UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

TESI DI LAUREA

GESTIONE A MINIMA POTENZA DEL TRAFFICO DATI DI UNA RETE DI SENSORI WIRELESS

RELATORE: CH.MO PROF. LUCA SCHENATO

LAUREANDO: RICCARDO SALA

CORSO DI LAUREA IN INGEGNERIA DELLE TELECOMUNICAZIONI

ANNO ACCADEMICO 2005-2006

Ai miei genitori

Indice

Abstract

Lo scopo di questo lavoro è la realizzazione di una rete di sensori wireless per monitoraggio ambientale, che minimizzi il consumo energetico mantenendo un'elevata affidabilità delle comunicazioni. La riduzione dei consumi energetici coinvolge aspetti riguardanti sia il software che l'hardware dei dispositivi. Solitamente non è possibile intervenire sull'hardware, d'altro canto se il dispositivo è ben progettato non è neanche necessario farlo.

I moderni dispositivi per reti di sensori wireless: i motes, sono gia predisposti per un basso consumo energetico, inoltre esportano delle interfacce software che consentono ai programmatori di intervenire pesantemente sul consumo. Qui verranno sviluppate tecniche software per l'ottimizzazione spinta dei consumi.

L'obiettivo è stato raggiunto procedendo essenzialmente in due fasi. Prima di tutto si interviene sulla potenza dei trasmettitori cercando di minimizzare il raggio di azione dei dispositivi al minimo indispensabile per la comunicazione. Si ottengono in questo modo due importanti vantaggi: la riduzione delle interferenze tra i sensori ed un minor consumo energetico, data la minore potenza irraggiata.

In un secondo momento si organizzano in modo sincrono le comunicazioni sulla rete. Si suddivide il tempo in cicli e i cicli in slots. Durante ogni slot i motes possono essere coinvolti in una ed una sola delle seguenti attività: ricezione, trasmissione oppure stand-by. Come vedremo, per la maggior parte del tempo i motes si trovano in stand-by consentedo loro di consumare notevolmente meno.

Il metodo proposto è stato testato su un'applicazione per il rilevamento delle temperature ambientali in ambito residenziale.

Dalle prove effettuate si evidenziano le ottime caratteristiche di affidabilità dell'algoritmo. Infatti la perdita media di pacchetti dopo l'invio di più di mille letture per ogni sensore si è attestata intorno allo 0.1%.

Il consumo medio di potenza da parte dei motes nel caso peggiore, ovvero per il motes

con il consumo più elevato è risultato di 13.4 mW comunque inferiore dell’80% rispetto al consumo senza funzioni di risparmio enegetico.

Introduzione

Il crescente sviluppo tecnologico ha consentito la realizzazione di circuiti integrati a scala di integrazione sempre maggiore a costi sempre minori. Questo comporta innumerevoli vantaggi. Innanzitutto si possono ridurre notevolmente le dimensioni dei dispositivi che quindi diventano sempre più portatili. Diminuiscono i consumi energetici perchè potendo realizzare percorsi elettrici più corti aumenta la conducibilità. Si possono inoltre aumentare le frequenze di funzionamento dei dispositivi, quindi otteniamo maggiori capacità di calcolo e la possibilità di trasmettere a frequenze più elevate con un migliore rapporto segnale/rumore.

Tanta evoluzione tecnologica ha rivoluzionato anche il mondo delle *Sensor Network* ovvero reti di sensori. Una rete di sensori è costituita da una serie di sensori "intelligenti" con una minima capacità di calcolo, dislocati nei punti di interesse. I principali compiti di un sensore sono quelli di lettura e campionamento delle grandezze fisiche di interesse ed il loro invio ad una base station. Sarà poi la base station, solitamante costituita da un calcolatore, a memorizzare i dati e a compiere le eventuali elaborazioni.

Sono ad oggi disponibili reti di sensori cablati su sistemi a bus come PROFIBUS [5], WorldFIP [12] o CAN [14]. Gli impieghi di queste reti di sensori sono molteplici, si va dalla gestione e controllo dell'automazione nei processi industriali, al monitoraggio ambientale, alla telesorveglianza e altro ancora. Poichè i sensori sono in grado di svolgere le loro attività autonomamente, si è voluto compiere un passo successivo e renderli anche fisicamente indipendenti, pertanto si è passati da un sistema a bus, ad un sistema wireless a basso consumo che possa essere alimentato anche a batterie. Sono nate così le *Wireless Sensor Network.*

Una *Wireless Sensor Network* è appunto una rete di sensori collegati tra loro mediante comunicazioni radio. Il componente fondamentale di questa rete è il *mote*. Esso è dotato di tutto l'hardware necessario per gestire il campionamento di sensori, possiede una seppur minima unità di elaborazione al suo interno, una memoria dove memorizzare temporaneamente i dati provenienti dai sensori, e soprattutto il necessario per comunicare

via radio. Una differenza fondamentale delle reti wireless rispetto a quelle su bus è che per estendere la portata della rete è necessario implementare un sistema di comunicazioni multi-hop. In una rete multi-hop i dati provenienti da un motes possono sfruttare le capacità radio anche degli altri motes per raggiungere la base station. In questo modo ogni motes si trova a funzionare oltre che da sensore anche da ripetiore.

La possibilità di poter svincolare la rete di sensori dal cablaggio presenta innumerevoli vantaggi, e nuove possibilità di impiego. Si possono ad esempio distribuire i sensori anche in ambienti dove la stesura di una connessione sarebbe difficoltosa per le condizioni dell'ambiente o per la presenza di sostanze chimiche che deteriorerebbero il cavo o lo renderebbero particolarmente costoso. Si possono dotare di sensori anche apparecchiature in movimento. Si può effettuare il tracking dei dispositivi, ovvero si dispongono alcuni motes in posizioni note ed altri li si possono posizionare su apparati in movimento. In base alla traccia radio rilevata dai motes fissi si può risalire alla traiettoria dei dispositivi mobili. Un altro aspetto da non sottovalutare assolutamente è la possibilità di intervenire rapidamente sulla loro disposizione: non essendo vincolati a nulla possono essere semplicemente spostati. Il basso costo e l'assenza del cablaggio ne rende possibile una fitta distribuzione, fornendo una quantità di dati notevolmente superiore ai sistemi tradizionali.

Il limite maggiore di questi dispositivi è forse rappresentato dal fatto che essendo per lo più alimentati a batterie hanno un'autonomia limitata e la sostituzione delle batterie spesso può avere costi proibitivi per la loro collocazione. Occorre studiare un sistema che consenta di ridurre il consumo energetico dei dispositivi per ottenere un'autonomia che renda conveniente il loro utilizzo. Risulta infatti poco opportuno utilizzare i protocolli e le tecniche adottate dalle reti di sensori tradizionali in cui i consumi non sono mai stati un problema primario. Bisogna implementare altre soluzioni pensate appositamente per questo genere di reti.

Sono state realizzate in alcuni casi anche reti di sensori wireless autoalimentate, quindi a manutenzione zero, come quella ad energia solare proposta in [15].

L'hardware come detto è già stato progettato ottimizzato per il risparmio energetico, tuttavia anche il software deve essere adeguato di conseguenza. A livello di protocollo di comunicazione viene utilizzato l' IEEE 802.15.4 pensato apposta per questi dispositivi. Senza dilungarci molto su questo protocollo che verrà analizzato al capitolo successivo, possiamo dire che implementa una gestione a basso consumo delle comunicazioni radio oltre a tutto quello che serve per le trasmissioni wireless: crittografia, gestione del traffico sincrono, comunicazione affidabile, CRC, ecc. ecc.

Ad un livello superiore troviamo il software, che in base a come è realizzato decide

in gran parte il consumo finale del dispositivo e quindi la sua autonomia. I motes hanno varie modalità di funzionamento, ognuna con un proprio consumo. Ad esempio, quando non serve, si può decidere di spegnere il chip radio che è uno dei componenti che consuma di più, oppure si può mettere tutto il motes in stand-by durante i periodi di inattività. Una cosa è certa, se non si adottano tecniche opportune, il consumo del dispositivo anche se basso, riduce l'autonomia a poche decine di ore che per molte applicazioni possono risultare insufficienti.

Questo problema è particolarmente sentito in applicazioni di acquisizione ciclica prolungata di dati come ad esempio il monitoraggio ambientale. In questi casi i motes devono stare accesi anche per giorni ad acquisire i dati provenienti dai sensori. Anche se i motes sono accesi l'attività in realtà è assai limitata, per la maggior parte del tempo i motes non hanno nulla da fare, pertanto possono essere messi in stand-by. Lo stesso discorso vale per l'integrato radio: è molto importante che venga spento quando non viene utilizzato.

Un altro parametro su cui intervenire per ridurre i consumi è la potenza dei trasmettitori. Solitamente i motes consentono di impostare la potenza del trasmettitore su più livelli. Come spesso accade i motes nelle reti di sensori wireless si trovano a distanze più corte di quanto non siano i raggi di azione dei loro trasmettitori. Ad esempio, se volessimo misurare la temperatura delle stanze di un edificio dovremo mettere un motes per ogni stanza, anche se il raggio di azione di un motes è sufficiente per coprire tutta la superficie dell'edificio. In questo caso è possibile ridurre la potenza dei trasmettitori.

É proprio su queste considerazioni che si basa il metodo proposto. Come vedremo, per ridurre il consumo si è pensato di ridurre automaticamente le potenze dei trasmettitori al minimo indispensabile per formare la rete di sensori, poi in un secondo momento si organizzano le comunicazioni in modo da tenere spenti il più possibile i motes.

Organizzazione del lavoro

Primo capitolo. In questo capitolo viene fornita una panoramica del software e dell'hardware a disposizione. Viene brevemente esposto l'ambiente di sviluppo ed il simulatore, il protocollo di rete usato ed il linguaggio di programmazione.

Secondo capitolo. Qui viene descritta dettagliatamente la procedura di minimizzazione della potenza dei trasmettitori. Prima di tutto i motes devono scoprire quali sono i loro vicini e come possono comunicare tra di loro, poi attraverso lo scambio reciproco di queste informazioni, tramite un algoritmo di routing distribuito creano le connessioni logiche tra i dispositivi che serviranno poi per trasmettere i dati alla base station.

Terzo capitolo. Una volta creati i collegamenti logici, si deve organizzare la trasmissione

dei dati. Questo capitolo descrive il metodo scelto per questo compito. Come vedremo si è scelto di suddividere il tempo in cicli di acquisizione ed i cicli in brevi time-slot. Lo scopo di questa suddivisione è quello di definire delle porzioni temporali nelle quali i motes devono stare accesi e altre nelle quali possono essere messi in stand-by.

Quarto capitolo. All'interno del quarto capitolo viene svolta l'analisi delle prestazioni della soluzione proposta. Vengono riportati i consumi medi di energia e l'affidabilità della comunicazione facendo un paragone con le soluzioni attualmente presenti in letteratura. Per semplicità di trattazione vengono considerate due semplici reti di esempio: una costituita da una fila di motes in linea retta, e un'altra con in motes disposti ai vertici di una stella. Le situazioni di impiego reali saranno una combinazione di queste due topologie, pertanto non è difficile estendere la trattazione agli specifici casi di interesse.

Quinto capitolo. Qui vengono riportati i risultati di una serie di prove sperimentali effettuate con la procedura elaborata in questa tesi. Le prove prevedevano l'acquisizione periodica della temperatura dei locali di un appartamento. Per ogni prova vengono riportati i grafici delle rilevazioni e della disposizione dei motes. Nelle prime prove si è anche messo in evidenza il cammino multi-hop scelto dai vari motes per raggiungere la base station.

Sesto capitolo. Conclusioni. Verranno brevemente riassunti i risultati ottenuti e verranno presentati alcuni possibili sviluppi. In particolare si parlerà della necessaria sinronizzazione dei motes oltre che di un possibile accorgimento per ridurre ultriormente anche se di poco, i consumi.

Capitolo 1

Wireless Sensor Networks

In questo capitolo viene fatta una breve panoramica sull'hardware ed il software a disposizione. In particolare, vengono introdotti i dispositivi usati per le sperimentazioni: i *Tmote Sky* della *MoteIV*. Si passerà poi ad illustrare il protocollo di comunicazione usato: lo standard *IEEE 802.15.4* nelle sue caratteristiche essenziali, ed infine verranno introdotti il sistema operativo dei motes ed il loro linguaggio di programmazione.

1.1 Standard e protocolli

Già nell'introduzione abbiamo visto cosa si intende per *Wireless Sensor Network*, adesso analizzeremo in dettaglio quali sono gli standard attualmente utilizzati per implementare queste tipologie di reti. Lo standard uniformemente riconosciuto da tutti i produttori è il cosidetto *ZigBee*. Nato da un'alleanza tra i principali produttori di hardware per queste reti, *ZigBee* definisce tutto il necessario per poter implementare robuste applicazioni standard per *Wireless Sensor Network*. In Figura 1.1 possiamo vedere lo stack di rete usato dai motes. Sono riportati i vari livelli e gli standard ai quali aderiscono. Si vede che *ZigBee* definisce sostanzialmente le API (Application Program Interface), la crittografia e la tipologia logica di rete. Per maggiori delucidazioni sullo standard *ZigBee* si faccia riferimento a [16].

Per quello che riguarda le comunicazioni a basso livello *ZigBee* prevede l'uso dello standard IEEE 802.15.4 (vedi [7]). Questo standard è stato concepito appositamante per consentire la connessione di dispositivi wireless a basso consumo. Le principali caratteristiche di questo protocollo sono:

- Data-rate possibili: 250 Kbps, 40 Kbps, 20 Kbps

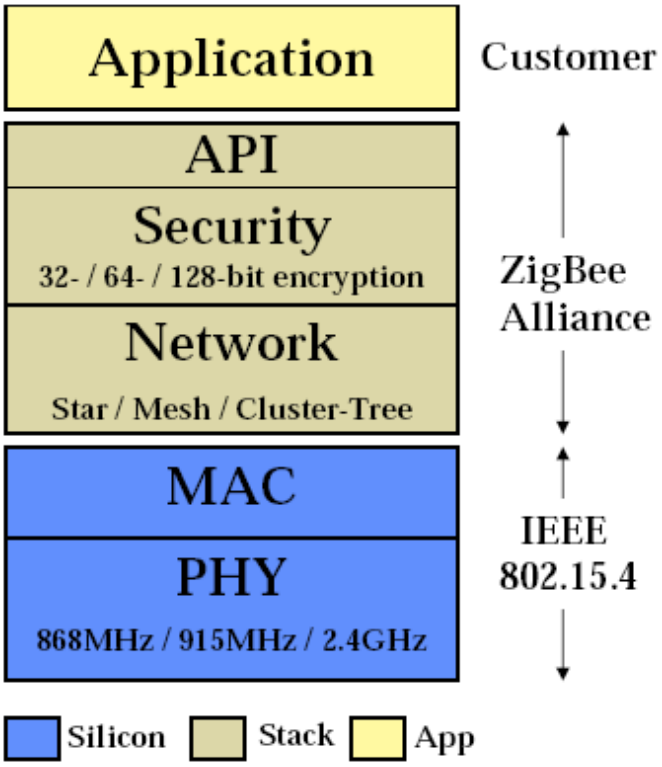

Figura 1.1: *Stack ZigBee. [Yuan Yuxiang, Department of Electrical Engineering - Keio University]*

- supporto per topologie a stella e peer-to-peer
- spazio di indirizzamento corto a 16 bit, o lungo a 64 bit
- uso del CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) per gestire l'accesso al mezzo
- supporto a livello hardware per comunicazioni affidabili con *Acknowledge*
- comunicazioni a basso consumo energetico
- monitoraggio dell'energia e della qualità della comunicazione: ED (*Energy Detection*) e LQI (*Link Quality Detection)*
- possibilità di allocare 16 canali sulla banda ISM (Industrial Standard Medical) a 2450 MHz, 10 sulla 915 MHz ed uno ad 868 MHz.

Una nota sulle frequenze utilizzate è d'obbligo. I dati riportati nel precedente elenco sono quelli definiti dallo standard che deve avere valenza mondiale. Le frequenze ISM ovvero le frequenze non a pagamento purtroppo non sono le stesse in tutto il mondo, ci sono delle piccole differenze. In Tabella 1.1 è riportata un'interessante comparazione tra le frequenze disponibili nei principali stati. Sono inoltre indicati i livelli di potenza massima consentiti per i trasmettitori e gli organi competenti.

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. (IEEE 802.15.4 [7])*

É interessante notare come la banda 902-928 MHz in Europa non sia disponibile e soprattutto come a seconda dello stato possano variare le potenze massime ammissibili e quindi i raggi di copertura dei dispositivi.

Non è difficile inoltre prevedere un'estensione del protocollo alla "prossima" banda ISM, ovvero quella dei 5725-5850 MHz, che per fortuna è disponibile in tutto il mondo.

Le topologie supportate differiscono per il fatto che nella prima topologia tutti i motes comunicano con la base station e solo con essa, quindi sono esplicitamente evitate le comunicazioni punto-punto tra un motes e l'altro. Nella seconda, quella di tipo peer-to-peer tutti i motes possono comunicare tra di loro, e non è detto che ogni motes possa comunicare con la base station. Due esempi di queste topologie sono riportati in Figura 1.2 e Figura 1.3, dove con un cerchietto pieno sono indicate le base-station mentre con un cerchietto vuoto i motes. Nella seconda figura si nota che solo due motes sono direttamente collegati alla base station (link a tratto continuo), mentre gli altri per poter comunicare con essa hanno bisogno di passare da altri motes (link tratteggiati), dando luogo così ad una rete multi-hop.

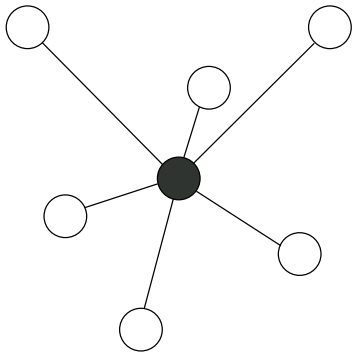

Tabella 1.2: *Topologia a stella.*

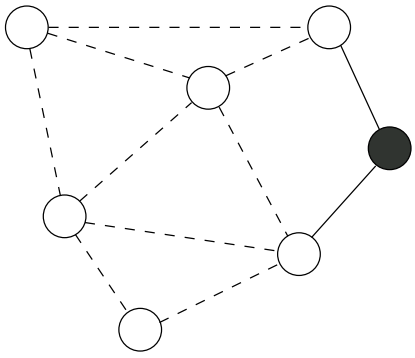

Tabella 1.3: *Topologia peer-to-peer.*

Come abbiamo visto i bit-rate, previsti da questo protocollo non sono particolarmente elevati, si raggiungono al massimo 250 Kbps. Se lo paragoniamo ad altri protocoli che la-

vorano sulle stesse frequenze, vediamo che ad esempio il Bluetooth raggiunge i 750 Kbps mentre l'802.11, cioè il protocollo delle reti wireless di calcolatori raggiunge addirittura i 54Mbps.

La spiegazione di questa scelta è riconducibile al fatto che, al contrario dei suoi "fratelli", 802.15.4 è stato progettato per trasmissioni a basso consumo. Questo vuol dire che sono previsti sistemi di modulazione più lenti, schemi di codifica meno pesanti e pachetti dati più corti, il tutto riduce il consumo energetico della trasmissione a scapito della velocità.

Per fare un esempio concreto possiamo vedere come sono strutturati i pacchetti inviati da 802.15.4. Il protocollo definisce quattro tipi di pacchetto: BEACON, DATA, ACK, MAC COMMAND. Senza soffermarci eccessivamente sui tipi di pacchetti, vediamo come si presenta la struttura di un pacchetto DATA, che sarà poi quello usato dal nostro programma.

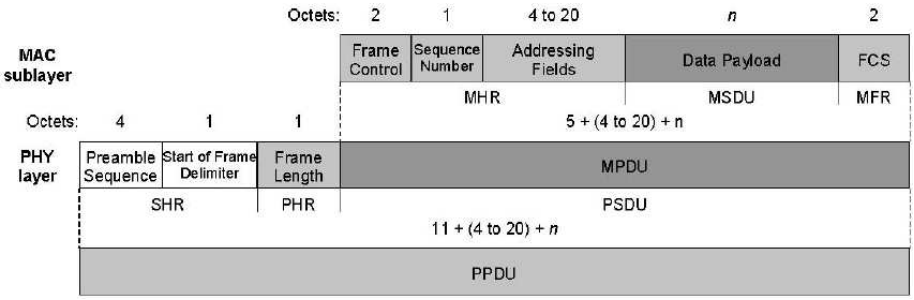

Figura 1.2: *Visione schematica di un DATA frame 802.15.4. (IEEE 802.15.4 [7])*

Dallo schema di Figura 1.2 si vede che l'overhead introdotto dal protocollo è assai limitato, infatti ad ogni pacchetto dati vengono aggiunti solo 15 bytes (se si usano indirizzi corti da quattro bytes). Inoltre la parte contenente le informazioni: il *Data Payload* deve essere al massimo di 127 bytes. Questo limite di fatto impone bassi bit-rate però consente di realizzare ricevitori e trasmettitori con hardware più semplice, buffer più piccoli, e quindi con consumo energetico ridotto.

1.2 Hardware

I motes usati, i *Tmote Sky* (`http://www.moteiv.com/products-tmotesky.php`) della *MoteIV corp.* (`www.moteiv.com`) sono dispositivi di ultima generazione che oltre ad un basso consumo energetico incorporano molte funzionalità aggiunti-

ve rispetto ai precedenti modelli. Vediamo a grandi linee quali sono le caratteristiche principali di questi motes:

- MCU (Micro Controller Unit) MSP430 della *Texas Instruments*, funzionante alla frequenza di 8 MHz. Questa MCU incorpora al suo interno una memoria RAM da 16KB ed una flash ROM da 48KB
- convertitori ADC e DAC a 12 bit, integrati nella MCU
- antenna integrata nel PCB (Printed Circuit Board), con connettore di espansione per antenna esterna. L'antenna interna consente un raggio di azione di 50 metri indoor e 125 outdoor
- basso consumo energetico (vedi Capitolo 4)
- porta USB. Usata sia per programmare il dispositivo che per lo scambio di dati con il calcolatore.
- integrato per comunicazioni wireless su 802.15.4: *Chipcon CC2420*
- sensori di umidità, temperatura e intensità luminosa integrati
- connettore di espansione, compatibile con l'interfaccia I_2C
- 1MB di memoria flash esterna alla MCU, utilizzabile anche per immagazzinare i dati letti dai sensori

Nella Figure 1.3 e 1.4 sono riportate le immagini del lato superiore ed inferiore del motes. Si può notare, evidenziato in rosso, il piccolo sensore di temperatura (ed umidità) usato nelle prove sperimentali del capitolo 5.

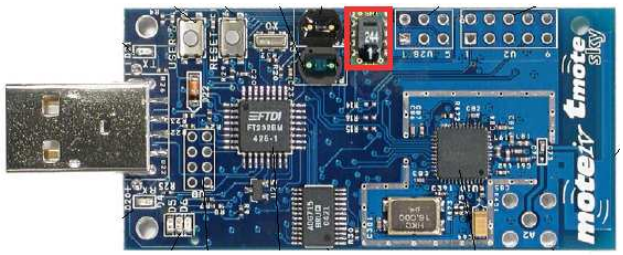

Figura 1.3: *Lato superiore del PCB (a grandezza naturale). (Vedi: [13])*

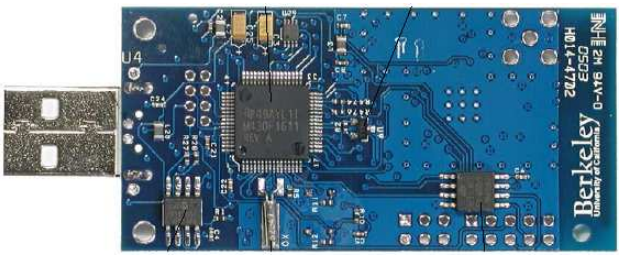

Figura 1.4: *Lato inferiore del PCB. (Vedi: [13])*

L'integrato radio, il *Chipcon CC2420*, è in grado da solo si gestire tutto lo strato fisico (PHY) e buona parte del MAC del protocollo 802.15.4. Implementa a livello hardware la decodifica degli indirizzi, l'invio di dati con acknowledge, la crittografia, il controllo di integrità dei pacchetti e parecchie altre cose. Può essere attivato o disattivato secondo le esigenze e può regolare la potenza di uscita del trasmettitore su 31 livelli diversi. Il fatto che gran parte delle operazioni legate alle comunicazioni radio siano fatte in hardware da un integrato apposito ha il duplice vantaggio di lasciare più libera la MCU per altri compiti e di diminuire i consumi perchè ci sono meno componenti e meno flussi di dati tra MCU e chip radio.

In Figura 1.4 è riportato un tipico pacchetto dati 802.15.4, con evidenziati i campi che sono gestiti dall'hardware (HW) del CC2420 e quelli gestiti dal software (SW). Il campo indirizzi può essere lasciato all'hardware o per esigenze particolari, gestito via software.

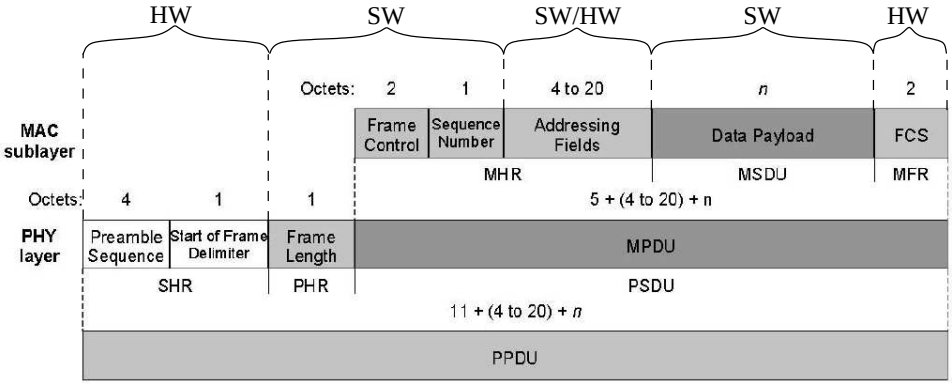

Tabella 1.4: *Visione schematica di un DATA frame 802.15.4 con evidenziati i campi gestiti dal software (SW) e quelli gestiti dall'hardware (HW).*

1.3 Software

1.3.1 Sistema operativo

Vediamo quali devono essere le caratteristiche di un sistema operativo adatto ad un dispositivo di questo genere.

Innanzitutto deve occupare poca memoria, l'MCU ha a disposizione solo 10KB di memori RAM e 48KB di ROM, ed in queste devono trovare posto sia il sistema operativo che le applicazioni. In particolare la RAM è usata anche per ospitare temporaneamente i dati provenienti dai sensori prima che vengano spediti via radio o archiviati nella flash ROM. Indicativamente il sistema operativo dovrebbe mantenersi più piccolo di qualche chilobyte.

Dovrebbe essere in grado di gestire al meglio la concorrenza degli eventi poichè un motes prevalentemente deve rispondere velocemente ad eventi generati dall'hardware e dal software piuttosto che portare a termine lunghe procedure di calcolo. Ad esempio l'arrivo di un pacchetto radio è un evento da gestire rapidamente prima dell'arrivo successivo, così come lo sono le letture dei sensori, o lo scadere dei timer software. Bisogna quindi che il sistema operativo abbia una gestione temporale orientata agli eventi.

Sono inoltre richieste robustezza e stabilità. Questo è vero in generale per ogni sistema, ma diventa più importante per applicazioni come queste in cui l'hardware, non sempre è facilmente raggiungibile per la manutenzione.

La scelta della *MoteIV* è stata di equipaggiate i motes con **TinyOS** (`www.tinyos.net`). TinyOS è un sistema operativo modulare open-source che risponde egregiamente a tutte queste richieste.

Per limitare l'occupazione di memoria TinyOS ha adottato una struttura modulare. Il programmatore ha la possibilità di caricare sul motes solamente i componenti del sistema operativo che effettivamete servono. Ad esempio, se si usano le funzionalità radio si carica l'opportuno modulo, lo stesso dicasi per la porta USB, d'altro canto se non si ha bisogno di qualche sensore, come ad esempio quello di intensità luminosa non si carica il modulo e quindi si risparmia memoria. Sostanzialmente si crea un sistema operativo personalizzato per ogni applicazione.

I moduli esportano *interfacce* software mediante le quali è possibile farli interagire tra di loro. In particolare queste interface sono pensate per lavorare in risposta agli eventi come vedremo nella sezione successiva.

TinyOS è open-source, questo significa che il codice sorgente del sistema stesso è disponibile gratuitamente, e può essere modificato a piacimento dal programmatore. Esiste

infatti una vasta comunità di programmatori che costantemente aggiornano ed espandono TinyOS alla quale chiunque può partecipare. La disponibilità del codice sorgente oltre che garantire trasparenza e sviluppo impossibili altrimenti, fornisce anche una notevole base di apprendimento per chi si avvicina per la prima volta a questo sistema, infatti per realizzare la propria applicazione si può riutilizzare codice scritto da altri ed espanderlo secondo le proprie esigenze.

1.3.2 Linguaggio di programmazione

Il linguaggio di programmazione di TinyOS è *nesC* [9], anche se in realtà con TinyOS non c'è una netta distinzione tra sistema operativo e applicazione.

L'approccio classico per un calcolatore prevede di avere un sistema operativo in esecuzione e da esso caricare il software applicativo. Quindi come spesso accade il sistema operativo è fatto con un linguaggio mentre le applicazioni possono essere fatte indipendentemente con altri linguaggi, basic, java, C ecc. ecc.

TinyOS al contrario, vine compilato assieme all'applicazione. I moduli suddetti sono agganciati all'applicazione. Implementare applicazioni per TinyOS significa combinare opportunamente i moduli del sistema operativo secondo le specifiche richieste aggiungendo il proprio codice. Quindi l'applicazione diventa un tutt'uno con il sistema operativo. Questa soluzione consente di utilizzare al meglio le risorse in termini di efficienza del codice e risparmio di memoria.

Pertanto il linguaggio con cui è realizzato il sistema è anche quello usato per le applicazioni ed è appunto nesC.

NesC è ha una sintassi molto simile al C, è uno dei tanti linguaggi *C-like*. Si differenzia dal C perchè è appositamente pensato per lavorare con i moduli di NesC. Come detto, i moduli esportano delle interfacce che non sono altro che le "specifiche" con cui interagire con i moduli. Funzionano in modo simile alle interfacce java o alle classi astratte del C++. Qui sotto è riportato un esempio di interfaccia, si tratta della *SplitControl* usata per inizializzare molti componenti. Si vede che ci sono comandi (*command*) ed eventi (*event*). Gli eventi sono appunto gli eventi generati dal modulo, mentre i comandi vengono usati per rispondere agli eventi o in generale per far compiere delle azioni al modulo. In questo caso il programmatore può chiamare esplicitamete il comando *init()* per inizializzare il componente e successivamente può intercettare l'evento *initDone()* per sapere quando è terminata l'inizializzazione ed andare avanti con il codice opportuno.

```
interface SplitControl
{
  command result_t init();
  event result_t initDone();

  command result_t start();
  event result_t startDone();

  command result_t stop();
  event result_t stopDone();
}
```

Listato 1.1: *Interfaccia SplitControl*

In Figura 1.5 è mostrato il grafico di una semplice applicazione d'esempio di TinyOS. I moduli sono rappresentati con degli ovali con al centro il nome del modulo, mentre le interfacce sono riportate sugli archi. Ad esempio: il modulo BlinkM è collegato al modulo LedsC tramite l'interfaccia Leds. Questa forte schematizzazione "imposta" dal linguaggio mantiene il programma ordinato e di facile gestione.

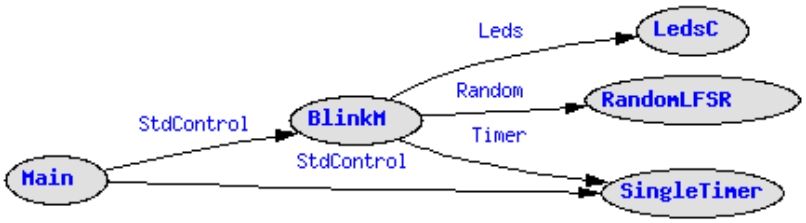

Figura 1.5: *Blink: una semplice applicazione dimostrativa che fa lampeggiare un led sul motes.*

Per una chiara introduzione al linguaggio nesC si può fare riferimento a [9], oppure alla documentazione ufficiale presente in TinyOS.

1.3.3 Il simulatore: TOSSIM

L'ambiente di sviluppo dei TmoteSKY, TinyOS-nesC, come abbiamo visto è un ambiente potente e flessibile. A questi strumenti si aggiunge anche un potente ambiente di simulazione di reti di motes: TOSSIM [10].

TOSSIM simula al calcolatore il comportamanto di un'intera *Wireless Sensor Network*, consentendo di esaminare sul terminale del calcolatore una discreta quantità di

informazioni su quello che sta accadendo ai motes della rete simulata. Queste informazioni possono essere ad esempio il contenuto dei pacchetti inviati o ricevuti dai motes, lo stato dei leds, dei timer o le letture dei sensori (anch'essi simulati)

L'aspetto più interessante di questo simulatore è che al contrario di altri prodotti simili non crea un ambiente che simuli la realtà in cui far girare l'applicazione emulata. Piuttosto TOSSIM cerca di emulare il comportamento delle applicazioni scritte in nesC a livello di bit. Di ogni componente hardware presente sul motes è fornito un modello software che ne riproduce il comportamento. Ad esempio abbiamo il modello della MCU ed il modello dell'integrato radio. L'applicazione vine compilata con questi moduli software in modo nativo per il calcolatore: in sostanza viene creato un eseguibile che simula un intera rete.

Le comunicazioni radio vengono simulate semplicemente impostando il bit-error dei links tra un motes e l'altro. Questo approccio a prima vista semplicifativo, consente invece un'emulazione abbastanza precisa della realtà. L'approccio classico seguito da altri simulatori, prevede di simulare i singoli fenomeni, come la percentuale di pacchetti persi, il fenomeno del nodo nascosto (vedi [2]) e altri.

Con l'approccio seguito da TOSSIM la simulazione di questi fenomeni avviene di conseguenza. Ad esempio, non occorre simulare la perdita di pacchetti della comunicazione, basta simulare il bit-error del canale, la perdita di pacchetti avverà di conseguenza. L'unica cosa che deve essere precisa è la descrizione statistica di come varia il bit-error in funzione della distanza reale tra i motes. Ad esempio se vogliamo simulare le comunicazioni tra due motes disposti a 50 metri l'uno dall'altro dobbiamo sapere qual'è il bit-error atteso del link per poterlo inserire nel simulatore.

Questa descrizione statistica è stata raccolta in base ad una serie di prove sperimentali eseguite in [4], per comodità il grafico delle prove è riportato in Figura 1.6.

In queste prove sperimentali si sono ricavate le perdite di pacchetti di vari link. In base a questi dati e conoscendo la struttura dei pacchetti del protocollo 802.15.4 sono stati successivamente ricavati i bit-error. Ogni punto nero è una prova. La riga rossa mostra che la perdita di pacchetti (loss-rate) in funzione della distanza segue un andamento grossomodo a gaussiana. Si nota anche come verso il centro ci sia una forte componente casuale, ovvero un link lungo 20 piedi (circa 6 metri) può presentare una perdita di pacchetti, del 100% come del 10%.

Questi risultati sperimentali sono stati approssimati con una funzione ed inseriti in TOSSIM.

Durante la stesura del codice dell'applicazione si è potuto usare il simulatore ma

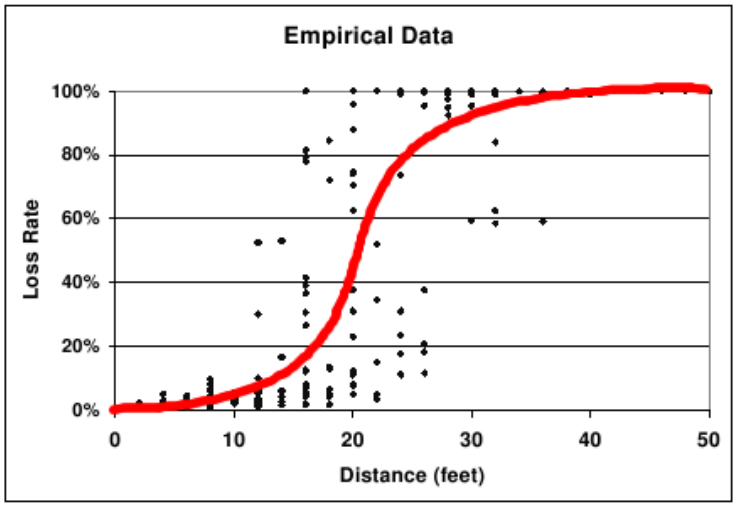

Figura 1.6: *Perdita di pacchetti del link in funzione della distanza.*

solo fino ad un certo punto. Il problema è che come detto nell'introduzione, per ridurre il consumo energetico si fa uso della riduzione della potenza dei trasmettitori dei motes. Purtroppo questa funzione non è utilizzabile nel simulatore. Per poterlo fare sarebbe stato necessario creare un complesso programma che si interfacci con il simulatore variando a *runtime* i parametri dei link. Si è preferito allora proseguire con la sperimentazione diretta sull'hardware. A parte questa lacuna TOSSIM rimane un ottimo strumento per lo sviluppo e il *debugging* delle applicazioni per TinyOS.

Dopo questa breve panoramica sugli strumenti a disposizione possiamo iniziare dal prossimo capitolo a vedere come funziona l'applicazione.

Capitolo 2

Impostazione adattiva a minima potenza dei link: AMPL

La prima fase di set-up della rete di sensori consiste nell'individuazione da parte dei dispositivi della topologia della rete. In questo capitolo vengono descritte quelle che sono le procedure eseguite dai dispositivi che portano alla completa conoscenza dei loro vicini e del *costo* per raggiungerli.
Chiameremo questa prima procedura *AMPL*: *Adaptive Minimum Power Link.*
Al termine di questa fase, viene creato un albero di copertura minimale o *MST* (*Minimum Spanning Tree*) della rete di sensori, secondo un'opportuna funzione costo che definiremo più avanti.
Successivamente le comunicazioni tra i sensori potranno essere di tipo punto-punto e non solo broadcast. Questo ci consente di implementare una rete di sensori multi-hop, in cui i dati "saltano" da un sensore all'altro fino a raggiungere la base-station.

2.1 Discovery

Prima di tutto è necessario che tutti i dispositivi sappiano quando devono iniziare il processo di creazione della sensor network. Bisogna in qualche modo impartire un comando di start che arrivi a tutti i dispositivi ed avvi la procedura.

Occorre qui formulare alcune ipotesi fondamentali. La prima è che tutti i dispositivi siano in grado di comunicare almeno alla massima potenza in modo tale che la topologia della rete risulti un grafo connesso a cui appartenga la base station. La seconda è che la topologia sia supposta invariante nel tempo, così come le caratteristiche del mezzo trasmissivo.

Il dispositivo che si occupa di dare lo start è la Base Station, e lo fa inviando in broadcast un particolare pacchetto: un pacchetto di tipo *discovery*. Nel messaggio di discovery l'unico dato contenuto è l'indirizzo del mittente poichè la funzione di questo messaggio è solo quella di avvertire che il setup della rete è iniziato.

All'inizio tutti i sensori sono in ricezione, e con i trasmettitori impostati alla massima potenza. I sensori che sono nelle vicinanze della base station ricevono il messaggio di *discovery* e lo replicano sempre in broadcast agli altri. Viene realizzato il flooding della rete.

Le comunicazioni in queste prime fasi non possono che avvenire in broadcast in quanto ogni nodo non ha ancora cognizione di quali siano i suoi vicini.

Per assicurarsi che il comando arrivi a tutti i nodi della rete il messaggio di *discovery* viene reinviato N_D volte. Per diminuire le possibilità di collisione ogni reinvio vine fatto con un ritardo t_r casuale uniformemente distribuito nell'intervallo: $t_r \in U[0, T_D]$. La scelta di T_D deriva da considerazioni empiriche. Supponedo che un nodo abbia N_k vicini, il traffico medio sulla rete al nodo in questione risulta: $\lambda = (N_k + 1)\frac{2}{T_D}$. Poichè per inviare un pacchetto servono almeno $10\,ms$, bisogna scegliere T_D in modo che λ risulti inferiore ad $\frac{1}{10^{-3}}$.

Il tempo di completamento di questa fase dipende oltre che dalle costanti T_D e N_D dalla dimensione della rete in termini di numero di hops massimo che occorre fare per raggiungere i sensori più lontani dalla base station. Supponendo che il messaggio si propaghi entro il secondo reinvio, come avviene in pratica, otteniamo che la durata media è

$$T_{DISC} = \frac{T_D}{2} N_{hops}$$

Nelle simulazioni effettuate con $T_D = 250ms$, $N_D = 3$ ed una diecina di motes, anche posti in fila, questo tempo si è sempre mantenuto entro i tre secondi. Il flow-chart di questa procedura è riportato in appendice B, Figura B.1.

2.2 Pinging

Dopo un tempo T_{DISC} che da modo alla rete di propagare il *discovery* a tutti i dispositivi, i nodi iniziano la trasmissione dei pacchetti di *ping*.

Lo scopo dei messaggi di *ping* è quello di testare la qualità del link in termini di probabilità di ricezione di pachetto tra un nodo e quelli che saranno i suoi vicini in modo da poter stabilire l'identificativo dei vicini e per quali livelli di potenza del trasmettitore sono raggiungibili.

L'algoritmo procede così: si invia un blocco di N_P *ping*, si scala di un valore il livello di potenza e si invia un altro blocco di *ping*, e così via fino ad esaurire i livelli. Come visto in precedenza il circuito integrato che si occupa delle trasmissioni, il *CC2420* consente di variare la potenza del trasmettitore su una gamma di 31 livelli mappati su una scala di $25dBm$. Per rendere più veloce la convergenza dell'algoritmo di è pensato di limitare i livelli disponibili ad otto, mappandoli sempre sui $25dBm$, come mostrato in Tabella 2.1.

PA_LEVEL	Potenza di uscita [dBm]	Consumo di corrente [mA]
31	0	17.4
27	-1	16.5
23	-3	15.2
19	-5	13.9
15	-7	12.5
11	-10	11.2
7	-15	9.9
3	-25	8.5

Tabella 2.1: *Livelli di potenza dei trasmettitori. Nella prima colonna sono riportati i valori da passare alla funzione che imposta la potenza.*

Il campo dati di un pacchetto di *ping* è costituito da tre campi: l'ID (identificativo) del dispositivo che ha trasmesso il *ping*, il livello di potenza a cui è stato trasmesso e il numero del pacchetto, per ora usato solo a scopi di debugging. L'invio dei *ping* vine effettuato ogni T_P millisecondi con una lieve deviazione casuale del 10% massimo per evitare il più possibile collisioni. T_P non può essere scelto tanto piccolo, perchè in questo tempo il nodo deve trasmettere un pacchetto ed anche ricevere quelli provenienti dagli altri nodi. Sperimentalmente si è visto che mantenendo T_P sopra ai $500ms$ non si hanno rilevanti perdite di pacchetti per collisioni.

E' ora necessario un feedback che consenta ad un nodo di sapere quanti e quali dei suoi *ping* sono arrivati a destinazione e soprattutto a chi sono arrivati.
In un primo momento si era pensato ad un feedback immediato tramite degli *acknowledge* di risposta, ossia ogni nodo all'arrivo di un *ping* rispondeva con un *ack*. In questo modo si poteva interrompere l'invio dei ping quando, dopo un certo *time out* non si ricevevano più *ack*. La soluzione però non si è rivelata molto brillate. Il problema è che l'invio di un *ack* in risposta ad ogni *ping* aumenta drasticamente il traffico della rete, che essendo *wireless* avviene su un unico mezzo condiviso: l'etere. Infatti, supponendo ci siano N

dispositivi comunicanti l'invio di N_p *ping*, uno per ogni dispositivo, comporta l'invio di

$$N_a = N_p(N_p - 1)$$

ack di risposta. Un aumento del traffico significa anche un aumento delle probabilità di collisione dei pacchetti, con conseguente aumento delle ritrasmissioni per effetto del *CSMA* delle schede, quindi una maggiore perdita di pacchetti sia di *ping* che di *ack*. L'arrivo o meno dei *ping* a destinazione è usato per valutare l'affidabilità del link quindi è fondamentale in questa fase garantire la minor perdita possibile di pacchetti per effetto delle collisioni, altrimeni si andrebbe a sfalsare il risultato. In sostanza si vuole che i pacchetti vengano persi solo a causa del rapporto segnale-rumore (*SNR*) del mezzo.

L'invio degli *ack* non va bene perchè oltre ad aumentare a dismisura il traffico come visto, raddoppia anche la probabilità di perdita di un *ping*, infatti il *ping* vine considerato non arrivato sia che si sia perso effettivamente il *ping* che l'*ack*. Non solo, nell'istante in cui un nodo trasmette non può ricevere contribuendo ulteriormente alla perdita di pacchetti trasmessi o ricevuti. Anche il vantaggio di sospendere l'invio dei *ping* quando non arrivano più *ack* si è rivelato inutile in quanto ogni nodo non può sapere nulla degli altri e quindi deve aspettare comunque che tutta la rete abbia terminato questa fase per procedere alla fase successiva.

Si è optato allora per un sistema di feedback differito, ossia vengono inviati prima tutti i *ping* e poi in un secondo momento tutti i feedback. In pratica ogni dispositivo man mano che arrivano i *ping* aggiorna una tabella (*pingTable*) nella propria memoria, in cui sono mantenute le informazioni riguardo i propri vicini. Un esempio è mostrato nella Tabella 2.2.

	sourceMoteID	**powerLevels**							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0x0005	0	0	18	20	20	20	20	20
2	0x0002	0	17	19	20	20	20	20	20
3	0x0007	20	20	20	20	20	20	20	20
4	0x0009	0	0	0	0	17	20	20	20

Tabella 2.2: *Esempio di pingTable*

All' arrivo del primo *ping* da parte di un vicino si crea un nuovo record della tabella mentre ai successivi si oggiornano i record. Il primo campo che compare è l'indirizzo del vicino (*sourceMoteID*), mentre negli altri campi ci sono i contatori che tengono traccia

del numero di *ping* arrivati dal vicino suddivisi per livello di potenza. Le potenze della Tabella 2.1 sono state caricate in un vettore, in Tabella 2.2 al posto delle poteze effettive compaiono gli indici di questo vettore.

Nell'esempio si vede che dal primo vicino, il cinque, il nodo ha ricevuto 0 *ping* a potenza 0 che è la più bassa, 0 a potenza 1, 18 a potenza 2, 20 a potenza 3, ecc. ecc.

La *base station* pur non inviando *ping* crea comunque la propria tabella *pingTable* da inviare ai dispositivi che le sono prossimi.

Una serie di osservazioni sperimentali ha dimostrato che la probabilità di perdita di pacchetto è modellabile come una gaussiana "stretta", pertanto il link ha un comportamento del tipo "o funziona bene o funziona male", anche se raramente, per opportune distanze si può presentare una via di mezzo. Questo comportamento è visibile anche osservando la tabella 2.2.

2.3 Creazione della tabella delle adiacenze

Dall'arrivo dell'ultimo ping si attende un tempo PT_D scaduto il quale i dispositivi inviano i record di queste tabelle ai rispettivi destinatari, cioè ai mittenti dei ping. Il traffico generato da questi invii è estremamente ridotto, e soprattutto non si va a sovrapporre con quello dei *ping*.

Il calcolo del tempo di PT_D dipende da T_P, N_P e dal numero di livelli di potenza che si è deciso di considerare, cioè otto. Quindi

$$PT_D = 8\, N_P\, T_P(1 + 0.1)$$

lo 0.1 che compare nella formula è dovuto all' incertezza imposta del 10% su T_P.

Questi invii vengono effettuati ad intervalli di T_T millisecondi (un secondo nelle simulazioni) sempre per evitare collisioni derivanti dall'eccessivo traffico e viene richiesta inoltre la conferma tramite acknowledge. Questo perchè è estremamante importante garantire l'arrivo delle tabelle ai destinatari, altrimenti si rischia di scartare dei link che magari erano ottimi. In caso di mancata ricezione dell'acknowledge entro un secondo, si ha la ritrasmissione fino ad un massimo di $N_T = 5$ volte. La durata di questa fase dipende essenzialmente dal numero dei vicini e dalle ritrasmissioni che si sono dovute effettuare. Solitamente non sono necessarie ritrasmissioni a meno che la topologia della rete non sia cambiata, ma questo non è contemplato nelle ipotesi iniziali, quindi indicativamante dura un numero di secondi pari al numero dei vicini del motes con più vicini.

Alla ricezione dei record della *pingTable* da parte dei vicini i dispositivi possono crearsi la tabella delle adiacenze che contiene le informazioni su quali siano i vicini

e a quali potenze sia necessario impostare i trasmettitori per comunicare con loro. In Tabella 2.3 viene riportato un esempio di tabella delle adiacenze.

Per poter creare questa tabella è necessario definire quando un link ad una data potenza è affidabile o meno. Si è scelto di imporre una soglia del 10% sui ping arrivati, cioè una coppia link/potenza è ritenuta affidabile se non si è perso più del 10% dei ping.

	Destinazione	**Potenza**
1	0x0003	2
2	0x0005	1
4	0x0004	1
5	...	...

Tabella 2.3: *Esempio di tabella delle adiacenze*

Nella prima colonna si trova l'ID del vicino mentre nella seconda c'è il livello di potenza necessario per raggiungerlo. In Appendice B, Figura B.2, si trova il flow-chart di questa procedura.

2.4 Routing

Un aspetto fondamentale delle reti multi-hop è che al contrario delle tipologie a stella, i motes usano gli altri motes per far arrivare i dati alla base station aumentando in questo modo il raggio di azione della rete.

Per fare questo è necessario che ogni sensore individui un "percorso" attraverso gli altri sensori che conduca alla base station. Questo compito è svolto appunto dall'algoritmo di routing.

Poichè le scelte ad ogni hop possono essere molteplici bisogna fissare una funzione di costo del link. L'algoritmo scelto cercherà di individuare il percorso che porta alla base station minimizzando il costo ***totale*** del percorso.

Come funzione di costo si è scelto di minimizzare il consumo di energia quindi il livello di potenza dei trasmettitori. Tuttavia dimiuendo le potenze dei trasmettitori le celle di comunicazione dei dispositivi diminuiscono il loro diametro causando un aumentando del numero di hops necessari per far giungere i dati alla base station.

La letteratura esistente è prodiga di algoritmi di routing, per una lista non certamente completa si può fare riferimento ad [11].

Nel nostro caso la scelta è ricaduta sul *Link State Routing* per una serie di motivi:

parte dell'algoritmo è già implementata nelle fasi precedenti, è un algoritmo distribuito quindi non necessita di un nodo della rete che faccia da coordinatore, è ampiamente utilizzato da molte applicazioni e non è particolarmente complesso da implementare. Per una descrizione competa dell'algoritmo si faccia riferimento sempre ad [11], in particolare al cap. 5.2.5 .

La nostra implementazione procede nel seguente modo. Ogni dispositivo crea una struttura, *baseStationPath* (vedi Appendice C), in cui ci sono le informazioni in suo possesso per raggiungere la base, che sono: il prossimo vicino nel percorso verso la base station, il costo totale del percorso fino al nodo e il numero di hop necessari. Quest'ultimo parametro per il momento non viene utilizzato se non a scopi informativi, però potrebbe servire in futuro se si decidesse di minimizzare il numero di hops anzichè il consumo.

Scaduto un certo tempo necessario per consentire a tutta la rete di aver creato queste strutture, i nodi che hanno come vicino la base station inviano ai propri vicini la *baseStationPath*, i vicini ricevuta l'informazione aggiornano la propria *baseStationPath* e la inviano ai loro vicini. A questo punto ogni volta che un sensore riceve una *baseStationPath* aggiornata da un suo vicino può fare due cose: se il percorso risulta migliore di quello della propria *baseStationPath* la aggiorna e invia l'aggiornamento ai vicini, altrimenti ignora il pacchetto. I reinvii non sono immediati ma come al solito sono differiti di un ritardo casuale per diminuire il traffico della rete.

Un'ulteriore vantaggio nell'uso di AMPL è che riducendo il raggio di azione dei motes, diminuiscono anche i nodi che possono interferire a tutto vantaggio dell'affidabilità dei link.

Purtroppo non esiste un modo per sapere con certezza quando tutti i nodi hanno terminato di scambiarsi queste informazioni, dipende infatti dalla topologia della rete oltre che dal numero di nodi. Occorre fissare un time-out in base a considerazioni empiriche.

Termiata questa fase ogni nodo può comunicare con la base station lungo un percorso a bassa potenza costruito utilizzando gli altri nodi come ripetitori.

Capitolo 3

Sincronizzazione e gestione del traffico ciclico

Una volta creati i collegamenti e impostate le potenze dei trasmettitori non rimane che inviare i dati. Per farlo però occorre fissare le modalità e i tempi degli invii. La questione non è banale, infatti bisogna tenere a mente che lo scopo primario è quello di minimizzare il consumo di energia dei dispositivi, quindi non è possibile tenerli sempre accesi. D'altronde si tratta di una rete multi hop quindi i sensori sono usati anche da ripetitori, ma finchè un sensore è spento, o meglio in stand-by, non riceve e quindi non può fare da ripetitore per gli altri. Vedremo come la questione è stata risolta sincronizzando la trasmissione dei dati in maniera ciclica.

3.1 Periodicità e latenza delle comunicazioni

Per poter spegnere i dispositivi pur consentendo loro di trasmettere dati su una rete multi hop, per forza è necessario che il traffico sia sincrono, cioè i motes devono sapere quando possono trasmettere e quando devono ricevere. Un modo per ottenere comunicazioni sincrone è quello di adottare una gestione ciclica della trasmissione dei dati.

Si fissa un periodo di campionamanto e si inviano tutti i dati all'interno del periodo. In questo modo è sufficiente che i motes sappiano qual'è la durata del periodo e l'istante iniziale per poter inviare ciclicamante i dati senza bisogno di altre sincronizzazioni. Questo è vero se i motes mantengono la sincronizzazione entro certi limiti almeno per la durata dell'acquisizione di tutti i dati. In caso di acquisizioni lunghe sarà necessario ripetere la sincronizzazione come vedremo in dettaglio più avanti.

La gestione ciclica della comunicazioni genera però una certa latenza nell'arrivo dei

messaggi. Un messaggio generato in un ciclo non può arrivare immediatamente ma deve attendere uno o più cicli per pervenire a destinazione. Questo può non essere tollerabile per segnali di allarme o per acquisizioni in tempo reale che prevedano un feedback immediato, per queste applicazioni occorre usare una gestione asincrona del traffico, a scapito della durata delle batterie.
La latenza dei dati verrà esaminata più avanti nel corso del Paragrafo 4.3.1.

3.2 Consumo dei dispositivi e scelta della sincronizzazione

Come vedremo dettagliatamante nel Paragrafo 4.1 i motes di ultima generazione consentono un notevole risparmio energetico se posti in stand-by. Per avere un termine di paragone, i motes che sono stati usati, se alimentati a batterie hanno una durata di circa 85 ore tenendoli sempre accesi, mentre se fossero sempre in stand-by la durata (teorica) salirebbe a 32000 ore, ovvero 56 anni! In questo caso sarebbero sicuramante preponderanti altri fattori come l'autoscarica delle batterie, invecchiamento dei componenti, obsolescenza, ecc. ecc. Occorre quindi una qualche forma di sincronizzazione fra i motes che consenta di far arrivare i dati alla base station e di spegnerli quando sono inattivi.

Prenderemo in esame due possibili soluzioni. La prima, di tipo *Store and Forward*, prevede di fissare un *duty-cycle* della rete ossia di tenere accesi per una piccola parte del periodo di acquisizione tutti i motes e di spegnerli nel restante.
La seconda consiste nel suddividere il periodo di acquisizioni in *time slot*. Ogni sensore rimane acceso solo durante alcuni slot secondo opportune modalità che vedremo in seguito. Si è scelto di implementare un'algoritmo di quest'ultimo tipo chiamato *Flexible Power Scheduling: FPS* sviluppato in [3].

3.3 Store and forward

Questo semplice algoritmo è stato il primo ad essere utilizzato in questo genere di applicazioni (vedi [3]). Un'analisi prestazionale più dettagliata è disponibile più avanti, all'interno del capitolo 4, qui viene solo brevemente riportato il funzionamento.
Si fissa un periodo di acquisizione, e si impone che per una parte di esso i motes siano *tutti* accesi, pronti a ricevere e a trasmettere, mentre per il restante siano *tutti* in stand-by, ossia si fissa il duty-cycle della rete.

Occorre però fissare questo valore in base ad un compromesso consumo/traffico sulla

rete. Valori elevati del duty-cycle significano consumi elevati, mentre valori troppo bassi generano situazioni di traffico eccessivo che possono causare perdita di pacchetti.

Lo svantaggio principale è che il duty-cycle deve essere lo stesso per tutti i motes, e questo limita fortemente la validità dell'approccio, infatti il traffico di pacchetti transitanti per un nodo non è uguale per tutti i nodi della rete. I nodi perferici devono inviare solo il proprio pacchetto dati, ossia la lettura del sensore, mentre i nodi in prossimità della base station si trovano a dover veicolare tutto il traffico della rete più il loro. Il dover imporre un duty-cycle comune a tutti causa uno spreco di energia per i primi o un traffico elevato per i secondi.

Dai primi test sperimentali è risultata subito una scelta poco conveniente per l'elevato numero di pacchetti persi anche per valori piuttosto elevati di duty-cycle, pertanto si è preferito adottare un altro algoritmo, anche se più complesso ma che consenta una gestione migliore delle risorse: il Flexible Power Scheduling.

3.4 Flexible Power Scheduling

3.4.1 Descrizione generale dell'algoritmo originale

Flexible Power Scheduling è un protocollo per la gestione distribuita della potenza in una rete di sensori wireless, che contemporaneamente fornisce supporto per una domanda variabile delle risorse della rete. Il protocollo fornisce uno scheduling delle comunicazioni basato solo su informazioni locali. L'assegnazione e la modifica dei time slot, ovvero delle porzioni temporali assegnate ai nodi è basata su un algoritmo di tipo *Supply and demand* che consente l'allocazione e la cancellazione degli slot senza la necessità di reinizializzare l'intera rete.

Le comunicazioni con questo protocollo si intendono principalmente in un senso solo: dai motes alla base station. Il protocollo può essere esteso anche per comunicazioni bidirezionali, mentre non può gestire comunicazioni di tipo punto-punto tra i nodi della rete.

3.4.2 Schedulig in FPS

La divisione temporale in slot risolve molti dei problemi delle reti multi-hop. Mantiene il traffico di rete incorrelato garantisce una minore perdita di pacchetti e soprattutto fornisce una migliore gestione dell'energia poichè è noto quando il nodo debba trasmettere, quando ricevere e quando possa essere spento.

Una suddivisione globale in slot prevederebbe l'utilizzo di un gestore centralizzato con uno schedule globale statico, e la necessità di una sincronizzazione precisa tra i dispositivi. Questo potrebbe essere possibile per reti centralizzate, ma risulta molto più difficoltoso per reti multi-hop.

Al contrario FPS fornisce una topologia ad albero con un'algoritmo distribuito che implementa uno schedule locale e richiede solo una sincronizzazione locale a "a grana grossa".

Anche con l'introduzione dei time slot non si eliminano completamente le collisioni, è possibile che in alcuni punti della rete due nodi stiano trasmettendo contemporaneamente a causa del fenomeno degli *hidden-nodes*. La gestione di queste collisioni è demandata al MAC dell'802.11.4 che fa uso del CSMA/CD, una tecnica che consente di prevenire e rilevare le collisioni (vedi [6], Par. 4.4). La combinazione di una prima sincronizzazione grossolana e del protocollo a livello MAC riduce notevolmente le collisioni e rende la trasmissione più robusta.

3.4.3 Concetti base di FPS

Il tempo è suddiviso in slots. Ogni slot *s* ha una lunghezza T_s. La numerazione degli slot è periodica di modulo *m*, ad esempio lo slot *m+s* è chiamato semplicemente slot *s*. Un ciclo *c* pertanto ha durata $T_c = mT_s$.

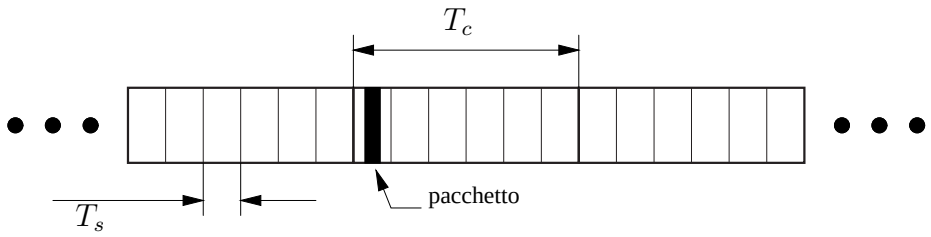

Figura 3.1: *Terminologia usata in FPS*

Ogni nodo mantiene uno schedule locale sulle operazioni da compiere in un ciclo. Durante il corso di un ciclo il nodo può essere in uno di questi tre stati: ricezione, trasmissione o idle ossia stand-by.
Si definisce come *domanda* di un nodo, il numero di slot in ricezione più il traffico generato dal nodo stesso, mentre l'*offerta* è il numero di slot in trasmissione. I nodi partono con una domanda iniziale di almeno 1. Se poniamo $b(n)$ il numero di slot (in un ciclo) in cui un nodo non è in idle, possiamo definire il duty cycle del nodo n: $\delta(n)$ come:

$$\delta(n) = \frac{b(n)}{m} \tag{3.1}$$

Dall'equazione 3.1 si evince che ci sono due modi per diminuire il duty cycle: aumentare il numero di slot per ciclo *m* o ridurre il numero di slot non idle. Ogni nodo interviene su quest'ultimo parametro per variare la propria attività.

3.4.4 Semplice esempio esplicativo

Supponiamo di avere la rete di motes in Figura 3.2. I quadrati rappresentano i nodi con una domanda iniziale di 1. I cerchietti rappresentano la domanda aggiuntiva originata al nodo, mentre le linee sono le comunicazioni tra i nodi.

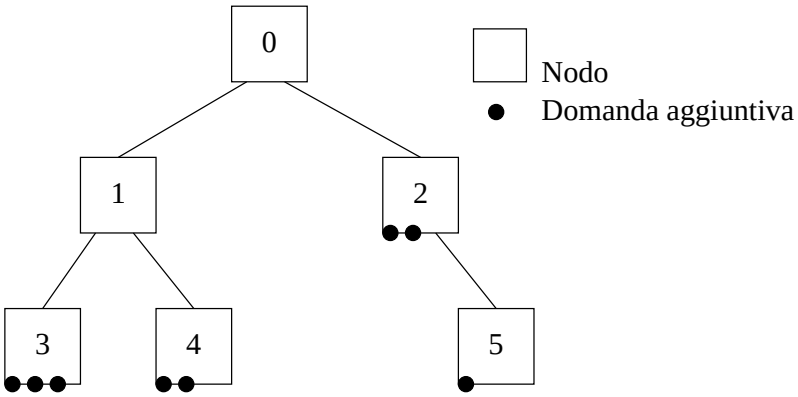

Figura 3.2: *Rete usata nell'esempio*

Supponiamo che il nodo 0 sia la base station, e che tutti gli altri nodi abbiano una domanda iniziale di 1, quindi ogni nodo ha almeno uno slot di comunicazione con il proprio genitore. I nodi iniziano sempre con un'unità extra di domanda. Iniziando dal basso, i nodi 3 e 4 richiedono entrambi slot di trasmissione con il genitore che è il nodo 1. Percui l'effettiva domanda al nodo 1 è 8, che dovrà quindi chiedere 8 slot di comunicazione alla base station. Guardando l'altro ramo, si vede che il nodo 2 richiede 5 slot di comunicazione con la base station.
Già da questo semplice esempio appare chiaro che i nodi in prossimità della base station hanno un duty cycle più elevato di quelli periferici.

3.4.5 Descrizione dell'algoritmo

Stato dei nodi

Ogni nodo mantiene il proprio schedule degli slot per la trasmissione e per la ricezione. Si definisce come *reservation slot* lo slot corrispondente ad una ricezione/trasmissione tra

genitore e figlio.

Ogni istante (slot) un nodo può essere in uno dei seguenti sei stati:

1. *Transmit* (T) - Trasmette un messaggio al nodo genitore, rimane nello schedule finchè la topologia della rete non cambia.

2. *Receive* (R) - Riceve un messaggio da un nodo figlio, rimane nello schedule finchè la topologia della rete non cambia.

3. *Advertisement* (A) - Manda in broadcast un messaggio di *Advertisement* da parte di un genitore, rimane nello schedule per un ciclo al massimo.

4. *Transmit Pending* (TP) - Invia un messaggio di *Reservation Request* al genitore, rimane nello schedule un ciclo al massimo.

5. *Receive Pending* (RP) - Riceve un *Reservation Request* da un figlio, rimane nello schedule per due cicli al massimo.

6. *Idle* (I) - Dopo che tutta la domanda al nodo è stata sodisfatta il nodo può entrare in stand-by durante gli slot idle.

Quando un nodo genitore invia in broadcast un *Advertisement* per una prenotazione (*Reservation*) marca il *Reservation Slot* nel proprio schedule come *Receive Pending*. Se un figlio ha una domanda ancora da soddisfare marca il *Reservation Slot* nel proprio schedule come *Transmit Pending* ed invia una *Reservation Request* durante il *Reservation Slot*.
Il nodo genitore risponde immediatamente al *Reservation Request* con un messaggio di *Reservation Confirm*. A questo punto il genitore segna il *Reservation Slot* come *Receive* mentre il figlio lo segna come *Transmit*.
In Figura 3.3 è riportato un esempio di questo procedimento.

Sono rappresentati tre cicli: 1,2,3, con i rispettivi schedule del nodo padre: P e del nodo figlio: F. Nel primo ciclo il padre seleziona a caso uno slot (il 7 nell'esempio), lo marca come *Receive Pending* e sempre in uno slot a caso precalcolato, invia un *Advertisement* con questa informazione ai figli (il 2 nell'esempio). Il figlio, che ha ancora una domanda di traffico insoddisfatta, riceve l' *Advertisemen*te si segna lo slot indicato dall'*Advertisement* come *Transmit Pening*.

Nel ciclo successivo, durante lo slot marcato come *Transmit Pending*, il figlio invia al padre un messaggio di *Reservation Request* (RR), e il genitore da la conferma rispondendo immediatamente con un *Reservation Confirm* (RC).

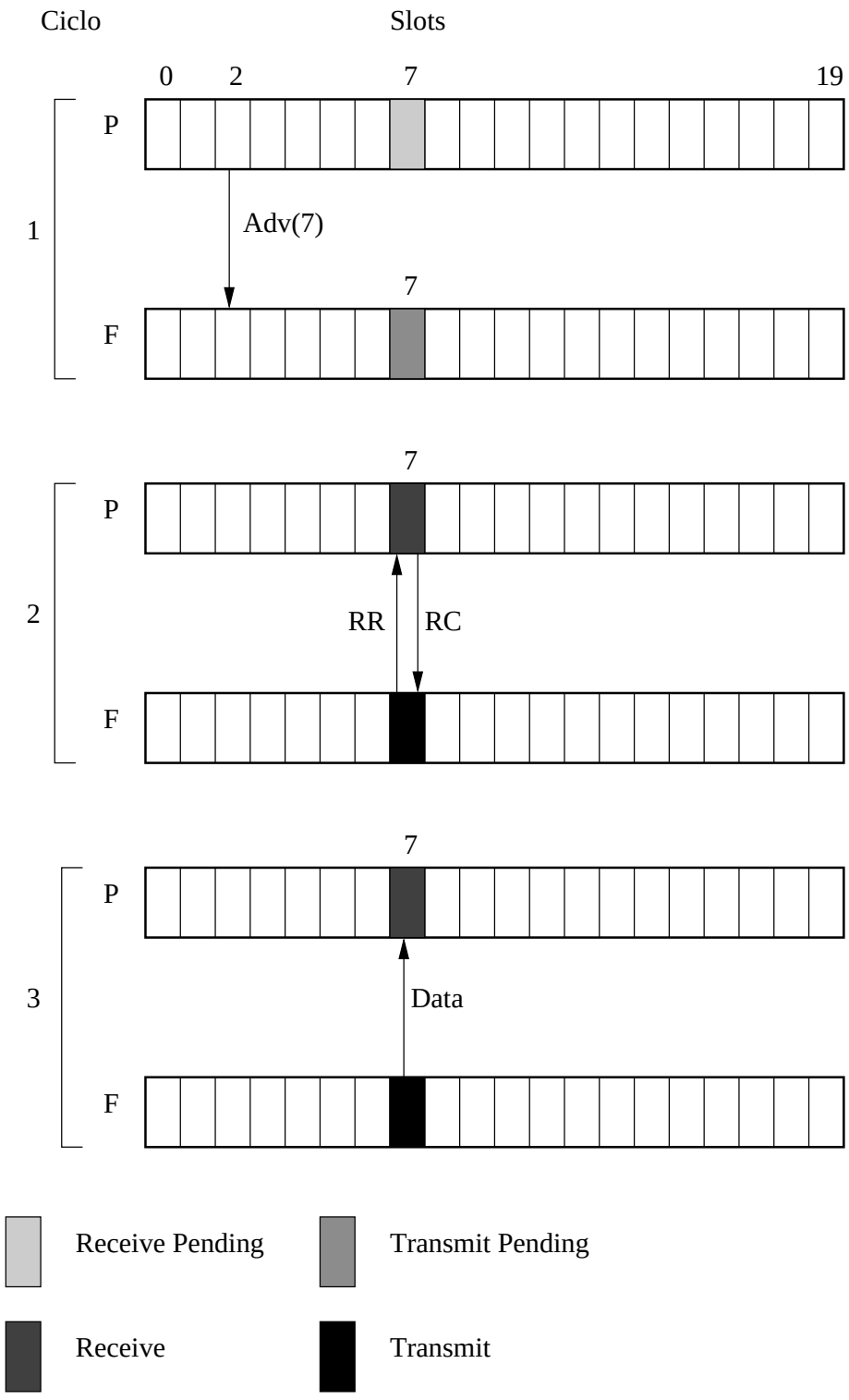

Figura 3.3: *Operazione di prenotazione di uno slot tra due nodi*

A questo punto nel terzo ciclo e in tutti i successivi il figlio può trasmettere al padre i dati nello slot che per lui è marcato come *Transmit*, e per il padre *Receive*, ovvero lo slot numero 7.

3.4.6 Sincronizzazione

Quando un figlio seleziona il genitore sincronizza i suo timer-contatore degli slot e il numero di slot con quello del genitore. Periodicamente è necessario ripetere la sincronizzazione a causa del drift tra i dispositivi. È necessaria solo una sincronizzazione grossolana poichè i time slot sono relativamente grandi. Si possono adottare vari metodi. Il più semplice anche se non il migliore consiste nell'attivare periodicamante il nodo figlio, diciamo ogni N_s cicli per ricevere un *Advertisement* da parte del padre e rieseguire la sincronizzazione del timer.

3.4.7 Inizializzazione

Non è necessaria alcuna inizializzazione globale. Tutto inizia dalla base station che seleziona un *Reservation Slot*, rilascia un *Advertisement* e rimane in ascolto per un *Reservation Request* da parete dei figli. Se un nodo è nuovo, ossia ha tutti gli slot idle, rimane in ascolto per uno o più cicli di un *Advertisement*, quando lo riceve invia un *Reservation Request*, per cercare di accaparrarsi lo slot.

I nodi continuano a rispondere agli *Advertisement* finchè non hanno sodisfatto tutta la loro domanda iniziale. Quando hanno soddisfatto la domanda iniziale iniziano a loro volta ad emettere *Advertisement* per i loro figli e a mettersi in stand-by durante gli slot idle.

3.4.8 Algoritmo

La Figura 3.4 mostra il flow chart dell'algoritmo seguito all'inizio di un ciclo da ogni nodo n dopo l'inizializzazione, mentre la Figura 3.5 mostra quello per ogni slot. In Figura 3.6 troviamo le operazioni da eseguire al termine di un ciclo.

Una slot è ufficialmente riservato quando un nodo genitore riceve ed accetta una richiesta durante uno slot RP. Nel caso arrivi più di una richiesta, viene accettata solo la prima. Il genitore invia una conferma immediata ed incrementa la sua domanda di uno ed imposta come R questo slot. Il figlio incrementa la sua offerta di uno ed imposta lo slot come T. La prenotazione non ha bisogno di essere rinegoziata e rimane attiva per sempre.

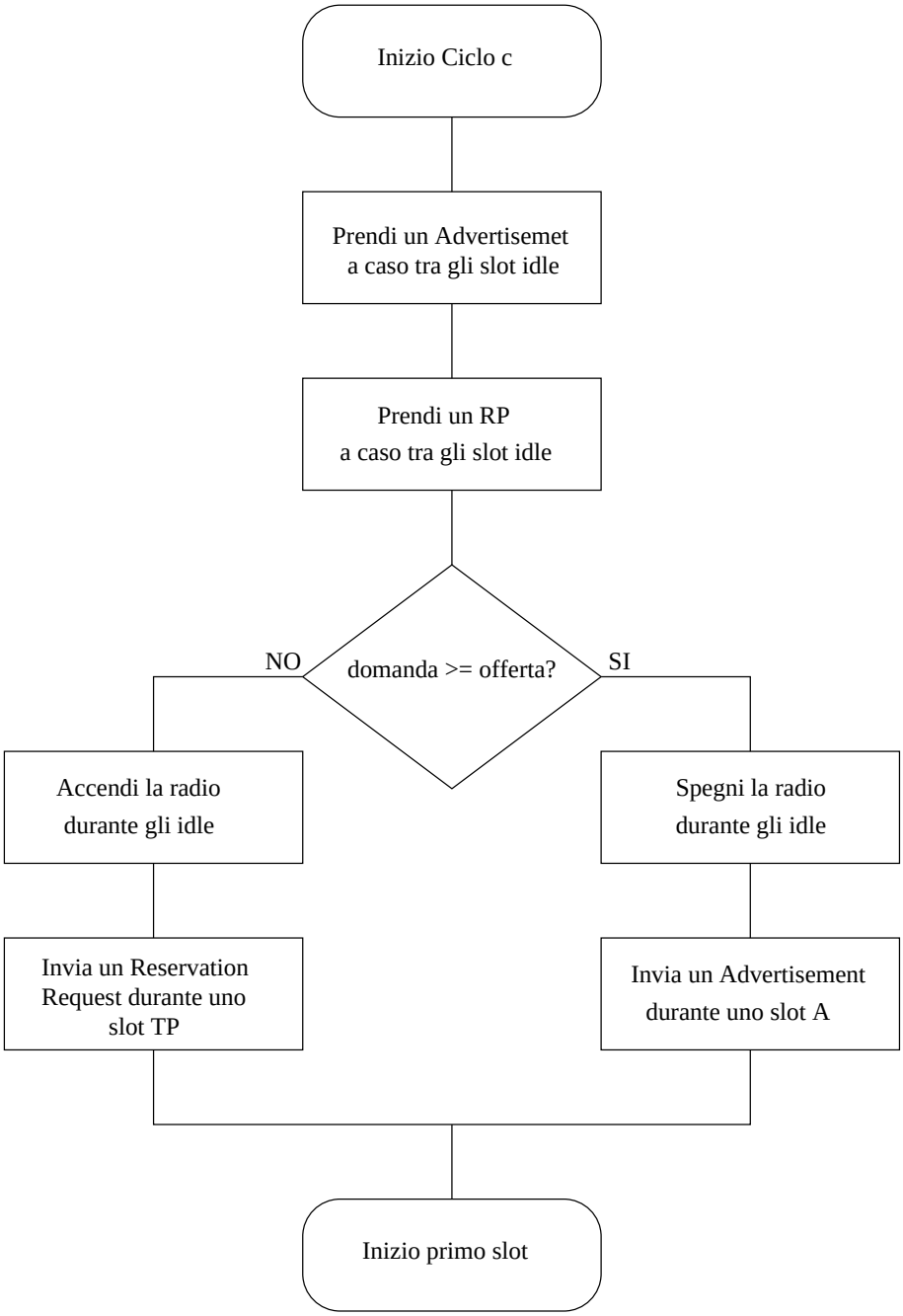

Figura 3.4: *Algoritmo FPS: inizio ciclo*

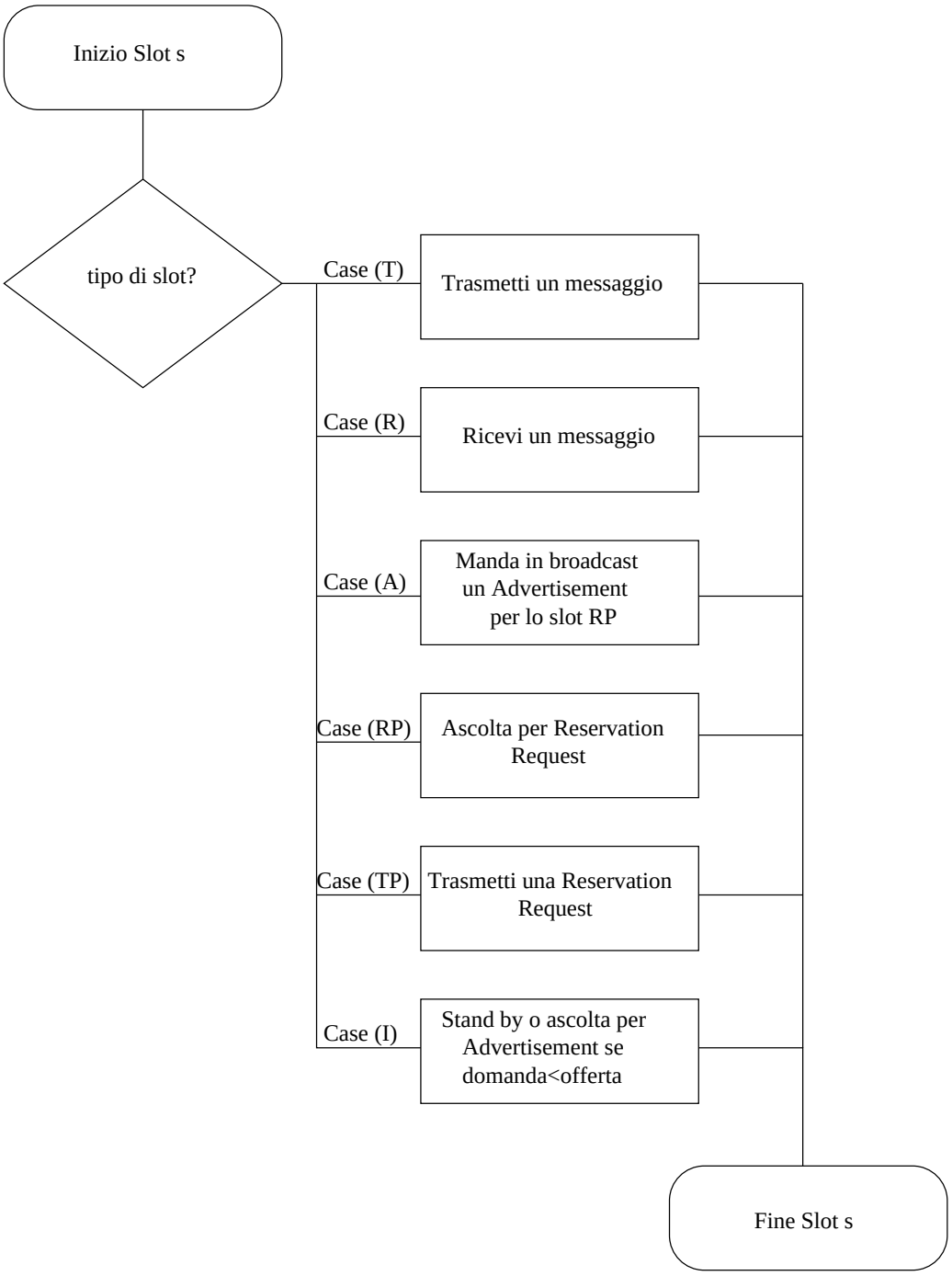

Figura 3.5: *Algoritmo FPS: slot*

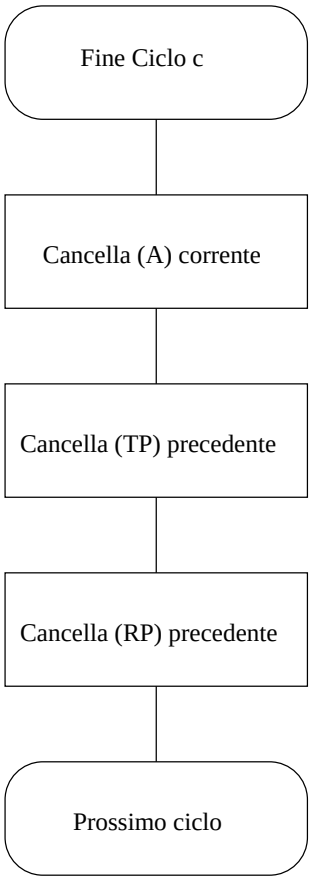

Figura 3.6: *Algoritmo FPS: fine ciclo*

Un nodo può ricevere anche una richiesta di diminuzione della domanda durante uno slot R, nel qual caso imposta lo slot come idle.

3.4.9 Collisioni e perdite di messaggi

La funzione primaria di questo protocollo è essenzialmente quella di determinare quando trasmettere e quando ricevere. La larghezza degli slot è mantenuta al minimo per consentire la trasmissione di al massimo due pacchetti in uno slot, l'accesso al canale in questo case è affidato al CSMA del MAC.

Collisioni possono accadere a causa del fenomeno del terminale nascosto che si può verificare quando due nodi figli, fuori dalla portata dei trasmettitori l'uno con l'altro, rispondono allo stesso *Advertisement*. Quando un figlio non riesce ad ottenere la prenotazione per un certo numero di cicli può assumere che questo tipo di collisioni stia accadendo, in questo caso il figlio potrebbe inviare le richieste (*Reservation Request*) successive con probabilità $P_{req} < 1$.

3.5 Implementazione del protocollo FPS

L'implementazione del protocollo FPS qui proposta ricalca essenzialmente quanto esposto. Le uniche differenze riguardano la dinamicità della topologia della rete.

Poichè si è supposto che la topologia della rete rimanga costante nel tempo, si sono levate da FPS tutte le caratteristiche che hanno a che fare con questo aspetto. D'altronde prima di procedere con questo protocollo si sono minimizzate le potenze dei trasmettitori, se la rete cambiasse bisognarebbe ripetere l'intera procedura iniziale. Le principali differenze sono le seguenti.

- I nodi non possono de-allocare gli slot riservati. La domanda iniziale di ogni nodo è costante ed è pari ad uno slot, ossia quello che serve per trasmettere la lettura del sensore di temperatura.

- L'invio dei dati è sempre fatto a minima potenza mentre l'invio dei messaggi necessari per la prenotazione dello slot è fatto a massima potenza. L'invio a massima potenza è necessario perchè si tratta di messaggi in broadcast quindi destinati a più nodi. In alternativa si potrebbe eseguire un invio personalizzato con il giusto livello di potenza del trasmettitore per ogni destinatario ma aumenterebbe il traffico in rete e occuperebbe più slots.

In ogni caso, questo aspetto non altera molto i consumi dato che una volta impostata la rete, non si ha più traffico dovuto alle prenotazioni ma solo traffico dati.

Capitolo 4

Risparmio energetico, autonomia

In questo capitolo cercheremo di valutare da un punto di vista applicativo il metodo proposto.

Per procedere sono necessarie alcune considerazioni fondamentali sull'hardware a disposizione che verranno riassunte nella prima sezione. Seguirà un confronto tra l'utilizzo dell'algoritmo *FPS puro* e la versione qui sviluppata mentre nell'ultima parte ci occuperemo di confrontare l'approccio seguito da *FPS* e il metodo classico di tipo *store and forward* che non fa uso della suddivisione temporale in slot.

4.1 Organizzazione dell'hardware

I motes a disposzione, sono come abbiamo visto i *TmoteSky* (*Telos*) prodotti dall'americana *MoteIV*. Il design innovativo di questi prodotti consente di mantenere i consumi energetici entro livelli decisamente bassi. In Tabella 4.1 è possibile vedere l'evoluzione di questi prodotti; si nota che il *Telos* monta un microcontrollore della *Texas Instruments*, il TI MSP430 che fornisce tutto il necessario per la gestione dei processi e della memoria.

Per le comunicazioni via radio, si è scelto di adottare l'integrato CC2420 della *Chipcon*. Questo integrato anch'esso concepito per un basso consumo energetico implementa in hardware tutto il necessario per gestire lo strato PHY e una parte anche del MAC del protocollo 802.15.4.

Confrontando i dati inerenti al consumo di corrente dei vari motes, si nota come il *Telos* sia in media con gli altri, almeno con i più recenti, quindi su questo fronte non si hanno miglioramanti, tuttavia guardando la tensione minima di funzionamento si vede che il dispoitivo può funzionare anche solo con 1.8 volts mentre gli altri hanno bisogno di

Mote Type	WeC	René	René2	Dot	Mica	Mica2Dot	Mica 2	Telos
Year	1998	1999	2000	2000	2001	2002	2002	2004
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128			TI MSP430
Program memory (KB)	8		16		128			48
RAM (KB)	0.5		1		4			10
Active Power (mW)	15		15		8		33	3
Sleep Power (μW)	45		45		75		75	15
Wakeup Time (μs)	1000		36		180		180	6
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M25P80
Connection type	I^2C				SPI			SPI
Size (KB)	32				512			1024
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250
Modulation type	OOK				ASK	FSK		O-QPSK
Receive Power (mW)	9				12	29		38
Transmit Power at 0dBm (mW)	36				36	42		35
Power Consumption								
Minimum Operation (V)	2.7		2.7		2.7			1.8
Total Active Power (mW)	24				27	44	89	41
Programming and Sensor Interface								
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	16-pin
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB
Integrated Sensors	no	no	no	yes	no	no	no	yes

Figura 4.1: *Confronto tra diversi tipi di motes.*

almeno 2.7 volts. Questa caratteristica ci consente, come vedremo, di ottenere autonomie di funzionamento ben maggiori.

Esaminando la curva di scarica di una moderna batteria alcalina (Fig. 4.2) si vede che la tensione di cut-off è intorno agli 0.9 volts. Poichè le due batterie che alimentano il mote sono in serie otteniamo che il cut-off totale è di 1.8 volts che è esattamente la tensione minima di funzionamento dei *Tmote Sky*, quindi il motes è in grado di sfruttare fino in fondo le batterie. Gli altri motes invece hanno bisogno di 2.7 volts ossia di 1.35 volts per batteria, riuscendo così a sfruttare solo il 50% della carica.

Abbiamo visto che l'hardware è ottimizzato per il basso consumo, precisamente, dal datasheet del *Tmote Sky* e del CC2420 risultano i seguenti consumi.

Stato	Consumo	Unità
MCU On, Radio Rx	21.8	mA
MCU On, Radio Tx , potenza max (0dBm)	19.5	mA
MCU On, Radio Tx , potenza min (-25dBm)	10.3	mA
MCU On, Radio Off	1.8	mA
MCU Idle, Radio Off	0.054	mA

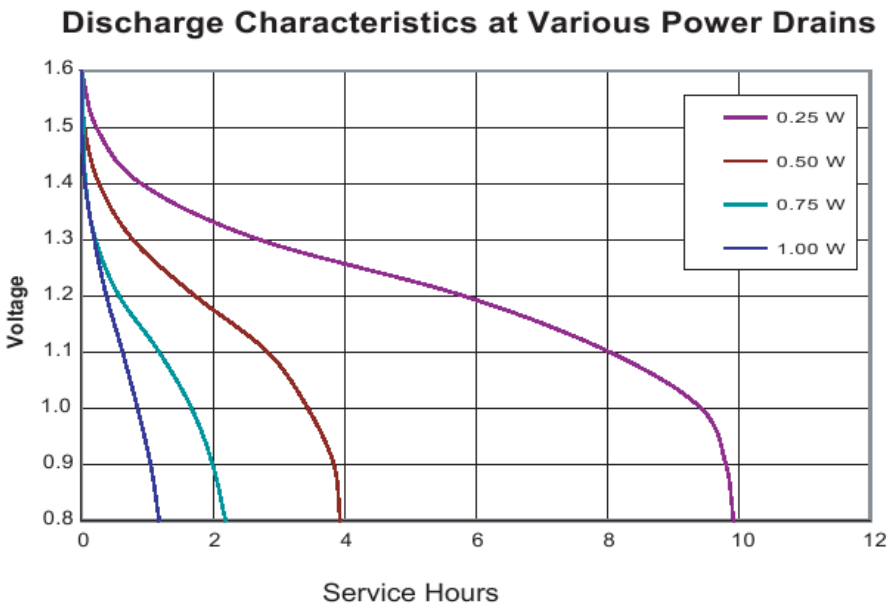

Figura 4.2: *Curva di scarica di una batteria alcalina (Duracell, vedi [8]).*

4.2 Analisi dei consumi energetici

Tenendo presente quanto esposto poc'anzi cercheremo di ricavare il consumo medio dell'applicazione.

Per rendere più egevole la trattazione faremo riferimento a due topologie semplici, nella prima supponiamo di avere tutti i motes dipsosti su di una linea retta, nella seconda invece, che siano tutti su di una circonferenza con la base station al centro. Nelle applicazioni reali avremo reti più complesse che però possono essere ricondotte ad una serie di composizioni di queste due topologie.

4.2.1 Topologia lineare

Supponiamo di disporre tutti i motes equidistanziati su di una linea retta, come in Figura 4.3, in cui sono rappresentati cinque motes e la base station. Supponiamo che T_c sia il periodo di campionamento e che sia suddiviso in n_s slots. Definiamo i consumi di corrente come:

- I_{Rx}, il consumo in ricezione
- $I_{Tx,max}$, il consumo in trasmissione con il trasmettitore al massimo
- $I_{Tx,min}$, il consumo in trasmissione con il trasmettitore al minimo

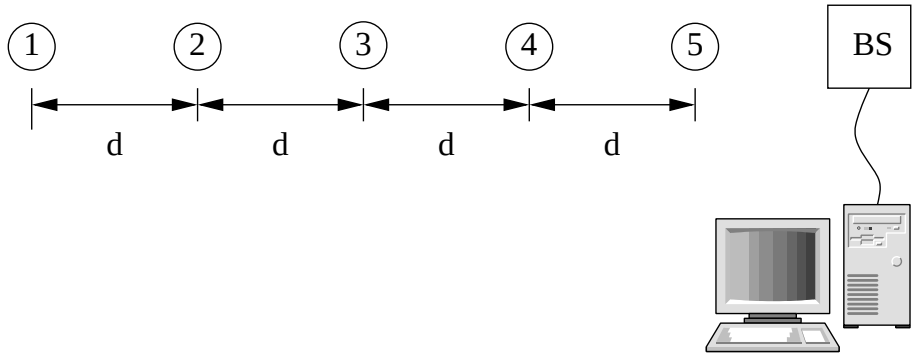

Figura 4.3: *Topologia semplice: cinque motes posti su una linea retta.*

- I_{idle}, il consumo con il dispositivo in stand-by

Con queste premesse andiamo ad analizzare il consumo medio di potenza della rete nei tre casi: applicando l'algoritmo FPS "puro", l'FPS con impostazione del link a minima potenza, e con l'algoritmo di tipo *store and forward* che non fa uso della divisione in slot del periodo di campionamento.

FPS

Per questo algoritmo bisogna distinguere tra i motes esterni, ovvero quelli che fanno solo da sensori/trasmettitori e i motes interni cioè che fungono anche da ripetitori. Con riferimento alla Figura 4.3 si tratta del motes 1 e di 2,3,4,5 rispettivamante. Il motes 1 durante un ciclo deve solo trasmettere quindi si trova ad avere attivo uno slot occupato per l'invio dell' *Advertisement* ed uno per l'invio della lettura del sensore, quindi il consumo di corrente medio è

$$I_1 = \frac{2I_{Tx,max} + (n_s - 2)I_{idle}}{n_s}. \tag{4.1}$$

Per i nodi intermedi bisogna considerare anche gli slot occupati per la ritrasmissione dei dati provenienti dagli altri sensori, supponendo di voler trasmettere le letture di n_r sensori si ha

$$\begin{aligned} I_{2,3,4,5} &= \frac{2I_{Tx,max} + (n_s - n_r - 2)I_{idle} + n_r I_{Rx} + n_r I_{Tx,max}}{n_s} \\ &= \frac{(n_r + 2)I_{Tx,max} + n_r I_{Rx} + (n_s - 2n_r - 2)I_{idle}}{n_s}. \end{aligned} \tag{4.2}$$

Per proseguire con l'esempio abbiamo bisogno di fissare T_c e n_s, prendiamo $T_c = 10sec$, ed $n_s = 100$, come nelle prove sperimentali che seguiranno. Il tempo di slot pertanto

risulta $T_s = \frac{T_c}{n_s} = 100$ ms.
Andando ad inserire T_c, n_s ed i dati dei *Tmote Sky* nelle 4.1 e 4.2 otteniamo

$$I_1 = 0.44\,mA \qquad I_5 = 2.09\,mA \tag{4.3}$$

che sono i motes con il consumo minore e maggiore.
Il consumo di corrente come si vede dalle equazioni 4.1 e 4.2 dipende anche dal numero di slot n_s. Ad esempio, se avessimo raddoppiato il numero di slot avremmo ottenuto:

$$I_1 = 0.22\,mA \qquad I_5 = 1.045\,mA \tag{4.4}$$

FPS con impostazione del link a minima potenza

Prima di eseguire l'algorimo dell'FPS si fa partire la procedura che minimizza la potenza dei trasmettitori, AMPL. Supponiamo che i motes siano posti a distanza ravvicinata e che la procedura abbia assegnato ad ogni motes di usare il nodo successivo più vicino alla base station per comunicare con essa, e che la potenza dei trasmettitori sia stata impostata al minimo.
Questa è solo una situazione ideale, non è detto che il motes più "conveniente" per raggiungere la base station sia propio il successivo, potrebbe essere ad esempio quello dopo ancora o perfino quello prima, come gia osservato in precedenza. Sicuramente mediamente questo è vero.

L'algoritmo è il medesimo del paragrafo precedente quindi le formule rimangono simili. L'unica variazione è la potenza dei trasmettitori che viene mantenuta al minimo. Pertanto

$$I_1 = \frac{I_{Tx,max} + I_{Tx,min} + (n_s - 2)I_{idle}}{n_s} \tag{4.5}$$

dove il primo addendo e dovuto alla trasmissione dell'*Advertisement* che avviene sempre a potenza massima, e il secondo alla trasmissione del dato letto. Per i nodi intermedi otteniamo

$$I_{2,3,4,5} = \frac{I_{Tx,max} + (n_r + 1)I_{Tx,min} + n_r I_{Rx} + (n_s - 2n_r - 2)I_{idle}}{n_s} \tag{4.6}$$

Inserendo i parametri precedenti otteniamo

$$I_1 = 0.35\,mA \qquad I_5 = 1.63\,mA. \tag{4.7}$$

Chiaramente nel caso di motes molto distanti, in cui i link siano stati impostati comunque alla massima potenza, i risultati sarebbero stati coincidenti con l'FPS puro.

L'*Advertisement* è inviato a potenza massima perchè essendo un messaggio in broadcast, è destinato a tutti i vicini e pertanto deve coprire la distanza maggiore possibile.

Abbiamo visto che l'uso di AMPL ed FPS porta ad una possibile riduzione dei consumi rispetto al solo uso di FPS, e soprattutto non peggiora le cose, pertanto il suo utilizzo è sempre raccomandabile.

Store and Forward

Questo semplice algoritmo prevede di dotare tutti i motes di funzioni da ripetitore, senza però coordinare le attività degli stessi. Come descritto in precedenza, ogni dispositivo trasmette il proprio dato con un ritardo casuale ad ogni ciclo. Contemporaneamente riceve i dati provenienti dagli altri motes e li replica al suo sucessore verso la base station.

Potrebbe essere una valida alternativa se non si avesse l'esigenza di minimizzare il consumo energetico. Per questo motivo i motes non possono essere sempre accesi quindi è necessario fissare il *duty-cicle* della rete, ossia si decide di mantenere accesi i dispositivi solo per una parte del ciclo di acquisizione.

Il problema è proprio la scelta del *duty-cycle*, valori elevati aumentano i consumi, mentre valori troppo piccoli aumentano notevolmente il traffico sulla rete con conseguente aumento delle collisioni. Il valore dovrebbe quindi essere scelto vagliando una serie di parametri come la lunghezza massima della rete, cioè il numero di hops massimo che un dato deve fare per raggiungere la base station, la quantità di motes che costituisce la rete e la durata del ciclo di acquisizione. Un altro parametro importante per la scelta è la percentuale di pacchetti persi che può essere tollerata.

Purtroppo questi fattori, a parte l'ultimo, non sono noti a priori, quindi sarebe necessaria una fase preventiva di valutazione della rete per poter produrre una stima del *duty-cycle* da imporre.

Volendo minimizzare i consumi potremo, sempre seguendo il nostro esempio, imporre un *duty-cycle* $d = \frac{10}{100} = 0.1\,(10\%)$ in modo da poter paragonare il consumo con quello del nodo 5 nell'FPS, che tra l'altro è quello che consuma di più.

In generale il consumo medio è dato da

$$I = d \cdot I_{Rx} \tag{4.8}$$

mentre il traffico totale al nodo più vicino alla base, se poniamo N il numro di sensori, $n_r = N - 1$ il numero delle ricezioni e $n_t = N$ le trassioni, divenata

$$\lambda = \frac{n_r + n_t}{d\,T_c} \qquad [pkt/sec] \tag{4.9}$$

senza contare le ritrasmissioni dovute alle collisioni. Con i dati del nostro esempio abbiamo ($N = 5$, $T_c = 10sec$) con $d = \frac{10}{100}$ per i *Tmote Sky* si ottiene $I = 2.18\,mA$, mentre il traffico sulla rete al nodo 5 diventa $\lambda = 9$ [pkt/sec], che dai da sperimentali è gia da considerarsi un valore troppo elevato.

Faciamo ora alcune considerazioni sulla durata delle trasmissioni. La durata temporale dell'invio di un pacchetto dipende dal tempo impiegato dalla MCU per costruire il pacchetto dati ed inviarlo al trasmettitore, dal tempo che ci mette il trasmettitiore ad impostare i registri per l'invio ed in fine dal tempo impiegato dalla trasmissione del pacchetto. Di questi tempi solo l'ultimo è facile da calcolare, gli altri dipendono dall'hardware e dal software.

Il calcolo del tempo di invio come detto è semplice, infatti un pacchetto dati è da 8 bytes, ovvero: due per l'ID del mittente del pacchetto, due per l'ID del sensore che ha fatto la lettura, due per il contatore delle misure e due per il valore letto. A questi 8 bytes vanno aggiunti altri 9 bytes dallo strato MAC e 6 dal PHY per un totale di 23 bytes. Poichè la trasmissione avviene a 250 Kbps il tempo di invio del pacchetto è di

$$T_{data} = \frac{21 \cdot 8}{250000} = 0.73ms$$

sicuramente trascurabile rispetto agli altri due fattori. Va inoltre considerato che i motes non sono *full duplex*, cioè nell'istante in cui il motes trasmette non può ricevere, e vice versa. Questo unito alle ritrasmissioni per effetto del CSMA può causare la perdita di una buona parte dei pacchetti.

A titolo di esempio citiamo i dati sperimantali ottenuti in [3], riguardanti una rete di sei motes che inviano 100 pacchetti da 36 byte con una frequenza di un paccetto ogni 3.2 secondi ed un time slot da 80 ms. I pacchetti erano spediti attraverso 3 hops, con l'eccezione del mote due che inviava attraverso due hops. La Tabella 4.1 mostra il numero medio degli arrivi usando l'algoritmo FPS e lo *store and forward*.

Node	1	2	3	4	5	6
FPS	96.00	96.91	96.09	97.45	94.55	97.55
Naïve	28.64	11.55	54.36	18.45	18.18	16.91

Tabella 4.1: *Numero medio di pacchetti ricevuti alla base station dopo dieci tentativi. Vedi [3].*

Osservando la tabella si vede chiaramente l'elevato numero di pacchetti persi dallo *store and forward* rispetto ad FPS.

4.2.2 Topologia a stella

Supponiamo ora di disporre i motes equidistanziati su una circonferenza come in Figura 4.4. Manteniamo le stesse impostazioni dell'esempio precedente, quindi T_c, n_s, I_{Rx},

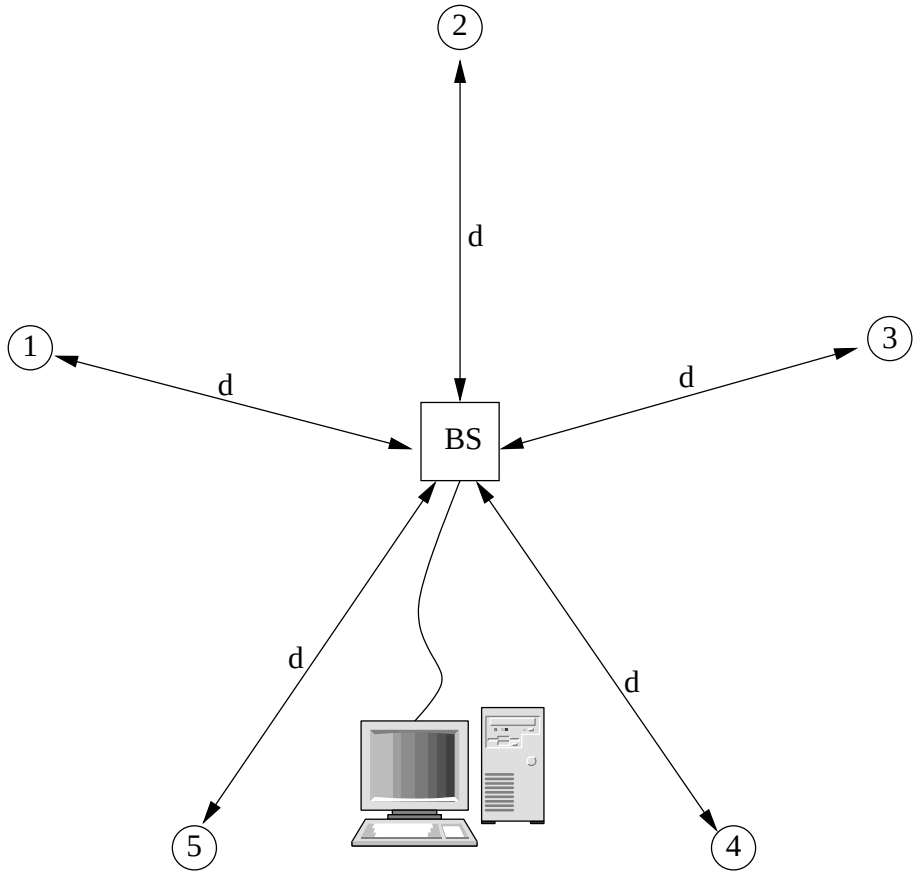

Figura 4.4: *Topologia semplice: cinque motes posti in configurazione a stella con al centro la base station (BS).*

$I_{Tx,max}$, $I_{Tx,min}$, I_{idle} rimangono le stesse e ripetiamo l'analisi con i tre algoritmi.

FPS

Essendo i motes equidistanti dalla base si suppone che tutti possano comunicare con la base direttamente pertanto non è necessario il multi hop. L'algoritmo si limita ad assegnare le risorse temporali ai sensori e a gestirne il coordinamento.

Il consumo medio di corrente per uno qualsiasi dei motes è di

$$I = \frac{2I_{Tx,max} + (n_s - 2)I_{idle}}{n_s} \tag{4.10}$$

poichè uno slot è occupato dal'*Advertisement*, uno dalla trasmissione dei dati mentre gli altri sono liberi e quindi in *standby*. Inserendo i valori numerici otteniamo $I = 0.44\,mA$ In questo caso particolare si vede che il consumo rimane basso ed omogeneo per tutti i motes. Tuttavia impostando il link si puo fare ancora meglio.

FPS con impostazione del link a minima potenza

In questo caso l'algoritmo del calcolo delle potenze minime dovrebbe porgere risultati simili per ogni motes. Darebbe dei risultati uguali se il diagramma di irraggiamento dei motes fosse isotropico e se fossimo in assenza di riflessioni di alcun tipo. Come visto questo non è posibile pertanto ci saranno sempre delle direzioni preferenziali.

Supponendo che i motes siano vicini le potenze che vengono impostate sono quelle minime, quindi il consumo medio che è dato da

$$I = \frac{I_{Tx,max} + I_{Tx,min} + (n_s - 2)I_{idle}}{n_s} \tag{4.11}$$

poichè in uno slot trasmette l'*Advertisement* a massima potenza, uno il dato a minima potenza e i restanti sono *idle*. La 4.11 per i *Tmote Sky* porge $I = 0.35\,mA$, inferiore del 20% rispetto all'FPS puro. Come per la configurazione in linea retta nel caso peggiore cioè con i motes posti a grande distaza, le prestazioni sarebero state coincidenti.

Store and Forwad

Questa è la situazione più favorevole per lo *Store and Forward*, non si hanno ritrasmissioni perchè nessun motes è impiegato da ricevitore quindi il traffico di rete rimane contenuto. Infatti questa topologia è quella di una rete single hop per la quale lo *Store and Forward* può risultre una buona scelta.

Il calcolo del consumo medio è molto semplice, basta imporre il traffico desiderato, ossia un'intensità degi arrivi che garantisca un livello di pacchetti persi accettabile. Per quanto visto precedentemente sui tempi di trasmisssione / ricezione questo dato dovrebbe essere ricavato empiricamente in funzione della lunghezza dei pacchetti da trasmettere. Una buona scelta dettata da considerazioni sperimentali potrebbe essere $\lambda = 3$ [pkt/sec].

Con i dati precedenti, ovvero con un periodo di acquisizione di $T_c = 10$ secondi, e $N = 5$ motes, occorre tenere accesi i dispositivi per un tempo

$$T_{ON} = d\,T_c = \frac{N}{\lambda} = 1.7\,sec$$

a cui corrisponde un consumo medio sul ciclo di

$$I = \frac{d\,T_c I_{Rx} + (T_c - T_{ON}) I_{idle}}{T_c} = 0.37\,mA. \tag{4.12}$$

4.3 Analisi delle prestazioni

4.3.1 Comparazione delle prestazioni

La valutazione delle prestazioni di un algoritmo dipende essenzialmente da cosa ci si aspetta da esso. Per procedere nella valutazione faremo riferimento agli esempi semplificati del capitolo precedente, che ben rappresentano delle celle base si una rete di acquisizione dati.

Come spesso accade non esiste una soluzione ottima per tutti i casi. Ogni algoritmo presenta vantaggi e svantaggi a seconda dell'applicazione che si vuole implementare.

Le soluzioni basate su FPS hanno come svantaggio principale che i dati dai sensori arrivano alla base con un certo ritardo che può anche essere notevole, infatti la procedura di associazione di un nodo con il successore verso la base station richiede come visto tre fasi:

- ricezione dell'*Advertisement*
- invio di un *Receive Request* e ricezione di un *Receive Confirm*
- invio del dato.

Questi passaggi avvengono in slot casuali in cicli successivi, quindi nel caso peggiore possono trascorrere $3T_c$ secondi prima che un nodo riesca ad inviare il primo dato al nodo succesivo. Nel caso di una rete con n_h hops il ritrdo massimo con cui i dati arrivano è

$$T_r = 3T_c\, n_h. \tag{4.13}$$

Con riferimanto alla tipologia in linea del paragrafo precedente nel caso peggiore, ossia per il motes 1 il ritardo è $T_{r,1} = 120$ secondi.

Vediamo che con reti anche non eccessivamante estese i pacchetti possono accumulare ritardi anche di qualche minuto. Per diminuire questi tempi di propagazione si può intervenire sul periodo di campionamento T_c riducendolo, con il risultato però, di ritrovarsi un numero maggiore si campioni anche se inutili.

Confrontiamo ora i consumi energetici di queste soluzioni. Per comodità in Tabella 4.2 sono riportati i consumi degli esempi precedenti.

Retta			Stella	
	mote 1	mote 5	tutti i motes	
FPS	0.44 mA	2.09 mA	FPS	0.44 mA
FPS + link	0.35 mA	1.63 mA	FPS + link	0.35 mA
Store & Fw	2.18 mA		Store & Fw	0.37 mA

Tabella 4.2: *Tabella riassuntiva dei consumi degli esempi*

Si nota innanzitutto che la configurazione che consente di consumare meno energia è quella qui proposta, che fa uso dell'FPS per la suddivisione in time slot e minimizza precedentemente la potenza dei trasmettitori. Lo *Store and forward* invece funziona bene per la topologia a stella, mentre è decisamente da evitare per le reti multi hops.

Altra cosa importante è che con le prime due soluzioni ogni mote consuma il minimo indispensabile, mentre nella terza il consumo è identico per tutti. Questo fattore potrebbe influire nel caso si avessero più fonti di alimentazione, ad esempio batterie diverse, per poter scegliere la fonte migliore per ogni motes.

Se ipotizziamo di alimentare i dispositivi con moderne batterie alcaline la cui capacità è di circa 1800mAh, otteniamo le autonomie di Tabella 4.3.

Retta			Stella	
	mote 1	mote 5	tutti i motes	
FPS	4090	860	FPS	4090
FPS + link	5140	1100	FPS + link	5140
Store & Fw	820		Store & Fw	4860

Tabella 4.3: *Autonomie possibili con due batterie alcaline, espresse in ore*

Nel caso migliore ovvero per il motes 1 della prima topologia l'uso di FPS e del setup del link ha portato ad un incremento dell'autonomia di ben 4320 ore rispetto a *Store & Forward*, mentre negli altri casi si sono avuti miglioramenti più o meno consistenti. Senza contare che la percentuale di pacchetti persi con l'uso di FPS è decisamente inferiore allo *Store and Forward* come mostrato in Tabella 4.1.

In sostaza gli algoritmi che prevedono una suddivisione temporale in slots come FPS introducono inevitabilmente dei ritardi, pertanto vanno applicati a situazioni in cui la tempestività dei dati non sia un parametro fondamentale.

Sicuramente per applicazioni di monitoraggio di fenomeni a variazione lenta, come ad esempio la temperatura ambientale rappresentano un'ottima scelta perchè minimizzano sia i consumi energetici che la perdita di pacchetti. Peraltro in situazioni in cui ci siano da propagare segnali veloci come ad esempio gli allarmi, non sono assolutamante idonei.

Un'altro tipo di applicazioni per le quali sono particolarmente indicati sono le acquisizioni ad alta velocià di fenomeni a variazione veloce in cui non interessa avere subito i dati ad esempio per intervenire rapidamante, ma si sia piuttosto interessati ad avere tutti i dati per compiere in seguito un'analisi del fenomeno. Esempi di queste applicazioni si possono ritrovare ovunque nei processi industriali (pressioni in autoclavi e pistoni, distanze su bracci meccanici, vibrazioni ecc. ecc.). In questo caso si apprezza la bassa perdita di pacchetti di questi algoritmi.

4.3.2 Conclusioni

L'impostazione del link a bassa potenza unito a FPS consente di risparmiare ancora qualcosa soprattutto in reti fortemente connesse ovvero in cui ci siano molti motes che possono comunicare gli uni con gli altri. Basti pensare all'esempio della topologia a stella, in cui data la vicinanza dei motes si è supposto che tutti comunicassero con la base anche alla minima potenza, si è ottenuto un guadagno di più di mille ore rispetto ad FPS puro.

Pertanto, in situazioni in cui non si abbiano esigenze di dinamicità della topologia della rete, per implementare una rete di sensori wireless multi-hop l'approccio qui seguito rappresenta senz'altro una valida opzione. L'unione di AMPL e FPS consente di ridurre i consumi agendo su due fronti: riduzione della potenza irradiata e riduzione del duty cycle dei motes. I consumi ottenuti con questa tecnica risultano comunque inferiori all'uso di FPS "puro" senza introdurre peggioramenti come perdite di pacchetti o altro.

Capitolo 5

Analisi Sperimentale

Gli algoritmi proposti nei capitoli precedenti sono stati testati su uno scenario reale di monitoraggio ambientale di una abitazione. Si sono rilevate ciclicamante le temperature di alcune stanze dell'abitazione. Sono state effettuate due tipi di prove. Nelle prime sono stati impiegati pochi dispositivi per lo più vicini tra loro ed è stata fatta un'acquisizione di poche decine di campioni. Lo scopo di queste prove era verificare il comportamento di *AMPL*.
La seconda serie di prove ha coinvolto tutto il programma in un monitoraggio a frequenza piuttosto elevata della temperatura. I campionamenti talvolta sono stati effettuati anche con un periodo di dieci secondi. Questa frequenza di campionamento eccessiva per un'applicazione di monitoraggio ambientale è stata scelta per mettere alla prova la procedura, ed avere delle statistiche più "fitte" sulla percentuale di pacchetti persi. Il sistema si è comporatato bene, dimostrando una bassissima perdita di pacchetti, sempre al di sotto del 5%, sicuramente buona per un'applicazione di questo genere.

5.1 Prime prove sperimentali: algoritmo AMPL

Per poter valutare il funzionamento dell'algoritmo AMPL è necessario conoscere due cose: quali sono i livelli di potenza stimati dai nodi per raggiungere gli altri nodi della rete e il percorso che ogni nodo ha selezionato per raggiungere la base station.

Poichè queste informazioni sono contenute a livello locale nella memoria dei motes occorre un sistema per poterle visualizzare. Non essendo chiaramente dotati di display, l'unico modo per esportare informazioni sullo stato è quello di inviare dei pacchetti di debug, con le informazioni che servono.

Questa procedura ha un solo inconveniente: trattandosi di reti multi hop i pacchetti di

debug vengono inviati alla base veicolati assieme al traffico dati, sul percorso scelto dai motes. Se l'algoritmo che crea il percorso non funziona non è possibile neanche inviare i pacchetti di debug, e quindi è difficile capire quello che non funziona.

Ecco perchè i primi test sono stati effettuati a distanze ravvicinate, perchè così si è potuto inviare direttamante in broadcast alla base station i pacchetti di debug.

5.1.1 Motes in linea retta a breve distanza

I motes sono stati disposti secondo la topologia mostrata in Figura 5.1. I motes sono stati numerati ed etichettati. Nel corso delle prove vengono distribuiti a caso, saranno poi i motes stessi a scoprire che numero hanno i loro vicini. L'unico motes che fa eccezione è lo zero che per convenzione è la base station, ossia il motes collegato al calcolatore.

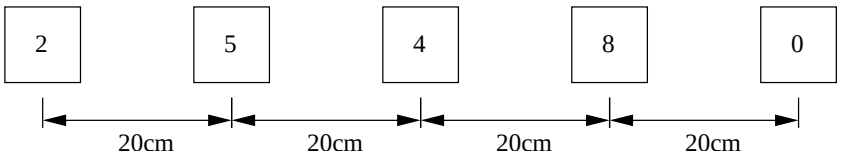

Figura 5.1: *Prima topologia sperimentata: motes disposti in linea retta.*

Lo scopo di questa prova è stato quello di valutare quale fosse la costruzione dell'albero di copertura minimale costruito da AMPL con una topologia particolarmente semplice. In linea teorica l'albero risultante avrebbe dovuto essere costituito da un ramo solo con i motes in linea, nell'ordine dalla base station al più lontano.

In pratica il motes 2 avrebbe dovuto usare 5 come genitore, ovvero come successore nel percorso verso la base station. Il 5 avrebbe dovuto usare il 4, il quattro l'8 e l'otto lo 0. In realtà l'albero creato è quello di Figura 5.2, dove i collegamenti selezionati dai motes sono rappresentati con delle rette. Tra parentesi su ogni retta è indicato il livello di potenza selezionato dal nodo su una scala di otto livelli da 1 a 8 compresi.

Al contrario di quanto si sarebbe detto l'albero è costituito da due rami. Il comportamento anomalo è quello del nodo 4 che avrebbe dovuto passare dal nodo 8, ma in realtà comunica direttamente con la base station. Questo significa che il link tra 4 è 0 è migliore di quello tra 4 e 8, nonostante la minore distanza tra questi ultimi.

Questo comportamento sperimentale è corretto, infatti come abbiamo avuto modo di vedere all'inizio, il diagramma di radiazione dell'antenna integrata sullo stampato del *Tmote Sky* non è perfettamente isotropico nello spazio. In Figura 5.3 è evidenziata in rosso la posizione dell'antenna sul circuito stampato,

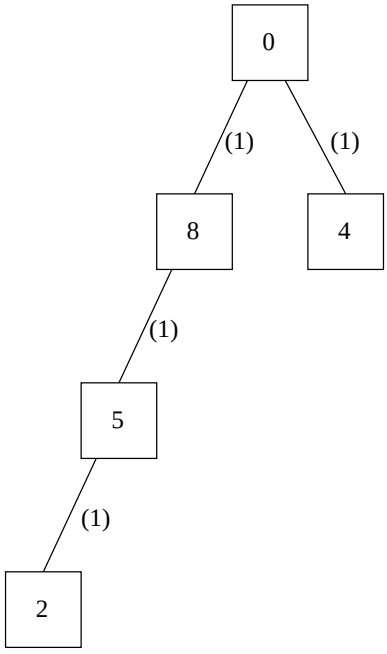

Figura 5.2: *MST, dalla topologia Figura 5.1, con le schede in orizzontale. I quadrati sono i motes con il loro rispettivi numeri identificativi, mentre le rette rappresentano i links e tra parentesi tonde sono riportati i costi (livelli di potenza) dei link.*

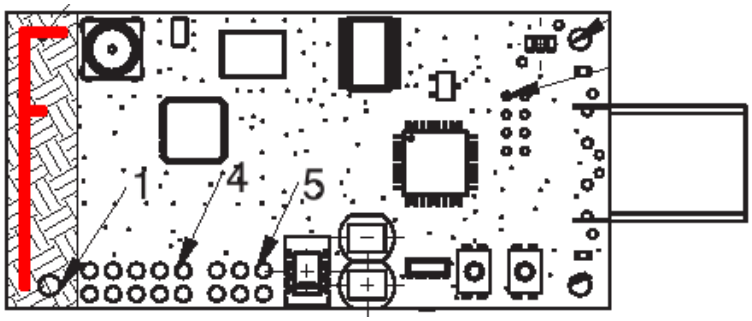

Figura 5.3: *Antenna disegnata sullo stampato del TmoteSky (in rosso). [Dal datasheet del Telos TmoteSky].*

mentre la Figura 5.4 mostra il diagramma di radiazione sul piano orizzontale.

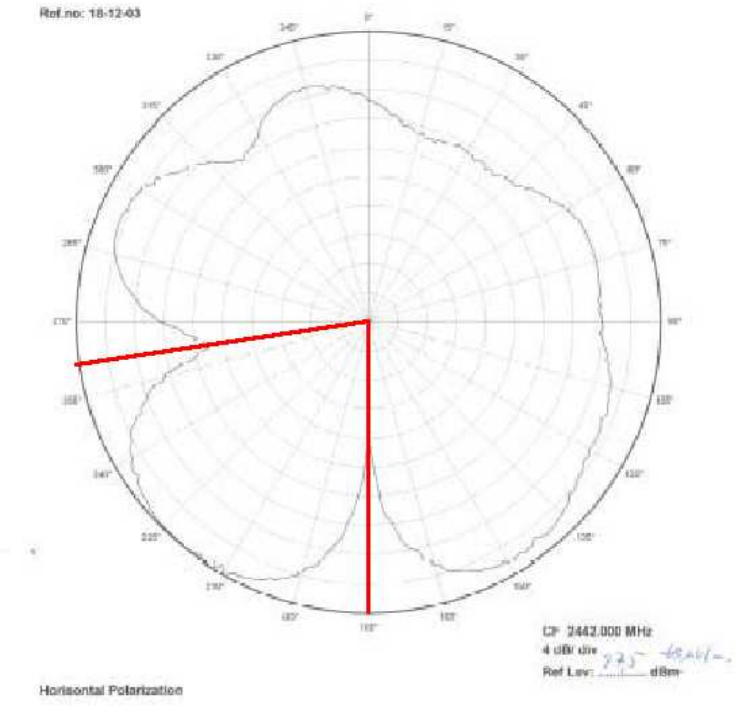

Figura 5.4: *Diagramma di radiazione orizzontale. [Dal datasheet del Chipcon CC2420].*

In particolare possiamo notare la presenza di due punti di minimo (evidenziati in rosso). Se per caso due motes hanno le antenne orientate grossomodo lungo queste direzioni, la portata delle comunicazioni si riduce. Questo fenomeno si avverte maggiormente con le potenze dei trasmettitori impostate al minimo, poichè la trasmissione è sensibile a variazioni anche di pochi centimetri.

Posizionando i motes in verticale, come in Figura 5.5, si ha una radiazione più uniforme e l'albero ottenuto è effettivamente quello che ci si aspettava, ossia quello della Figura 5.6.

5.1.2 Seconda topologia con motes ravvicinati

La seconda prova prevedeva il posizionamento dei motes sempre a breve distanza, vedi Figura 5.7. Anche questa volta per rendere più uniforme la radiazione i motes sono stati posti in verticale. La distanza tra 0 e 7 non è riportata perchè i motes erano pressochè appoggiati.

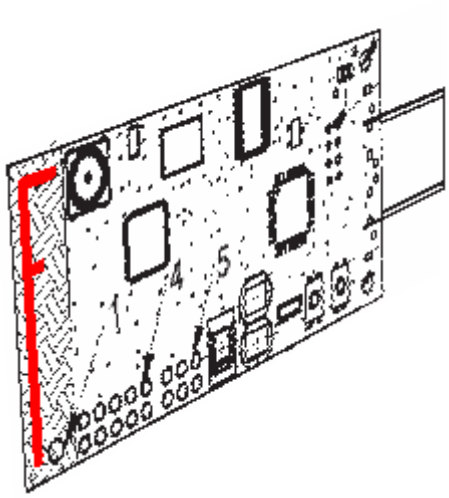

Figura 5.5: *Motes posizionato in verticale, per una radiazione più uniforme.*

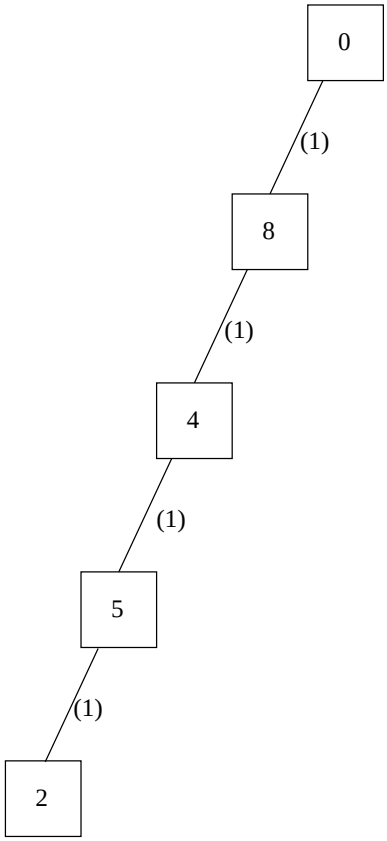

Figura 5.6: *MST, dalla topologia di Figura 5.1, con le schede in verticale.*

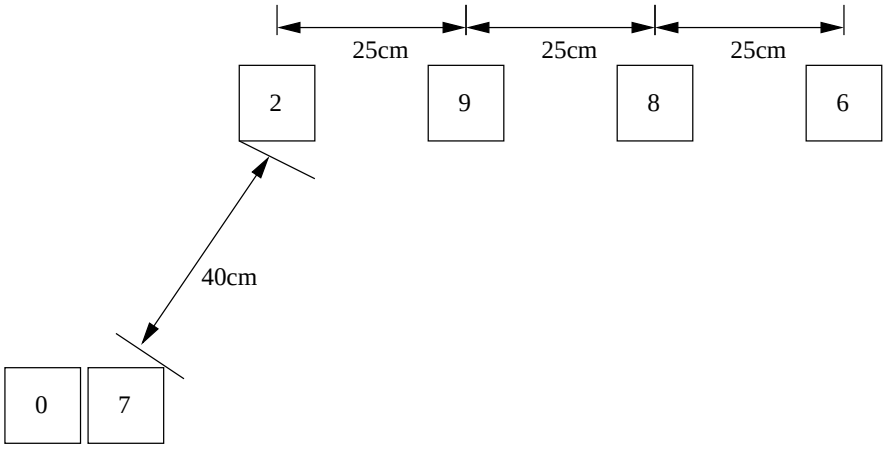

Figura 5.7: *Posizionamento dei motes nella seconda prova.*

L'albero ottenuto è mostrato in Figura 5.8. Come si può notare l'albero era grossomodo quello che ci si poteva aspettare.

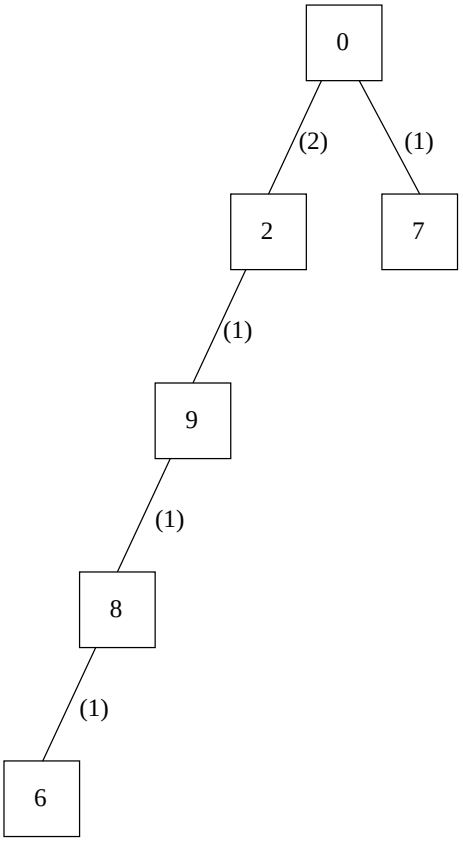

Figura 5.8: *MST, dalla topologia Figura 5.7.*

5.2 Prove complete per l'acquisizione di dati

Vengono ora presentate una serie di prove di acquisizione dati di temperatura eseguite in ambiente residenziale. L'appartamento oggetto delle prove consta di due piani le cui planimetrie sono mostrate in Figura 5.2.

Le prime prove, della durata di quindici minuti circa, avevano lo scopo di valutare il corretto funzionamento di *AMPL* unito ad *FPS*. Sono state eseguite posizionando alcuni motes su un piano. L'intervallo di campionamento ed invio dei dati è di 20 secondi mentre la durata dello slot è $T_s = 100ms$ più che sufficiente per trasmettere il pacchetto dati.

Non è stata posta alcuna cura sul posizionamento dei motes, essi sono stati posizionati in orizzontale e con orientazione casuale, in modo da simulare il più possibile quelle che potrebbero essere le reali condizioni di impiego.

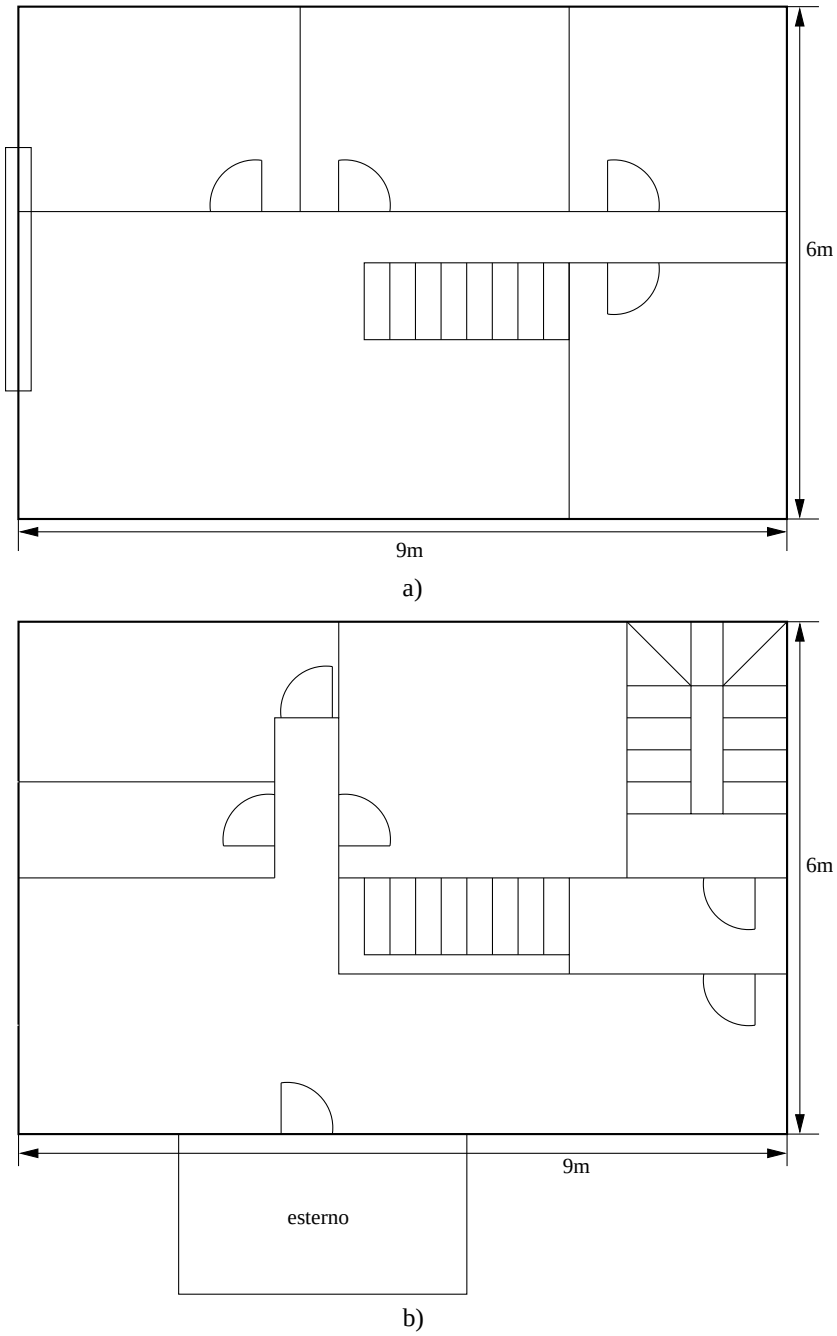

Figura 5.9: *Planimetrie: a) piano superiopre, b) piano inferiore.*

5.2.1 Prova n°1

In questa prova sono stati posizionati quattro motes come mostrato in Figura 5.10. I motes sono rappresentati con dei rettangoli neri, mentre la base station è raffigurata con un computer schematizzato. La disposizione dei motes è la seguente: 4 e 5 sono posizionati in punti diversi della stessa stanza, il 9 è in un'altra stanza più piccola, mentre il 7 è sul davanzale esterno della finestra.

Le frecce rappresentano i percorsi scelti dai motes per raggiungere la base station. Si vede ad esempio che il motes 9 ha scelto il 5 come successore, mentre gli altri comunicano direttamente. I dati raccolti dai sensori sono riportati nel grafico 5.11.

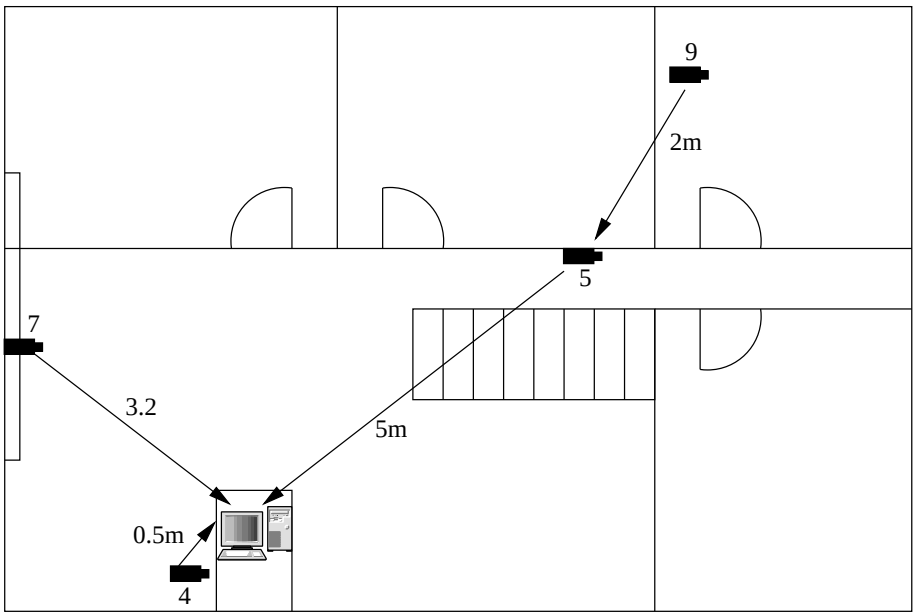

Figura 5.10: *Primo prova: posizionamento dei motes.*

Si vede che il 5 e il 7 riportano sostanzialmente temperature costanti durante tutto lo svolgimento della prova. Il nodo 7 posizionato all'esterno segna un grado in meno rispetto alla temperatura intera dell'abitazione. Esaminando attentamente la traccia del nodo 5 si può notare l'assenza della ventiquattresima lettura: si tratta di un pacchetto perso. Per applicazioni di questo genere perdere una lettura su 50 è da considerarsi comunque tollerabile, inoltre questo è stato l'unico pacchetto perso di tutti i sensori.

I motes 9 e 4 presentano temperature non costanti. Il nodo 9 infatti è stato posto sotto una finestra in modo da evidenziare quali possano essere le escursioni termiche in presenza di luce diretta. Per un intervallo di tempo di 20 letture circa, corrispondenti grossomodo a sei minuti, ha riportato temperature oltre i 34 gradi centigradi con un picco

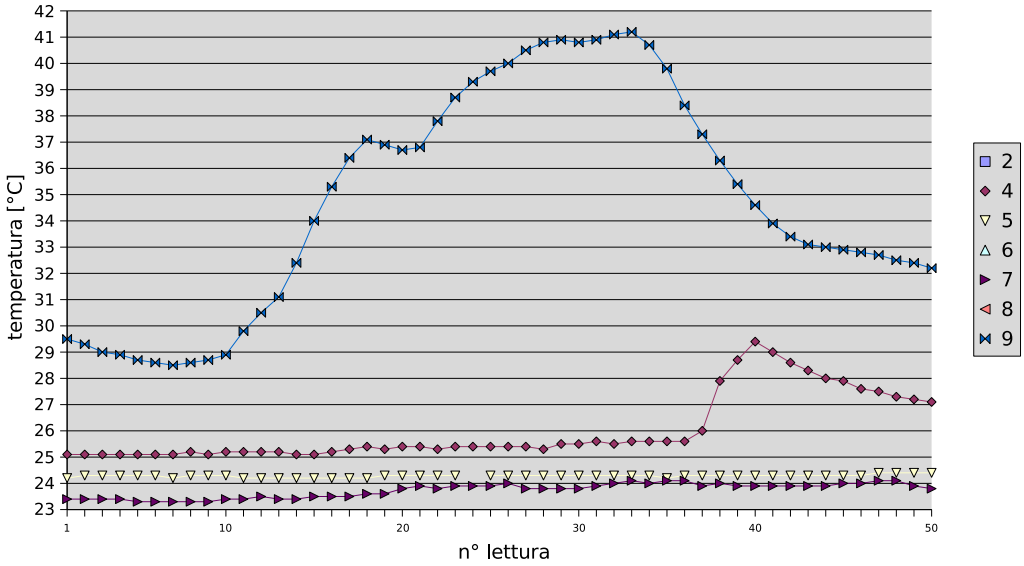

Figura 5.11: *Grafico dei dati raccolti dai sensori nella prima prova. Il periodo tra una misura e l'altra è di 20 sec.*

di 41. Questo è stato fatto intenzionalmente; come è noto per applicazioni di rilevamento di temperature ambientali i sensori vanno posti distanti da fonti di luce diretta.

Lo strano picco al termine della traccia del nodo 4 invece, non è stato affatto intenzionale. Questo motes era in prossimità del notebook collegato alla base station, è il brusco aumento di temperatura è esattamante in corrispondenza dell'accensione della ventolina di raffreddamento del notebook.
L'analisi di questa traccia, anche se parziale offre uno spunto di riflessione su quale possa essere l'inerzia termica del dispositivo. Si vede che la curva di salita è abbastanza ripida, valutabile intorno agli $0.05\,°C/sec$, mentre quella di discesa è più dolce e dall'andamento esponenziale decrescente. Probabilmente il sensore della *Sensirion* montato sui *Tmote Sky*, essendo elettronico (si tratta essenzialmente di una termocoppia), ha bassa inerzia termica. Tuttavia poichè l'aria in uscita dal notebook può raggiungere temperature elevate è plausibile che per il breve tempo che è fuoriuscita abbia scaldato l'intero motes rendendo la curva di discesa sensibilmente più lenta.

Da questo si evince che per valutare la velocità di risposta del sensore potrebbe essere importate valutare l'inerzia termica dell'intero dispositivo piuttosto che solo quella del sensore, che da quanto visto risulta sensibilmente minore.

5.2.2 Prova n°2

Quattro motes sono stati posizionati come in Figura 5.12.

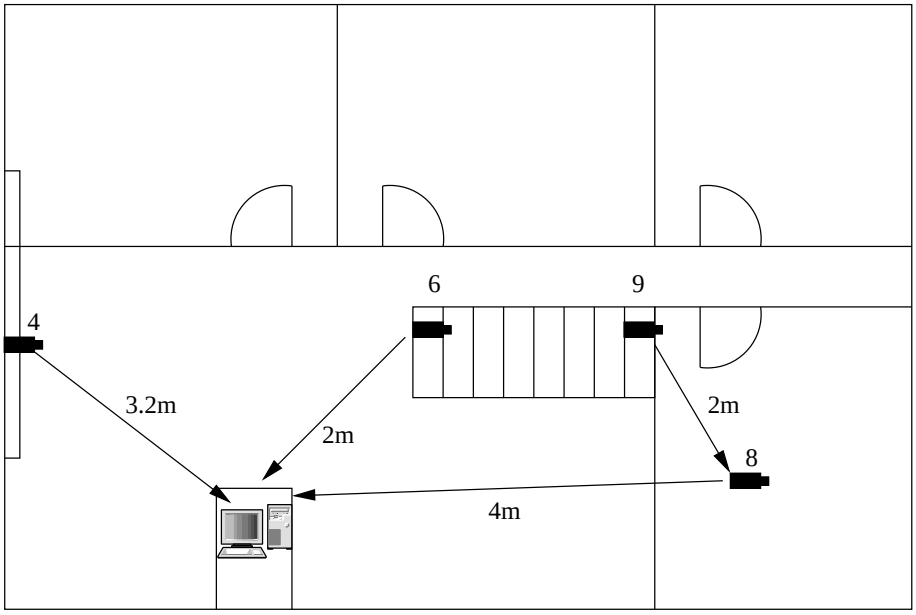

Figura 5.12: *Seconda prova: posizionamento dei motes.*

Il motes 4 è sempre sul davanzale come nella prova precedente, il 6 e il 9 sono sulla scala che conduce al piano inferiore sul primo e sull'ultimo gradino rispettivamente. Mentre l'8 è al piano inferiore.

La cosa "strana" di questa prova è che il motes 9 anzichè comunicare direttamente con la base station o passare tramite il 6, trova più conveniente passare per l' 8 che è il più lontano. Come già affermato in precedenza in ambienti ristretti influiscono pesantemente sulle radiocomunicazioni anche le riflessioni sulle pareti, pertanto risultati simili anche se non prevedibili sono certamente plausibili. L'analisi delle temperature riportata in Figura 5.13 non presenta particolari di rilievo se non la perdita di qualche pacchetto da parte di tutti i nodi a parte il 6.

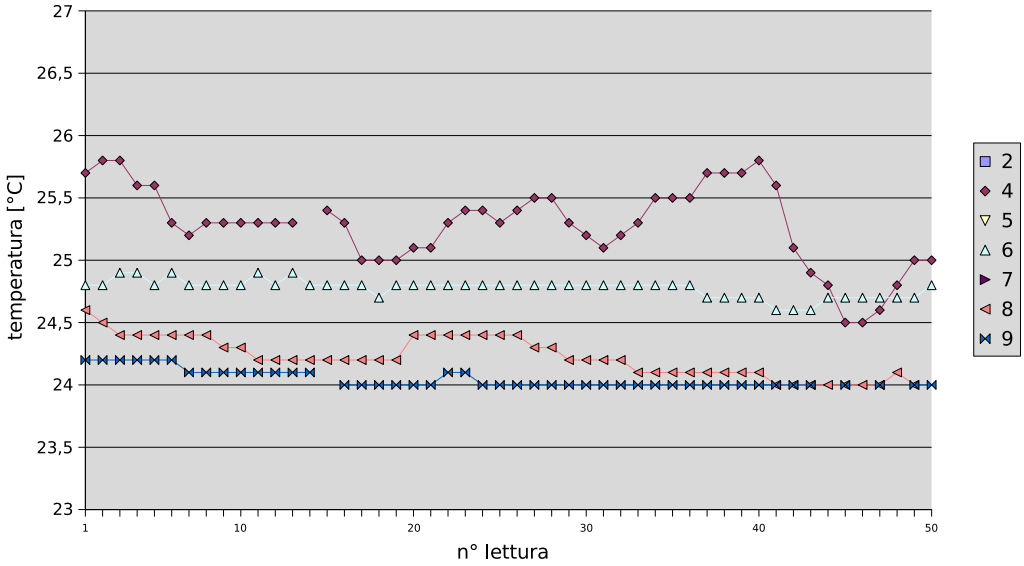

Figura 5.13: *Grafico dei dati raccolti dai sensori nella seconda prova.*

5.2.3 Prova n°3

Quest'ultima prova è stata effettuata per valutare le capacità multi-hop della rete. I motes sono stati predisposti su di un percorso che si snoda su due piani come si vede in Figura 5.14. Il motes 4 è al piano superiore, il 9 a metà scala mentre i restanti sono al piano inferiore. Dalle frecce riportare in Figura 5.14 si nota che le cose sono andate più o meno come ci si aspettava, il percorso logico creato dai dispositivi infatti, coincide con il percorso geografico che porta alla base station.

Sull'analisi dei dati riportata in Figura 5.15 si possono fare un paio di considerazioni. Una banale, sulle temperature che sono in calo, questo è dovuto semplicemente al fatto che la prova è stata eseguita verso sera. La seconda forse più interessate è che non si sono evidenziate perdite di pacchetti nonostate il nodo 8 ad esempio, abbia dovuto fare ben quattro hops per raggiungere la base station.

Probabilmente anche questo è dovuto all'orario delle prove. Come è noto, per una serie di fattori, le comunicazioni wireless la sera e soprattutto la notte sono più affidabili perchè meno soggetti ad interferenze. Pertanto quando si realizzano acquisizioni a lungo termine sarebbe meglio eseguire il setup della rete, ossia l'AMPL, nel momento peggiore, ovvero in pieno giorno.

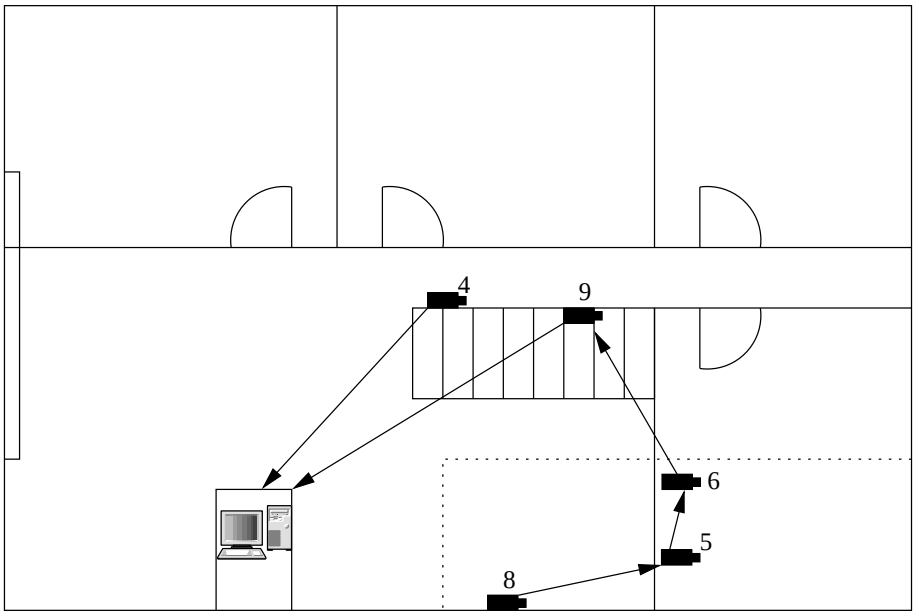

Figura 5.14: *Terza prova: posizionamento dei motes.*

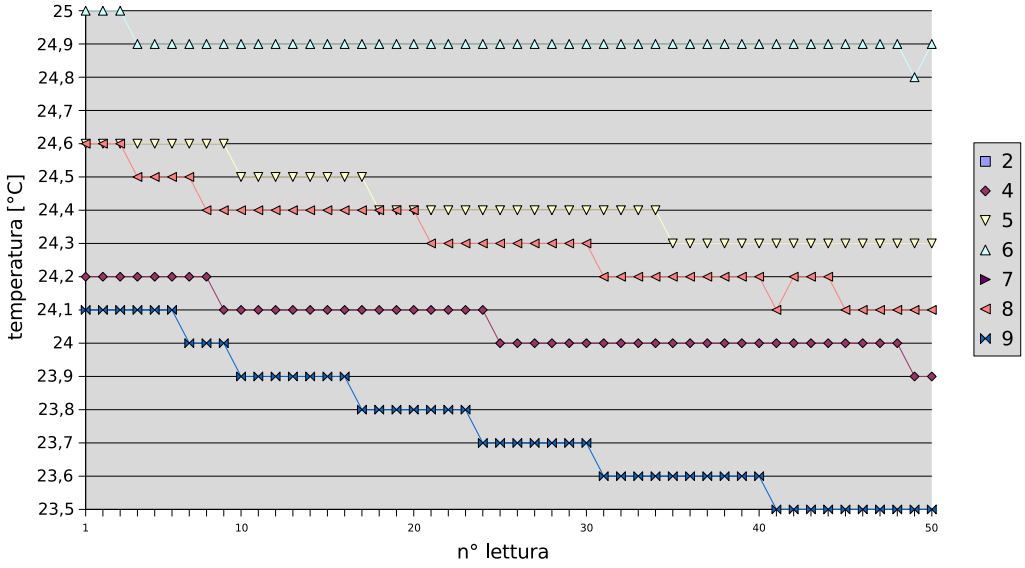

Figura 5.15: *Grafico dei dati raccolti dai sensori nella terza prova.*

5.3 Prova di acquisizione prolungata

L'ultima prova presentata prevede di fornire una valutazione più approfondita del comportamento dell'applicazione. Sono stati impiegati sette motes distribuiti su due piani come mostrato in Figura 5.3. La frequenza di acquisizione è stata incrementata ad un campione ogni 10 secondi, mentre gli slot sono da $T_s = 200ms$. La durata complessiva della prova è stata di tre ore per un totale di 1080 campioni acquisiti per ogni motes. Il grafico delle temperature è mostrato in Figura 5.17. Possiamo notare che le temperature dei motes posti all'esterno: il 4 e il 6 sono più alte di quelle interne e presentano variazioni di un paio di gradi dovute a fenomeni climatici (vento, nuvole, ecc. ecc.), mentre quelle dei motes interni tendono a mantenere una temperatura costante.

Vale la pena notare lo strano comportamento dei motes 2,5,7,8 ovvero i motes posti al piano inferiore. Questi motes contrariamente al 9 che era al piano superiore presentano un transitorio iniziale della durata di ben 20 minuti prima di stabilizzarsi sulle loro temperature, questo è dovuto al fatto che prima di iniziare la prova si trovavano tutti al piano superiore, quindi tutti alla temperatura del motes 9 più alta di circa un grado rispetto quella del piano sotto. Quando si effettuno misurazioni di questo tipo bisognerebbe tenere conto anche della presenza di questi transitori, e magari scartare i valori iniziali.

É interessante anche analizzare la percentuale di pacchetti persi riportata in tabella.

Motes						
2	4	5	6	7	8	9
0.6%	5.0%	0.4%	0.0%	0.8%	0.1%	0.0%

Si possono apprezzare le ottime prestazioni del time slotting di FPS, infatti la percentuale di pachetti persi è assai bassa. Questo potrebbe suggerire che sarebbe stato possibile una riduzione ulteriore dello slot per portarlo magari ad 80ms che è il valore usato in [3], il che avrebbe men che dimezzato il consumo energetico mantenendo comunque la percentuale di pacchetti persi entro valori accettabili (vedasi Tabella 4.1).

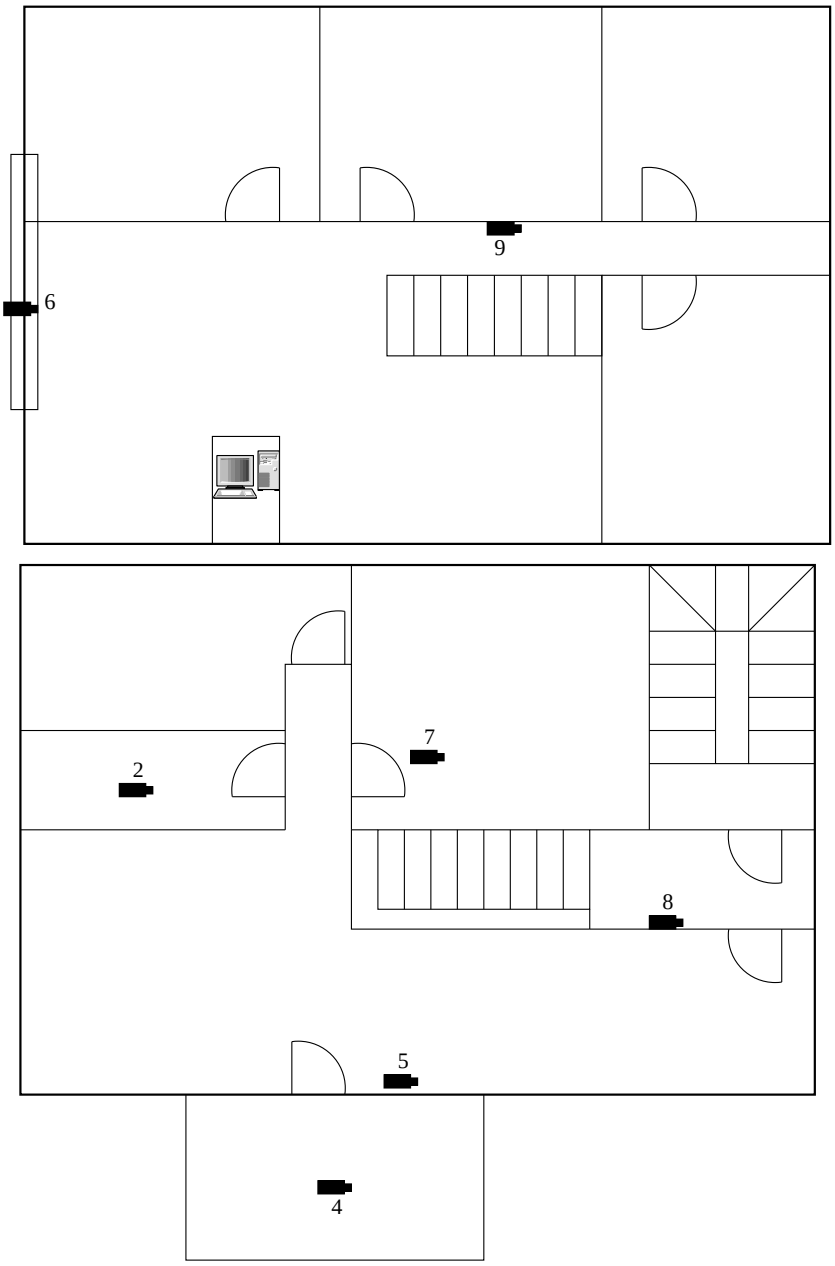

Figura 5.16: *Distribuzione dei motes nel piano superiore e inferiore rispettivamente.*

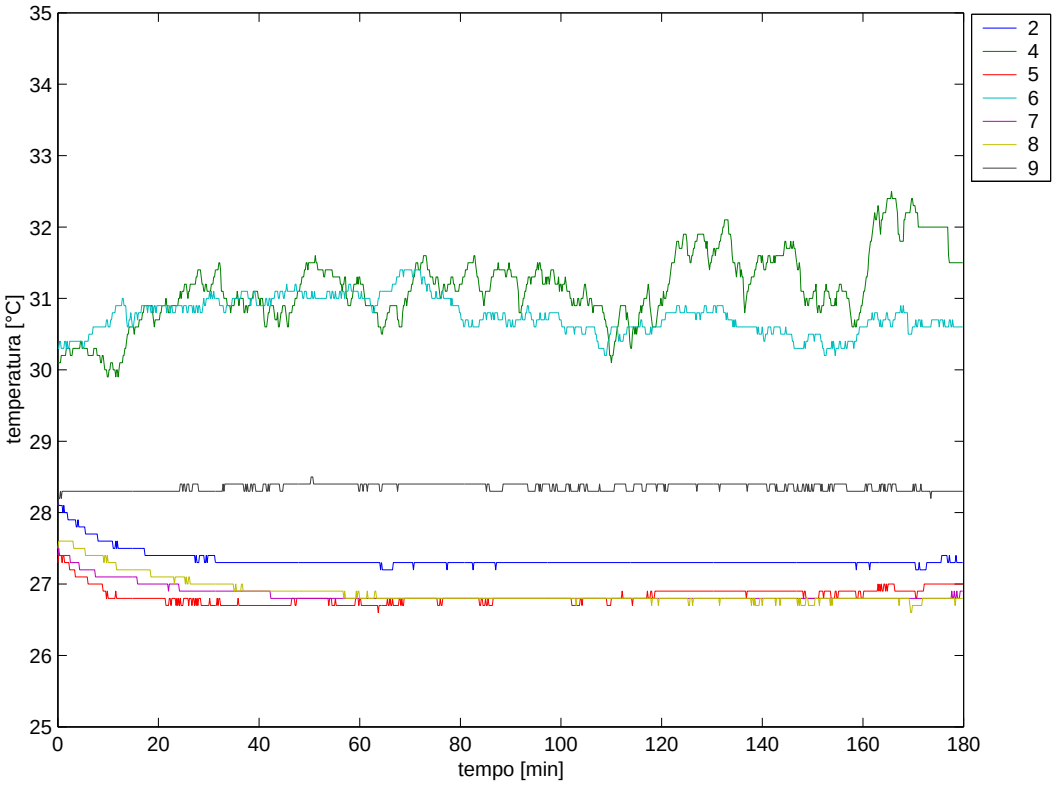

Figura 5.17: *Grafico dei dati raccolti nell'ultima prova*

Capitolo 6

Conclusioni e sviluppi futuri

Come emerso dall'analisi teorica e dalle prove sperimentali, la soluzione proposta si è comportata nel complesso molto bene. Riassumeremo nel corso di questo capitolo le principali caratteristiche.

Come visto precedentemente nell'introduzione, non è possibile realizzare una rete di sensori wireless senza l'uso di una qualche tecnica di riduzione dei consumi. Qui si è scelto di intervenire su due fronti.

Considerando il fatto che uno dei componenti che consumano maggiormente all'interno di un motes è il trasmettitore/ricevitore si è pensato come prima cosa di ridurre al minimo le potenze dei trasmettitori con l'algoritmo AMPL. Valutre il risparmio energetico introdotto da questa soluzione è molto difficile poichè dipende essenzialmente dalla topologia della rete in questione. Il vantaggio è che la soluzione sicuramente non introduce peggioramenti, quindi conviene sempre utilizzarla.

Il secondo fronte su cui si è intervenuti è il duty-cicle dei motes. Si cerca cioè di mantenere i motes spenti per la maggior parte del tempo, per poi accenderli quando necessario. Per fare questo è stato necesario sincronizzare le comunicazioni tra i motes perchè per avere la comunicazione tra due dispositivi occorre che siano accesi entrambi nello stesso perido.

Si è scelto di utilizzare l'algoritmo FPS. Questa soluzione consente un notevole risparmio energetico e contemporaneamante mantiene estremamente ridotto il numero di pacchetti persi, che per una rete wireless tende sempre ad essere elevato. Per avere un'idea della validità della soluzione basta osservare la Tabella 5.3, si vede che la percentuale di letture perse si mantiene in ogni caso non superiore al 5%.

C'è però, un'importante lacuna da colmare: la sincronizzazione temporale tra i motes. Come abbiamo visto nel paragrafo 3.4.6, tra nodo genitore ed un figlio occorre vi sia una

sincronizzazione almeno grossolana. Cerchiamo ora di valutare quanto possa essere la deviazione massima ammissibile.

In Figura 6.1 è riportato il diagramma temporale di un motes, dove i cicli hanno durata T_c, e gli slots T_s.

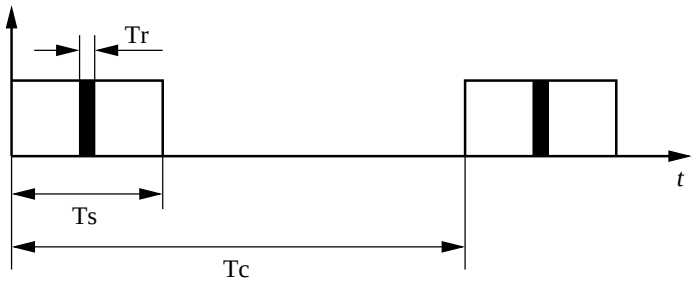

Figura 6.1: *Diagramma temporale di un motes. Le trasmissioni/ricezioni avvengono nella finestra temporale di ampiezza* T_r.

Il messaggio del figlio può essere ricevuto dal genitore solo se quest'ultimo si trova acceso nell'istante in cui avviene la trasmissione. Come abbiamo visto gli slot sono larghi a sufficienza per consentire la trasmissione di almeno due messaggi, quindi finchè i motes sono sincronizzati non ci sono problemi. A lungo andare però i motes tendono a perdere il sincronismo a causa del drift degli oscillatori, pertanto si è scelto di fare l'invio del messaggio al centro dello slot, cioè dopo un tempo $T_s/2$ (vedi rettangolo pieno nero di Figura 6.1). Questo ci consente di avere una certa tolleranza sulla desincronizzazione. Infatti se supponiamo che il motes abbia bisogno di un tempo T_r per ricevere un messaggio, otteniamo che i due motes possono tollerare una desincronizzazione di

$$\Delta T = \frac{T_s - T_r}{2} \tag{6.1}$$

secondi.

Dall'equazione 6.1 si deduce che se vogliamo rendere la comunicazione più tollerante ai drift degli oscillatori dobbiamo aumentare la larghezza degli slot: T_s. Così facendo però aumentiamo il tempo che i motes rimangono accesi e pertanto i consumi. Piuttosto che aumentare T_s risulta alora conveniente risincronizzare i motes ad intervalli regolari.

Per sapere ogni quanto sincronizzare i motes bisogna conoscere la deriva massima degli oscillatori. Per i *Tmote Sky* abbiamo indicativamente che l'oscillatore ha una deriva massima di: $d = 10\mu s/s$ (Vedi [1] Par. 6.7).

Il sincronismo tra i motes è garantito per un tempo

$$T_{sync} = \frac{\Delta T}{d}. \tag{6.2}$$

Se poniamo $T_s = 200ms$, e $T_r = 10ms$ che sono dei tempi ragionevoli, otteniamo $T_{sync} \cong 26\, ore$.

Per ovviare a questo problema si può adottare una delle soluzioni proposte in [3]. Come prima soluzione si potrebbe fare in modo che il nodo figlio conosca qando trasmette il padre, questa informazione gli potrebbe assere passata ad esempio nell'advertisement, e si mettesse in ascolto ad intervalli regolari per risincronizzarsi con il padre. Poichè come visto non serve una sincronizzazione precisa, questi intervalli potrebbero essere ad esempio di due ore.

Un'altra soluzione più semplice potrebbe essere che il nodo figlio si metta periodicamente in ascolto degli advertisement del padre per effettuare la sincronizzazione su di essi. Questa soluzione è meno efficiente della precedente perchè prevede di tenere acceso il ricevitore per un ciclo intero, mentre nella precedente bastava uno slot. Tuttavia, poichè la sincronizzazione è fatta ad intervalli molto lunghi (26 ore), le due soluzioni non differiscono di molto nei consumi, quindi la seconda potrebbe essere preferibile perchè di più facile implementazione.

Un utlimo accorgimento per ridurre ancora di qualcosa i consumi potrebbe essere quello di sospendere dopo un certo periodo di tempo l'invio degli *Advertisement*. Poichè la rete è supposta invariante nel tempo, è lecito supporre che dopo l'invio di un numero *n* di *Advertisement* senza ricevere alcun *Receive Request* la richiesta di slot da parte dei figli sia completamente soddisfatta. A questo punto gli *Advertisement* diventano inutili, quindi si può sospenderne l'invio. Si risparmia in questo modo uno altro slot per ogni ciclo.

Un'altro aspetto che meriterebbe un approfondimento è rappresentato dalle costanti utilizzate dal programma. Molte di esse, come ad esempio le costanti di tempo, sono state ricavate in base a parametri sperimentali. Varrebbe la pena di indagare sulla possibilità di modallare queste costanti su quelli che possono essere i parameri principali di una rete.

Appendice A

Schemi logici

A.1 Schema generale dell'applicazione

In quest'appendice è riportato lo schema logico di collegamento dei componenti dell'applicazione. Il modulo principale è HtlM, che gestisce le varie fasi dell'implementazione. Si possono notare in particolar modo il componente HumidityC che fornisce il controllo sul sensore di temperatura ed umidità ed i componenti CC24220C e UART che consentono di comunicare via radio e via USB rispettivamente. La porta USB in realtà è usata solo dalla base station per trasferire i dati provenienti dai sensori al calcolatore.

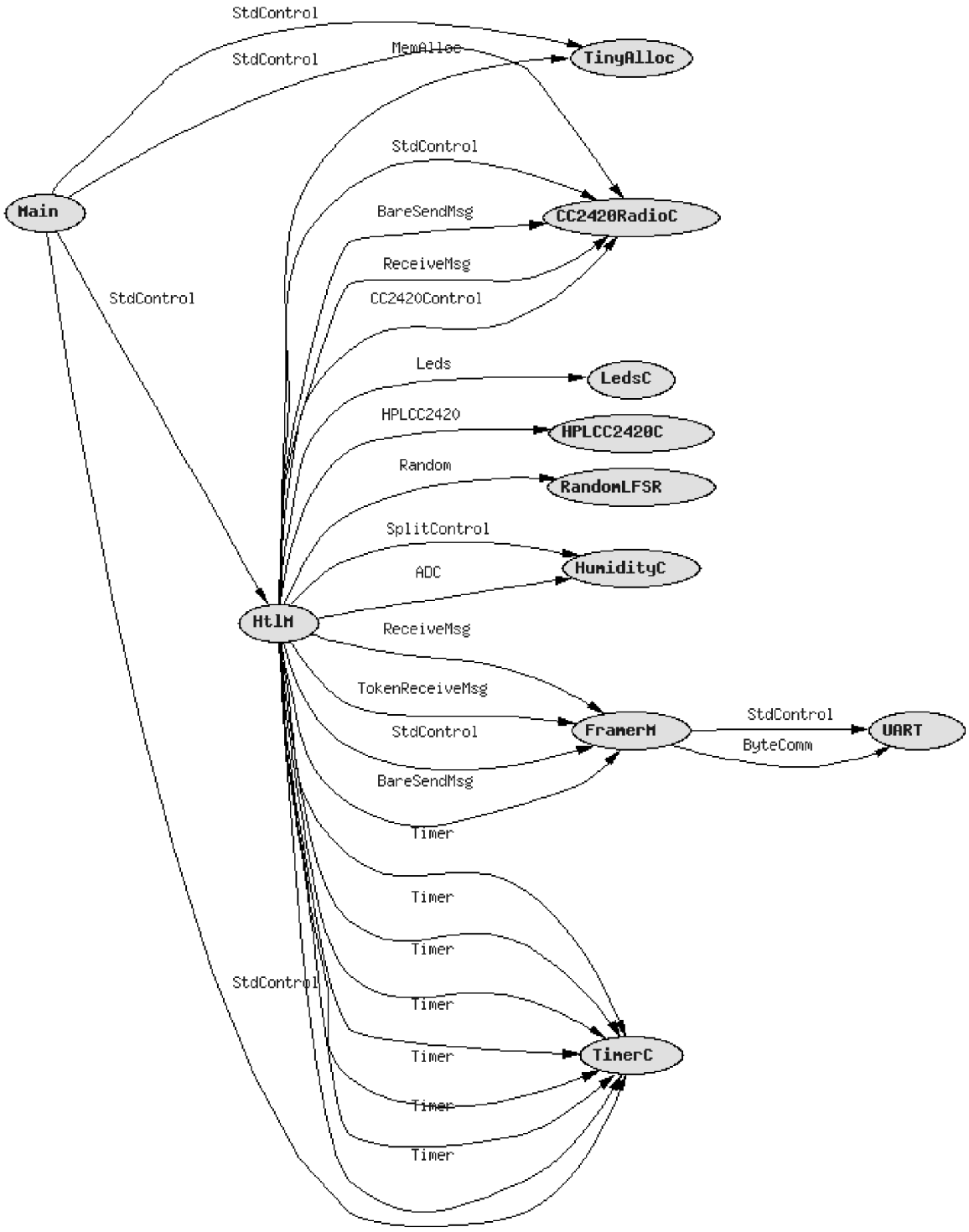

Figura A.1: *Schema delle interfacce e dei moduli usati dall'applicazione.*

Appendice B

Diagrammi di flusso dell’algoritmo AMPL

Vendgono riportati i diagrammi di flusso di alcune parti di AMPL.

B.1 Diagramma di flusso della fase di *Discovery*

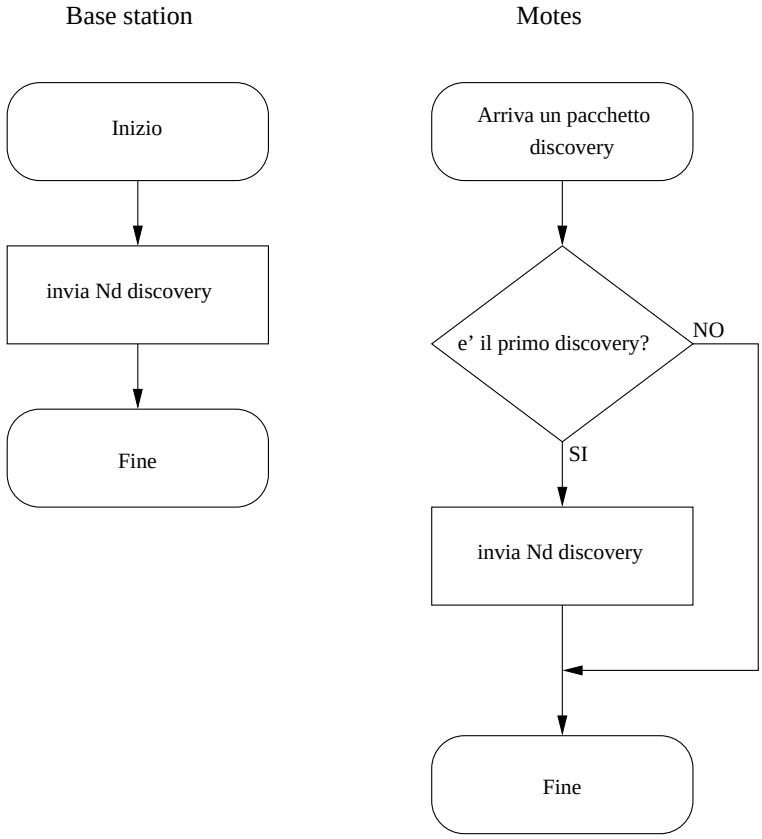

Figura B.1: *Flow-chart del codice eseguito dalla Base station e dai motes durante la fase di* Discovery *in AMPL.*

B.2 Diagramma di flusso della fase di creazione dell percorso verso la base station (*baseStationPath*)

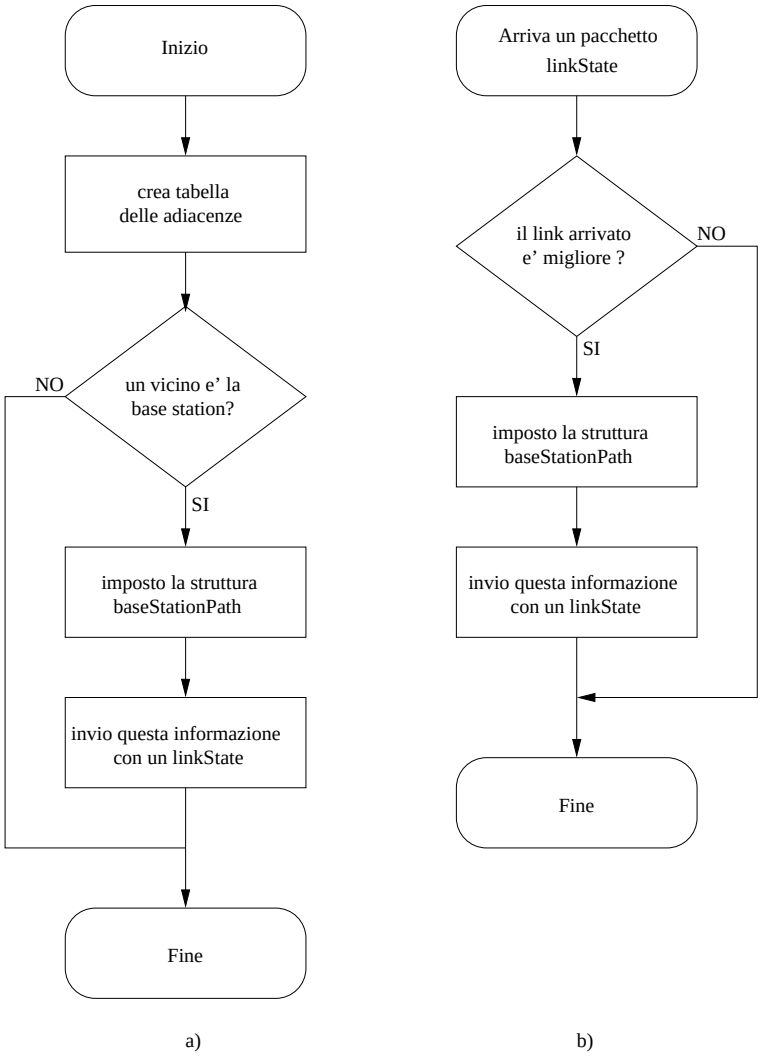

Figura B.2: *Flow-chart del codice eseguito dai motes durante la fase di invio delle tabelle di linkState e di creazione della baseStationPath in AMPL.*

Appendice C

Codice *nesC* di alcune parti dell'applicazione

C.1 Struttura *baseStationPath*

Struttura *baseStationPath*, usata per contenere le informazioni necessarie a raggiungere la base station. Il campo *powerLevel* rappresenta il costo totale del percorso per raggiungere la base station ed è dato dalla somma dei costi (livelli di potenza dei trasmettitori) di tutti i link componenti il percorso.

```
struct baseStationPath{
   /*  indice del record della adjTable contenente
       il vicino per raggiungere la base st.
   */
   uint8_t index;
   uint8_t powerLevel; /* costo totale */.
   uint8_t n_hop; /* numero di hop */
};
```

Si può notare come il campo *index* sia stato dichiarato come `uint8_t` che significa intero senza segno ad 8 bit. Questo limita il numero di vicini di un nodo a 255, però richiede poca memoria, quindi consente di mantenere più compatto il codice. Se si dovesse avere la necessità di un numero maggiore di vicini bisognerebbe impostare *index* di tipo `uint16_t`, che consente un massimo di 65535 vicini per nodo.

C.2 File di configurazione *Htl.h*

Tutti i parametri e le strutture dati dell'applicazione sono contenuti nel file *Htl.h*. Modificando questo file è possibile variare le costanti di tempo, la dimensione dei buffer, il numero di ping, le ritrasmissioni, ecc. ecc. Per ogni constante è presente un rapido commento, mentre altri commenti sono presenti nel codice, dove le costanti vengono utilizzate.

```
#ifndef HTL_H
#define HTL_H

#define HTL_GROUP    0xAA
#define MAX_ADJ        32 //numero massimo di vicini ammesso
#define MAX_POWER_LEVELS  8  // livelli di potenza possibili

#define TX_QUEUE_DIM   10  // dimensione della coda di invio
#define RX_QUEUE_DIM   10  // dimensione della coda di ricezione
#define UART_QUEUE_DIM 10  // dimensione della coda USB

/* tipi di messaggio */
enum {
  PING_MSG=        0x01,
  DISCOVERY_MSG=   0x03,
  PINGTABLE_MSG=   0x04,
  ACK_MSG=         0x05,
  LSP_MSG=         0x06,
  DATA_MSG=        0x07,
  RESET_MSG=       0x0A,
  ADV_MSG=         0x0B,
  RR_MSG=          0x0C,
  RC_MSG=          0x0D,
};
/* costanti varie */
enum {
  ND=        3,     /* numero di pacchetti di discovery da inviare */
  TD=        250,   /* periodo di invio pacchetti di discovery */
  TDRND=     2048,
  TDISC=     5000,  /* tempo prima di far partire la fase di ping */
  TDISCRND=  128,
  NP=        20,    /* MAX 255, se no devi mettere uint16_t in pingBuf */
  TP=        500,   /* periodo di invio pacchetti ping */
```

```
  TPRND=     1024,
  MIN_NP=    15,    /* numero di ping minimo che devono arrivare per
                       considerare affidabile la comunicazione */
  PT_DELAY=      30000U,  /* ritardo prima di iniziare la fase di invio
                             delle tabelle */
  PT_DELAYRND=   128,
  BASE_PT_DELAY= 110000U, /* ritardo della base station prima di iniziare
                             la fase di invio dalle tabelle */
  NT=            4,       /* numero di tabelle di ping da inviare */
  TT=            1000,    /* periodo di invio pacchetti tabelle */
  TTRND=         512,
  TLSP_DELAY_START= 40000UL, /* ritardo prima di iniziare la fase di
                                distribuzione degli LSP */
  TLSP_DELAY=  30000,   /* ritardo prima di inviare il primo LTSP per
                           permettere init di baseStation */
  TLSPRND=     64,
  DATASTART=   220000UL,/* usato dalla base station per far iniziare
                           la tx dei dati */
  NS=      3,      /* numero di volte che viene replicato un suyc */
  TS=      250,    /* periodo di invio pacchetti di discovery */
  TSRND=   2084,
  TDATA_TXRND= 32, /* costante per il tempo aleatorio massimo per
                      l'invio del dato */
  TDATA_RETXRND= 64, /* costante per tempo aleatorio massimo per
                        la ritrasmissione di un pkt */
  FPS_TC=  10000,  /* periodo di campionamento dell'FPS */
  N_SLOTS= 50,     /* numero di slot */
  FPS_TS=  FPS_TC/N_SLOTS,/* durata di un time slot */
  T_RESET= 5000,          /* tempo di attesa prima di resettare la rete */
  POWER_SETUP_DELAY= 50  /* ritardo per dare il tempo al motes di
                            impostare la temperatura */
};
/* costanti per il primo timer */
enum {
  T1_DISCOVERY,
  T1_PING,
  T1_PING_START,
  T1_PINGTABLE,
  T1_LSP,
  T1_RESET_MSG,
```

```
  T1_SEND_MIN,
  T1_RESET,
  T1_TXON,
};

/* costanti per il secondo timer */
enum {
  T2_LSP,
  T2_DATA_START,
  T2_DATA,
  T2_FPS_START,
  T2_FPS
};
/* costanti per il terzo timer */
enum {
  T3_FPS
};

enum {
  ACK_PINGTABLE= 0x01,
  ACK_DEBUG=     0x02
};
typedef enum slot_type{
  SLOT_T,
  SLOT_R,
  SLOT_A,
  SLOT_TP,
  SLOT_RP,
  SLOT_I
} slot_t;

enum {OFF,ON};

struct pingTableRecord{
  uint16_t sourceMoteID; /* id del mote che ha inviato i ping */
  uint8_t powerLevels[MAX_POWER_LEVELS];
};
/* struttura con informazioni sul percorso per raggiungere la base station*/
struct baseStationPath{
  uint8_t index;
```

```
  uint8_t powerLevel;
  uint8_t n_hop;
};
/* messaggio di ping */
struct PingMsg
{
  uint16_t sourceMoteID;
  uint8_t powerLevel;
  uint8_t pingNumber;
};
struct AckMsg{
  uint16_t sourceMoteID;
  uint8_t type;
};
/* messaggio di discovery */
struct DiscoveryMsg{
  uint16_t sourceMoteID;
};

/* messaggio di pingTable */
struct PingTableMsg{
  uint16_t sourceMoteID;           /* sorgente del pacchetto */
  struct pingTableRecord record;  /* record della tabella pingTable */
};
struct adjTableRecord{
  uint16_t dest;       /* vicino */
  uint8_t powerLevel;  /* potenza necessario per raggiungerlo */
};
struct LinkStateMsg{
  uint16_t sourceMoteID; /* sorgente del pacchetto */
  uint8_t indiceVicino;  /* per usi futuri */
  uint8_t lsmNumber;     /* numero del pacchetto */
  uint8_t powerLevel;    /* costo del percorso fino alla base st. */
  uint8_t n_hop;         /* numero di hops per raggiungere la base st */
};
struct DataMsg{
  uint16_t sourceMoteID;  /* sorgente del pacchetto */
  uint16_t sensorMoteID;  /* ID del mote che ha fatto la lettura */
  uint16_t temperature;   /* temperatura */
  uint16_t counter;       /* contatre delle misure */
```

```
};

struct ResetMsg{
  uint16_t sourceMoteID;  /* sorgente del pacchetto */
};
struct AdvMsg{
  uint16_t sourceMoteID;
  uint8_t adv_slot;
  uint8_t reservation_slot;
};
struct RCMsg{
  uint16_t sourceMoteID;
  uint8_t reservation_slot;
};
#endif
```

Bibliografia

[1] Basso Alessio. Sincronizzazione temporale in reti di sensori wireless, 2006.

[2] Adam Wolisz Andreas Willing, Kristen Matheus. Wireless technology in industrial networks. *IEEE/ACM Transactions on Networking*, 93(6):1133–1134, June 2005.

[3] Eric Brewer Barbara Hohlt, Lance Doherty. Flexible power scheduling for sensor networks. Article ACM 1-5813-846-6/04/0004, University of California, Berkeley, april 2004. (disponibile all'indirizzo `http://www.cs.berkeley.edu/~hohltb/papers/ipsn_hohlt.pdf`).

[4] D. Culler D. Estrin D. Ganesan, B. Krishnamachari and S. Wicker. An empirical study of apidemic algorithms in large scale multihop wireless networks. February 2002. (disponibile all'indirizzo `http://citeseer.nj.nec.com/genesan02empirical.html`).

[5] Union Tecnique de l'Electricitè. *General Purpose Field Communications System: PROFIBUS*. Number EN-50170. Union Tecnique de l'Electricitè, 1996.

[6] Robert Gallager Dimitri Bertsekas. *Data Networks*. Prentice Hall, second edition edition, 1992.

[7] Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans)(ieee 802.15.4 standard). Technical Report ISBN 0-7381-3686-7, The Institute of Electrical and Electronics Engineers, Inc., may 2003.

[8] Duracell Labs inc. Alkaline-manganese dioxide batery (aa type). Datasheet. (disponibile all'indirizzo `http://www.duracell.com`).

[9] The *nesC* language: A holistic approach to networked embedded systems. Article 1-58113-662-5/03/0006, ACM, june 2003. (disponibile all'indirizzo `http://nescc.soureforge.net`).

[10] Nelson Lee Philip Levis. Tossim: A simulator for tinyos networks. Article, University of California, Berkeley, june 2003. (disponibile all'indirizzo `http://www.tinyos.net/tinyos-1.x/doc/nido.pdf`).

[11] Andrew S. Tanenbaum. *Computer Networks*. Pearson Education Internationl, fourth edition edition, 2001.

[12] J.P. Thomasse. *The WorldFIP Fieldbus in the Industrial Information Technology Handbook*. R. Zurawski Ed. Boca Raton, FL, 2005.

[13] Tmote sky. Datasheet, MoteIV Corp, june 2006.

[14] G. Cena A. Valenzano. *Operating principles and features of CAN for high speed communications*. 1993.

[15] David Culler Xiaofan Jiang, Joseph Polastre. Perpetual environmentally powered sensor networks. Technical report, University of California, Berkeley, april 2005. (disponibile all'indirizzo `http://www.polastre.com/pubs.html`).

[16] Zigbee specification. Protocol Spec. 053474r06, Version 1.0, ZigBee Alliance, june 2005.

Elenco delle figure

Elenco delle tabelle

Ringraziamenti

Giunto al termine di questa lunga esperienza è doveroso dedicare un paio di righe a tutti coloro che durante questi anni mi sono stati vicini. Da chi si comincia? Ma dai genitori ovviamente, loro erano con me ancora prima che nascessi e lo sono ancora, mi hanno aiutato a crescere... mi hanno insegnato a fare le divisioni, che per un ingegnere è importante! Sono loro che *mi hanno fatto studiare*, ed anche se non sono stato proprio uno studente modello, pare che che alla fine ce l'abbia fatta. Grazie.

Ringrazio mio fratello, per aver vigilato sul mio italiano durante tutti i miei anni scolastici e accademici.

I compagni di studio di sempre, Simone ed Elvis, con loro ho condiviso gran parte dei giorni di studio ma anche delle serate Padovane. Ringrazio Elvis per la grande pazienza dimostrata in questi anni di convivenza, e poi perchè avere una *pantera rosa* da 80 Kg che ti gira per casa era sempre stato il mio sogno. Simone che con i suoi "scherzetti" mi ha aiutato a rendere ad Elvis la vita ancora più difficile e con il quale abbiamo condiviso le lunghe ore di studio di molti esami. Ricordo in particolare i primi esercizi di *Comunicazioni elettriche* in cui riuscivamo sempre ad avere due risultati diversi ovviamente entrambi sbagliati!

Ringrazio Luca per i bei momenti passati in appartamento e soprattutto per avere spiegato così bene alle nostre vicine del piano di sopra come la pensavamo su di loro.

Diego, l'inquilino abusivo, l'incomparabile "compagno di giochi", con il quale ho condiviso gran parte delle mie attività ludico-idiote di tutti questi ultimi dieci anni. Abbiamo cominciato appesi come scimmie alle attrezzature di un percorso della salute e siamo finiti imbragati su una parete di roccia... pensare al futuro è preoccupante.

Illen per aver dimostrato a tutti che con perseveranza e forza di volontà si può ottenere **qualsiasi** cosa. Le devo più di un esame, in particolare ricordo Analisi II, senza la sua tenacia avrei dovuto rifare lo scritto.

Gli amici del Thomas! Credo che gli anni passati al Thmoas siano tra i più belli della mia vita. Come posso non ringraziare Loretta e Federica, le nostre mamme acquisite.

Sempre presenti, certe giornate storte basta anche solo vedere qualcuno che ti chiede “tutto bene?”, che la risposta diventa “Si!”. Un ringraziamento speciale va a Loretta e al suo decolte, perchè nelle nostre “lunghe“ corse sull’argine con lei a fianco, potevo gustarmi le facce di stupore-bramosia dei passanti (maschi) che incontravamo.
Grazie anche a Ale, Giorgia, Kally l’optoDj, Enrico e tutti quelli che hanno reso il mio soggiorno di Via Belzoni quello che è stato.

Vengo ora agli amici più ”recenti“. Mi riferisco a quelli che ho conosciuto grazie a Valentina, ovvero Simone (Mok), Francesca, Alberto, Piera, Stefano e Alice. Ringrazio tutti per il loro supporto morale durante il periodo del servizio civile. In particolare ringrazio Mok per essersi offerto in mille modi per aiutarmi nella stesura della tesi, non me ne voglia ma non sapevo proprio cosa fargli fare: le tesi di Ingegneria devono essere esteticamente mediocri, uniformi e noiose, con il suo genio me l’avrebbe sicuramente ”rovinata“! Mi fa piacere aver saputo che per sfogarsi ha deciso di deviare la sua arte su un grande foglio di carta per ritrarmi nelle pose più volgari che la sua mente possa concepire, sono sicuro che il risultato sarà di grande effetto.

Ecco fatto, penso di aver ringraziato tutti... scherzo! Vedo già Valentina dubbiosa che si chiede se non mi sarò dimenticato di lei...

Come posso dimenticarmi di Valentina, sto scrivendo seduto alla sua scrivania! A parte gli scherzi, le devo un grazie speciale, per la sua disponibilità, e per avermi sopportato in questo periodo difficile. La ringrazio per le volte che mi ha fatto studiare e anche e soprattutto per quelle che non mi ha fatto studiare. Ringrazio inoltre i genitori, Adriana, Gigi, e la sorella Paola, per la sincera ospitalità che mi hanno dimostrato in questi ultimi anni di studi. Ringrazio anche Anna, Dino e Caterina che hanno sacrificato un giorno di tranquillità domestica per consentirmi di festeggiare con voi nel migliore dei modi questo evento.

Devo anche ringraziare il mio relatore, Luca Schenato per la grande disponibilità dimostrata durante tutto il periodo di tesi, fino alla laurea.

A voi, e a tutti quelli che sicuramente avrò dimenticato, Grazie.

Padova, 5 luglio 2006