia del numero ciclo corrente
                post ricChiamataTask();
            }
        }
        if (len == sizeof(txTerminata)) {                       // arriva un pkt di tipo "txTerminata"
            tellTxTerminata = *((txTerminata*) payload);
            if ((tellTxTerminata.code == codeTxTerminata) && (!taskTxTerminataBusy)
            && (tellTxTerminata.mitt <= numNodi)) {
                atomic{taskTxTerminataBusy=TRUE;}
```

```
            post ricTxTerminataTask();
        }
    }
    if (len == sizeof(BlinkToRadioMsg)) {              // arriva un pkt di tipo "BlinkToRadioMsg" (dati)
        tmpBlinkToRadioMsg = *((BlinkToRadioMsg*) payload);
        if ((tmpBlinkToRadioMsg.code == ((uint8_t)codeBlinkToRadioMessage))
        && (!taskBlinkToRadioBusy)) {
            ricevuti[c].serialNumber = tmpBlinkToRadioMsg.serialNumber;
            ricevuti[c].batteryMitt = tmpBlinkToRadioMsg.battery;        // livello batteria mittente
            call ReadTemperature.read();
            call ReadVoltage.read();
            ricevuti[c].batteryDest = valBat;                  // livello batteria Destinatario
            ricevuti[c].temperature = valTemperature;      // temperatura
            ricevuti[c].pTx = tmpBlinkToRadioMsg.pTx;
            ricevuti[c].dest = (uint8_t)TOS_NODE_ID;       // indirizzo destinatario
            ricevuti[c].mitt = call AMPacket.source(msg);  // indirizzo mittente
            ricevuti[c].lqi = call SetRadio.getLqi(msg);       // LQI
            ricevuti[c].rssi = call SetRadio.getRssi(msg);     // RSSI
            c++;                          // incremento l'indice per scorrere il vettore dei pkt dati ricevuti
        }
    }
    return msg;
}

event void ReadVoltage.readDone(error_t result, uint16_t val) {
                                            // lettura livello batterie
    valBat=((uint8_t)(val*0.059534883));
}

event void ReadTemperature.readDone(error_t result, uint16_t val) {
                                            // lettura temperatura
    valTemperature=(val*(1.5/4096)-0.986)/0.00355;
}

event void AMSendFlooding.sendDone(message_t* msg, error_t error) {
    if( call Packet.payloadLength(msg) == sizeof(chiamata)) {
                                            // sendDone flooding pkt "chiamata"
        atomic{busy = FALSE;} // radio libera
    }
    if( call Packet.payloadLength(msg) == sizeof(txTerminata)) {
                                            // sendDone flooding pkt "chiamata"
        tellTxTerminata = *(txTerminata*)(call Packet.getPayload(msg, NULL));
        floodingTxTerminata[tellTxTerminata.mitt] = tellTxTerminata.serialNumber;
                                            // segno flooding corrente
        atomic{busy = FALSE;} // radio libera
    }
}

event void AMSendSpedisco.sendDone(message_t* msg, error_t error) {
    if (call Packet.payloadLength(msg) == sizeof(BlinkToRadioMsg)) {
                                            // sendDone inoltro serie di pkt dati
        singoliPkt++;                 // incremento il contatore di pkt inviati con stesso livello di potenza
        atomic{busy = FALSE;}       // radio libera
    }
}

event void AMControl.startDone(error_t error) {
}

event void AMControl.stopDone(error_t error) {
}

event void Write.eraseDone(error_t error) {
```

```
    }

    event void CC2420Config.syncDone( error_t error ) {
        call AMControl.start();
    }

    event void Write.syncDone(error_t error) {
    }

}
```

BIBLIOGRAFIA

[1] D. Lymberopoulus, Q. Lindsey, A. Savvides.
An empirical analysis of radio signal strength variability in IEEE 802.15.4 networks using monopole antennas
Embedded Networks and Applications Lab, Yale University, New Haven

[2] Moteiv Corporation.
Tmote Sky Datasheet

[3] Chipcon AS Products.
CC2420 Datasheet

[4] ST Microelectronics.
M25P80 Datasheet

[5] Texas Instruments.
MSP430 Datasheet

[6] Philip Levis.
TinyOS Programming

[7] H. Hasheimi.
The indoor radio propagation channel
Proceeding of IEEE

[8] L. E. Foong, C. W. L. Xiao.
A study of radio signal behaviours in complex environments
Computer Science Department, Michigan State University

[9] G. Zhou, T. He, S. Krishnamurthy, J. Stankovic.
Impact of radio irregularity on wireless sensor networks
Department of Computer Science, University of Virginia

[10] J. Ma, Q. Chen, D. Zhang, L. M. Ni.
An empirical study of radio Signal Strength in sonsor networks using MICA2 nodes
Department of Computer Science end Engineering, Hong Kong University of Science and Technology

[11] LAN/MAN Standards Committee.
IEEE Standards 802.15.4. Technical report
IEEE Computer Society

Elenco delle Figure

Elenco delle Tabelle