

POLITECNICO DI TORINO

III Facoltà di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

**Sviluppo di una piattaforma di comunicazione energy aware per reti di sensori wireless**



**Relatore:**

Prof. Giovanni Malnati

**Candidati:**

Daniele CLORALIO  
Gianfranco COSTAMAGNA

**Supervisore Aziendale**

Dott. Daniele Marietta Bersana

NOVEMBRE 2012

*Alle nostre famiglie*

# Sommario

Nell'elaborato viene illustrata la progettazione di una rete di sensori wireless energy aware. L'obiettivo è la creazione di una rete a basso consumo energetico, i cui nodi siano in grado di mantenersi tramite una fonte energetica rinnovabile.

Una fase iniziale è stata dedicata allo studio del problema, sia dal punto di vista hardware, sia software.

La scelta di un'adeguata piattaforma hardware ha previsto lo studio di numerosi dispositivi, evidenziando pregi e difetti di ognuno di essi, per selezionare quelli che maggiormente si adattassero ai requisiti richiesti, tra i quali il più importante è il minor consumo di energia.

La ricerca dell'hardware si è conclusa con la scelta di una piattaforma che comprende un microcontrollore, un'antenna e un battery meter: sono stati selezionati, rispettivamente, un MPS430, un BQ27510-G2 e un CC2500.

Si è quindi proceduto allo studio di un algoritmo di routing che permettesse il maggior risparmio energetico possibile, ma allo stesso tempo garantisse alla rete stabilità, resistenza ai guasti, una bassa latenza e una rapida convergenza; tutto ciò, insieme ai limitati consumi, è stato ottenuto a discapito del throughput.

Dopo aver studiato gli algoritmi maggiormente diffusi nel mondo delle reti wireless, apprendendone i pregi e i difetti, se ne è progettato uno che prendesse come punto di partenza i pregi di quelli analizzati e si adattasse ai requisiti richiesti.

Implementato l'algoritmo progettato, si è passati alla fase di testing, la quale ha fornito ottimi risultati, illustrati nel capitolo conclusivo.

# **Ringraziamenti**

Un ringraziamento speciale a Daniele Marietta Bersana che ci ha seguito con preziosi consigli durante l'intero lavoro in Reply.

Un ringraziamento va anche ai suoi colleghi, in particolare a Marco Maddaleno e Nicodemo Belcastro per i consigli tecnici e l'aiuto nell'utilizzo delle attrezzature.

Grazie a Massimo Pennazio, compagno di studi esperto di elettronica e micro-controllori, che ci ha indirizzato verso la scelta di una scheda MSP430 low cost e fornito supporto per alcuni problemi elettronici.

# Indice

<b>Sommario</b>	<b>iii</b>
<b>Ringraziamenti</b>	<b>iv</b>
<b>1 Descrizione del problema</b>	<b>1</b>
1.1 Le reti di sensori wireless . . . . .	1
1.1.1 Introduzione . . . . .	1
1.1.2 L'efficienza energetica . . . . .	3
1.1.3 Creazione di una rete di sensori perenne . . . . .	5
<b>2 Stato dell'arte</b>	<b>7</b>
2.1 Introduzione . . . . .	7
2.2 Protocolli Data Centrici . . . . .	9
2.2.1 Flooding and gossiping . . . . .	9
2.2.2 SPIN . . . . .	10
2.2.3 Directed Diffusion . . . . .	11
2.2.4 Energy-aware routing . . . . .	13
2.2.5 Rumor Routing . . . . .	15
2.2.6 Gradient-based routing . . . . .	17
2.2.7 CADR . . . . .	18
2.2.8 COUGAR . . . . .	20
2.2.9 ACQUIRE . . . . .	22
2.3 Protocolli gerarchici . . . . .	24
2.3.1 LEACH . . . . .	24
2.3.2 PEGASIS . . . . .	25
2.3.3 PEGASIS Gerarchico . . . . .	26
2.3.4 TEEN . . . . .	28

2.3.5	APTEEN . . . . .	30
2.3.6	Energy-aware routing for cluster-based sensor networks . . . . .	31
2.3.7	Self-organizing protocol . . . . .	34
2.4	Protocolli basati sulla posizione . . . . .	37
2.4.1	MECN . . . . .	37
2.4.2	SMECN . . . . .	39
2.4.3	GAF . . . . .	41
2.4.4	GEAR . . . . .	44
2.5	Network flow and QoS-aware protocols . . . . .	47
2.5.1	Maximum lifetime energy routing . . . . .	47
2.5.2	Maximum lifetime data gathering . . . . .	48
2.5.3	Minimum cost forwarding . . . . .	50
2.5.4	Sequential assignment routing (SAR) . . . . .	51
2.5.5	Energy-aware QoS routing protocol . . . . .	51
2.5.6	SPEED . . . . .	52
2.6	ZigBee . . . . .	53
2.6.1	Introduzione . . . . .	53
2.6.2	Protocolli . . . . .	56
2.6.3	Software . . . . .	57
2.6.4	Comunicazione e dispositivi di rilevamento . . . . .	59
2.7	AODV . . . . .	59
2.7.1	Funzionamento . . . . .	60
2.8	Z-MAC . . . . .	61
2.8.1	B-MAC . . . . .	64
2.9	Valutazioni generali e scelte implementative . . . . .	66
<b>3</b>	<b>Architettura</b> . . . . .	<b>68</b>
3.1	Architettura . . . . .	68
3.1.1	Descrizione generale . . . . .	68
3.1.2	Divisione degli slot temporali . . . . .	70
3.1.3	Sincronizzazione temporale . . . . .	71
3.1.4	Sincronizzazione nelle reti wireless . . . . .	73
3.1.5	Meccanismi elementari di sincronizzazione . . . . .	74
3.1.6	Protocolli di sincronizzazione . . . . .	76

3.2	L'algoritmo di routing . . . . .	83
3.2.1	Modelli di consegna dei dati . . . . .	83
3.2.2	Presupposti . . . . .	85
3.2.3	Descrizione dell'algoritmo . . . . .	86
<b>4</b>	<b>Materiali e metodi</b>	<b>99</b>
4.1	Configurazione hardware . . . . .	99
4.1.1	Microcontrollore . . . . .	99
4.1.2	Battery meter . . . . .	105
4.1.3	RF antenna . . . . .	107
4.1.4	RTC . . . . .	110
4.1.5	Architettura scelta . . . . .	111
4.1.6	La scheda su eagle . . . . .	111
4.2	Comunicazione tra i componenti . . . . .	111
4.2.1	I <sup>2</sup> C . . . . .	111
4.2.2	SPI . . . . .	125
4.2.3	I timer dell'MSP430 . . . . .	129
<b>5</b>	<b>Conclusioni</b>	<b>132</b>
5.1	Conclusioni . . . . .	132
5.1.1	Consumi . . . . .	132
5.1.2	Test . . . . .	138
<b>Bibliografia</b>		<b>144</b>

# Capitolo 1

## Descrizione del problema

### 1.1 Le reti di sensori wireless

#### 1.1.1 Introduzione

Una rete di sensori wireless (spesso abbreviata in WSN, ovvero wireless sensor network) indica una determinata tipologia di rete che, caratterizzata da una architettura distribuita, è realizzata da un insieme di dispositivi elettronici autonomi in grado di prelevare dati dall'ambiente circostante e di comunicare tra loro.

Questi dispositivi, tipicamente caratterizzati da piccole dimensioni, bassa potenza e costi contenuti sono chiamati nodi sensori, (in inglese sensor node o, nell'America settentrionale, mote). Essi costituiscono i nodi della rete e sono formati da componenti in grado di rilevare grandezze fisiche (sensori di posizione, temperatura, umidità ecc.), di elaborare i dati e di trasmetterli in radio frequenza verso un coordinatore che renderà disponibili queste informazioni a uno o più computer.

Un esempio di architettura è illustrato in Figura 1.1

La comunicazione tra i nodi della rete, realizzata tramite tecnologia wireless a corto raggio, è solitamente di tipo asimmetrico in quanto i nodi comuni inviano le informazioni raccolte a uno o più nodi speciali della rete, detti nodi sink, i quali hanno lo scopo di raccogliere i dati e trasmetterli a un server o a un calcolatore, o elaborarli direttamente loro stessi per prendere decisioni. Una comunicazione avviene in genere autonomamente da parte del nodo quando si verifica un dato evento, ma può anche venire indotta dal nodo sink tramite l'invio di un'opportuna richiesta verso i nodi interessati. Un tipico esempio può essere il controllo della temperatura all'interno di un edificio: i vari nodi monitorano a intervalli regolari la temperatura nel punto in cui sono posizionati e la comunicano al sink, il quale decide se aumentare il riscaldamento o meno.

Le reti di sensori possono essere utilizzate in numerose applicazioni, tuttavia è necessario prestare attenzione ad alcuni requisiti fondamentali di questo tipo di reti che necessitano di un algoritmo apposito che si deve differenziare notevolmente dagli algoritmi che permettono lo scambio di informazioni nelle normali reti di calcolatori cablate.

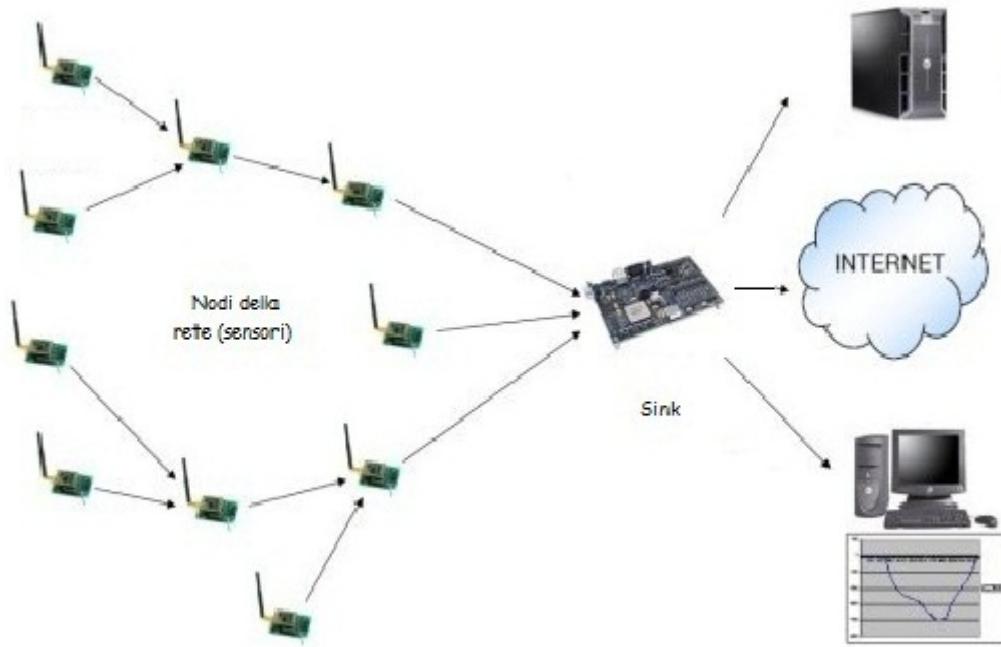


Figura 1.1. Esempio di topologia di una rete di sensori wireless.

I principali requisiti da prendere in considerazione nella progettazione di un algoritmo per una rete di sensori wireless sono che:

- I nodi non conoscono la loro posizione nella rete;
- I nodi sono soggetti a guasti;
- La topologia di una rete di sensori può cambiare frequentemente a causa di guasti ai nodi o della loro mobilità;
- I nodi utilizzano un paradigma di comunicazione broadcast mentre la maggior parte delle reti sono basate su una comunicazione di tipo punto-punto;
- I nodi sono limitati rispetto ad alimentazione, capacità di calcolo e memoria;
- I nodi necessitano di una stretta integrazione con le attività di rilevamento.

I nodi sensore usano, per interagire tra loro, una comunicazione wireless. Alcuni nodi posizionati lontano dal sink potrebbero non essere in grado di comunicare direttamente con questo a causa della limitata potenza di trasmissione a disposizione. Per questo motivo, questa tipologia di rete necessita di algoritmi pensati e realizzati in maniera specifica per gestire la comunicazione e l'instradamento dei dati. Quindi, non dobbiamo modellare una rete di sensori come un database distribuito dove tutti i nodi sono passivi, ma come un insieme distribuito di nodi che collaborano tra loro e dove ciascuno ha capacità attive programmabili. Questo permette a tutti i nodi di coordinarsi l'uno con l'altro per collaborare nella raccolta delle informazioni e per

invierle al sink. In questo modo i nodi sensore diventano attivi e autonomi. I nodi sensore sono sparpagliati in un’area chiamata area di “sensing”. Ogni sensore, all’interno di questa zona, ha la capacità di accumulare e di instradare i dati fino al nodo sink e da questo fino all’utente finale.

La posizione dei nodi all’interno della rete non deve essere predeterminata in quanto questa tecnologia può essere utilizzata in posti difficilmente accessibili o in operazioni di soccorso in luoghi disastrati per i quali è necessaria una disposizione casuale dei nodi. Ciò significa che gli algoritmi e i protocolli utilizzati nelle reti di sensori devono possedere capacità auto organizzative. I sensori, conoscendo le caratteristiche di trasmissione del sink e sfruttando il suo segnale (che possono, o meno, sentire), possono effettuare un autoapprendimento di posizione, permettendo quindi la distribuzione random di questi nella rete. L’autoapprendimento di posizione dei sensori è quindi una delle caratteristiche più importanti di questi e, vista la scarsa quantità di energia di cui è dotato un sensore, si deve cercare di ottimizzare al meglio gli algoritmi che permettono a un sensore di conoscere la propria posizione, abbassando il loro tempo di apprendimento, ovvero cercare di minimizzare il tempo in cui un sensore deve rimanere attivo e il numero di volte in cui esso si deve attivare.

Le reti di sensori possono essere utilizzate su un’ampia schiera di applicazioni che vanno da quella militare a quella scientifica, industriale, medica e domestica.

Lo scopo fondamentale di una rete di sensori è di produrre su un periodo esteso di tempo, una informazione globale significativa ottenuta da una serie di dati locali provenienti dai singoli sensori. È importante notare che la rete deve essere realizzata in modo da garantirne l’integrità per un periodo di tempo che sia il più lungo possibile, allo scopo di ottenere informazioni accurate anche in caso di attacco alla rete da parte di organi esterni o di cedimenti hardware.

Il fatto che un singolo sensore sia dotato di una piccola quantità di energia non deve impedirgli di inviare le informazioni elaborate, che verranno raccolte e unite alle informazioni provenienti dagli altri sensori. Un’importante via da seguire consiste nel rilevare il maggior quantitativo possibile di dati locali, evitando la trasmissione di pacchetti inefficienti attraverso la rete. Esistono diverse possibili tecniche che possono essere usate per connettere la rete con l’esterno, in particolare per trasmettere le informazioni che essa raccoglie. Nelle reti di sensori generalmente ci sono entità speciali chiamate nodi sink, che agiscono come nodi gateway, a lungo raggio d’azione e con energia infinita.

### 1.1.2 L’efficienza energetica

I dati raccolti dai sensori in una rete sono normalmente inoltrati verso una stazione (gateway o collettore) che collega tale rete ad altre, in cui i dati possono essere elaborati e visualizzati. Quando sensori e collettore si trovano relativamente vicini è possibile garantire una comunicazione diretta tra di loro. Tuttavia, la maggior parte delle applicazioni richiede il dispiegamento di un gran numero di sensori per coprire superfici ampie, imponendo un approccio alla comunicazione basato sulla propagazione delle informazioni da un nodo a un altro (multi hop). Pertanto, i singoli nodi non solo generano e disseminano i propri dati, ma servono anche come

ripetitori delle informazioni generate da altri sensori. Il processo di stabilire dei percorsi da una sorgente a una destinazione (tipicamente il nodo collettore), attraverso uno o più ripetitori, è detto routing e costituisce la principale responsabilità dello strato di rete dello stack protocolare adottato nella comunicazione. Quando i nodi sono disposti in modo deterministico (ovvero sono posti in posizioni predeterminate), la comunicazione con il gateway può avvenire lungo percorsi definiti a priori. Tuttavia, quando i nodi sono disposti casualmente all'interno dell'ambiente o sono mobili, le topologie risultanti divengono impredicibili e non uniformi. In questo caso diventa essenziale per i nodi auto organizzarsi, cooperando per determinare la propria posizione, identificare i vicini e scoprire i cammini verso il collettore. Il progetto di un protocollo di routing per una rete di sensori introduce alcune sfide per via delle peculiarità di tale tipo di rete, quali la scarsità delle risorse e l'inaffidabilità del mezzo di comunicazione. Ad esempio, la limitata capacità di elaborazione e di memorizzazione e le ridotte disponibilità di banda ed energia impongono soluzioni di instradamento leggere e, al tempo stesso, adattative e flessibili: occorre infatti poter per far fronte a cambiamenti repentina della topologia dovuti al non funzionamento di uno o più nodi o al loro riposizionamento. Inoltre, a differenza di quanto avviene nelle reti cablate, potrebbe non essere disponibile uno schema di indirizzamento globale che consente di identificare univocamente un nodo e la topologia della sottorete a cui è associato.

Il vincolo più frequentemente associato al progetto di una rete di sensori è l'esigenza di operare con quantità di energia limitata. I nodi tipicamente sono alimentati da batterie che devono essere sostituite o ricaricate attingendo energia da una fonte rinnovabile. In alcuni casi, nessuna delle due opzioni è appropriata, per cui il nodo cesserà di funzionare nel momento in cui la propria sorgente energetica si sarà esaurita e, di conseguenza, non farà più parte della rete. La strategia applicata per contenere il consumo energetico dipende in modo significativo dal fatto che la batteria possa essere ricaricata o meno. Nel caso di batterie non ricaricabili, un nodo deve poter funzionare fino a che non è trascorso il proprio "tempo di missione"; alternativamente deve essere possibile/sostenibile sostituire la batteria. La durata del "tempo di missione" può variare da poche ore (ad esempio, per sensori usati a supporto dalle missioni militari) ad anni (come nel caso del monitoraggio ambientale e strutturale). Al fine di garantire la massima durata in vita di un nodo occorre che tutte le sue componenti (hardware, software, protocolli di rete) siano progettate per aumentarne l'efficienza energetica. Nel caso di un processore CMOS, il consumo energetico è principalmente dovuto alla commutazione e alle perdite, secondo la relazione:

$$E_{CPU} = E_{SWITCH} + E_{LEACKAGE} = C_{total}V_{DD}^2 + V_{DD}I_{leackage}\Delta t$$

dove  $C_{total}$  è la capacità totale,  $V_{DD}$  la tensione di alimentazione,  $I_{leackage}$  è la corrente di perdita e  $\Delta t$  è la durata della computazione.

Alcune tecniche per controllare la perdita di energia includono lo spegnimento progressivo dei componenti non attivi e tecniche software come ad esempio il Dynamic Voltage Scaling (DVS), ovvero l'aumento o la diminuzione della tensione di alimentazione in funzione del carico computazionale.

Lo strato MAC permette ai nodi l’accesso al canale wireless. Alcune implementazioni di tale strato sono basate sulla contesa, ovvero ogni nodo può cercare di accedere al mezzo in un momento qualunque, dando origine potenzialmente a collisioni con altri nodi. In tal caso, lo strato MAC deve introdurre dei meccanismi che permettano alla trasmissione di avere successo: per esempio l’ascolto del canale prima dell’inizio della trasmissione, l’attesa di un tempo variabile di backoff e/o l’utilizzo di messaggi di conferma della ricezione. I limiti di questi approcci includono l’incremento dei consumi e l’inserimento di ritardi variabili. Inoltre, i nodi devono ascoltare il canale continuamente, per evitare di perdere potenziali messaggi. Altre implementazioni di protocolli MAC sono prive di contesa, quindi regolano in modo stringente l’accesso al mezzo e consentono ai sensori di spegnere completamente la radio nei periodi in cui non è attesa alcuna comunicazione.

Oltre ai protocolli di comunicazione, l’obiettivo dell’efficienza energetica si raggiunge con il progetto del sistema operativo (ridotta dimensione della memoria, commutazione efficiente tra attività differenti), del middleware (eventualmente utilizzato per facilitare lo sviluppo del sistema distribuito), dei meccanismi di sicurezza e delle applicazioni stesse. Ad esempio, l’elaborazione all’interno della rete (in-network processing) viene spesso utilizzata per eliminare dati ridondanti, provenienti da sensori vicini, e per aggregare le misure di più sensori. Questo porta a un compromesso tra calcolo (sia in termini di numero di operazioni eseguite sia di celle di memoria occupate) e comunicazione (trasmissione dei dati originali oppure delle loro elaborazioni) che può essere spesso sfruttato per ottenere risparmi energetici.

Poiché la potenza di trasmissione di un dispositivo wireless è proporzionale al quadrato della distanza che si intende coprire (o anche maggiore se si è in presenza di ostacoli), risulta essere energeticamente conveniente coinvolgere eventuali nodi intermedi per permettere la comunicazione fra due nodi. Se si è scelto di adottare una politica di spegnimento ciclico mirata al risparmio energetico, tale approccio può risultare complesso o introdurre ritardi anche considerevoli. È possibile mitigare tale difficoltà utilizzando strategie di “risveglio a richiesta”, basate sull’utilizzo di due radio, una a basso consumo per la gestione delle richieste di riattivazione, l’altra per la comunicazione vera e propria. Le tecniche di spegnimento adattativo cercano, invece, di impedire che tutti i nodi possano entrare in modalità a basso consumo contemporaneamente, garantendo la presenza di un sottoinsieme (variabile nel tempo) di nodi attivi che costituiscono la dorsale di comunicazione.

L’ampia scala e i vincoli energetici di molte reti di sensori impediscono l’utilizzo di algoritmi centralizzati per la gestione della rete, come il mantenimento della topologia e il calcolo dell’instradamento. I sensori, invece, collaborano con i propri vicini per prendere decisioni locali che, sebbene non globalmente ottime, risultano essere più efficienti energeticamente di quelle centralizzate oltre a offrire un maggior grado di resilienza.

### 1.1.3 Creazione di una rete di sensori perenne

In questo ambito si colloca la rete di sensori che è stata progettata. In particolare l’obiettivo è stato quello di realizzare una rete di sensori wireless adatta al trasporto di piccole quantità di dati. Al fine di evitare interventi esterni, da parte di operatori

umani, per cambiare le batterie ai sensori, i nodi della rete sono caratterizzati da un consumo estremamente ridotto di energia (energy aware).

La rete si deve auto configurare al momento dell'accensione generale e quando vengono aggiunti dei sensori senza alcun intervento esterno. In caso di interruzione del funzionamento di un nodo, la rete deve essere in grado di convergere e non interrompere il funzionamento generale: le eventuali rotte passanti per tale nodo subiranno una modifica permettendo di continuare a instradare correttamente i dati. Inoltre non è da escludere lo spostamento di un sensore dalla sua attuale posizione in un'altra zona: l'obiettivo, come nel caso precedente, è una rapida convergenza della rete. È necessario ricordare che la convergenza della rete, in seguito a un cambiamento di topologia, comporta un inutile spreco di energia da parte dei nodi, i quali devono capire se sia ancora possibile raggiungere il sink o se, invece, sia necessario calcolare una nuova rotta. Per questo motivo il transitorio durante il periodo di convergenza deve avere un tempo più breve possibile.

L'auto configurazione della rete è garantita da un algoritmo che riesce a risolvere tutti i casi sopra citati.

Tuttavia la caratteristica principale della rete progettata, ovvero che sia perpetua, implica che ogni nodo debba avere un'energia teoricamente infinita, dato che è da escludere la possibilità di cambiare la batteria in caso si scarichi completamente. Si è reso quindi indispensabile procedere alla scelta di una piattaforma hardware che permettesse un consumo di energia il più basso possibile, compatibile con le risorse fornite da energie rinnovabili, come ad esempio piccoli pannelli fotovoltaici oppure generatori termoelettrici.

# Capitolo 2

## Stato dell'arte

### 2.1 Introduzione

Le reti di sensori, per la loro particolare natura, introducono una pluralità di sfide che occorre gestire per permetterne un utilizzo efficiente ed efficace nei diversi campi di applicazione. Spesso, tali sfide riguardano requisiti non funzionali e, come tali, richiedono soluzioni ottimizzate in tutti gli strati della pila protocollare, tenendo conto della generale scarsità di risorse che caratterizza questo tipo di reti. Le tecniche di instradamento, in particolare, sono state oggetto di un notevole interesse nella comunità di ricerca e numerosi algoritmi sono stati proposti. Spesso, tali algoritmi, si basano su tecniche di aggregazione dei dati o dei nodi e sull'uso di informazioni di localizzazione. Nello stato dell'arte attuale quindi esistono molti algoritmi e protocolli di instradamento dei dati che si possono definire “energy aware”, ma tutti questi algoritmi cercano di consumare poco pur mantenendo un elevato throughput con la conseguenza di ridurre la vita nelle loro implementazioni ad un tempo oltre il quale si rende necessaria la sostituzione o la ricarica delle batterie.

È possibile classificare i protocolli di instradamento in diversi modi. Alcune classificazioni sono basate sull'organizzazione della rete (ad esempio, protocolli gerarchici, privi di gerarchia e basati sulla posizione) e altre sul modo con cui viene definito un percorso (ad esempio, protocolli reattivi, proattivi e ibridi). I protocolli di instradamento energy aware verranno suddivisi nelle seguenti categorie [3]:

- Protocolli data-centrici, nei quali viene utilizzato uno schema di nomi per poter identificare i dati disponibili nella rete. In questo modo, ciascun nodo può richiedere l'invio di dati specifici, riducendo quindi il numero di trasmissioni ridondanti.
- Protocolli gerarchici, nei quali i nodi che costituiscono la rete vengono suddivisi in cluster. Il centro del cluster, solitamente, si occupa di aggregare i dati per ridurre il numero di trasmissioni e, quindi, limitare il consumo di energia. In alcuni di questi, viene scelto come centro del cluster un sensore che sia dotato di maggiore energia, banda e memoria.

- Protocolli basati sulla posizione, i quali prevedono la disseminazione dei dati in aree di interesse. La consegna dei dati avviene utilizzando le informazioni sulla posizione dei nodi.
- Protocolli basati sulla modellazione del flusso di rete o sulla qualità del servizio erogata.

La prima classe di algoritmi tende a ottimizzare l’uso della rete riducendo la quantità di informazione che deve essere trasmessa. Ipotizzando che solo una frazione dei dati raccolti possa essere di interesse, in un dato momento, ad un agente posto all’esterno della rete, si provvede ad etichettare ciascun blocco di informazioni con un identificatore simbolico, eventualmente strutturato in modo forte (con una gerarchia ontologica dal generale al particolare) o debole (attraverso l’ausilio di marcatori o tag che evidenziano una qualche caratteristica dell’informazione raccolta) e a trasmettere a tutti i nodi della rete solo la descrizione del dato richiesto, utilizzando una qualche forma di linguaggio di interrogazione. I nodi che hanno i dati richiesti provvedono ad inoltrarli al richiedente. Ogni nodo mantiene una finestra di dati attivi, corrispondenti alle ultime osservazioni. Ogni nuovo dato raccolto da un nodo va a sostituire uno di quelli precedenti: se un dato raccolto non è di interesse di nessuno, esso viene semplicemente scartato e non genera traffico inutile. Analogamente, se un dato è conosciuto da più di un nodo, è possibile evitare la sua trasmissione ridondante. Algoritmi di questo tipo sono quelli detti “Directed Diffusion” e “SPIN”. Gli algoritmi gerarchici separano i nodi in sotto-regioni per separare le aree di osservazione dell’ambiente in cui la rete è dispiegata. I nodi presenti in ciascuna sotto-regione, o cluster, eleggono un proprio rappresentante che diventa responsabile per la gestione della sotto-rete (con le funzioni di aggregazione dei dati e consegna delle richieste) e della trasmissione dei dati raccolti all’interno del cluster stesso. Esempi di questo tipo di algoritmi sono dati da “LEACH”, “PEGASIS” e “TEEN”. Gli algoritmi basati sulla posizione (come “GAF” e “GEAR”) si basano sull’uso delle informazioni relative alla posizione dei nodi per identificare ed inoltrare i dati verso una specifica destinazione all’interno della rete. In base all’ambiente in cui la rete viene dispiegata, è possibile utilizzare diverse tecniche per determinare la posizione dei vari nodi che la compongono. È possibile servirsi di ricevitori GPS (anche se il costo ed il consumo energetico aggiuntivo e la possibile difficoltà a sintonizzare il segnale in ambienti interni rendono tale scelta poco adeguata), oppure servirsi di algoritmi semplificati basati sull’intensità del segnale ricevuto dai diversi nodi o, ancora, definire tale posizione in fase di setup della rete (anche se tale opzione di solito non è accettabile per quei casi d’uso in cui la rete deve auto-organizzarsi). L’ultima classe di algoritmi (ad esempio, “SAR” e “SPEED”), utilizzano modelli del traffico di rete e si servono di meccanismi basati sulla qualità del servizio per supportare i propri requisiti di instradamento. La ricerca di soluzioni ottime dal punto di vista dell’utilizzo di energia spesso non è un’opzione percorribile, vista la scarsità di risorse a disposizione dei singoli nodi e, in generale, risulta sufficiente ottenere algoritmi in grado di garantire soluzioni sostenibili (cioè sub-ottime) ma di semplice implementazione, tali da garantire la massima sopravvivenza dei nodi e della rete nel suo complesso.

L’algoritmo proposto risulta essere di natura ibrida rispetto alla classificazione precedente, incorporando aspetti di instradamento gerarchico, dipendenza dalla posizione e modello dei flussi di rete.

Di seguito saranno descritti e confrontati alcuni protocolli appartenenti a queste classi.

## 2.2 Protocolli Data Centrici

Nella maggior parte delle reti di sensori, i nodi sono meno importanti delle informazioni che generano. Pertanto nei protocolli data centrici, l’accento viene posto sulla raccolta e la disseminazione di informazioni che contengono specifici attributi, piuttosto che di dati provenienti o destinati ad un determinato sensore. In questa sezione verranno analizzati protocolli data centrici per reti di sensori non gerarchiche, in cui cioè tutti i nodi hanno lo stesso ruolo nel processo di instradamento e collaborano per la creazione dei percorsi.

### 2.2.1 Flooding and gossiping

Il flooding e il gossiping [18] sono due meccanismi classici per la trasmissione di dati in reti di sensori che non richiedono l’utilizzo di algoritmi di instradamento e non necessitano di politiche di manutenzione dei percorsi.

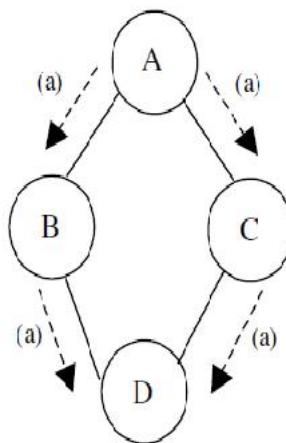


Figura 2.1. Problema dell’implosione.

Nel flooding, ciascun sensore inoltra i pacchetti ricevuti a tutti i propri vicini. Questo processo continua fino a quando i dati non giungono al collettore oppure non viene raggiunto il numero massimo di ritrasmissioni per quel pacchetto. Questo algoritmo è molto facile da implementare, ma ha i seguenti svantaggi [19]:

- È soggetto a fenomeni di implosione, causato dalla ricezione di messaggi ripetuti: come mostrato in Figura 2.1, il nodo *A* invia un pacchetto a tutti

i propri vicini ( $B$  e  $C$ ).  $B$  e  $C$ , a loro volta, lo inoltrano a  $D$ , che riceve due copie dello stesso dato.

- È soggetto a fenomeni di sovrapposizione, come mostrato in Figura 2.2: due sensori ( $A$  e  $B$ ) osservano aree sovrapposte, quindi inviano a  $C$  pacchetti contenenti in parte gli stessi dati, introducendo una ridondanza inutile.
- È soggetto al fenomeno detto resource blindness, cioè non tiene in considerazione eventuali vincoli di consumo energetico, in quanto ciascun nodo inoltra i pacchetti a tutti i propri vicini.

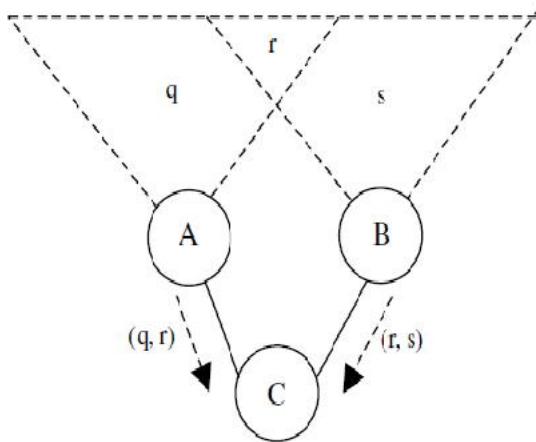


Figura 2.2. Fenomeno di sovrapposizione.

Il gossiping è una versione modificata del flooding, in cui un nodo inoltra i dati ricevuti ad un sottoinsieme dei propri vicini, che a loro volta lo invieranno ad un sottoinsieme dei loro. Con questo meccanismo è possibile evitare il fenomeno dell’implosione, in quanto il numero di trasmissioni è limitato con la scelta dei vicini a cui inviare il dato, aumentando però il ritardo nella propagazione dei dati e non garantendo la consegna finale.

### 2.2.2 SPIN

SPIN (Sensor Protocols for Information via Negotiation) [20] utilizza uno schema di nomi in cui i dati sono associati a descrittori di alto livello o metadati, il cui formato non è definito dal protocollo, ma dal livello applicativo. Quando un nodo ha a disposizione un dato da trasmettere (sia perché ne è la sorgente, sia perché lo ha ricevuto da un proprio vicino), pubblicizza la disponibilità di tale dato inviando ai propri vicini un messaggio ADV contenente i metadati associati ad esso (Figura 2.3, *a* e *d*). I nodi interessati (ad esempio quelli che non lo hanno ancora ricevuto) inviano, quindi, un messaggio di richiesta REQ al mittente (Figura 2.3, *b* ed *e*). Quando la sorgente riceve un messaggio di richiesta, invia la risposta contenente il dato (Figura 2.3, *c* e *f*). Il meccanismo di negoziazione dei metadati permette di evitare gli svantaggi del flooding, cioè l’invio di informazioni ridondanti, la sovrapposizione delle

arie di rilevazione e la resource blindness, portando ad un considerevole risparmio energetico a patto che la dimensione dei metadati sia considerevolmente inferiore a quella dei dati. In questo protocollo, ciascun nodo deve mantenere solamente la lista dei propri vicini (alla distanza di un hop): i cambiamenti di topologia sono quindi localizzati. Questo meccanismo non garantisce la consegna dei dati: ad esempio, se i nodi interessati ad un particolare dato sono separati dalla sorgente da un gruppo di nodi che invece non richiedono il dato, esso non verrà mai consegnato. Questo protocollo, quindi, non è utilizzabile nelle applicazioni che richiedono la consegna affidabile dei pacchetti ad intervalli regolari.

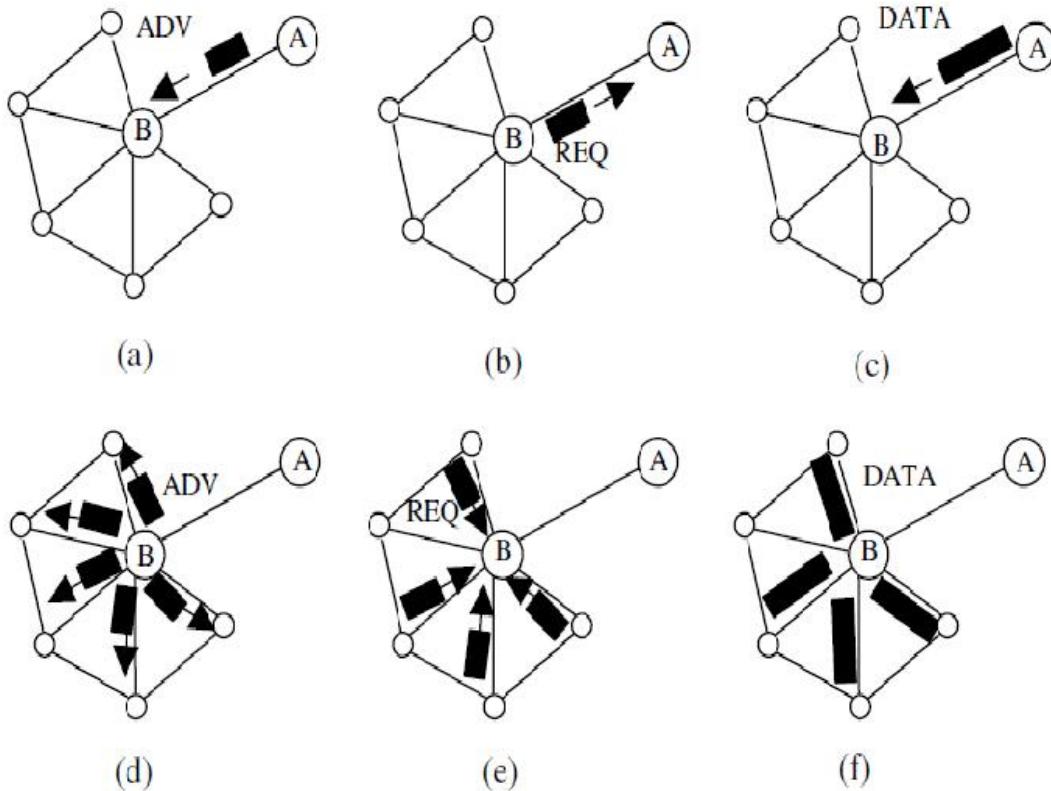


Figura 2.3. Il protocollo SPIN.

### 2.2.3 Directed Diffusion

Il protocollo Directed Diffusion [22] [13] prevede che i dati generati dai sensori siano identificati da coppie attributo-valore. Quando un nodo desidera ottenere dei dati, invia un messaggio di interesse in cui indica le coppie attributo-intervalli di valore che li descrivono. Questa richiesta viene inoltrata a tutti i nodi della rete (attraverso il flooding) oppure ai nodi appartenenti ad una specifica regione della rete (routing geografico). Quando la richiesta arriva ad un nodo che può soddisfarla, esso attiva i propri sensori per poter raccogliere i dati richiesti. Quando tale operazione è terminata, viene creato un messaggio contenente i dati che è inoltrato al nodo richiedente utilizzando il percorso inverso seguito dal messaggio di interesse. I nodi

intermedi possono aggregare i dati, ad esempio combinandoli con quelli ricevuti da altri sensori. La caratteristica principale del protocollo Directed Diffusion è che la propagazione dei messaggi di interesse e dei dati è determinata da interazioni localizzate, che coinvolgono nodi vicini o all’interno di sottoaree della rete. Il protocollo Directed Diffusion prevede le diverse fasi:

- Diffusione di un messaggio di interesse (Figura 2.4 a), che contiene la richiesta di dati ed una lista di coppie attributo-intervallo di valore (es. nome dell’oggetto, intervallo, durata, area geografica).
- Impostazione dei gradienti (Figura 2.4 b): quando un nodo riceve un messaggio di interesse crea al proprio interno un vettore gradiente che indica la direzione dalla quale ha ricevuto tale messaggio. Ciò è possibile in quanto tale algoritmo presuppone la conoscenza della posizione fisica dei nodi.
- Fase di rafforzamento (Figura 2.4 c). Poiché spesso i messaggi di interesse raggiungono la destinazione seguendo percorsi diversi, è necessario scegliere il percorso da utilizzare per la trasmissione dei dati al nodo richiedente. In questa fase, vengono anche creati dei percorsi di backup, che possono essere utilizzati quando il percorso principale tra un nodo sensore ed il collettore risulta non attivo. La costruzione di percorsi di backup può essere eseguita utilizzando diverse tecniche, ad esempio in [15] viene proposta la creazione più percorsi di backup, in modo che in caso di fallimento di uno di essi, possa essere scelto un percorso alternativo senza costi aggiuntivi. Per poter verificare il corretto funzionamento dei percorsi alternativi, è necessario periodicamente trasmettere su di essi dati a bassa velocità. Queste trasmissioni causano un consumo di energia aggiuntiva, ma evitano il consumo di energia dovuto alla ricerca di un nuovo percorso in caso di rottura di un link.

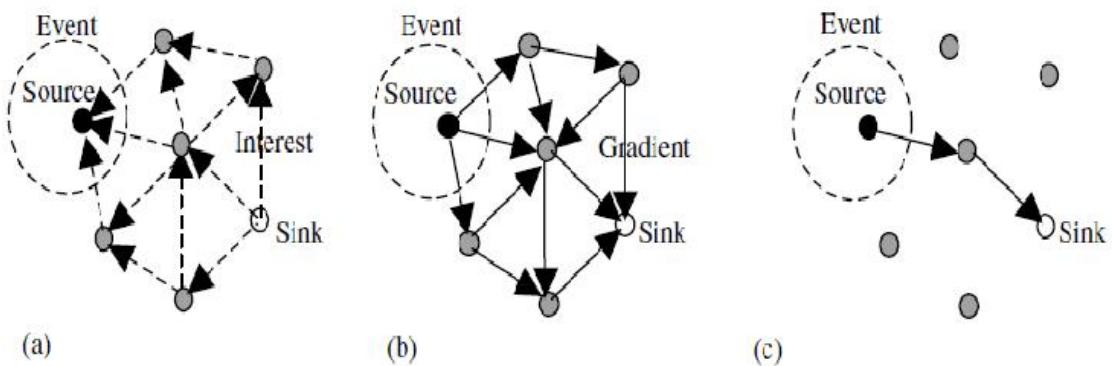


Figura 2.4. Il protocollo Directed Diffusion.

Nel protocollo Direct Diffusion, il collettore interroga i sensori per ottenere i dati desiderati, mentre nel protocollo SPIN sono i sensori stessi ad informare gli altri nodi della disponibilità di un nuovo dato.

Questo protocollo non è utilizzabile nelle applicazioni che richiedono una continua trasmissione dei dati, in quanto necessita di una richiesta esplicita da parte del

collettore. Il protocollo non descrive lo schema dei nomi che deve essere utilizzato che deve essere definito per ciascuna applicazione. Infine, il processo di ricerca di una corrispondenza fra i dati disponibili su un sensore e le richieste ricevute potrebbe sovraccaricare i nodi della rete. Inoltre, l’overhead introdotto dal meccanismo dei gradienti risulta particolarmente oneroso sia in termini di complessità dell’algoritmo che di consumo energetico.

### 2.2.4 Energy-aware routing

L’utilizzo esclusivo di percorsi a consumo energetico minimo può non essere ottimale dal punto di vista della durata in vita della rete e della connettività a lungo termine. Per aumentare la durata della rete, il protocollo proposto in [37] prevede l’utilizzo occasionale di percorsi sub-ottimali, in modo da preservare l’energia dei sensori posti sul cammino ottimo e da garantire un utilizzo più equo delle risorse energetiche presenti su vari sensori della rete. Per ottenere ciò, vengono creati molteplici percorsi fra una sorgente ed una destinazione e a ciascuno di essi viene assegnata una probabilità di utilizzo, che dipende da metriche energetiche. Ogni volta che un dato viene trasmesso da una sorgente alla destinazione, il protocollo sceglie casualmente uno di questi percorsi. Come nel caso del protocollo Directed Diffusion, si assume che la topologia della rete sia nota a priori a tutti i nodi e che lo spazio sia coperto uniformemente dai nodi della rete che sono in grado di raggiungere tutti i propri vicini posti ad una distanza inferiore ad una soglia prefissata (ovvero che la propagazione sia isotropa).

Il protocollo prevede tre fasi:

- Fase di setup o propagazione dell’interesse. In questa fase vengono effettuati flooding localizzati in modo da trovare tutti i percorsi fra sorgente e destinazione e i rispettivi costi energetici. Inoltre, vengono popolate le tabelle di routing.
- Fase di comunicazione dei dati: i dati vengono mandati dalla sorgente alla destinazione, scegliendo in modo probabilistico i percorsi in base ai costi energetici calcolati nella fase precedente.
- Fase di mantenimento dei percorsi: vengono eseguiti periodicamente flooding localizzati per mantenere attivi tutti i percorsi.

La fase di setup, a sua volta, è composta dalle seguenti fasi:

- Il nodo destinazione invia richieste di dati nella direzione della sorgente tramite flooding. Il costo per raggiungere la destinazione viene posto a 0:

$$Cost(N_D) = 0$$

- Ogni nodo intermedio  $N_i$  inoltra la richiesta solamente ai vicini che sono più prossimi alla sorgente rispetto a se stesso e che sono più lontani rispetto al

nodo destinazione. Quindi, le richieste sono inviate solamente ai nodi  $N_j$  che soddisfano i seguenti vincoli sulla distanza  $d$ :

$$\begin{aligned} d(N_i, N_S) &\geq d(N_j, N_S) \\ d(N_i, N_D) &\leq d(N_j, N_D) \end{aligned}$$

- Quando un nodo riceve una richiesta, calcola la metrica energetica per il nodo da cui è stato ricevuto il pacchetto e la somma al costo totale del percorso. Ad esempio, se la richiesta è inviata dal nodo  $N_i$  al nodo  $N_j$ ,  $N_j$  calcola il costo del percorso come segue:

$$C_{N_j, N_i} = Cost(N_i) + Metric(N_j, N_i)$$

La scelta della metrica energetica impatta notevolmente sulle prestazioni del protocollo. La metrica utilizzata in [37] tiene in considerazione sia i costi di trasmissione che quelli di ricezione, oltre che l’energia residua sui due nodi.

- Vengono inseriti nella tabella di forwarding solamente i cammini con i costi minori, che soddisfano il seguente vincolo:

$$FT_j = \{i | C_{N_j, N_i} \leq \alpha \cdot (\min(C_{N_j, N_k}))\}$$

gli altri cammini trovati vengono scartati.

- Il nodo  $N_j$  assegna una probabilità a ciascun vicino  $N_i$ , attraverso il quale può raggiungere la destinazione, inversamente proporzionale al costo del percorso  $N_j N_i$ ; tale probabilità viene salvata nella tabella di routing ed è calcolata come:

$$P_{N_j, N_i} = \frac{1/(C_{N_j, N_i})}{\sum_{i \in FT_j} 1/(C_{N_j, N_i})}$$

- Il nodo  $N_j$  calcola il costo medio per raggiungere la destinazione tramite i vicini che si trovano nella propria tabella di forwarding mediante la formula:

$$Cost(N_j) = \sum_{i \in FT_j} P_{N_j, N_i} C_{N_j, N_i}$$

Questo costo medio viene riportato nel campo di costo del pacchetto di richiesta e inoltrato verso i nodi sorgente.

Nella fase di comunicazione, il nodo sorgente sceglie un proprio vicino, a cui inviare i pacchetti di dati, in base alle probabilità assegnate a ciascuno di essi nella fase precedente. Quando un nodo intermedio riceve un pacchetto di dati da inoltrare al collettore, sceglie uno dei propri vicini come prossimo hop in base alle probabilità memorizzate nella propria tabella di forwarding. Questo processo continua fino a quando il pacchetto non raggiunge il collettore.

Rispetto al protocollo Directed Diffusion, il protocollo energy aware routing fornisce un risparmio energetico dovuto al fatto che la trasmissione dei dati avviene

su di un singolo percorso, scelto in modo casuale tra alternative multiple, mentre nel protocollo Directed Diffusion i dati vengono inviati su più percorsi, a velocità diverse: su uno i dati vengono inviati a velocità più elevata, mentre sui rimanenti viene utilizzata una bassa velocità di trasmissione. Tuttavia, contrariamente a ciò che succede nel protocollo Directed Diffusion, l’utilizzo di un singolo percorso rende più onerosa la fase di calcolo di un nuovo percorso a fronte della caduta di un link. Inoltre, questo approccio richiede la raccolta di informazioni sulla localizzazione dei nodi e l’utilizzo di un meccanismo di indirizzamento fra i nodi.

### 2.2.5 Rumor Routing

Il protocollo Rumor Routing [6] è stato pensato principalmente per i contesti nei quali il routing geografico non è applicabile e cerca un compromesso fra l’utilizzo del flooding per la disseminazione delle richieste e l’utilizzo del flooding per la segnalazione di eventi. Il protocollo si basa sulla creazione di percorsi che possono portare a ciascun evento: in questo modo, quando viene generata una richiesta, essa può essere inoltrata casualmente fino a quando non incontra un percorso che la può portare all’evento. Questo meccanismo evita che la richiesta venga inoltrata tramite flooding in tutta la rete.

Ogni nodo mantiene una lista dei propri vicini e una tabella con l’elenco degli eventi di cui è a conoscenza e le relative informazioni per il forwarding. La lista dei vicini può essere creata e mantenuta attivamente, trasmettendo una richiesta, o passivamente, ascoltando le trasmissioni di altri nodi. Quando un nodo percepisce un evento, lo aggiunge alla propria tabella di eventi e gli associa una distanza pari a zero. Inoltre, il nodo genera in maniera probabilistica un agente.

Un agente è un pacchetto di lunga durata, che viaggia sulla rete, propagando le informazioni su eventi locali a nodi distanti. Esso contiene una tabella di eventi, simile a quella presente nei nodi, che contiene gli eventi che il pacchetto ha rilevato durante il proprio viaggio e le relative distanze. Tutte le volte che l’agente giunge ad un nodo, questa tabella viene aggiornata. In Figura 2.5 è mostrato un esempio di tale processo: quando l’agente viene trasmesso dal nodo *B*, contiene un percorso per *E1* di lunghezza 3, ma non ha indicazioni su come raggiungere l’evento *E2*. Quando l’agente arriva sul nodo *A*, apprende il percorso per raggiungere *E2* e aggiorna la tabella degli eventi del nodo *A*, ottimizzando il percorso per raggiungere *E1*. Se le trasmissioni avvengono in broadcast, i nodi vicini a quelli sul percorso dell’agente possono ricevere le informazioni contenute in esso e modificare di conseguenza le proprie tabelle degli eventi. Dopo la fase di sincronizzazione delle tabelle degli eventi, il TTL dell’agente viene decrementato e, se è maggiore di zero, esso viene inoltrato. Per propagare le informazioni sugli eventi più lontano possibile nella rete, il next hop dell’agente viene calcolato utilizzando il seguente algoritmo. L’agente mantiene una lista di nodi incontrati recentemente, che viene aggiornata tutte le volte che l’agente giunge su di un nodo, inserendo in essa tutti i vicini del nodo su cui si trova. Quando l’agente verrà inoltrato, verrà scelto come next hop un nodo che non si trova in questa lista, in modo da eliminare i loop. Questo meccanismo non garantisce la creazione di percorsi ottimali verso un evento.

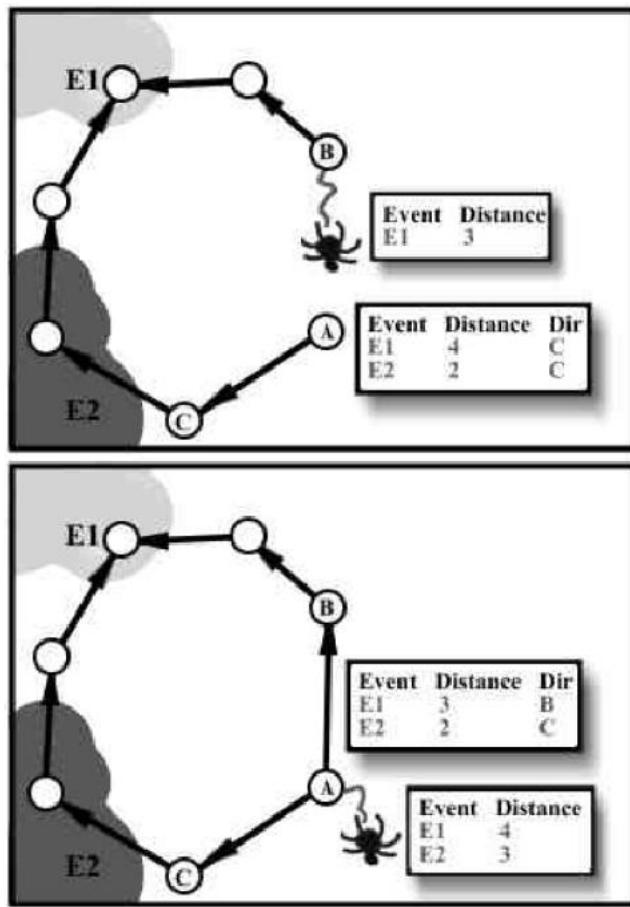


Figura 2.5. Processo di sincronizzazione della tabella degli eventi di un agente.

Ciascun nodo può generare in qualunque momento una richiesta indirizzata a un particolare evento. Se il nodo è a conoscenza di un percorso per l’evento, inoltra la richiesta sul relativo percorso. In caso contrario, la richiesta viene inoltrata ad un proprio vicino scelto casualmente. Questo processo continua fino a quando il TTL della richiesta non scade, o fino a quando non raggiunge un nodo che ha rilevato l’evento richiesto. Il meccanismo di inoltro delle richieste impiega lo stesso meccanismo utilizzato per gli agenti: viene mantenuta la lista dei nodi incontrati di recente, in modo da poter evitare di visitarli nuovamente nel caso in cui si debba scegliere casualmente il prossimo hop della richiesta. Se il nodo che ha originato la richiesta determina che essa non ha raggiunto una meta può provare a ritrasmetterla, rinunciare alla richiesta o richiedere il flooding del messaggio.

Il protocollo Rumor routing mantiene solo un percorso tra sorgente e destinazione, mentre in Directed Diffusion vengono mantenuti percorsi multipli, inviando su di essi i dati a basso bitrate. I risultati delle simulazioni hanno dimostrato che il rumor routing ha un risparmio energetico significativo rispetto al flooding e può anche gestire la caduta di un nodo. Tuttavia, questo protocollo è efficiente solo quando il numero di eventi è piccolo: in caso contrario, il costo di gestione degli agenti e delle tabelle degli eventi può non essere giustificato se non c’è abbastanza interesse per quegli eventi da parte dei collettori.

### 2.2.6 Gradient-based routing

Nel protocollo gradient-based routing [36] i nodi che desiderano ottenere dei dati li possono ottenere inviando un messaggio di interesse tramite flooding. Questo messaggio contiene un campo che indica il numero di hop che ha fino a quel momento percorso il quale viene aggiornato tutte le volte che il messaggio viene inoltrato da un nodo della rete. Tale campo permette ai nodi sorgente di sapere qual è il numero minimo di hop che li separano dal nodo collettore, viene detta height. La differenza fra la height di un nodo e quella del suo vicino è considerata il gradiente su quel link. I pacchetti contenente i dati vengono trasmessi sul link con gradiente più elevato.

Per ridurre il numero di messaggi da trasmettere e l’overhead dovuto alle intestazioni dei messaggi, il protocollo prevede la fusione di messaggi simili, senza però ricorrere alla creazione di cluster e alla selezione di un cluster head. I sensori che rilevano lo stesso evento solitamente si trovano nella stessa area e formano una nuvola di nodi attivati, quindi è possibile che i percorsi che connettono i nodi attivati al nodo collettore possano avere dei tratti comuni. Durante la trasmissione dei dati, i nodi che fanno parte di percorsi multipli creano una Data Combining Entity (DCE) e si occupano di fondere i dati da trasmettere al collettore. Questa tecnica è molto robusta ai fallimenti: se un nodo di una DCE si guasta, i pacchetti possono automaticamente essere indirizzati ad un altro nodo, che creerà a propria volta una nuova DCE.

Il protocollo Gradient-based routing prevede anche l’utilizzo di una delle seguenti tecniche di diffusione dei dati per equilibrare il traffico attraverso la rete e di conseguenza aumentare la durata del ciclo di vita della rete:

- Schema stocastico: quando un nodo deve trasmettere un pacchetto e nella propria tabella di forwarding esistono due o più link con lo stesso gradiente verso il collettore, il nodo ne sceglie casualmente uno di essi. In questo modo, si può bilanciare il carico dei nodi senza però allungare il percorso verso il collettore.
- Schema basato sull’energia: quando l’energia di un nodo scende sotto una determinata soglia, aumenta la propria height in modo da ridurre la probabilità che gli altri sensori lo utilizzino per inoltrare i dati. Questo cambiamento, può provocare l’aggiornamento delle height dei vicini, dando così inizio ad una fase di aggiornamento dei gradienti.
- Schema basato sul flusso: un nodo che sta ricevendo dei pacchetti da parte di un vicino, segnala a tutti gli altri vicini che la propria height è aumentata, scatenando il processo di aggiornamento dei gradienti. Il flusso di dati che era già attivo non risente dell’aggiornamento dei gradienti, mentre i nuovi flussi verranno instradati su altri nodi, evitando così che i nodi che fanno parte già di un flusso vengano utilizzati per inoltrare nuovi flussi di dati.

### 2.2.7 CADR

L’approccio descritto in [8] (Constrained Anisotropic Diffusion Routing) prevede l’introduzione di una misura dell’utilità dell’informazione per la selezione dei sensori da interrogare e per la creazione dinamica dei percorsi per l’instradamento dei dati. In questo modo, è possibile interrogare i sensori e instradare i dati nella rete in modo da massimizzare la diffusione delle informazioni, riducendo al minimo la latenza e l’occupazione di banda.

Il calcolo del percorso migliore per instradare sia le richieste che i dati può essere effettuato con due modalità, a seconda che siano presenti o meno informazioni di posizionamento globale dei nodi sensore.

Se il nodo che fa la richiesta ha una conoscenza globale delle posizioni di tutti i sensori della rete, può interrogare direttamente il sensore più vicino alla posizione in cui si verifica l’evento. La posizione ottimale  $x_0$  può essere calcolata utilizzando la funzione composta obiettivo  $M_c$ :

$$x_0 = \arg_x [\Delta M_c = 0]$$

Se il protocollo di routing supporta l’indirizzamento assoluto dei sensori, la richiesta può essere inviata direttamente al nodo che si trova più vicino alla posizione ottimale.

Se invece non sono disponibili informazioni globali sul posizionamento dei sensori, la richiesta viene inoltrata in base a decisioni individuali dei nodi e diretta alle regioni che soddisfano i vincoli definiti da  $M_c$ . La decisione locale del prossimo hop può essere basata su diversi criteri:

- Ciascun nodo (posizionato in  $x_k$ ) valuta la funzione obiettivo  $M_c$  per i propri vicini (cioè per i nodi che si trovano nel raggio di comunicazione) e sceglie il nodo  $j$  che massimizza localmente la funzione obiettivo:

$$\hat{j} = \arg_j \max(M_c(x_j)), \forall j \neq k$$

- Tra tutti i sensori che si trovano nel proprio raggio di comunicazione, il nodo ne sceglie uno che sia nella direzione del gradiente della funzione obiettivo  $\Delta M_c$ :

$$\hat{j} = \arg_j \max \left( \frac{(\Delta M_c)^T (x_k - x_j)}{|\Delta M_c| |x_k - x_j|} \right)$$

dove  $x_k$  rappresenta la posizione del nodo di routing corrente.

- Invece di utilizzare i gradienti locali della funzione obiettivo durante tutto il percorso di routing, la direzione scelta ad ogni hop può essere modificata in modo da puntare verso la direzione ottimale. La direzione ottimale può essere scelta utilizzando una media pesata del gradiente della funzione obiettivo, la distanza fra il sensore corrente e la posizione ottimale:

$$d = \beta \Delta M_c + (1 - \beta)(x_0 - x_j)$$

Il parametro  $\beta$  può essere scelto, ad esempio, come una funzione della distanza fra il nodo corrente e la posizione ottimale:  $\beta = \beta(|x_0 - x_j|)$ . Questo meccanismo consente l’adattamento della direzione di routing per raggiungere la posizione ottimale. Quando la distanza è grande, è più conveniente seguire il gradiente della funzione obiettivo, in modo da avere un rapido incremento nella diffusione delle informazioni, mentre quando la distanza è bassa, è più conveniente dirigere la richiesta direttamente alla posizione ottimale rispetto a seguire l’aumento del gradiente.

Per determinare quale nodo può fornire le informazioni più utili, bilanciando i costi energetici, viene utilizzato il protocollo IDSQ (Information-Driven Sensor Querying). Il protocollo prevede che ciascun nodo sia a conoscenza della propria posizione  $x_i$  e consta dei seguenti passi (Figura 2.6):

- Fase di inizializzazione (Figura 2.6 1). I nodi vengono suddivisi in cluster e in ciascuno di essi viene determinato il leader. Tale nodo conosce la posizione degli altri nodi appartenenti al cluster.
- Se il nodo non è il leader di un cluster (Figura 2.6 2a), attende la ricezione di richieste provenienti dal leader. Quando viene interrogato, elabora i propri dati e li trasmette al leader.
- Se il nodo è il leader di un cluster (Figura 2.6 2b) ed è stato rilevato un evento all’interno del cluster (ad esempio è stata rilevata una temperatura maggiore di una data soglia), il nodo passa nello stato di attivazione e calcola una rappresentazione del belief state utilizzando le misure, provenienti da altri sensori, che sono già a propria disposizione. In questa fase, il leader tiene traccia dei nodi le cui misure sono state incorporate nel calcolo del belief.
- Viene effettuata una valutazione del belief (Figura 2.6 3). Se viene considerato soddisfacente, utilizzando una metrica opportuna, il nodo termina la fase di computazione. Altrimenti, continuerà con fase di interrogazione dei nodi del cluster (Figura 2.6 4).
- Viene selezionato un sensore, tra quelli le cui misure non hanno già contribuito al calcolo del belief, in base alle caratteristiche del sensore stesso (che si assume essere conosciute dal nodo leader). Una volta selezionato tale nodo, il leader gli invia la richiesta e ne attende la risposta. Quando il leader la riceve, aggiorna il belief e la lista dei sensori già interrogati e ripete le operazioni a partire da (Figura 2.6 3) fino a quando non ottiene un belief soddisfacente.

Per ottenere le misure necessarie al calcolo del belief, vengono scelti solamente i nodi che possono fornire le informazioni più importanti, in modo da evitare trasmissioni non necessarie e ridurre il consumo energetico.

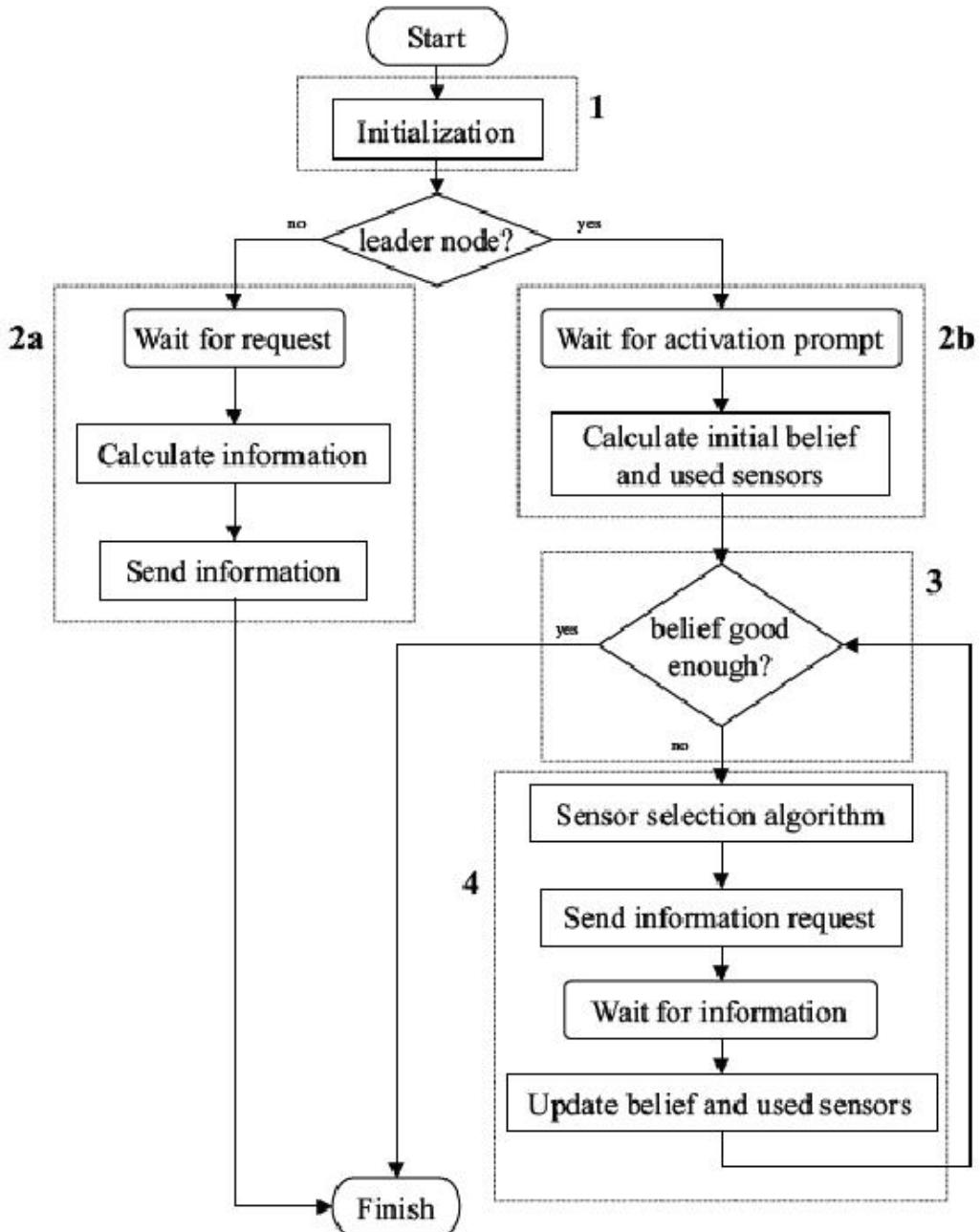


Figura 2.6. Schema di funzionamento del protocollo il protocollo IDSQ.

## 2.2.8 COUGAR

Il protocollo Cougar [43] tratta la rete come se fosse un enorme sistema di database distribuiti, dove ciascun sensore corrisponde ad un database che contiene una frazione dei dati disponibili. In questo sistema, l’utente può interrogare i sensori tramite query dichiarative, che risultano essere particolarmente adatte per l’interazione con una rete di sensori: gli utenti e le applicazioni inoltrano richieste senza sapere come i

dati siano generati e come essi vengano elaborati per fornire una risposta alla richiesta. Per permettere l’utilizzo di richieste dichiarative, il protocollo prevede un livello di gestione delle richieste, detto query layer, che è formato dall’insieme dei “query proxy” presenti su ciascun nodo. Dal punto di vista dell’architettura, il query proxy si trova fra il livello rete e quello applicativo e si occupa dell’elaborazione in-network delle richieste. Nelle reti di sensori con un numero di nodi significativo, la trasmissione di dati ad un nodo centrale che si occupa della loro gestione, il meccanismo di richiesta dei dati e la loro analisi provocano un notevole consumo di energia, dato che l’utilizzo del mezzo wireless è molto costoso in termini energetici. Dal momento che ciascun nodo ha capacità computazionali, una parte dell’elaborazione può essere spostata da un nodo che si trova al di fuori della rete sui nodi che compongono la rete stessa, come l’aggregazione dei dati o l’eliminazione di dati non significativi, al fine di ridurre il consumo energetico e aumentare la durata del ciclo di vita dei sensori.

Lo strato composto dai query proxy presenti su ciascun nodo interagisce sia con il livello applicativo che quello di instradamento. Sul nodo gateway è presente un ottimizzatore delle richieste che si occupa di pianificare l’elaborazione distribuita delle richieste che riceve dall’esterno. Questa pianificazione specifica sia il percorso che devono seguire i dati che le elaborazioni che i dati devono subire su ciascun nodo. Tale pianificazione viene disseminata a tutti i nodi rilevanti.

Ad esempio, si supponga di aver creato una richiesta  $Q$  a lunga durata che permetta di monitorare la temperatura media di una stanza ogni  $t$  secondi. La richiesta  $Q$  genera una notifica (ad esempio creando un record) se la temperatura media ha superato una determinata soglia definita dall’utente. Per valutare tale richiesta, l’ottimizzatore cercherà di fondere questa nuova richiesta con altre precedenti. Se la richiesta  $Q$  è unica nella rete, l’ottimizzatore crea una pianificazione  $QP$  per tale query, che contiene il nodo leader, cioè il nodo su cui avverrà la computazione della temperatura media. Il leader può essere scelto in base all’energia residua oppure in modo casuale utilizzando un algoritmo distribuito per la scelta del leader. Per poter provvedere al calcolo della temperatura, vengono generati due piani: uno per il leader e un altro per i nodi che si trovano nella regione di destinazione della richiesta.

Un esempio di architettura è illustrato in [2.7](#)

La Figura [2.7](#) mostra il piano di richiesta per un nodo che partecipa al calcolo della risposta, senza però essere il leader della query. Tali nodi periodicamente mandano i propri dati di temperatura al nodo leader, aggregandoli parzialmente in modo opportuno con i dati provenienti da altri sensori.

La Figura [2.8](#) mostra, invece, il query plan di un nodo leader. Esso è in grado di aggregare i dati provenienti da altri sensori e valutare se il risultato di tale aggregazione è superiore ad una determinata soglia. In caso affermativo, il leader si occupa di trasmettere il dato risultante dall’aggregazione al gateway. L’introduzione del query layer separa la fase di interrogazione dei sensori dal livello rete, ma introduce un overhead maggiore in termini di consumo di energia e di storage. Inoltre, la capacità di calcolo in-network richiede che i nodi siano sincronizzati: ad esempio, un nodo di raccolta dati deve aspettare che gli arrivino i pacchetti provenienti da tutte

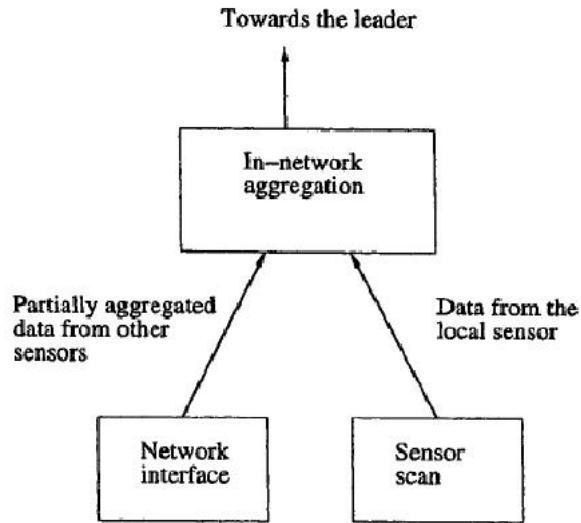


Figura 2.7. Piano di query presso un nodo sorgente.

le sorgenti, prima di inviare i dati al nodo di leader. Per garantire il corretto funzionamento del protocollo, è necessario introdurre meccanismi che rilevino quando un nodo leader non è in grado di svolgere tutti i propri compiti.

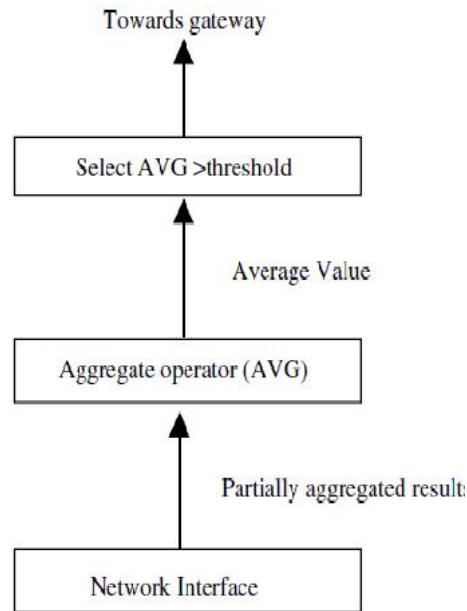


Figura 2.8. Piano di query presso un nodo leader.

## 2.2.9 ACQUIRE

Il protocollo ACQUIRE (ACtive QUery forwarding In sensoR nEtworks) [35] è un protocollo data centrico adatto per interrogazioni one-shot complesse, cioè che

consistono di diverse sotto richieste.

Quando il collettore invia una richiesta attiva (active query), essa viene inoltrata attraverso una serie di nodi. Quando un nodo riceve la richiesta, esso diventa un nodo attivo e utilizza i dati che ha ricevuto dai propri vicini (si considerano vicini i nodi con una distanza pari a  $d$  hop) per risolvere in modo parziale la richiesta. Il nodo attivo richiede la trasmissione di nuovi dati ai propri vicini in modo reattivo solamente nel caso in cui le informazioni in proprio possesso risultano essere obsolete. Dopo aver risolto la richiesta in modo parziale, il nodo invia la richiesta attiva ad un altro nodo, che può essere scelto in modo casuale oppure utilizzando altre tecniche (ad esempio, scegliendo il nodo in modo da massimizzare la probabilità che la query venga risolta). Man mano che la richiesta viene inoltrata nella rete, essa diventa più “piccola”, in quanto parti di essa vengono risolte durante il percorso. Quando arriva su un nodo che è in grado di esaudirla completamente (cioè rispondere alle richieste rimanenti), la richiesta entra nello stato “completed response” e viene inviata al collettore (utilizzando il percorso inverso a quello seguito dalla richiesta oppure calcolando il percorso più breve). Questo processo è raffigurato in Figura 2.9.

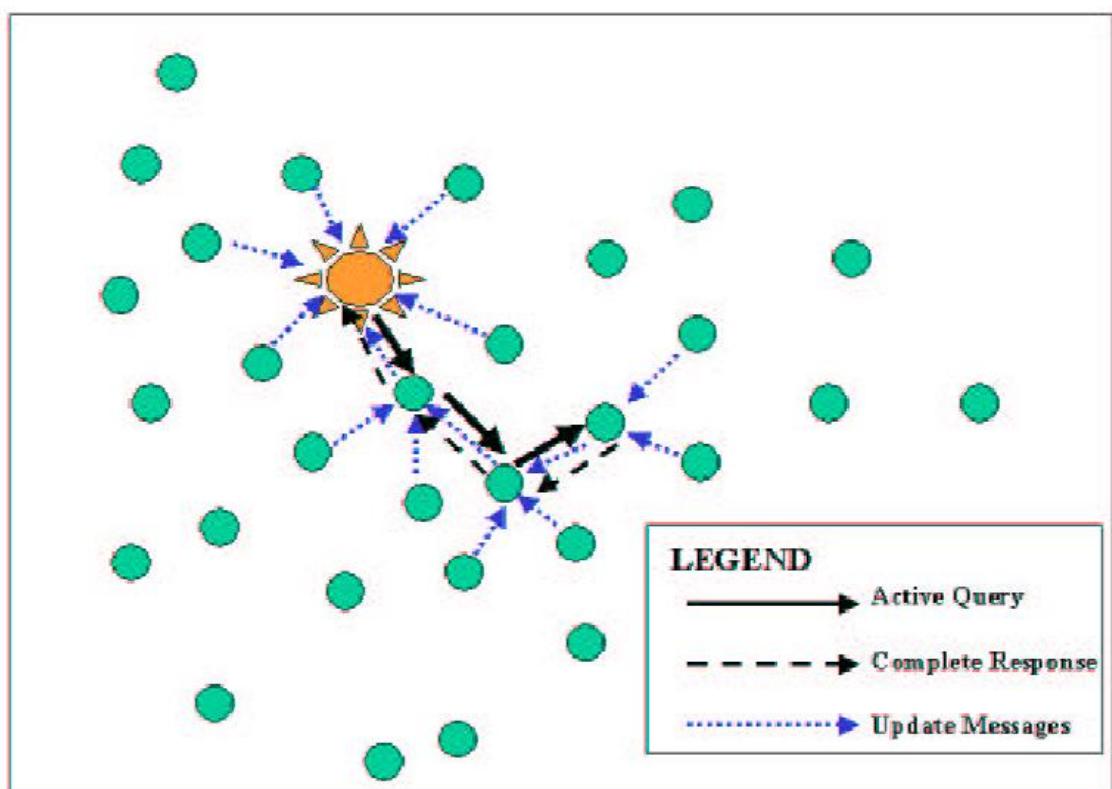


Figura 2.9. Funzionamento dell'algoritmo ACQUIRE (con  $d=1$ ).

## 2.3 Protocolli gerarchici

Come succede per altre reti di comunicazione, uno dei fattori più importanti nella progettazione di una rete di sensori è la scalabilità. In una rete single-tier, l’incremento del numero di sensori può portare al sovraccarico del gateway, causando un incremento della latenza e, quindi, il monitoraggio inadeguato degli eventi. Inoltre, l’architettura single-gateway non è utilizzabile per reti di sensori che coprono una vasta area di interesse, poiché i sensori, solitamente, hanno capacità di comunicazione a corto raggio. Per risolvere tali problematiche, sono stati proposti diversi protocolli di routing gerarchici. In tali protocolli, la rete è suddivisa in cluster: i nodi all’interno di un cluster aggregano e fondono i dati presenti nell’area, in modo da limitare il numero di messaggi trasmessi al collettore e, quindi, limitare il consumo di energia.

### 2.3.1 LEACH

LEACH (Low-Energy Adaptive Clustering Hierarchy) [21] è un protocollo auto-organizzante, adattivo di clustering che mira alla distribuzione uniforme del carico di energia tra i sensori della rete. In questo protocollo, i nodi appartenenti alla rete si auto-organizzano in cluster locali, dotati di un cluster-head, che si occupa di collezionare i dati e trasmetterli al collettore.

Gli algoritmi di clustering convenzionali, prevedono che i cluster-head siano scelti a priori nella fase di inizializzazione e rimangano tali per tutto il ciclo della vita della rete. Utilizzando tale tecnica, però, i cluster head sono soggetti ad un maggiore consumo energetico rispetto agli altri sensori e, quando la loro energia viene completamente consumata, i nodi dell’intero cluster non saranno più in grado di comunicare con il collettore. Per risolvere tale limitazione, LEACH prevede che il cluster head venga periodicamente cambiato in maniera casuale in modo da bilanciare il consumo energetico di tutti i nodi del cluster. Inoltre, il protocollo prevede una fase di data fusion locale per diminuire la dimensione dei dati che devono essere inviati al collettore, in modo da ridurre ulteriormente la dissipazione energetica e aumentare la durata del ciclo di vita della rete.

Il protocollo LEACH è costituito da una serie di round. Ciascun round è composto da una fase di setup, che prevede l’organizzazione dei cluster, e una fase steady-state, che prevede una serie di trasferimenti di dati verso il collettore. Nella fase di inizializzazione, cioè al momento della creazione dei cluster, ciascun nodo decide se diventare o meno il cluster-head per il round corrente, tenendo in considerazione la percentuale di nodi che svolgono il ruolo di cluster head (stabilità a priori) e il numero di volte in cui il nodo ha ricoperto tale ruolo fino a quel momento. Tale decisione viene presa da un nodo  $n$  scegliendo un numero casuale compreso tra 0 e 1. Se il numero è minore di una data soglia  $T(n)$ , il nodo diventa cluster head per il round corrente. La soglia viene definita come:

$$T(n) = \begin{cases} \frac{P}{1-P \cdot (r \bmod 1/P)}, & n \in G \\ 0 & \text{altrimenti} \end{cases}$$

dove  $p$  è la percentuale predefinita di nodi che svolgono il ruolo di centro del cluster (ad esempio  $P = 0,05$  indica che il 5% dei nodi sarà centro di un cluster),  $r$  è il ciclo attuale, e  $G$  è l’insieme di nodi che non sono stati centri di cluster negli ultimi  $1/p$  cicli. Utilizzando tale soglia, ciascun nodo diventerà cluster head una volta in  $1/p$  round.

Ciascun nodo che ha deciso di diventare cluster head per il ciclo corrente invia in broadcast un messaggio agli altri nodi. I nodi che ricevono tali messaggi e non hanno deciso di divenire cluster head decidono a che cluster aggregarsi in base al signal strength del messaggio ricevuto. Assumendo che la propagazione sia simmetrica, maggiore è il signal strength del pacchetto ricevuto, minore sarà l’energia necessaria per trasmettere un pacchetto al cluster head da cui è stato ricevuto il messaggio.

Quando un nodo ha deciso a quale cluster head aggregarsi, gli invia un messaggio di adesione. Durante questa fase, tutti i cluster head devono mantenere acceso il proprio ricevitore. Il cluster head, in base al numero di messaggi di adesione che riceve, pianifica la trasmissione dei vari nodi appartenenti al cluster e trasmette lo schedule a tutti i nodi del cluster. Questo schedule permette ai nodi che non hanno assunto il ruolo di head cluster di spegnere la propria radio per tutto il tempo in cui non devono trasmettere, riducendo così al minimo l’energia dissipata in singoli sensori.

Una volta creato lo schedule di trasmissione, può iniziare la fase di trasmissione dei dati: ciascun nodo può inviare i propri dati al cluster head, utilizzando il time slot che gli è stato riservato. Quando il nodo head ha ricevuto tutti i dati, può aggregare e fondere i messaggi ricevuti e inviare il pacchetto risultante al collettore. Dopo un determinato intervallo di tempo (stabilito a priori), verrà eseguita nuovamente la fase di inizializzazione dei cluster, dando così origine ad un nuovo round.

Il protocollo LEACH è completamente distribuito e non richiede la conoscenza globale della rete. Tuttavia, LEACH prevede che ciascun nodo possa trasmettere direttamente sia al centro del cluster sia al collettore. Pertanto, non è applicabile alle reti di sensori che si espandono su grandi regioni.

### 2.3.2 PEGASIS

Il protocollo PEGASIS (Power-Efficient GAthering in Sensor Information Systems) [27] è pensato per reti in cui tutti i nodi possono trasmettere direttamente a qualunque altro nodo della rete e possono raggiungere direttamente anche il collettore. Tale protocollo prevede che ciascun nodo trasmetta i propri dati al vicino più prossimo e che i nodi a rotazione assumano il ruolo di leader, cioè si incarichino di trasmettere tutti i dati ricevuti al collettore. Tale approccio permette un consumo energetico bilanciato in tutti i nodi della rete.

Invece di formare cluster di nodi come previsto dal protocollo LEACH, il protocollo PEGASIS prevede la formazione di catene di sensori. Tale suddivisione può essere determinata dai nodi stessi, utilizzando un algoritmo greedy, oppure dal collettore, che si occupa di creare le catene e inviare le informazioni relative a tutti i nodi della rete. In ogni round, ciascun nodo riceve dei dati da un solo vicino, li

fonde con i propri dati e li trasmette ad un altro nodo della catena. La fase di raccolta dei dati viene iniziata dal nodo leader e avviene utilizzando un token, in modo da impedire trasmissioni simultanee. Sotto viene rappresentato un esempio di tale meccanismo: il nodo  $c_2$  è il nodo leader del round corrente, quindi trasmette il token al nodo  $c_0$  che è il primo nodo della catena. Quando  $c_0$  riceve il token, trasmette i propri dati a  $c_1$ . Il nodo  $c_1$  aggrega i dati ricevuti con i propri e li trasmette a  $c_2$ . Il nodo  $c_2$  passa il token a  $c_4$  che a sua volta trasmette i propri dati a  $c_3$ . Il nodo  $c_3$  aggrega i dati provenienti da  $c_4$  con i propri e poi invia il messaggio a  $c_2$ . Il nodo  $c_2$  attende di ricevere i dati da entrambi i vicini, si occupa della loro aggregazione con i propri dati e infine invia il messaggio al collettore.

$$\begin{array}{c} c_0 \rightarrow c_1 \rightarrow c_2 \leftarrow c_3 \leftarrow c_4 \\ \downarrow \\ \text{Basestation} \end{array}$$

Rispetto al protocollo LEACH, PEGASIS utilizza il routing multi-hop tramite la formazione di catene e utilizza un solo nodo per round per la trasmissione dei dati al collettore. È stato dimostrato che PEGASIS ha prestazioni migliori rispetto a LEACH di circa il 100-300%, a seconda della dimensione e della topologia della rete utilizzata. Tale aumento delle prestazioni è ottenuto eliminando la procedura di formazione di cluster dinamici e riducendo il numero di trasmissioni e ricezioni tramite l’aggregazione dei dati.

Tuttavia, PEGASIS introduce una latenza consistente per i nodi che si trovano alle estremità della catena e il leader, che è unico in ciascuna catena, può diventare il collo di bottiglia per la trasmissione dei dati.

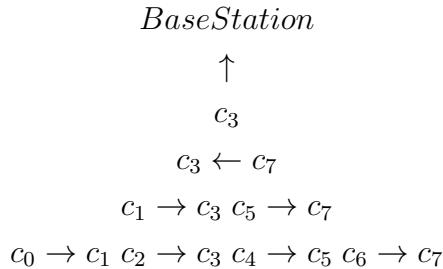
### 2.3.3 PEGASIS Gerarchico

Il protocollo PEGASIS Gerarchico [28] è un’estensione di PEGASIS, che mira a diminuire il ritardo nella trasmissione dei pacchetti verso il collettore e propone una soluzione per il problema della raccolta di dati, considerando metriche basate su *energia\*ritardo*.

Al fine di ridurre il ritardo, è ammessa la trasmissione simultanea di messaggi. Per evitare collisioni e possibili interferenze tra i sensori, è possibile utilizzare due tecniche differenti: la codifica dei segnali, ad esempio CDMA, oppure la trasmissione contemporanea di nodi separati spazialmente.

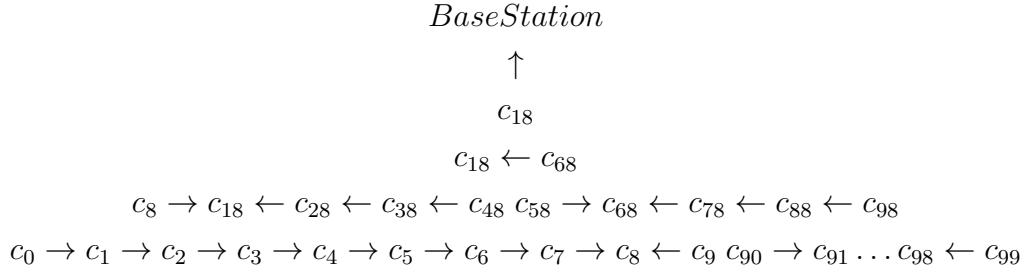
Nel caso in cui i nodi siano in grado di utilizzare un protocollo di tipo CDMA, è possibile minimizzare il ritardo utilizzando il maggior numero possibile di coppie di comunicazione per ciascun livello, costruendo una gerarchia composta da  $\log N$  livelli, dove  $N$  è il numero di nodi. Nel livello più basso, viene costruita una catena lineare tra tutti i nodi, come avviene in PEGASIS, in modo che i nodi adiacenti si trovino vicini anche nella catena. Per costruire tale catena, si assume che tutti i nodi

abbiano una conoscenza globale della rete e che venga utilizzato un algoritmo greedy. Per raccogliere i dati da inviare al collettore, in ciascun livello della gerarchia, ogni nodo trasmette i dati in suo possesso al nodo più vicino. I nodi che ricevono i dati andranno a comporre il livello successivo della gerarchia. Alla fine del processo, rimarrà un solo nodo, che si occuperà di inviare i dati ottenuti al collettore. Ad esempio, nella formula sottostante il nodo  $c_3$  è il leader designato per l’attuale round. Dato che il nodo  $c_3$  è il quarto nodo della catena (posizione dispari), tutti i nodi in una posizione pari invieranno i dati vicino alla loro destra. Nel livello successivo, il nodo  $c_3$  è ancora in una posizione dispari, quindi tutti i nodi in una posizione pari aggregheranno i propri dati con quelli ricevuti e li invieranno al nodo alla loro destra. Al terzo livello, il nodo  $c_3$  è in posizione pari, quindi il nodo  $c_7$  aggrega i dati e li trasmette al  $c_3$ . Infine, il nodo  $c_3$  combinerà i propri dati con quelli ricevuti da  $c_7$  e trasmette il messaggio al collettore.



In una rete in cui i nodi non supportano il protocollo CDMA, non è possibile utilizzare uno schema binario in quanto le interferenze nei livelli più bassi sarebbero troppe. Per migliorare la metrica *energia\*ritardo* è necessario utilizzare un protocollo che consenta le trasmissioni simultanee tra nodi distanti per minimizzare le interferenze senza però impattare troppo sul ritardo ottenuto. In questo scenario, è possibile creare gerarchia di catene a tre livelli per la raccolta dei dati da inviare al collettore. Nella sottostante, è mostrato un esempio di tale schema in una rete composta da 100 nodi, in cui possono avvenire 10 trasmissioni contemporanee e in cui ciascun nodo si occupa dell’aggregazione dei dati. In questo esempio, vengono costruiti, nel livello più basso della gerarchia, 10 gruppi composti da 10 nodi ciascuno. Il ritardo di trasmissione dei dati è pari a 9 unità al primo livello della gerarchia. Per ogni round della vita della rete, verrà scelto un nodo diverso per la trasmissione dei dati aggregati al collettore (BS). Per ciascun livello della gerarchia, viene scelto come nodo che trasmette al livello successivo della gerarchia il nodo il cui indice rappresenta il nodo leader modulo 10: essendo il leader  $c_{18}$ , i nodi che formano il primo livello della gerarchia manderanno i dati ai nodi  $c_8, c_{18}, c_{28}, \dots, c_{29}$ . Tali dieci nodi formano il secondo livello della gerarchia, che viene suddiviso a sua volta in due gruppi. I nodi  $c_{18}$  e  $c_{68}$  si occupano di raccogliere i dati a questo livello, quindi i messaggi subiranno un ritardo di 4 unità. Al terzo livello, il nodo  $c_{68}$  trasmetterà i propri dati a  $c_{18}$ , che si occupa di inviarli al collettore. Il ritardo totale

per la raccolta dei dati può essere, quindi, quantificato in 15 unità.



Anche se i protocolli PEGASIS e PEGASIS gerarchico consentono di evitare l’overhead per la creazione del clustering richiesto dal protocollo LEACH, necessitano comunque di un adeguamento dinamico della topologia nel caso di failure di un nodo, in quanto non viene monitorata l’energia disponibile su ciascun sensore: ogni sensore deve essere a conoscenza dello stato dei suoi vicini in modo da poter decidere dove instradare i dati. I meccanismi di regolazione della topologia possono introdurre un overhead significativo, soprattutto in reti altamente utilizzate. Inoltre, nella nostra rete non è valida l’assunzione che ciascun nodo della rete può comunicare con qualunque altro sensore e con il collettore.

### 2.3.4 TEEN

TEEN (Threshold sensitive Energy Efficient sensor Network protocol) [29] è un protocollo gerarchico progettato per rispondere a improvvisi cambiamenti nei parametri monitorati, come ad esempio la temperatura, adatto per applicazioni time-critical.

L’architettura della rete prevista da questo protocollo è basata su un raggruppamento gerarchico dei sensori: ciascun cluster è dotato di un cluster head che colleziona i dati provenienti dagli altri nodi, li aggrega e li invia al collettore oppure ad un cluster head del livello superiore.

Come mostrato in Figura 2.10, i nodi 1.1 e 1.2 sono cluster head di primo livello e formano un cluster, a loro volta, con il nodo 1. Il nodo 1 è il cluster head di secondo livello per tale cluster. Tale struttura viene ripetuta in modo da formare una gerarchia di cluster, in cui i cluster head dell’ultimo livello disponibile sono in grado di comunicare direttamente con il collettore. Il collettore è la radice della gerarchia e supervisiona l’intera rete. Le caratteristiche principali di tale architettura sono:

- tutti i nodi trasmettono solamente al proprio cluster head, risparmiando quindi energia;
- soltanto i cluster head si occupano di eseguire computazioni ulteriori sui dati, quindi tutti gli altri nodi non sprecano energia,
- i cluster head dei livelli superiori nella gerarchia devono trasmettere i dati a distanze crescenti, dato che oltre ad inviare i dati eseguono anche delle

computazioni sui dati, la loro energia viene consumata più velocemente rispetto agli altri nodi. Per avere una distribuzione equa di tale consumo, tutti i nodi diventano a turno cluster head per un periodo di tempo  $T$ , detto periodo di cluster.

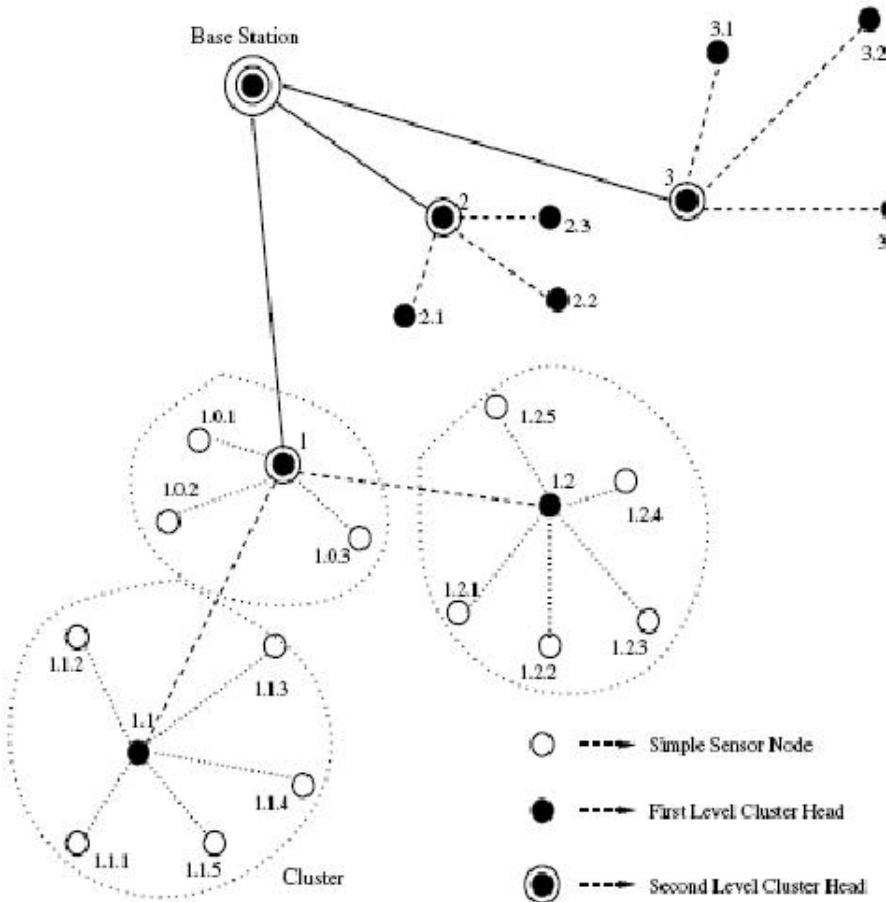


Figura 2.10. Clustering gerarchico in TEEN.

Per ogni periodo di cluster, il cluster head invia ai nodi del proprio cluster, oltre alle informazioni relative agli attributi da monitorare continuamente, due valori che riguardano i parametri monitorati: la soglia “hard” e quella “soft”. Quando la misura corrente effettuata da un nodo supera per la prima volta la soglia hard, il nodo accende il proprio trasmettitore e invia il pacchetto relativo al proprio cluster head. Le misure effettuate vengono memorizzate all’interno del nodo, in una variabile detta sensed value (SV). Successivamente alla prima trasmissione, il nodo invierà i dati nel periodo di cluster corrente se entrambe le seguenti condizioni sono vere:

- il valore corrente dell’attributo monitorato è maggiore della soglia hard;
- il valore corrente dell’attributo monitorato differisce da SV di una quantità maggiore o uguale della soglia soft.

Tutte le volte che il nodo trasmette un dato, il valore di SV viene impostato con la misura corrente del parametro monitorato. La soglia hard, quindi, permette di ridurre il numero di trasmissioni, in quanto viene effettuato un invio solo se il parametro monitorato è nel range di interesse. La soglia soft permette di ridurre ulteriormente il numero di trasmissioni, eliminando le trasmissioni che sarebbero invece state effettuate quando vi è un piccolo o nullo cambiamento negli attributi monitorati una volta che la soglia hard sia stata superata.

Le caratteristiche principali di tale protocollo sono:

- I dati raggiungono l’utente in modo pressoché istantaneo, quindi può essere utilizzato in applicazioni time critical.
- Il consumo energetico è minore rispetto ad approcci proattivi, in quanto la trasmissione dei messaggi (che è più onerosa dal punto di vista energetico rispetto al sensing) avviene meno frequentemente.
- La soglia soft può essere modificata in base agli attributi monitorati.
- Un valore minore della soglia soft causa una visione più accurata della rete, incrementando però il consumo energetico. Modificando tale valore, l’utente può controllare il trade-off fra l’efficienza energetica e l’accuratezza.
- All’inizio di ogni periodo di cluster, vengono inviati gli attributi e le relative soglie, permettendo quindi all’utente di modificare tali informazioni in modo dinamico.

Tuttavia, l’approccio TEEN non è applicabile nelle reti in cui sono necessari rapporti periodici, in quanto, se le soglie non sono raggiunte, non viene trasmesso alcun dato.

### 2.3.5 APTEEN

Il protocollo APTEEN (Adaptive Periodic Threshold-sensitive Energy Efficient Sensor Network protocol) [30] è un’estensione di TEEN che permette sia di raccogliere periodicamente dati sia di reagire a eventi time critical. L’architettura della rete è la stessa proposta da TEEN (si veda la Figura 2.10).

La prima fase del protocollo APTEEN prevede sia la creazione dei cluster sia la scelta dei cluster head. In seguito, in ogni periodo di cluster, ciascun cluster head trasmette le seguenti informazioni:

- l’elenco degli attributi (parametri fisici) da monitorare;
- le soglie “hard” e “soft” (per la descrizione di tali valori si veda la sezione riguardante il protocollo TEEN);

- lo schedule di trasmissione, in cui viene assegnato uno slot a ciascun nodo del cluster. Tale informazione è utile per ridurre il numero di collisioni: dato che i nodi che si trovano nella stessa area vengono aggregati nello stesso cluster, le loro misurazioni saranno simili e, se non venisse utilizzato uno schedule, cercherebbero di inviare i dati contemporaneamente;
- il count time, che indica l’intervallo massimo fra tra due invii successivi di un nodo.

Le caratteristiche principali di tale protocollo sono:

- L’invio periodico di dati permette all’utente di avere una descrizione dettagliata della rete e consente una reazione pressoché immediata delle situazione critiche (cioè quando i valori dei parametri misurati superano la soglia hard). APTEEN combina, quindi, l’approccio reattivo con quello proattivo.
- APTEEN è flessibile, in quanto consente all’utente di specificare sia il count time che i valori di soglia.
- Il consumo di energia può essere controllato dal count time e dai valori di soglia.
- La rete ibrida può emulare una rete proattiva o reattiva, in base ai valori di count time e soglia.

Queste caratteristiche sono introdotte aggiungendo un certo grado di complessità al protocollo.

### 2.3.6 Energy-aware routing for cluster-based sensor networks

Il protocollo descritto in [45] è basato sull’architettura di rete mostrata in Figura 2.11: i nodi sono raggruppati in cluster dotati di un nodo gateway che interagisce con un nodo centrale, il command node, che si occupa di gestire la rete. I gateway sono nodi particolari, in grado di comunicare con tutti gli altri nodi appartenenti al cluster ed caratterizzati da meno vincoli energetici rispetto agli altri sensori. La suddivisione in cluster della rete è effettuata dal command node, utilizzando un algoritmo non definito dal protocollo stesso. Il command node si occupa, inoltre, di comunicare al nodi gateway gli identificativi e la posizione dei sensori allocati nel relativo cluster. Ciascun sensore appartenente ad un cluster riceve i comandi e invia le misure effettuate al proprio gateway. I gateway, a loro volta, inviano al command node dei report, generati fondendo le informazioni dei vari nodi appartenenti al cluster.

In tale architettura, ciascun sensore è in grado di funzionare in modalità attiva o a basso consumo, cioè che i circuiti di rilevamento e di elaborazione possono essere o non essere alimentati. Inoltre, sia il trasmettitore sia il ricevitore possono essere indipendentemente attivati e disattivati e la potenza di trasmissione può essere programmata in base al range di trasmissione richiesto. Inoltre, si assume che ciascun

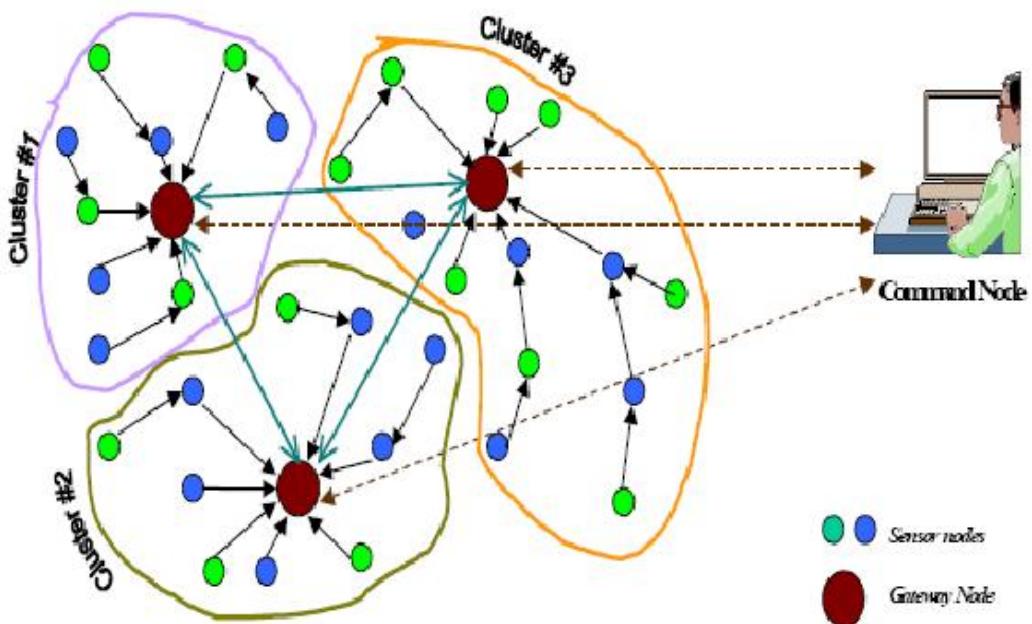


Figura 2.11. Architettura della rete.

nodo possa essere utilizzato per consegnare dati ad altri sensori, permettendo così la creazione di percorsi multihop. Nella fase di creazione dei cluster, i clock dei nodi sensori e dei gateway si sincronizzano, ad esempio tramite il GPS. Successivamente, viene utilizzato un sistema di controllo dell'accesso al canale basato sul tempo, in modo da poter garantire la sincronizzazione dei clock durante le successive fasi del protocollo.

In questo protocollo, il gateway si occupa di gestire l'organizzazione del proprio cluster, in particolare regolando lo scambio di pacchetti e decidendo l'instradamento dei pacchetti. Lo scambio di messaggi fra nodi è regolato utilizzando un meccanismo di tipo TDMA: il gateway comunica ad ogni nodo gli slot in cui si deve mettersi in ascolto sul canale e gli slot che può utilizzare per propria trasmissione. In questo modo, si evitano collisioni ed è possibile lo spegnimento del trasmettitore e del ricevitore negli slot non dedicati al sensore stesso.

L'instradamento di un pacchetto viene deciso in modo centralizzato dal nodo gateway, in quanto è a conoscenza della topologia del proprio cluster e inoltre ha meno vincoli energetici rispetto agli altri nodi. Si suppone che il gateway abbia informazioni circa l'energia residua su ciascun nodo e sulla latenza massima che è accettabile per un dato tipo di messaggio.

I nodi sensore di un cluster possono essere in uno dei seguenti stati:

- Sensing: il nodo monitora l'ambiente e invia dati al gateway ad un tasso costante.
- Relaying: il nodo non monitora l'ambiente, ma il circuito di trasmissione è alimentato in modo da poter trasmettere i dati che provengono da altri nodi attivi.

- Sensing/relayng: esegue entrambe le funzionalità descritte in precedenza.
- Inactive: il nodo ha i circuiti di rilevamento e di comunicazione non alimentati.

La transizione nei diversi stati di un nodo viene stabilita dal nodo gateway in base a considerazioni sull'attuale organizzazione dei sensori, sul livello di energia residua e sulle prestazioni desiderate della rete. L'energia residua di un nodo viene stimata dal gateway ogni volta che riceve un pacchetto da un dato nodo. Le attività della rete possono essere suddivise in una fase di invio dei dati e una fase di instradamento. Nel ciclo di dati, i nodi che stanno monitorando l'ambiente inviano i propri dati al gateway. Nella fase di routing, invece, il gateway determina lo stato di ciascun nodo del cluster e invia eventuali messaggi di transizione di stato e le informazioni di inoltro dei dati. Dato che la stima dell'energia residua effettuata dal gateway può scostarsi dal valore effettivo, a causa dell'inaccuratezza del modello o alla perdita di pacchetti, i nodi inviano periodicamente i propri dati energetici al gateway, utilizzando una frequenza di invio bassa.

In Figura 2.12 è mostrato un esempio di cluster di sensori per un'applicazione target-tracking. Per ciascun nodo è indicato il proprio stato e sono rappresentati i percorsi per la raccolta dei dati da parte del gateway.

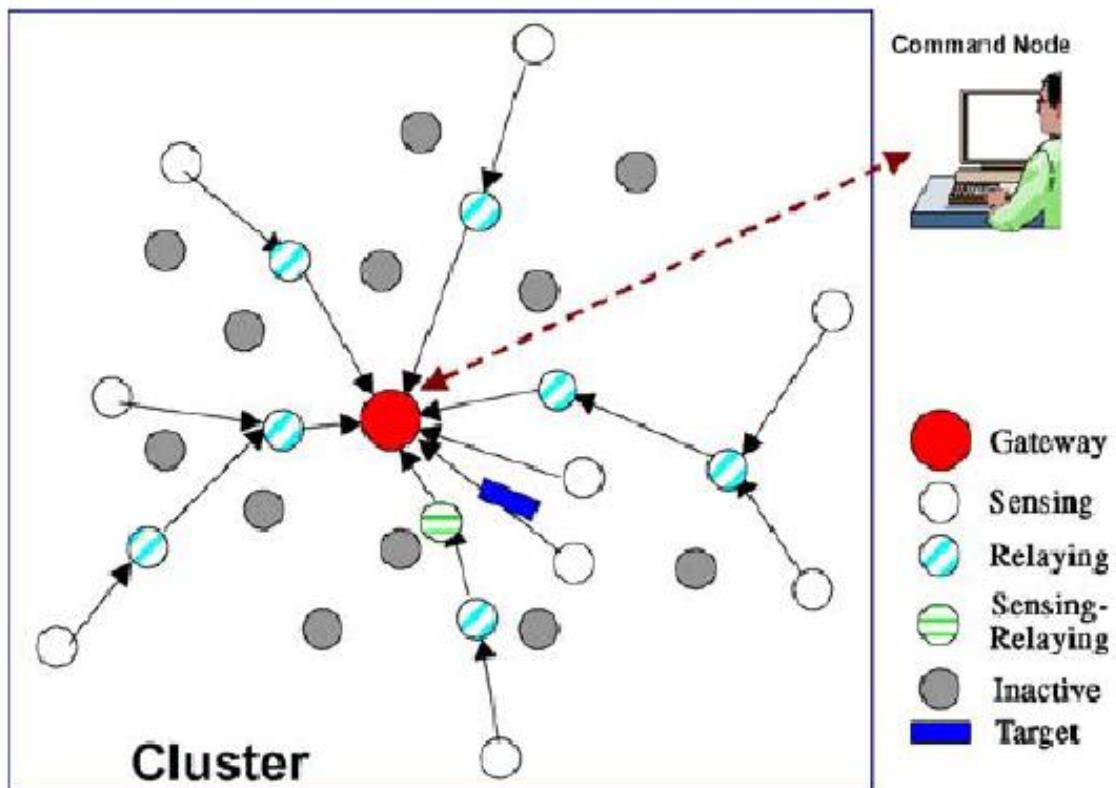


Figura 2.12. Esempio di cluster in una rete di sensori.

Ciascun link viene etichettato con un costo, che viene calcolato tenendo conto del consumo energetico, del ritardo e altre metriche di prestazioni. Utilizzando questi

valori, il gateway calcola il percorso di costo minimo tra ciascun nodo sensore ed il gateway. Quando l’applicazione richiede che l’insieme di sensori che monitorano l’ambiente sia modificato o quando un nodo attivo esaurisce l’energia disponibile, è necessario ripetere la fase di creazione dei percorsi.

Una variante di questo approccio [46] mira a limitare il ritardo di trasmissione, assumendo che ciascun nodo garantisca un range di trasmissione minimo. I risultati delle simulazioni hanno dimostrato che tale approccio ha prestazioni migliori sia di quando vengono utilizzate metriche basate sul risparmio energetico (ad esempio tempo di vita della rete), sia metriche composite (ad esempio, throughput e ritardo end-to-end). I risultati sperimentali hanno anche indicato che combinando l’approccio di routing con l’accesso al canale time-based è possibile aumentare ulteriormente la vita della rete di circa un ordine di grandezza. Tuttavia, tale approccio presuppone l’utilizzo di un modello di propagazione semplice, che potrebbe richiedere l’impiego di molti gateway per garantire un’elevata copertura di sensori.

### 2.3.7 Self-organizing protocol

Il protocollo proposto in [40] è pensato per essere utilizzato in reti eterogenee di sensori, cioè composte sia da nodi mobili che fissi, la cui architettura è rappresentata in Figura 2.13.

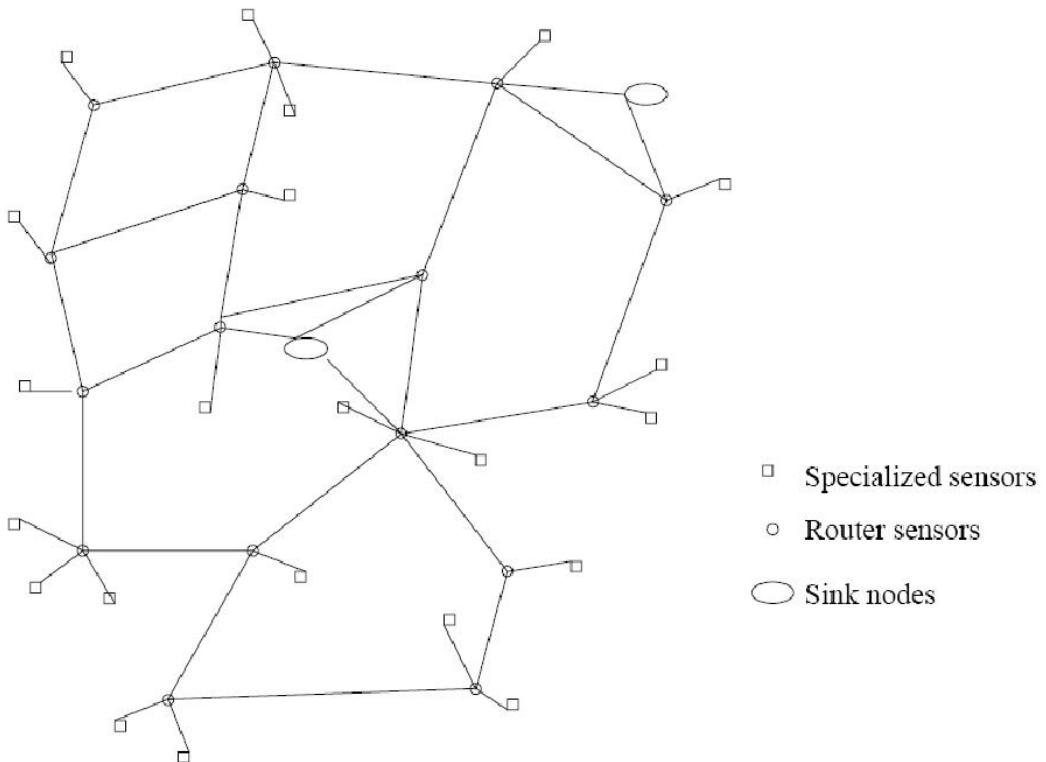


Figura 2.13. Architettura della rete: nodi collettore, router e specializzati.

I sensori presenti nella rete possono essere catalogati nelle seguenti categorie:

- Nodi collettori: hanno capacità di storage molto grandi, capacità computazionale elevata, connettività verso la wide-area (Internet) e non hanno vincoli energetici stringenti. Tali nodi sono in grado di contattare selettivamente dei nodi della rete (ad esempio per richiedere l’attivazione di attuatori specifici) oppure di inviare in broadcast messaggi a tutti i nodi della rete.
- Nodi specializzati: sono nodi che sono in grado di monitorare parametri ambientali, come la temperatura, la pressione, l’umidità, ecc. Ciascun nodo viene identificato a partire dalla propria classe di dispositivo, che indica le funzionalità che può supportare. Tali nodi, in un sistema auto configurante, possono essere mobili. È necessario che ciascun nodo di questa categoria sia raggiungibile da un nodo router, in modo da prendere parte alla rete.
- Nodi router: tali nodi sono stazionari e costituiscono il backbone per la comunicazione. Si suppone la stazionarietà per evitare una continua riorganizzazione del backbone, che potrebbe essere molto dispendiosa. I nodi router si occupano del processo di disseminazione e di instradamento dei dati, ma non si occupano di monitorare l’ambiente.
- Nodi di aggregazione: questi nodi raccolgono i dati provenienti da diversi sensori e li aggregano prima di inoltrarli ad ulteriori nodi. L’aggregazione può avvenire su nodi specializzati oppure sui nodi router.

L’organizzazione della rete viene effettuata dai nodi router, mentre i nodi specializzati memorizzano solamente l’indirizzo del nodo router più vicino ad essi. L’algoritmo di auto-organizzazione della rete prevede le seguenti fasi: discovery, organizzazione, mantenimento e auto-organizzazione.

Nella fase di discovery, ciascun nodo determina i propri vicini, fissando il proprio raggio di trasmissione. Infatti, è opportuno limitare il numero di vicini per evitare un aumento della latenza e la diminuzione del throughput della rete. Inoltre, dato che l’energia consumata è proporzionale al quadrato della distanza fra ricevitore e trasmettitore, viene scelto un raggio di trasmissione adeguato al livello energetico del nodo.

La determinazione del vicinato avviene secondo i seguenti passi:

- Ciascun nodo sceglie un raggio di trasmissione basso e invia un beacon che contiene l’indicazione se si tratta di un nodo speciale o di un nodo router.
- I nodi che ricevono tale messaggio, rispondono inviando le proprie coordinate geografiche (ottenute tramite GPS).
- Se il numero di risposte è inferiore ad una determinata soglia, il nodo invia il beacon utilizzando un raggio di trasmissione maggiore. Questo processo continua fino a quando non viene raggiunto un numero di vicini compresi tra la soglia minima e la soglia massima.

La fase di organizzazione si compone delle seguenti sotto fasi:

- Formazione della gerarchia: i nodi si aggregano formando dei gruppi e viene costruita una gerarchia a partire da tali gruppi.
- Assegnazione degli indirizzi: viene assegnato un indirizzo a ciascun nodo, in base alla propria posizione nella gerarchia.
- Generazione delle tabelle di routing su ogni nodo, tale tabella ha dimensioni dell’ordine  $O(\log n)$ .
- Generazione degli alberi di broadcast: viene costruito un albero di trasmissione che include tutti i nodi presenti nella rete, che viene poi convertito in un grafo aciclico diretto.

La fase di mantenimento viene effettuata su ciascun nodo e consiste nell’aggiornamento delle tabelle di routing e dei costi relativi a ciascun nodo, in modo da mantenerle consistenti. Il mantenimento può essere eseguito in modalità attiva o passiva. Nel monitoraggio attivo, ciascun nodo invia periodicamente ai propri vicini un beacon. Un nodo non riceve tale messaggio da un dato vicino per più di sei intervalli di tempo, il nodo assume che il link non è più disponibile e avvia una procedura di riorganizzazione, per far fronte al cambiamento di topologia. Nel monitoraggio passivo, invece, un nodo controlla lo stato di un determinato vicino, inviandogli un messaggio, a cui dovrebbe seguire un messaggio di risposta. Tale tecnica permette di ridurre il consumo energetico, limitando il numero di trasmissioni. Ciascun nodo invia periodicamente a tutti i vicini il proprio costo (calcolato ad esempio tenendo conto dell’energia residua), che viene utilizzato dai nodi che lo ricevono per aggiornare le tabelle di routing. Al fine di garantire la tolleranza ai guasti, viene monitorato il livello energetico dei nodi per identificare in anticipo la failure di uno di essi: quando si riscontra un livello energetico troppo basso, viene calcolato un nuovo albero di broadcast che non includa tale nodo. La costruzione di alberi e grafi di broadcast avviene utilizzando l’algoritmo local Markov loops (LML).

La fase di auto-riorganizzazione avviene a fronte di un cambiamento della topologia della rete.

A seconda del problema riscontrato possono essere messe in atto diverse strategie:

- Quando un nodo subisce una failure, i propri vicini rilevano tale situazione, ad esempio perché non ricevono più beacon da tale nodo. In questa situazione, i vicini aggiornano i record presenti nelle proprie tabelle di routing in cui il next hop è il nodo che ha subito la failure. Se il nodo in questione non era una foglia dell’albero di trasmissione, viene ricalcolato l’albero in modo che esso diventi una foglia.
- La failure di un link avviene quando un nodo diventa non più raggiungibile da parte di un altro nodo. In questo caso, entrambi i nodi aggiornano le proprie tabelle di routing e viene aggiornato anche l’albero di trasmissione.
- Se avviene il partizionamento di un gruppo, cioè il gruppo risulta essere suddiviso in due o più sotto gruppi disconnessi fra di loro, tutti i sotto gruppi si riorganizzano e tentano di aggregarsi ad altri gruppi nelle vicinanze. In tale situazione, i nodi otterranno nuovi indirizzi.

- Se un nodo rileva che tutti i propri vicini hanno subito una failure, ripeterà la fase di discovery del vicinato.

## 2.4 Protocolli basati sulla posizione

La maggior parte dei protocolli di routing per reti di sensori utilizzano informazioni sulla posizione dei nodi sensori. Nella maggior parte dei casi le informazioni sulla posizione sono necessarie per il calcolo della distanza tra due nodi, in modo da poter stimare il consumo di energia del percorso scelto. Dal momento che non esiste uno schema di indirizzamento per le reti di sensori, come IP, e i nodi sono spazialmente distribuiti in una regione, le informazioni di localizzazione possono essere utilizzate per l’instradamento dei pacchetti in modo da evitare eventualmente le trasmissioni inutili (ad esempio, inoltrando il pacchetto solamente nelle aree di interesse).

### 2.4.1 MECN

Il protocollo MECN (Minimum Energy Communication Network) [34] è un protocollo di rete distribuito ottimizzato per ottenere il minimo consumo energetico in una rete di sensori mobili che supporta la comunicazione peer-to-peer. In tale architettura, ciascun nodo della rete esegue un semplice schema di ottimizzazione locale, garantendo la completa connettività in tutta la rete e minimizzando il consumo energetico. Per il corretto funzionamento del protocollo, si suppone che i sensori che compongono la rete siano dotati di un modulo GPS a bassa potenza, che garantisca un posizionamento con errore inferiore ai 5m.

Nei sistemi peer-to-peer, ogni nodo è sia un fornitore che un collettore di informazioni, quindi, ogni nodo può sia inviare che ricevere messaggi da qualsiasi altro nodo. La rete sottostante deve quindi essere fortemente connessa, cioè deve esistere un percorso che collega ciascuna coppia di nodi. Nelle reti mobili, è necessario aggiornare dinamicamente i percorsi per mantenere la connettività nella rete. Un protocollo che garantisce la connettività forte nella rete è detto auto configurante. Il protocollo MECN è stato progettato per garantire tale caratteristica, consumando però la minor quantità di energia possibile. Il protocollo MECN prevede che esista un unico nodo che si occupi di raccogliere i dati provenienti dagli altri nodi della rete, detto “master site”.

Il protocollo MECN si basa sulla ricerca localizzata di percorsi che possano far parte del percorso a minimo consumo energetico verso il master site, evitando analizzare tutti i nodi della rete: ciascun nodo analizza i link a propria disposizione e tiene in considerazione solamente quelli vantaggiosi. Per ciascun nodo, viene calcolata la regione di relay: trasmettendo attraverso i nodi di questa regione si ottiene una maggiore efficienza energetica rispetto alla trasmissione diretta. La regione di relay per una coppia di nodi  $(i, r)$  è mostrata in Figura 2.14 ed è definita come:

$$R_{i \rightarrow r} \equiv (x, y) | P_{i \rightarrow r \rightarrow (x, y)} < P_{i \rightarrow (x, y)}$$

Dove  $P_{i \rightarrow r \rightarrow (x,y)}$  indica la potenza richiesta dal nodo  $i$  per la trasmissione di dati verso  $(x,y)$  utilizzando il nodo  $r$  come nodo di relay, mentre  $P_{i \rightarrow (x,y)}$  indica la trasmissione diretta fra i nodi  $i$  e  $(x,y)$ .

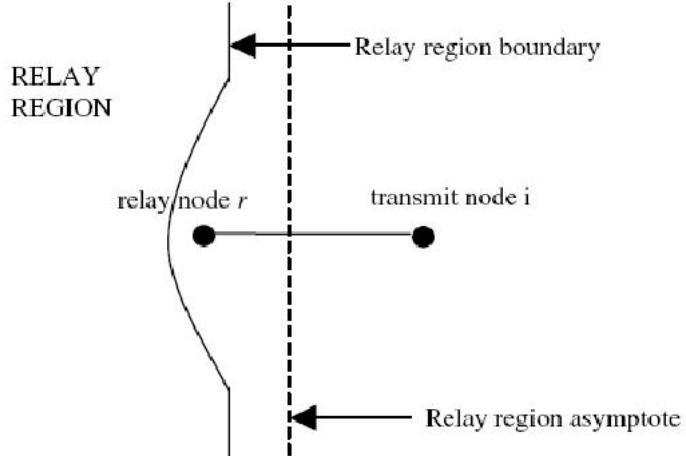


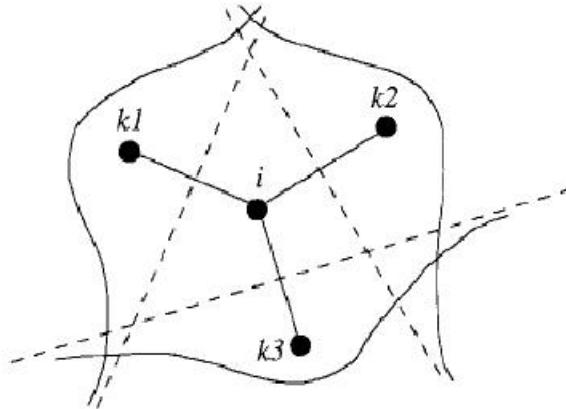
Figura 2.14. Regione di relay per una coppia di nodi  $(i, r)$ .

Per il calcolo del percorso a consumo energetico minimo, il protocollo MECN prevede due fasi: la prima prevede una ricerca localizzata su ciascun nodo per il calcolo del grafo di enclosure, mentre la seconda prevede la distribuzione di informazioni di costo fatta dal nodo master site a tutti i rimanenti nodi della rete.

Nella prima fase, per calcolare il grafo di enclosure, ciascun nodo deve determinare il proprio vicinato e calcolare la propria enclosure, definita come l’unione di tutte le regioni di relay dei nodi che possono essere raggiunti da  $i$  (Figura 2.15). Oltre i confini di tale regione non esistono nodi che possono essere utilizzati da relay per risparmiare energia rispetto alla trasmissione diretta. Il calcolo dell’enclosure richiede la conoscenza della posizione dei nodi vicini, quindi ciascun nodo deve inviare la propria posizione all’interno della propria regione di ricerca. Tale regione viene definita come l’area in cui il segnale trasmesso da un nodo può essere correttamente decodificato da tutti i nodi presenti. In questa fase, ciascun nodo invia un beacon contenente la propria posizione e si mette in ascolto dei messaggi provenienti da altri nodi. Quando riceve correttamente un beacon, ottiene la posizione del nodo sorgente e ne calcola la regione di relay. Se tale nodo è presente in una delle regioni di relay dei nodi già considerati, il nodo viene scartato, altrimenti il nodo che ha ricevuto il beacon aggiorna il proprio grafo di relay.

Nella seconda fase, vengono calcolati i percorsi ottimali sul grafo delle enclosure, tramite l’algoritmo di Belmann-Ford distribuito per il calcolo del cammino minimo e utilizzando come costo le informazioni sul consumo energetico. In questa fase, ciascun nodo invia il proprio costo ai vicini, definito come la minima potenza necessaria per stabilire un percorso verso il nodo master site. Ciascun nodo utilizza queste informazioni per calcolare il costo dei propri link.

Il protocollo MECN è auto-riconfigurante e quindi si può adattare dinamicamente ai guasti dei nodi o al dispiegamento di nuovi sensori: ogni nodo può eseguire la

Figura 2.15. Enclosure del nodo  $i$ .

prima fase dell’algoritmo, aggiornando i collegamenti a costo minimo considerando i nodi che si sono aggiunti o che non sono più disponibili.

#### 2.4.2 SMECN

Il protocollo SMECN (Small Minimum Energy Communication Network) [26] è un’estensione del protocollo MECN che permette, data una rete di comunicazione, di calcolare una sottorete tale per cui per qualunque coppia di nodi  $(u, v)$  connessi nella rete originaria esista un percorso a energia minima che li connetta. La rete calcolata da questo algoritmo è una sottorete di quella che sarebbe calcolata utilizzando il protocollo MECN, ma il carico computazionale è minore di quello di MECN.

In SMECN si assume che la rete sia completamente connessa, ma a differenza di MECN, non è necessario che ciascuna coppia di nodi possa comunicare direttamente (ad esempio a causa di ostacoli fra i due nodi). L’obiettivo di questo protocollo è la costruzione di un sottografo a minima energia  $G$ , a partire dal grafo che rappresenta l’intera rete  $G'$ .

Un cammino  $r = (u_0, \dots, u_k)$  dal nodo  $u_0$  al nodo  $u_k$  è a minima energia se il suo costo  $C(r)$  è minore o uguale al costo  $C(r')$  di tutti i cammini  $r'$  presenti in  $G'$  fra il nodo  $u_0$  e  $u_k$ . Un sottografo  $G = (V, E)$  è a minima energia se per tutte le coppie  $(u, v) \in V$  esiste un percorso  $r$  in  $G$  che sia a minima energia in  $G'$ . Il grafo  $G$  è un grafo fortemente connesso perché contiene percorsi che connettono ciascuna coppia di nodi. Ciascun nodo della rete, quindi, deve costruire il sottografo  $G$  (o almeno una sottoparte rilevante) in modo distribuito. Nel sottografo a minima energia  $G = (V, E)$  di  $G'$ , un arco  $(u, v) \in E$  è ridondante se esiste un percorso da  $u$  a  $v$  in  $G$  che abbia lunghezza maggiore di 1 e per cui il costo  $C(r)$  è minore del costo dell’arco. Viene definito  $G_{min} = (V, E_{min})$  il sottografo di  $G'$  tale per cui non esistono link ridondanti. Tale sottografo  $G_{min}$  è il sottografo di dimensioni minime che soddisfano il vincolo di minima energia. Il protocollo SMECN mira a trovare un sottografo  $G'$  che contenga  $G_{min}$  (in modo da garantire che  $G'$  sia a minima

energia), senza però costruirlo direttamente, in quanto sarebbe molto costoso in termini energetici.

Un arco  $(u, v) \in E'$  è *k-ridondante* se esiste un percorso  $r$  in  $G'$  tale che  $|r| = k$  e  $C(r) \leq C(u, v)$ . Inoltre,  $(u, v) \in E_{min}$  se e solo se non è *k-ridondante* per qualunque valore di  $k > 1$ . Si definisce  $E_2$  come l’insieme di archi di  $E'$  che non sono *2-ridondanti*. In SMECN, viene costruito un grafo  $G = (V, E)$  tale per cui  $E \supseteq E_2$ . Dato che  $E \supseteq E_{min}$ ,  $G$  è un grafo a minima energia.  $E_2$  può essere costruito usando un semplice algoritmo: ogni nodo  $u$  invia in broadcast, utilizzando la massima potenza  $p_{max}$ , un messaggio di *neighbor discovery* (NDM) che contiene la propria posizione. Se un nodo  $v$  riceve tale messaggio, risponde inviando a  $u$  la propria posizione. Definendo  $M(u)$  come l’insieme di nodi che rispondono a  $u$  e  $N_2(u)$  i vicini di  $u$  che appartengono a  $E_2$ , si ha che  $N_2(u) \subseteq M(u)$ . Un nodo  $v$  appartiene a  $N_2(u)$  se non esiste un nodo  $w \in M(u)$  tale per cui  $C(u, w, v) \leq C(u, v)$ .

Dato che il nodo  $u$  conosce la posizione di tutti i nodi di  $M(u)$ , può facilmente calcolare  $N_2(u)$ . Questo algoritmo è semplice, ma richiede la trasmissione alla massima potenza dei messaggi NDM: se la rete è composta da pochi nodi, potrebbe essere necessario utilizzare la massima potenza, in modo da poter raggiungere dei nodi che appartengano a  $E_2$ , ma se la rete è densa questa operazione potrebbe essere utilizzata una potenza inferiore. Il protocollo SMECN fornisce un algoritmo simile a quello presente in MECN, che permette di ridurre la potenza di trasmissione necessaria. Si assume che se un nodo  $u$  trasmette utilizzando la potenza  $p$ , venga a conoscenza dei nodi presenti nella regione  $F(u, p)$ , se non vi sono ostacoli e l’antenna è omnidirezionale, si può supporre che tale regione sia circolare e che abbia un diametro  $d_p$ . L’algoritmo che viene eseguito su ciascun nodo  $u$  è schematizzato in Figura 2.16.

Il nodo  $u$  invia in broadcast un messaggio NDM utilizzando una potenza  $p_0$  e ottiene le risposte da tutti i nodi presenti nella regione  $F(u, p_0)$ . Quindi, verifica se  $F(u, p_0) \supseteq R_{F_{u,p_0}}(u)$ , dove  $R$  è la regione di relay. In caso negativo, il nodo invia nuovamente un messaggio NDM usando una potenza maggiore e ripete questa operazione fino a quando la condizione non risulta verificata. In questo algoritmo, viene calcolata la potenza  $p(u)$  minima che è necessaria per raggiungere tutti i nodi presenti in  $N(u)$ . L’insieme  $A$  è costituito da tutti i nodi che il nodo  $u$  ha trovato durante la ricerca, mentre  $M$  consiste dei nuovi nodi scoperti nell’iterazione corrente. La differenza principale rispetto a MECN è nel calcolo di  $\eta$ : in SMECN, viene calcolato come  $\eta = \bigcap_{v \in A} F(u, p_{max}) - R_{u \rightarrow v}$  al termine di ogni iterazione, mentre in MECN  $\eta = \bigcap_{v \in A - NonNbrs} F(u, p_{max}) - R_{u \rightarrow v}$ . Inoltre, in SMECN, l’insieme *NonNbrs* viene aggiornato solamente inserendo nuovi nodi e non togliendone, come invece può accadere in MECN (se esiste un nodo  $v \in R_{u \rightarrow v}$  e nell’iterazione successiva viene trovato un nodo  $t$  tale per cui  $w \in R_{u \rightarrow t}$ , il nodo  $v$  viene rimosso dall’insieme *NonNbrs*).

La sotto-rete costruita da SMECN per minimizzare il consumo di energia è considerabilmente più piccola (in termini di numero di archi) rispetto a quella costruita in MECN, se le trasmissioni sono in grado di raggiungere tutti i nodi in una regione circolare attorno al trasmittitore, riducendo quindi il numero di hop necessari per la trasmissione di un dato. Le simulazioni hanno dimostrato che SMECN utilizza

Algorithm SMECN

```

 $p = p_0;$ 
 $A = \emptyset;$ 
 $NonNbrs = \emptyset;$ 
 $\eta = F(u, p_{max});$ 
while  $F(u, p) \not\supseteq \eta$  do
     $p = Increase(p);$ 
    Broadcast NDM with power  $p$  and gather responses;
     $M = \{v | Loc(v) \in F(u, p), v \notin A, v \neq u\};$ 
     $A = A \cup M;$ 
    for each  $v \in M$  do
        for each  $w \in A$  do
            if  $Loc(v) \in R_{u \rightarrow w}$  then
                 $NonNbrs = NonNbrs \cup \{v\};$ 
            else if  $Loc(w) \in R_{u \rightarrow v}$  then
                 $NonNbrs = NonNbrs \cup \{w\};$ 
             $\eta = \eta \cap \bigcap_{v \in M} (F(u, p_{max}) - R_{u \rightarrow v});$ 
     $N(u) = A - NonNbrs;$ 
     $p(u) = \min\{p : F(u, p) \supseteq \eta\}$ 

```

Figura 2.16. Algoritmo SMECN.

meno energia rispetto a MECN e che il costo di mantenimento dei percorsi è minore. Tuttavia, trovare una sottorete con un numero minore di archi introduce un overhead maggiore nell’algoritmo.

### 2.4.3 GAF

GAF (Geographic Adaptive Fidelity) [42] è un algoritmo progettato principalmente per reti ad hoc mobili, ma che può essere applicato anche a reti di sensori consentendo di risparmiare energia spegnendo i nodi non necessari, senza intaccare la costruzione di percorsi di routing.

Il protocollo prevede che ciascun nodo utilizzi la propria posizione (indicata dal GPS) per associarsi con un punto in una griglia virtuale che rappresenta l’area di copertura: i nodi che si trovano nella stessa cella della griglia sono considerate equivalenti in termini di costo di instradamento e quindi è possibile far sì che alcuni di essi entrino in stato di sleep, al fine di conservare energia. Utilizzando questa tecnica, aumentando la densità dei nodi della rete, è possibile allungare notevolmente la durata della vita della rete.

La determinazione delle equivalenze tra nodi è un’operazione non banale: nodi che possono essere equivalenti per la comunicazione fra determinate coppie di nodi, potrebbero non esserlo per altre coppie. Ad esempio, in Figura 2.17 è rappresentata una rete in cui i nodi hanno un raggio di comunicazione poco superiore di due unità: per la comunicazione tra i nodi 1 e 4, i nodi 2 e 3 sono equivalenti, mentre per quella fra 1 e 5 è possibile utilizzare solo il nodo 3 (Figura 2.18). Per risolvere questo problema, GAF introduce il concetto di griglia virtuale, definita in modo che se due

celle A e B sono adiacenti, tutti i nodi di A possono comunicare con tutti i nodi di B e viceversa: in questo modo, tutti i nodi di ciascuna cella sono equivalenti per l’instradamento.

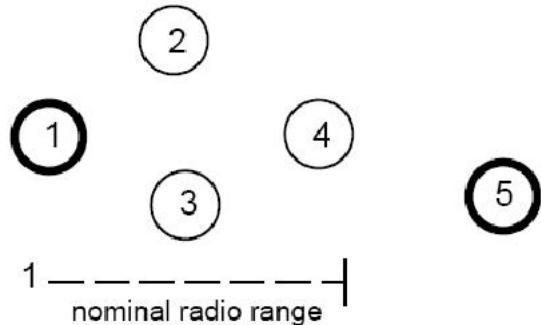


Figura 2.17. Esempio di nodi ridondanti in reti ad hoc.

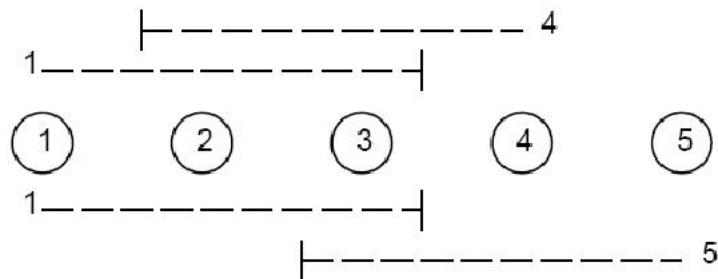


Figura 2.18. Problema dell’equivalenza fra nodi.

La Figura 2.19 mostra la griglia virtuale che può essere derivata dalla rete raffigurata in Figura 2.17: il nodo 1 può raggiungere i nodi 2, 3 e 4 e i nodi 2, 3 e 4 possono raggiungere il nodo 5, quindi i nodi 2, 3 e 4 vengono considerati equivalenti e due di essi possono essere messi in sleeping mode.

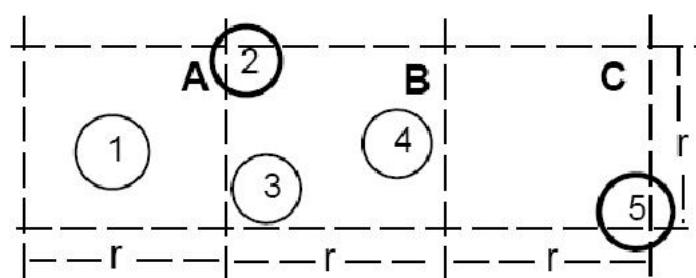


Figura 2.19. Esempio di griglia virtuale in GAF.

Il protocollo GAF definisce tre stati per i nodi: discovery, active e sleeping. Il diagramma a stati e le relative transizioni sono rappresentate in Figura 2.20. Lo

stato iniziale di un nodo è discovery: il circuito di trasmissione è acceso e vengono inviati dei messaggi volti a scoprire quali sono i nodi che appartengono alla stessa cella della griglia virtuale. Il messaggio di discovery è composto dall'identificativo del nodo, dall'identificativo della cella, il tempo di attivazione stimato ( $enat$ ) e lo stato del nodo. Quando un nodo entra in questo stato, inizializza un timer ( $T_d$ ) e al suo scadere, invia il messaggio di discovery entrando nello stato active. Questo timer può essere disattivato anche dalla ricezione di altri messaggi di discovery ed è utile per ridurre la probabilità di collisione di tali pacchetti. Quando un nodo è nello stato active, il nodo partecipa attivamente all'instradamento dei pacchetti. Quando il nodo entra in tale stato, definisce un tempo in cui rimarrà attivo, trascorso il quale tornerà nello stato di discovery, e invia periodicamente messaggi di discovery. Quando il nodo che è nello stato discovery o active rileva che esiste un altro nodo a lui equivalente che può gestire l'instradamento, entra nello stato di sleeping, in cui il modulo radio del nodo è spento, e vi rimane per un tempo definito dall'applicazione. Allo scadere di tale intervallo, il nodo entra nuovamente nello stato di discovery.

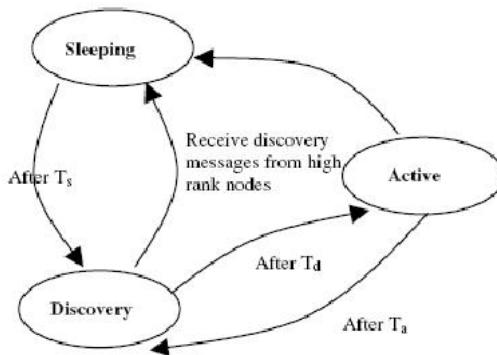


Figura 2.20. Transizione fra gli stati dei nodi previsti in GAF.

GAF utilizza la seguente tecnica per il bilanciamento del carico sui diversi nodi della rete. Un nodo rimane nello stato di active per un tempo pari a  $T_d$  e, successivamente, entra nello stato di discovery in modo da dare la possibilità ad altri nodi della propria cella di entrare nello stato active: un nodo che è appena entrato nella fase di discovery presumibilmente avrà meno energia residua dei propri vicini, che erano entrati nello stato di sleeping durante la fase di active del nodo stesso.

Per supportare la mobilità, ogni nodo della griglia stima il tempo di permanenza nella griglia e invia questo dato ai propri vicini. I vicini in modalità sleep regolano di conseguenza il loro tempo di sleep, al fine di garantire il mantenimento dei percorsi di instradamento. Prima che il tempo di permanenza di un nodo attivo sia trascorso, i nodi in stato sleeping si attivano ed uno di essi diventa active.

I risultati delle simulazioni mostrano che GAF ha prestazioni paragonabili ad un normale protocollo di routing per reti ad hoc in termini di latenza e perdite di pacchetti e aumenta la durata della rete grazie al risparmio energetico. Anche se GAF è un protocollo location-based, può anche essere considerato come un protocollo gerarchico, in cui i cluster sono formati in base alla posizione geografica. Per ogni area della griglia, infatti, un nodo si comporta come leader e trasmette dati

agli altri nodi. Il nodo leader, tuttavia, non si occupa di aggregare o fondere i dati come nel caso di altri protocolli gerarchici discussi in precedenza.

#### 2.4.4 GEAR

Il protocollo GEAR (Geographic and Energy-Aware Routing) [47] utilizza le informazioni di localizzazione per disseminare richieste in specifiche aree geografiche, per soddisfare richieste del tipo “*qual è la temperatura nella regione R nel periodo di tempo (t1, t2)*”. Per disseminare in modo efficiente una richiesta in una specifica regione, è necessario inserire informazioni di localizzazione nella richiesta e indirizzarla direttamente alla regione di interesse invece di disseminarla in tutta la rete. Il protocollo GEAR mira a inoltrare i pacchetti a tutti i nodi all’interno di un’area di interesse, senza necessitare di un database che contenga le informazioni di localizzazione di ciascun nodo. Il protocollo GEAR assume che ciascun pacchetto di richiesta contenga l’indicazione della regione di interesse, espressa, ad esempio, da rettangoli nello spazio bidimensionale e che ciascun nodo conosca la propria posizione, il proprio livello di energia residua, la posizione dei propri vicini e il loro livello energetico. Inoltre si assume che i link siano bidirezionali.

Il processo di inoltro di un pacchetto a tutti i nodi della regione di interesse consiste di due fasi:

1. Inoltro dei pacchetti verso la regione di destinazione: vengono utilizzate euristiche per la selezione del vicino al quale inoltrare il pacchetto basate su politiche di risparmio energetico e sulle informazioni di localizzazione.

Sono possibili due situazioni:

- a. Se esiste uno o più vicini che si trovano più prossimi alla regione di destinazione rispetto al nodo stesso, ne viene scelto uno come next hop.
- b. Se non esiste un nodo che soddisfa tale situazione, si verifica un buco. In questo caso viene scelto come next hop un nodo che minimizza una specifica funzione di costo.

2. Inoltro dei pacchetti all’interno della regione: quando il pacchetto ha raggiunto la regione di interesse, può essere diffuso in quella regione geografica utilizzando un forwarding ricorsivo geografico oppure con il restricted flooding, quando il precedente non converge a causa della bassa densità dei nodi. Un esempio è visualizzato in Figura 2.21.

In GEAR, ogni nodo che partecipa al processo di inoltro di un pacchetto  $P$  verso la regione di interesse  $R$  cerca di mantenere un bilanciamento del consumo energetico dei propri vicini, minimizzando il costo appreso  $h(N_i, R)$  dei propri vicini  $N_i$ . Ciascun nodo  $N$  mantiene il proprio costo appreso  $h(N, R)$  verso la regione  $R$  e invia questa informazione periodicamente ai propri vicini. Se un nodo non è a

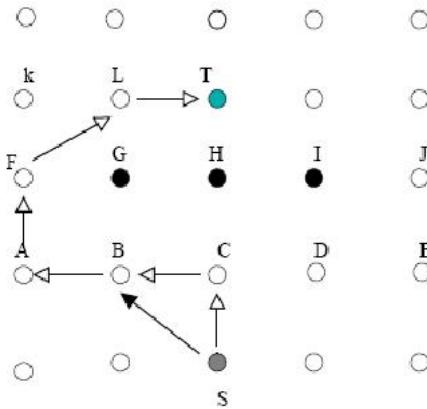


Figura 2.21. Instradamento attorno ad un buco.

conoscenza del costo  $h(N_i, R)$  per un vicino  $N_i$ , calcola il costo stimato  $c(N_i, R)$  che è definito come:

$$c(N_i, R) = \alpha d(N_i, R) + (1 - \alpha)e(N_i)$$

dove  $\alpha$  è un parametro che può essere modificato in base all’applicazione,  $d(N_i, R)$  è la distanza del nodo  $N_i$  dal centroide della regione  $R$  normalizzata rispetto al valore massimo di tale distanza dell’insieme dei vicini del nodo  $N$  e  $e(N_i)$  è l’energia consumata nel nodo  $N_i$  normalizzata rispetto al massimo consumo energetico nell’insieme dei vicini del nodo  $N$ . Quando un nodo sceglie come next hop il nodo  $N_{min}$ , aggiorna il proprio costo  $h(N, R)$  al valore  $h(N_{min}, R) + C(N, N_{min})$ , dove  $C(N, N_{min})$  è il costo di trasmissione di un pacchetto da  $N$  a  $N_{min}$  e può essere anche una combinazione del livello energetico residuo sui due nodi e la loro distanza. Quando non vi sono buchi nel percorso verso la regione di interesse  $R$ , il costo appreso è equivalente al costo stimato.

Quando un nodo  $N$  riceve un pacchetto che deve essere inoltrato, può trovarsi in una delle seguenti condizioni:

- almeno uno dei propri vicini è più vicino al centroide della regione di interesse rispetto a  $N$ . In questo caso, ne sceglie uno fra questi che minimizzi il costo appreso  $h(N_i, R)$ . Se non vi sono “buchi”, minimizzare il costo appreso porta ad ottenere un trade off tra la creazione del percorso più breve e il bilanciamento del consumo energetico;
- tutti i vicini sono più lontani dal centroide della regione di interesse rispetto a  $N$ : il nodo  $N$  si trova in un “buco”. Tale situazione è rappresentata in Figura 2.21, in cui la distanza minima fra due vicini è pari a 1 e ciascun nodo può raggiungere 8 sensori. I nodi neri ( $G, H, I$ ) non possono prendere parte al processo di inoltro dei pacchetti, ad esempio perché non hanno più energia residua. Quando il nodo  $S$  deve inviare un pacchetto verso la regione di interesse il cui centroide è rappresentato dal nodo  $T$ , deve scegliere uno dei propri vicini come next hop. All’inizio del processo, i costi stimati e i costi

appresi si equivalgono e i nodi  $B$ ,  $C$ ,  $D$  sono più vicini a  $T$  rispetto a  $S$ :

$$h(B, T) = c(B, T) = \sqrt{5}, h(C, T) = c(C, T) = 2, h(D, T) = c(D, T) = \sqrt{5}$$

Quindi  $S$  sceglie come next hop il nodo  $C$ .  $C$ , a sua volta, si trova in un buco, dato che tutti i propri vicini sono più lontani a  $T$  rispetto a  $C$  stesso, quindi sceglie come next hop uno di quelli che minimizzano il costo appreso, ad esempio il nodo  $B$ . A questo punto,  $C$  aggiorna il proprio costo  $h(C, T) = h(B, T) + C(C, B) = \sqrt{5} + 1$ . Quando il messaggio raggiungerà la destinazione, il costo appreso corretto è propagato un hop indietro in modo che l’instradamento per il pacchetto successivo possa essere aggiornato di conseguenza. Al termine di tale processo, quindi, quando  $S$  dovrà trasmettere un altro pacchetto a  $T$ , invierà il messaggio direttamente al nodo  $B$  invece che a  $C$  per aggirare il buco.

Quando il pacchetto raggiunge l’area di interesse, l’inoltro può essere effettuato utilizzando un algoritmo di flooding con rimozione dei duplicati, anche se è molto oneroso dal punto di vista energetico in quanto tutti i nodi devono inoltrare il pacchetto una volta, soprattutto nelle reti ad alta densità. In tali situazioni, viene utilizzato flooding ricorsivo geografico. In Figura 2.22, la regione  $R$  di interesse è rappresentata dal rettangolo esterno. Quando il nodo  $N_i$ , interno alla regione  $R$ , riceve un pacchetto  $P$  che deve essere inoltrato in  $R$ , suddivide la regione  $R$  in quattro sotto regioni (rappresentate dai rettangoli più piccoli), successivamente esso crea quattro copie del pacchetto e le inoltra alle quattro sotto regioni. Questo processo di suddivisione continua fino a quando le sottoregioni non risultano essere composte solo dal nodo corrente.

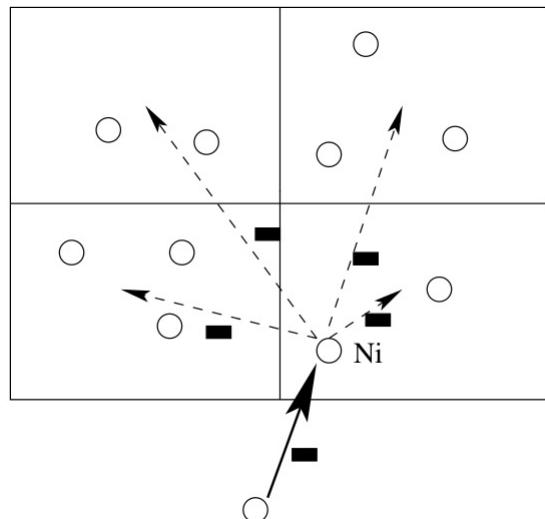


Figura 2.22. Esempio di forwarding geografico in GEAR.

## 2.5 Network flow and QoS-aware protocols

Anche se molti dei protocolli di routing proposti rientrano nelle classificazioni precedenti, ci sono altri protocolli che usano approcci differenti: in alcuni il setup della rete è modellato e risolto come se fosse un problema di flusso della rete stessa. Nei QoS invece si considerano i requisiti end-to-end quando si costruiscono i percorsi della rete di sensori.

### 2.5.1 Maximum lifetime energy routing

Il protocollo Maximum lifetime energy routing [7] definisce un’euristica che mira a massimizzare la durata della vita della rete, costruendo due percorsi a lunghezza minima per permettere l’inoltro di ciascun messaggio. Per ridurre il consumo energetico, viene utilizzato un algoritmo a due fasi per il calcolo del percorso di ciascun messaggio che deve essere trasmesso. Nel primo passo, vengono eliminati dal grafo  $G$  che rappresenta la rete di sensori tutti gli archi  $(u, v)$  tali per cui  $ce(u) < w(u, v)$ , dove  $ce$  indica l’energia corrente del nodo  $u$  e  $w(u, v)$  indica il costo di trasmissione diretta dal nodo  $u$  al nodo  $v$ , in quanto questi archi richiedono un’energia di trasmissione superiore a quella disponibile sul nodo. È possibile, quindi, calcolare il percorso  $P'_i$  a energia minima tra la sorgente e la destinazione del messaggio nel grafo risultante  $G' = (V, E')$ , utilizzando l’algoritmo di Dijkstra. Per ogni arco di  $P'_i$ , viene calcolata l’energia residua  $re(u) = ce(u) - w(u, v)$ . Sia  $\min(RE) = \min(re(u))|u \in P'_i \ \& \ u \neq t_i$ , dove  $t_i$  indica la destinazione del messaggio, e sia  $G'' = (V, E'')$  il grafo ottenuto rimuovendo da  $G'$  tutti gli archi  $(u, v) \in E'$  in cui  $ce(u) - w(u, v) < \min(RE)$ , in modo da evitare di tenere in considerazione nodi con scarsa energia residua. Nel secondo passo dell’algoritmo, viene calcolato il percorso da utilizzare per l’inoltro del messaggio, assegnando a ciascun arco in  $G''$  un peso. Il peso viene definito in modo da bilanciare il consumo energetico totale con il consumo energetico dell’energia di ciascun sensore. Sia  $eMin = \min\{(w(u, v)|(u, v) \in E'')\}$  l’energia richiesta dal sensore  $u$  per trasmettere il messaggio al vicino più prossimo in  $G''$  e

$$\rho(u, v) = \begin{cases} 0 & ce(u) - w(u, v) > eMin(u) \\ c & \text{altrimenti} \end{cases}$$

dove  $c$  è una costante non negativa.

Per ciascun nodo, viene definito  $\alpha(u) = \frac{\min(RE)}{ce(u)}$ . Il peso  $w''(u, v) \in E''$  assegnato ad un arco  $(u, v)$  è calcolato come  $w(u, v) = (w(u, v) + \rho(u, v))(\lambda^{\alpha(u)} - 1)$ , dove  $\lambda$  è un altro parametro non negativo dell’algoritmo. Come si può notare, questa funzione assegna un alto peso agli archi il cui uso porterebbe alla riduzione dell’energia residua di un dato nodo (tramite  $\rho$ ). Inoltre, viene assegnato un alto costo a tutti gli archi uscenti da un sensore la cui energia è piccola rispetto a  $\min RE$ , a causa del termine  $\lambda$ .

### 2.5.2 Maximum lifetime data gathering

L’algoritmo Maximum lifetime data gathering [24] suppone che la rete di sensori sia composta da  $n$  sensori ( $1, 2, \dots, n$ ) e da un collettore  $t(n+1)$ , le cui posizioni siano conosciute e fissate a priori. Inoltre, si assume che ciascun nodo generi un pacchetto di dati (di dimensioni pari a  $k$  bits) per unità di tempo (detto round) che deve essere consegnato al collettore. Ciascun nodo è in grado di trasmettere i propri dati a qualunque altro nodo della rete o al collettore. Tutti i nodi della rete sono dotati di una quantità di energia finita ( $E_i$ ), mentre il collettore non ha vincoli energetici.

Il ciclo di vita del sistema  $T$  è definito come il numero di round o di letture periodiche di dati dai sensori che precedono la prima failure di un nodo. Lo schedule per la raccolta dei dati specifica per ogni round come ottenere e instradare i dati al collettore. Uno schedule è caratterizzato da un albero di distribuzione che collega il collettore con tutti i nodi della rete. La durata del ciclo di vita del sistema dipende dalla durata della validità dello schedule. L’obiettivo è quello di ottimizzare la durata dello schedule.

Per raggiungere questo obiettivo, è stato proposto l’algoritmo Maximum Lifetime Data Aggregation (MLDA). L’algoritmo tiene conto della possibile aggregazione dei dati durante la creazione di instradamenti in modo da massimizzare la durata della rete. Considerando uno schedule  $S$  con  $T$  round, si definisce  $f_{i,j}$  il numero totale di pacchetti che un nodo  $i$  trasmette ad un altro sensore (o al collettore)  $j$  in  $S$ . Dal momento che uno schedule valido deve rispettare i vincoli energetici su ciascun sensore, per ciascun nodo  $i = 1, 2, \dots, n$  deve valere:

$$\sum_{j=1}^{n+1} f_{i,j} T x_{i,j} + \sum_{j=1}^n f_{i,j} R x_{i,j} \leq E_i$$

Quindi, per ciascun schedule  $S$ , viene creato un flusso di rete  $G = (V, E)$ . Sia  $S$  uno schedule con tempo di vita  $T$  e sia  $G$  il flusso di rete indotto da  $S$ , per ciascun sensore  $s$  il flusso massimo da  $s$  al collettore è maggiore o uguale a  $T$ . Per trovare il flusso di rete  $G$  con il massimo  $T$ , è necessario calcolare la capacità di ciascun arco di  $G$ . Il flusso di rete ottenuto con la massima durata del ciclo di vita è chiamato flusso di rete ottimale ammissibile. Un flusso di rete ammissibile ottimale può essere ottenuto utilizzando la programmazione lineare intera, utilizzando, oltre al tempo di vita  $T$  e alle capacità degli archi  $f_{i,j}$ , le seguenti variabili: per ciascun sensore  $k = 1, 2, \dots, n$  sia  $\pi_{i,j}^{(k)}$  la variabile che indica il flusso inviato da  $k$  al collettore tramite l’arco  $(i, j)$ . L’obiettivo è massimizzare  $T$  in modo da rispettare i vincoli energetici e i seguenti vincoli per tutti i valori di  $k = 1, 2, \dots, n$

$$\sum_{j=1}^n \pi_{j,i}^{(k)} = \sum_{j=1}^{n+1} \pi_{j,i}^{(k)} \text{ per tutti i valori di } i = 1, 2, \dots, n \text{ e } i \neq k$$

$$T + \sum_{j=1}^n \pi_{j,k}^{(k)} = \sum_{j=1}^{n+1} \pi_{k,j}^{(k)} \quad 0 \leq \pi_{i,j}^{(k)} \leq f_{i,j} \text{ per tutti i valori di } i = 1, 2, \dots, n \text{ e } j = 1, 2, \dots, n+1$$

$$\sum_{j=1}^n \pi_{i,n+1}^{(k)} = T \text{ dove tutte le variabili sono numeri interi.}$$

Una volta ottenuto un flusso di rete ammissibile, è necessario ottenere il relativo schedule, che è un insieme di alberi diretti che hanno come radice il collettore e

che coinvolge tutti i nodi. Ciascun albero, detto albero di aggregazione, stabilisce come i pacchetti vengono aggregati e trasmessi al collettore. Ciascun albero può essere utilizzato per uno o più round: il numero di volte in cui viene utilizzato viene detto  $f$  (tempo di vita dell'albero di aggregazione). Vengono, inoltre, definite profondità di un nodo sensore  $v$  la media delle proprie profondità in ciascuno degli alberi di aggregazione e profondità dello schedule la massima profondità dei nodi appartenenti allo schedule stesso.

La Figura 2.23 mostra un flusso ammissibile  $G$  con tempo di vita  $T = 100$  round e due alberi di aggregazione  $A_1$  e  $A_2$  con tempi di vita 60 e 40 round rispettivamente. Analizzando  $A_1$  è possibile notare che per ciascun round il sensore 2 trasmette un pacchetto al sensore 1, che a sua volta aggrega questi dati con i propri e li invia al collettore (4). Dato un flusso di rete ammissibile  $G$  con tempo di vita  $T$  e un albero  $A$  con radice nel collettore  $t$  con tempo di vita  $f$ , si definisce la riduzione rispetto a  $(A, f)$   $G'$  di  $G$  come il flusso di rete che si ottiene da  $G$  dopo aver ridotto le capacità dei propri archi che appartengono anche ad  $A$  di un fattore  $f$ .  $G'$  è ammissibile se il massimo flusso da  $v$  al collettore  $t$  è maggiore o uguale a  $T - f$  per ciascun vertice  $v$  di  $G'$ . Si noti che  $A$  non deve includere tutti i vertici di  $G$  e quindi non è necessariamente un albero di aggregazione.

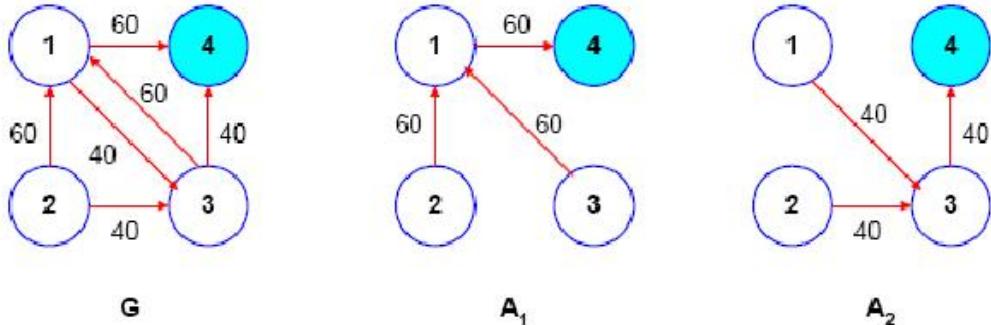


Figura 2.23. Flusso di rete ammissibile  $G$  con  $T = 100$  round e due alberi di aggregazione  $A_1$  e  $A_2$  con tempi di vita 60 e 40 round rispettivamente. La profondità dello schedule relativo è 2.

Inoltre, se  $A$  è un albero di aggregazione, con tempo di vita  $f$ , per un flusso di rete ammissibile  $G$  con tempo di vita  $T$  e la riduzione di  $G$  sia fattibile, allora il flusso ottenuto con la riduzione è un flusso di rete ammissibile con tempo di vita  $T - f$ . Per costruire uno schedule a partire da un flusso di rete ammissibile  $G$  con tempo di vita  $T$ , è possibile utilizzare un semplice algoritmo iterativo, se si utilizza un albero di aggregazione  $A$  che soddisfi i requisiti precedenti.

Per ottenere un albero di aggregazione  $A$  con tempo di vita  $f$  a partire da un flusso di rete ammissibile  $G$  con tempo di vita  $T \geq f$  viene utilizzato l'algoritmo riportato in Figura 2.24, che garantisce che  $A$  abbia la radice nel collettore e che la riduzione  $(A, f)$ -esima sia ammissibile. L'albero  $A$  viene creato come segue: inizialmente  $A$  contiene solamente il collettore. Fino a quando  $A$  non contiene tutti i sensori, viene aggiunto ad  $A$  un arco  $e = (i, j)$  dove  $i \notin A$  e  $j \in A$ , se la riduzione

$(A, f)$ -esima di  $G$  risulta ammissibile ( $A'$  è composto da  $A$  e dall’arco  $e$  mentre  $f$  è il minimo delle capacità degli archi in  $A'$ ).

```

GETTREE (Flow Network  $G$ , Lifetime  $T$ , Base Station  $t$ )
1   initialize  $f \leftarrow 1$ 
2   let  $A = (V_o, E_o)$  where  $V_o = \{t\}$  and  $E_o = \emptyset$ 
3   while  $A$  does not span all the nodes of  $G$  do
4       for each edge  $e = (i, j) \in G$  such that  $i \notin V_o$  and  $j \in V_o$  do
5           let  $A'$  be  $A$  together with the edge  $e$ 
6           // check if the  $(A', 1)$ -reduction of  $G$  is feasible
7           let  $G_r$  be the  $(A', 1)$ -reduction of  $G$ 
8           if  $\text{MAXFLOW}(v, t, G_r) \geq T - 1$  for all nodes  $v$  of  $G$ 
9               // replace  $A$  with  $A'$ 
10           $V_o \leftarrow V_o \cup \{i\}$ ,  $E_o \leftarrow E_o \cup \{e\}$ 
11          break
12      let  $c_{min}$  be the minimum capacity of the edges in  $A$ 
13      let  $G_r$  be the  $(A, c_{min})$ -reduction of  $G$ 
14      if  $\text{MAXFLOW}(v, t, G_r) \geq T - c_{min}$  for all nodes  $v$  of  $G$ 
15           $f \leftarrow c_{min}$ 
16      replace  $G$  with the  $(A, f)$ -reduction of  $G$ 
17  return  $f, G, A$ 
```

Figura 2.24. Costruzione di un albero di aggregazione  $A$  con tempo di vita  $f$  a partire da un flusso di rete ammissibile  $G$  con tempo di vita  $T$  tale per cui la riduzione  $(A, f)$  – esima di  $G$  sia ammissibile.

Una variante del problema stata proposta per le applicazioni in cui l’aggregazione dei dati non è possibile, ad esempio per flussi provenienti da sensori video. In questo caso, viene utilizzato il protocollo Maximum Lifetime Data Routing (MLDR) ed in cui si modellizza il problema del flusso di rete con vincoli energetici sui sensori. Le performance in termini della durata di vita del sistema di MLDR e MLDA sono state confrontate a quelle del Pegasus gerarchico. I risultati mostrano che entrambi hanno prestazioni nettamente migliori rispetto al Pegasus gerarchico. Tuttavia, in MLDA la latenza dei pacchetti è leggermente superiore rispetto al Pegasus gerarchico. Mentre MLDA ha prestazioni migliori rispetto agli altri protocolli, in termini di ciclo di vita del sistema, l’algoritmo è computazionalmente costoso per reti di sensori molto grandi.

### 2.5.3 Minimum cost forwarding

Questo protocollo [9] mira a trovare il percorso di costo minimo in una grande rete di sensori, definendo in ogni nodo un campo di costo che indica il minimo costo di trasmissione da quel nodo al nodo collettore. Il costo del link può tenere del ritardo, del throughput e del consumo energetico.

Il protocollo prevede due fasi. La prima fase, di setup, prevede l’impostazione dei parametri di costo in tutti i nodi. Tale fase ha inizio presso il collettore e, successivamente, si diffonde attraverso la rete. Ogni nodo regola il proprio valore

di costo sommando il costo del nodo da cui ha ricevuto il messaggio e il costo del collegamento. Tale adeguamento del costo non è effettuato attraverso il flooding, ma viene utilizzato un algoritmo di back-off al fine di limitare il numero di messaggi scambiati. La trasmissione del messaggio è differita per un periodo predefinito per consentire ai messaggi con un costo minimo di arrivare. L’algoritmo trova un costo ottimale per tutti i nodi verso il collettore utilizzando un solo messaggio in ogni nodo. Una volta che questi campi di costo sono impostati, non ci sarà bisogno di mantenere lo stato del next hop sui nodi. Questo garantisce la scalabilità dell’algoritmo.

Nella seconda fase, la sorgente manda in broadcast i dati ai propri vicini. I nodi che ricevono il messaggio broadcast, aggiungono il proprio costo di trasmissione (verso il collettore) al costo del pacchetto. Poi il nodo verifica il costo rimanente nel pacchetto. Se non è sufficiente a raggiungere il collettore, il pacchetto viene eliminato. Altrimenti, il nodo inoltra il pacchetto per i suoi vicini. Il protocollo non richiede alcun indirizzamento e percorsi di forwarding. I risultati delle simulazioni mostrano che i valori di costo ottenuti per ogni nodo da questo protocollo sono gli stessi ottenuti utilizzando il flooding. Quindi, si ottiene un forwarding ottimale con un numero minimo di messaggi.

#### 2.5.4 Sequential assignment routing (SAR)

SAR (Sequential Assignment Routing) [4] [38] è il primo protocollo per reti di sensori che include la nozione di QoS nelle decisioni di routing. Si tratta di un approccio basato su table-driven multi-path che tenta di ottenere l’efficienza energetica e la tolleranza ai guasti. Il protocollo SAR crea alberi radicati presso un vicino distante un hop dal collettore, prendendo in considerazione metriche di QoS, risorse energetiche ad ogni livello e la priorità di ogni pacchetto. Utilizzando gli alberi creati, vengono calcolati percorsi multipli dal collettore ai sensori. Uno di questi percorsi è selezionato in base alle risorse energetiche e QoS presenti sul cammino. Il failure recovery viene realizzato forzando la coerenza della tabella di routing tra i nodi a monte e a valle su ogni percorso. Una failure locale scatena una procedura automatica di ripristino del percorso a livello locale. Dato che il protocollo SAR prevede di mantenere percorsi multipli in modo da garantire la fault-tolerance e la facilità di ripristino dei percorsi, il protocollo soffre dell’overhead di gestione delle tabelle e degli stati presso ogni nodo sensore soprattutto quando il numero di nodi è molto elevato.

#### 2.5.5 Energy-aware QoS routing protocol

In questo protocollo [1] viene calcolato un percorso a costo minimo ed energeticamente efficiente che soddisfi determinati ritardi end-to-end durante la connessione. Il costo di collegamento utilizzato è una funzione che tiene in considerazione la riserva di energia dei nodi, l’energia di trasmissione, il tasso di errore e altri parametri di comunicazione.

Al fine di sostenere contemporaneamente sia traffico *best-effort* e *real-time*, viene utilizzato un modello di accodamento basato su classi. Il *bandwidth ratio r*, è definito

come un valore iniziale fissato dal gateway e rappresenta la quantità di banda da dedicare sia al traffico real-time che non real-time su un determinato link in uscita in caso di congestione. Come conseguenza, il throughput dei dati normali non diminuisce con la corretta regolazione del valore di  $r$ .

Il modello di accodamento è rappresentato in Figura 2.25. Il protocollo trova un insieme di percorsi a costo minimo, utilizzando una versione estesa dell’algoritmo di Dijkstra e sceglie da tale insieme un percorso soddisfi il ritardo end-to-end richiesto. Uno degli svantaggi di tale protocollo è che non prevede di utilizzare valori di  $r$  differenti, ma il valore di  $r$  viene scelto dal gateway e impostato su tutti i nodi, non fornendo da possibilità di regolare in modo flessibile o di condividere la banda su link diversi. Questo protocollo è stato esteso da [2] in modo da poter assegnare un valore di  $r$  diverso per ogni nodo, per ottenere un miglior utilizzo dei link.

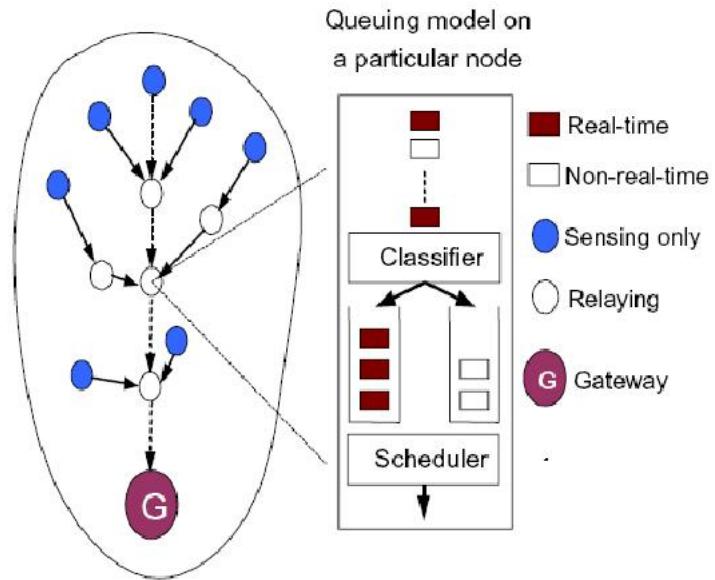


Figura 2.25. Modello di accodamento in un nodo sensore.

### 2.5.6 SPEED

Il protocollo di routing per reti di sensori, con gestione della QoS, descritto in [17] fornisce anche garanzie soft real-time end-to-end. Il protocollo prevede che ogni nodo mantenga informazioni sui propri vicini e che utilizzi il forwarding geografico per individuare i percorsi fra due nodi. Inoltre, SPEED mira a garantire una certa velocità di consegna per ogni pacchetto nella rete in modo che l’applicazione possa stimare il ritardo end-to-end, dividendo la distanza della sorgente dal collettore per la velocità del pacchetto prima di fare decidere il percorso su cui inoltrare il pacchetto. Inoltre, SPEED può fornire la congestion avoidance quando la rete è congestionata. Il modulo di routing di questo protocollo viene chiamato stateless geographic non-deterministic forwarding (SNFG) ed è accoppiato con quattro altri moduli a livello di rete, come mostrato in Figura 2.26. Per raccogliere le informazioni

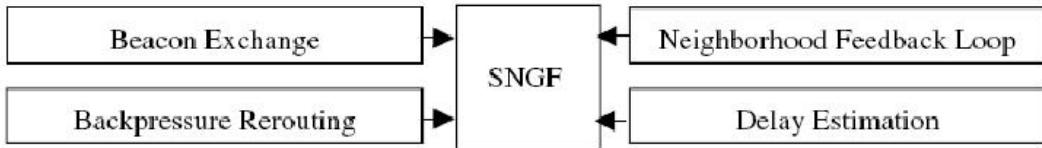


Figura 2.26. Moduli per il calcolo dei percorsi in SPEED.

sui nodi presenti nella rete e la loro ubicazione viene utilizzato un meccanismo basato sulla trasmissione di beacon. La stima del ritardo di consegna di un pacchetto viene fatta calcolando su ogni nodo il tempo trascorso fra l’invio di un pacchetto e la ricezione del relativo ACK. Analizzando i valori di ritardo su diversi percorsi, il protocollo SNGF sceglie come next hop il nodo che soddisfa il requisito di velocità richiesto dal pacchetto che deve essere trasmesso. Se nessun nodo soddisfa tale vincolo, viene verificato il relay ratio del nodo. Il modulo neighborhood feedback è responsabile di fornire il realy ratio che è calcolato analizzando la miss ratio dei vicini di un nodo (i nodi che non possono fornire la velocità richiesta). Se il relay ratio è inferiore di un numero generato casualmente tra 0 e 1, il pacchetto viene scartato. Il modulo il backpressure rerouting viene utilizzato per prevenire interruzioni di percorsi, cioè quando un nodo non riesce a trovare il next hop, e per eliminare la congestione, inviando messaggi di ritorno alla sorgente in modo che essi utilizzino percorsi differenti per l’inoltro dei pacchetti.

## 2.6 ZigBee

### 2.6.1 Introduzione

In telecomunicazioni ZigBee [16] è il nome di una specifica per un insieme di protocolli di comunicazione ad alto livello che utilizzano piccole antenne digitali a bassa potenza e basato sullo standard IEEE 802.15.4 per wireless personal area networks (WPAN). La relazione esistente fra ZigBee e IEEE 802.15.4-2003 è simile a quella esistente tra IEEE 802.11 e la Wi-Fi Alliance. La specifica ZigBee 1.0 è stata approvata il 14 dicembre 2004 ed è disponibile ai membri della ZigBee Alliance. Un tesseramento di livello base alla ZigBee Alliance costa 3500\$ e garantisce l’accesso alle specifiche.

ZigBee opera nelle frequenze radio assegnate per scopi industriali, scientifici e medici (ISM); 868 MHz in Europa, 915 MHz negli Stati Uniti e 2,4 GHz nella maggior parte del resto del mondo. La velocità di trasmissione dei dati varia tra 20 kb/s e 900 kb/s.

Questa tecnologia ha lo scopo di essere più semplice e più economica di altre WPAN come, ad esempio, Bluetooth. Il nodo ZigBee del tipo più complesso si dice che richieda solamente il 10% del codice necessario per un tipico nodo Bluetooth o Wi-Fi, mentre il più semplice dovrebbe richiedere intorno al 2%. Tuttavia, attualmente le dimensioni reali sono più alte e si aggirano intorno al 50% del codice necessario per Bluetooth. I produttori di chip ZigBee prevedono dispositivi da 128 kB.

Nel 2005 il costo stimato per il ricetrasmettitore di un nodo ZigBee era di circa 1.10\$ per il produttore, contando grossi volumi. La maggior parte dei dispositivi ZigBee richiedono però anche un microcontrollore, che fa alzare il costo totale. Quando fu lanciato (1998), per il Bluetooth si prevedeva un costo di 4\$-6\$ per grandi volumi, mentre il prezzo attuale per la fascia consumer è oggi sotto i 3\$.

La ZigBee Alliance ha cominciato a lavorare sulla versione 1.1. che mira ad avvantaggiarsi dei miglioramenti della specifica 802.15.4b, la cui più evidente miglioria è CCM\*, introdotto in alternativa al CCM mode (CTR + CBC-MAC). Il CCM\* gode delle stesse caratteristiche di sicurezza di CCM, fornendo però maggiore flessibilità nella scelta dei metodi di autenticazione e criptazione.

## Caratteristiche

ZigBee [11] è uno standard di wireless mesh network a basso costo e a basso consumo. Il basso costo della tecnologia le permette di essere ampiamente diffusa nelle attività di controllo wireless e nelle applicazioni di monitoraggio. L’utilizzo a bassa potenza inoltre permette una maggiore durata con batterie più piccole e le reti di tipo mesh offrono elevata affidabilità e un range di azione decisamente più ampio. I produttori di chip ZigBee tipicamente vendono dispositivi radio integrati con microcontrollori con memoria flash tra 60 KB e 256 KB.

Lo strato di rete ZigBee supporta nativamente sia reti a stella che ad albero, reti mesh e reti generiche. Ogni rete deve avere un dispositivo coordinatore, a cui viene affidato, all’atto della creazione, il compito di controllare i parametri della rete e la manutenzione di base. All’interno delle reti a stella, il coordinatore deve essere il nodo centrale. Le reti a albero e a maglia permettono inoltre l’utilizzo di router per estendere la comunicazione a livello di rete.

ZigBee si basa sul livello fisico e sul controllo di accesso del canale definito nello standard IEEE 802.15.4 (versione del 2003) per le WPAN a basso rate. La specifiche completano lo standard con l’aggiunta di quattro componenti principali: livello di rete, livello di applicazione, ZigBee device objects (ZDO) e oggetti applicativi definiti dal produttore che consentono la personalizzazione e l’integrazione totale per l’applicazione in questione con il mondo ZigBee.

Oltre ad aggiungere altri due livelli alla struttura sottostante (livello di rete e applicazione), il miglioramento più significativo è l’introduzione degli ZDO. Questi sono responsabili di una serie di compiti, tra cui memorizzare i ruoli dei dispositivi, gestire le richieste di adesione alla rete, rilevare i nuovi dispositivi e gestire la sicurezza. Uno schema dello stack protocollare di ZigBee è illustrato in Figura 2.27.

Poiché i nodi ZigBee possono passare dalla modalità sleep alla modalità attiva in 30ms o meno, la latenza può essere bassa e i dispositivi ne risentono di meno, in particolare rispetto ai ritardi di wake-up del Bluetooth, che sono in genere circa tre secondi. Poiché i nodi ZigBee possono dormire la maggior parte del tempo, il consumo medio è tipicamente molto basso, con conseguente lunga durata della batteria.

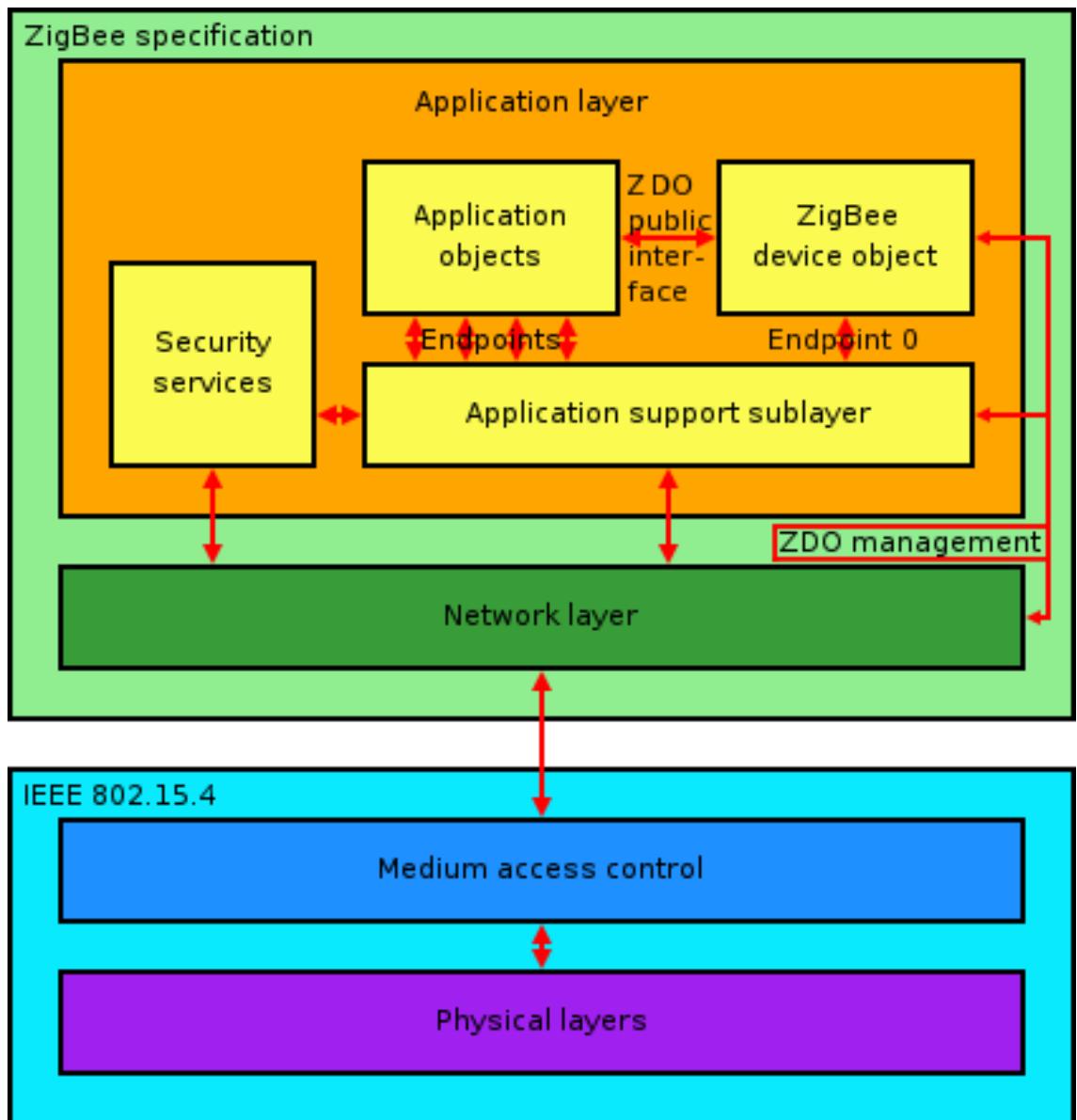


Figura 2.27. Stack del protocollo ZigBee.

### ZigBee Alliance

La ZigBee Alliance è un gruppo di aziende che mantiene e pubblica lo standard ZigBee. Il termine ZigBee non è riferito a un unico standard tecnico ma è un marchio registrato di questo gruppo. L'Alleanza pubblica vari profili delle applicazioni che permettono ai vari fornitori OEM di creare prodotti interoperabili tra loro. Per effettuare un paragone, il rapporto tra IEEE 802.15.4 e ZigBee è simile a quella tra IEEE 802.11 e la Wi-Fi Alliance.

Per scopi non commerciali, la specifica ZigBee è disponibile gratuitamente al pubblico. L'appartenenza all'entry level della ZigBee Alliance, chiamato Adopter, fornisce l'accesso alle specifiche inedite non ancora pubblicate e l'autorizzazione a creare prodotti per il mercato con le suddette specifiche.

I requisiti per l'appartenenza alla ZigBee Alliance causano una serie di problemi agli

sviluppatori di software open-source, perché entrano in conflitto sistematicamente con la GNU General Public Licence. In modo analogo, l’obbligo per lo sviluppatore di aderire alla ZigBee Alliance, entra in conflitto con molte altre licenze di software gratuiti.

### Tipi di dispositivo

Ci sono tre differenti tipi di dispositivo ZigBee:

- ZigBee Coordinator (ZC): è il dispositivo più “intelligente” tra quelli disponibili, costituisce la radice di una rete ZigBee e può operare da ponte tra più reti. Ci può essere un solo Coordinator in ogni rete. Esso è inoltre in grado di memorizzare informazioni riguardo alla sua rete e può agire come deposito per le chiavi di sicurezza.
- ZigBee Router (ZR): questi dispositivi agiscono come router intermedi passando i dati da e verso altri dispositivi.
- ZigBee End Device (ZED): includono solo le funzionalità minime per dialogare con il suo nodo parente (Coordinator o Router), non possono trasmettere dati provenienti da altri dispositivi; sono i nodi che richiedono il minor quantitativo di memoria e quindi risultano spesso più economici rispetto ai ZR o ai ZC.

### 2.6.2 Protocoli

I protocolli ZigBee [5] sono progettati per l’uso in applicazioni embedded che richiedano un basso transfer rate e bassi consumi. L’obiettivo attuale di ZigBee è di definire una Wireless mesh network non mirata, economica e autogestita che possa essere utilizzata per scopi quali il controllo industriale, le reti di sensori, la domotica, le telecomunicazioni (la Z-SIM è stata la prima SIM card creata con un nodo ZigBee). La rete risultante avrà un consumo energetico talmente basso da poter funzionare per uno o due anni sfruttando la batteria incorporata nei singoli nodi.

I protocolli si basano su di una recente ricerca nel campo degli algoritmi di routing (Ad-hoc On-demand Distance Vector) che puntano a costruire delle reti ad-hoc di nodi a bassa velocità. Nelle reti più grandi, la rete reale sarà formata da cluster di cluster, ma si potranno anche formare reti Mesh o cluster singoli. I profili correnti derivati dai protocolli ZigBee supportano sia reti “beacon enabled” che reti “non-beacon enabled”.

Nelle reti non-beacon enabled (quelle il cui beacon order è 15), viene utilizzato un meccanismo di accesso al canale di tipo CSMA/CA. In questo tipo di reti i ZigBee Router solitamente tengono i loro ricevitori sempre attivi, il che provoca un consistente consumo di energia. In pratica queste reti sono “miste”: alcuni dispositivi sono costantemente pronti a ricevere, mentre altri si limitano a trasmettere in presenza di un input esterno. L’esempio tipico di una rete di questo tipo è dato dagli interruttori wireless: un nodo ZigBee è all’interno di una lampada e può essere

costantemente in ricezione, avendo la possibilità della connessione diretta alla rete elettrica, mentre l’interruttore (al pari di un telecomando) alimentato a batteria può rimanere inattivo fino all’istante in cui vi è necessità di mandare un segnale. A quel punto si attiva, invia il comando, riceve un segnale di acknowledge e ritorna inattivo. In questo esempio la lampada sarà un ZR, se non un ZC, mentre l’interruttore sarà uno ZED.

Nelle reti beacon enabled, i nodi detti ZigBee Router trasmettono periodicamente dei beacon per confermare la loro presenza agli altri nodi. Tra un beacon e l’altro i nodi possono cambiare modalità per risparmiare energia, abbassando il duty cycle. Gli intervalli tra i beacon vanno da  $15,6 \text{ ms} \cdot 2^{14} = 251,65824 \text{ s}$  a  $250 \text{ kb/s}$ , da  $24 \text{ ms} \cdot 2^{14} = 393,216 \text{ s}$  a  $40 \text{ kb/s}$  e da  $48 \text{ ms} \cdot 2^{14} = 786,432 \text{ s}$  a  $20 \text{ kb/s}$ . Comunque operazioni a basso duty cycle con lunghi intervalli di beacon richiedono meccanismi di timing preciso, più difficili e costosi da realizzare.

In generale, i protocolli ZigBee minimizzano il tempo di attività del radiotrasmettitore, così da ridurre il consumo di energia. Nelle reti beacon enabled i nodi consumano energia solo nel periodo in cui c’è il beacon, mentre in quelle non-beacon enabled alcuni nodi sono sempre attivi (il loro consumo di energia è quindi alto) mentre altri sono per la maggior parte del tempo spenti.

I dispositivi ZigBee devono rispettare le norme dello standard IEEE 802.15.4-2003 Low-Rate Wireless Personal Area Network (WPAN). Esso specifica il protocollo di livello fisico (PHY), e la parte del livello data link del Medium Access Control (MAC). Questo standard opera nella banda ISM non licenziata 2,4 GHz, 915 MHz e 868 MHz. Nella banda 2,4 GHz ci sono 16 canali ZigBee, da 5 MHz ciascuno. La frequenza centrale per ogni canale può essere calcolata come  $FC = (2400 + 5 \cdot k) \text{ Mhz}$ , con  $k = 1, 2, \dots, 16$ . I trasmettitori radio usano una codifica DSSS. Si usa una modulazione BPSK nelle bande 868 e 915 MHz e una QPSK con offset (O-QPSK) che trasmette 4bit per simbolo nella banda 2,4 GHz. Il data rate over-the-air è di 250 kb/s per canale nella banda 2,4 GHz, 40 kb/s per canale nella banda 915 MHz e 20 kb/s nella banda 868 MHz. Il range è tra 10 e 75 m, dipendente dall’ambiente. La massima potenza trasmessa è in genere 0 dBm (1 mW).

La modalità di base di accesso al canale specificato da IEEE 802.15.4-2003 è il Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA). Questo significa che i nodi, controllano se il canale è libero, quando devono trasmettere. Vi sono alcune eccezioni all’uso del CSMA: i segnali di beacon, inviati secondo uno schema prefissato, i messaggi di acknowledge e le trasmissioni di dispositivi in reti beacon-oriented che hanno necessità di bassa latenza ed usano Guaranteed Time Slots (GTS) che per definizione non fa uso di CSMA.

### 2.6.3 Software

Il software è progettato per essere facile da sviluppare su microprocessori piccoli e poco costosi.

## Il livello rete

Le principali funzioni del livello di rete consistono nel permettere il corretto utilizzo del sottolivello MAC e fornire un’interfaccia adatta per l’uso all’immediato livello superiore, cioè il livello applicazione. Le capacità e la struttura di questo livello sono quelle tipicamente associati a tali livelli di rete, tra cui per esempio il routing.

Da un lato, il livello rete gestisce le unità di dati dello strato di rete dal payload del livello di applicazione ed esegue il routing secondo la topologia corrente. D’altra parte, vi è il livello di controllo, che viene utilizzato per gestire la configurazione di nuovi dispositivi e stabilire nuove reti: può determinare se un dispositivo adiacente appartiene alla rete e scopre nuovi vicini e nuovi router. Il livello di controllo può anche rilevare la presenza di un ricevitore, che permette la comunicazione diretta e sincronizzazione a livello MAC.

Il protocollo di routing utilizzato dal livello di rete è AODV (Ad-hoc On-demand Distance Vector). Per trovare il dispositivo di destinazione, trasmette una richiesta di instradamento a tutti i propri vicini. I vicini poi trasmettono la richiesta ai loro vicini e così via fino a raggiungere la destinazione finale. Una volta che la destinazione è raggiunta, invia la sua risposta indietro in unicast seguendo il percorso con il costo più basso per raggiungere il mittente. Una volta che il mittente riceve la risposta, potrà aggiornare la sua tabella di routing aggiungendo l’indirizzo di destinazione con il suo next hop e il costo del percorso.

## Il livello applicazione

Il livello applicazione è il più alto livello definito dalla specifiche, ed è l’interfaccia con cui il sistema ZigBee si presenta agli utenti finali. Esso comprende la maggior parte dei componenti aggiunti dalla specifica ZigBee: sia gli ZDO che le procedure di gestione, insieme con gli applicativi definiti dal produttore, sono considerati parte di questo livello.

Il ZDO è responsabile per la definizione del ruolo di un dispositivo come coordinatore o come dispositivo finale, come detto prima, ma anche per la scoperta di nuovi dispositivi nella rete e l’identificazione dei servizi offerti. Può anche stabilire collegamenti sicuri con dispositivi esterni e rispondere alle richieste di comunicazione di conseguenza.

Il sottolivello per il supporto applicativo (APS, ovvero Application Support Sublayer) è l’altro componente principale di questo livello, e come tale offre una ben definita interfaccia e una serie di servizi di controllo. Funziona come un ponte di collegamento tra il livello di rete e gli altri componenti del livello applicazione: mantiene aggiornate le tabelle di bindings sotto forma di database, che possono essere utilizzate per trovare gli appropriati dispositivi a seconda dei servizi che sono necessari e in base ai diversi servizi offerti dai dispositivi stessi. Dato che permette l’unione tra i due livelli specificati in precedenza, instrada anche messaggi attraverso lo stack protocollare.

### 2.6.4 Comunicazione e dispositivi di rilevamento

Affinché le applicazioni possano comunicare, i loro dispositivi devono utilizzare un protocollo comune (tipi di messaggi, formati e così via): questa serie di convenzioni sono raggruppate in profili. Il binding con un certo profilo viene deciso da corrispondenti identificatori di ingresso e di uscita del cluster, che sono unici nel contesto di un determinato profilo e associati ad un flusso in ingresso o in uscita dati in un dispositivo. Le tabelle di binding contengono coppie del tipo origine-destinazione. A seconda delle informazioni disponibili, la scoperta di un dispositivo può seguire strade diversi. Quando l’indirizzo di rete è noto, l’indirizzo IEEE può essere richiesto utilizzando una comunicazione unicast. Quando invece non lo è, le domande vengono inviate in broadcast (l’indirizzo IEEE farà parte del payload del pacchetto di risposta). Gli end node semplicemente risponderanno con l’indirizzo richiesto, mentre un nodo coordinatore o un router invierà anche gli indirizzi di tutti i dispositivi ad esso associati.

Questo protocollo di discovery permette a dispositivi esterni di trovare i dispositivi in una rete e scoprire i servizi che essi offrono, quali endpoint rispondono quando verrà richiesto un particolare servizio.

L’uso di identificatori di cluster rafforza il legame tra le entità complementari per mezzo delle tabelle di binding, che sono gestite dai nodi coordinatori ZigBee: la tabella deve essere sempre a disposizione all’interno di una rete perciò viene mantenuta dai coordinatori che hanno più probabilità di avere a disposizione un’alimentazione permanente. Alcune applicazioni possono anche richiedere la gestione di backup, che saranno gestiti da strati di livello superiore.

Quando un dispositivo richiede il binding è necessario che ci sia un collegamento di comunicazione già stabilito; dopo che è stato stabilito si decide se aggiungere un nuovo nodo di rete è a seconda dell’applicazione e delle varie politiche di sicurezza. La comunicazione può avvenire subito dopo l’associazione. L’indirizzamento diretto utilizza sia l’indirizzo associato alla radio che l’identificatore del nodo finale, mentre l’indirizzamento indiretto può usare ogni campo rilevante (indirizzo, endpoint, cluster e vari attributi) e richiede che siano inviati al coordinatore della rete, che mantiene le associazioni e traduce le richieste di comunicazione. L’indirizzamento indiretto è particolarmente utile per mantenere le informazioni su alcuni dispositivi molto semplice e ridurre al minimo il loro bisogno di risorse per memorizzarli. Oltre a questi due metodi, la trasmissione in broadcast a degli endpoint è sempre disponibile: viene utilizzato l’indirizzo di un gruppo per comunicare con quegli endpoint appartenenti allo stesso tipo di gruppo.

## 2.7 AODV

AODV, acronimo di Ad-hoc On-demand Distance Vector, è un algoritmo di routing per reti ad-hoc mobili e supporta sia l’instradamento unicast che multicast.

Si basa su un protocollo di tipo reactive poiché ricerca dei percorsi nella rete solo su richiesta, al contrario dei protocolli più comuni in Internet e nelle reti cablate che individuano tutti i nodi ed i percorsi possibili della rete indipendentemente dal loro



Figura 2.28. XBee, un esempio di dispositivo radio prodotto dalla Digi International.

uso (protocolli di tipo proactive).

Come suggerisce il nome, AODV è un derivato per reti ad-hoc del protocollo Distance Vector.

### 2.7.1 Funzionamento

Ogni nodo possiede un proprio numero di sequenza (sequence number) che cresce monotonamente nel tempo e che garantisce l’assenza di cicli nei percorsi utilizzati. Inoltre ogni componente della rete adibito alla funzionalità di routing memorizza un suo indice dei percorsi, che contiene l’indirizzo del prossimo nodo in direzione della destinazione (next hop), il suo numero di sequenza e la distanza complessiva indicata in salti (hops), o eventualmente in altre metriche atte alla misurazione della qualità del collegamento.

In AODV la rete rimane completamente silente finché non è richiesta una connessione per l’inoltro di un pacchetto dati. Quando è necessario cercare dei percorsi sulla rete, AODV ricorre ai seguenti pacchetti definiti dal suo protocollo:

- Route request (RREQ),

- Route reply (RREP),
- Route error (RERR).

Questi messaggi possono essere implementati come semplici pacchetti UDP, per cui il routing si basa comunque sul Internet Protocol (IP). I pacchetti RREQ vengono inviati in broadcast dal nodo sorgente, per cui si genera un’esplosione di messaggi che vengono inoltrati attraverso tutta la rete. Quando un nodo della rete riceve un pacchetto di richiesta può inviare un pacchetto di RREP attraverso un percorso temporaneo fino al nodo richiedente, che potrà dunque sfruttare l’informazione appena ricevuta. Generalmente ogni nodo confronta i diversi percorsi in base alla loro lunghezza e sceglie il più conveniente. Se un nodo non è più raggiungibile viene generato un messaggio di RERR per avvertire il resto della rete. Ogni RREQ ha un “time to live” che limita le volte che può essere ritrasmesso. Inoltre AODV implementa un meccanismo di backoff binario nel caso il nodo non riceva risposta al suo RREQ, per cui le richieste vengono ripetute a intervalli di tempo crescenti linearmente fino ad un massimo stabilito dall’implementazione.

Il vantaggio principale di AODV è quello di non generare traffico nel caso di percorsi già stabiliti e funzionanti. In effetti, l’algoritmo stesso è del tutto ininfluente finché non risulta necessario inviare un pacchetto ad un nodo di cui non si conosce il percorso. Al di là di questo, il routing basato su distance vector risulta semplice dal punto di vista computazionale e non richiede grandi quantitativi di memoria. Tuttavia il protocollo richiede tempi più lunghi rispetto ad altri protocolli per stabilire una connessione tra due nodi di una rete.

## 2.8 Z-MAC

Zebra-MAC (Z-MAC) [32] è un protocollo ibrido che cerca di combinare i pregi di CSMA e TDMA. Come CSMA, Z-MAC ottiene un alto utilizzo del canale e una bassa latenza con poche contese, mentre come TDMA, ottiene un alto utilizzo del canale con un numero elevato di contese, riducendo le collisioni tra due hop vicini ad un basso valore. Un tratto distintivo di Z-MAC è la sua robustezza agli errori di sincronizzazione, agli errori nell’assegnazione degli slot e a variazioni della topologia della rete nel tempo. Nel peggiore dei casi le sue performance scendono fino a quelle del CSMA. Si può trovare una implementazione di Z-MAC in TinyOS.

Un canale radio non può essere utilizzato in trasmissione da più di un nodo per volta nel suo raggio di interferenza. I nodi vicini possono causare “conflitti” o interferenze di segnale ad alcuni nodi se trasmettono nello stesso periodo di tempo e sullo stesso canale. Nelle reti di sensori wireless controllare l’accesso al canale (conosciuto come MAC, Multiple Access Control), ha un ruolo essenziale nel determinare la capacità di utilizzo del canale stesso, i ritardi della rete e il consumo stesso. Il MAC influenza altresì la congestione e l’equità di utilizzo del canale.

Le reti di sensori wireless possono venire utilizzate in applicazioni molto diverse, dalle applicazioni di monitoring ad una frequenza molto bassa fino a bitrate molto alti (nell’ordine di decine o centinaia di Mb al secondo).

Il CSMA (Carrier Sense Multiple Access) è un paradigma molto comune nelle reti di sensori wireless, perché è semplice, flessibile, robusto e non richiede particolari sincronizzazioni di clock o informazioni della topologia globale. Questi vantaggi però hanno come conseguenza l’aumento di errori di trasmissioni in caso di collisioni, che causano il rapido degradamento del canale e della banda aggregata. Un altro problema del CSMA è quello del terminale nascosto, illustrato in 2.29: il nodo *B* è in grado di sentire i segnali inviati da *A* e *C*, ma questi ultimi due nodi non si percepiscono a vicenda a causa dell’eccessiva distanza tra di loro. Ciò comporta che *A* e *C*, supponendo erroneamente che il canale sia libero, trasmetteranno contemporaneamente generando una collisione (percepita solo da *B*). Il “terminale nascosto” causa una seria degradazione del throughput quando le reti di sensori su cui viene applicato hanno un alto tasso di trasferimento dati.

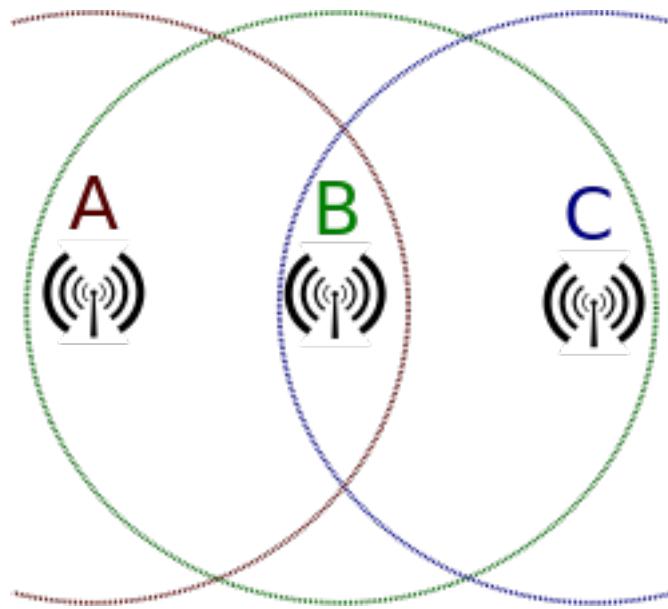


Figura 2.29. Esempio del problema del “terminale nascosto”.

Il problema del “terminale nascosto” può essere risolto tramite l’RTS/CTS (Request to Send / Clear to Send), un meccanismo opzionale del 802.11 che prevede l’invio di un pacchetto di tipo RTS da parte di un nodo prima che esso inizi la trasmissione dei dati: se riceverà come risposta CTS potrà iniziare a trasmettere. Questa soluzione comporta un overhead notevole in termini di pacchetti scambiati e tempo di attesa.

Il TDMA (Time-Division Multiple Access), invece, risolve il problema del “terminale nascosto” senza introdurre un overhead di messaggi perché riesce a schedulare le trasmissioni dei nodi vicini in diverse finestre temporali. Una conseguenza di questo può essere, ad esempio, la difficoltà nel trovare una schedulazione temporale efficiente e scalabile (in genere effettuata da un nodo master). In aggiunta sviluppare una schedulazione efficiente con un alto grado di concorrenza e un grande utilizzo del canale è considerato molto complesso (un problema *NP-completo*). Il TDMA richiede altresì una sincronizzazione dei clock tra i nodi.

Z-MAC cerca di sopperire ai lati negativi di CSMA e TDMA utilizzando CSMA come base dell’algoritmo, e usando TDMA come aiuto ai sensori in caso di contesa. In Z-MAC l’assegnazione dello slot temporale è effettuata durante il deployment, con un alto overhead iniziale. La sua applicazione migliore è dunque in reti di sensori wireless nelle quali non cambiano con alta frequenza il numero e la posizione dei sensori stessi, anche se è molto robusto ai cambiamenti di topologia della rete di sensori.

Z-MAC utilizza la versione distribuita di RAND, D-RAND [33], un algoritmo efficiente e scalabile per lo scheduling del canale, per assegnare uno slot temporale a ogni nodo presente nella rete. L’assegnazione degli slot tramite D-RAND garantisce che la trasmissione dei dati da parte di un nodo ai suoi vicini a distanza di un hop, non interferirà con i nodi a distanza di due hop, superando il problema del “terminale nascosto” esposto in precedenza.

Questa suddivisione in slot non è però rigida: a differenza di TDMA, in Z-MAC un nodo può trasmettere anche in slot non assegnati a lui, previo carrier-sensing in attesa di avere il canale libero.

Un nodo si può trovare in due modalità: Low Contention Level (LCL) oppure High Contention Level (HCL). La scelta tra le due modalità è fatta sulla base di quanti pacchetti di risposta (ack) vengono persi dai vicini a distanza un hop e quante collisioni vengono rilevate sul canale. In aggiunta può avvenire l’invio di un messaggio di tipo explicit contention notification (ECN) da parte di un vicino il quale si trova in modalità HCL e ci notifica di assumere lo stesso comportamento. In LCL qualsiasi nodo può concorrere alla trasmissione in qualunque slot, mentre in HCL solo il possessore di uno slot e i suoi vicini a distanza un hop possono prendere parte alla contesa. In caso di contesa in entrambe le modalità però il canale verrà occupato dal nodo che aveva lo slot assegnato (a causa della sua priorità maggiore). Se uno slot non è stato assegnato a nessun nodo, o il possessore non ha dati da trasmettere, gli altri nodi possono inviare in tale slot.

La priorità è gestita aggiustando la dimensione della finestra di contesa iniziale in modo che il proprietario della finestra abbia sempre la possibilità di trasmissione precedente a tutti gli altri. Lo scopo è di ridurre le collisioni nel caso in cui un nodo abbia dati da trasmettere durante uno slot non suo.

Grazie al mescolamento di CSMA e TDMA, Z-MAC diventa molto robusto agli errori di sincronizzazione, agli errori di assegnazione degli slot, agli errori sul canale e ai cambiamenti di topologia rispetto all’utilizzo del solo TDMA. Nel peggior dei casi le prestazioni degradano a quelle del CSMA.

Z-MAC è strutturato con una fase di setup nella quale vengono eseguite le seguenti operazioni: neighbour discovery, slot assignment, local frame exchange, global time synchronization. Queste operazioni vengono eseguite una volta sola durante la fase di setup, e non vengono eseguite nuovamente fino a quando non viene rilevato un cambiamento significativo nella rete. L’idea è che il costo iniziale nell’eseguire queste operazioni sia compensato da un throughput più elevato e un’efficienza maggiore durante la trasmissione dei dati.

Le fasi sono quindi:

- neighbor discovery e slot assignment: quando il nodo  $N$  si accende esegue un’operazione di neighbor discovery dove periodicamente manda un ping in broadcast ai suoi vicini per ottenere da essi la lista dei nodi che hanno nelle vicinanze. Un messaggio di ping contiene la lista dei propri vicini, perciò dopo questa fase  $N$  sarà a conoscenza dei nodi che distano due hop da lui. Questa informazione è essenziale in quanto costituisce l’input dell’algoritmo D-RAND, usato per lo slot assignment.
- local frame exchange: quando un nodo ha ottenuto il proprio slot temporale, deve decidere quando userà tale slot per trasmettere. Questo periodo è chiamato time frame del nodo.
- global time synchronization: dopo che un nodo sceglie uno slot temporale e un time frame si sincronizza con i suoi vicini, in modo tale che la finestra di trasmissione 0 coincida temporalmente con tutte le finestre 0 dell’intera rete.

Benchè in Z-MAC ogni nodo possa trasmettere in qualsiasi slot, anche assegnati ad altri nodi, non viene definito uno scheduling per le tempistiche di ricezione. Z-MAC si appoggia infatti sulla modalità LPL (Low Power Listening) del B-MAC per quel che concerne l’attivazione dell’antenna per la ricezione. Perciò il consumo energetico dello Z-MAC, dovuto principalmente ai periodi di inattività dei nodi, è pressocchè paragonabile a quello del B-MAC, presentato di seguito.

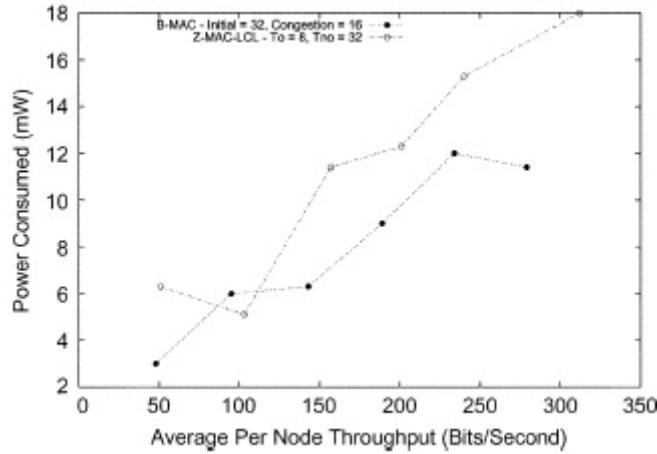


Figura 2.30. Consumo energetico di Z-MAC e B-MAC.

### 2.8.1 B-MAC

B-MAC (Berkeley Medium Access Control) [23] è un protocollo altamente leggero, configurabile e asincrono per gestire l’accesso al canale condiviso nelle reti di sensori wireless.

B-MAC è basato su duty cycle tra tempo di wake-up e tempo di sleep, un doppio meccanismo di backoff per limitare le collisioni e un preambolo esteso per segnalare l’invio di un pacchetto.

Questo protocollo gestisce l’arbitrio del canale con il CCA (Clear Channel Assessment) e con un sistema di backoff, utilizza un meccanismo di ack chiamato LLA (Link Layer Acknowledgment) per l’affidabilità e LPL (Low Power Listening) per permettere una comunicazione a basso consumo di energia.

Un nodo esegue una stima del canale, detta soglia CCA, quando esso è libero. Se, in un determinato istante di tempo, il livello di rumore è superiore a questa soglia, il canale è considerato occupato. B-MAC permette ai nodi di non tenere l’antenna costantemente accesa: viene fatto un campionamento a istanti periodici di tempo del canale per valutare se il canale sia libero o meno, senza consumare energia ascoltando continuamente il canale stesso.

Il protocollo prevede che, se un nodo ha un pacchetto da trasmettere, precede il pacchetto di dati in questione con un preamble che è leggermente più lungo del periodo di sleep di un ricevitore. Il mittente è così sicuro che il destinatario si attiverà durante l’invio del preamble e ne rileverà la trasmissione, restando pronto in ricezione per ricevere i dati. La Figura 2.31 illustra questo funzionamento.

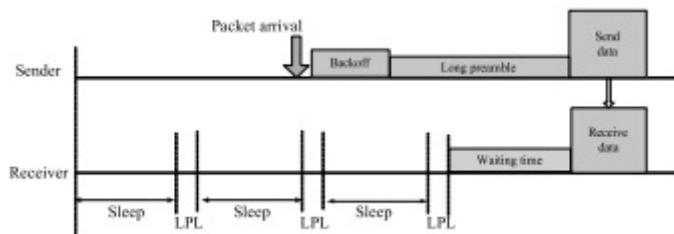


Figura 2.31. Esempio di trasmissione in B-MAC.

Un nodo può essere in uno stato attivo o inattivo. Essere in uno stato inattivo implica non avere alcun pacchetto nella propria coda da trasmettere. Viceversa, lo stato di un nodo quando deve inviare uno o più pacchetti è chiamato attivo. Lo stato attivo è composto ulteriormente dalla fase di backoff iniziale, dal backoff di congestione, da una fase in cui il nodo è “congelato”, dallo stato di trasmissione e di collisione. La transizione tra gli stati è illustrata in 2.32.

Quando un nodo ha un pacchetto da inviare, inizia un periodo di backoff di durata casuale, la cui lunghezza è scelta in maniera uniforme in base ai valori presenti in una finestra di backoff. Durante tale intervallo, il nodo non effettua il CCA per non consumare energia. Se il canale non è libero dopo questo backoff iniziale, il nodo entra nel backoff di congestione. Viene scelto nuovamente un intervallo di backoff, all’interno di un’altra finestra di backoff (finestra di congestione). Durante il backoff di congestione, viene effettuato il CCA per determinare se il canale sia o meno occupato: nel primo caso viene decrementato un contatore di backoff. Quando tale valore raggiunge zero, il nodo inizierà la trasmissione.

Questo doppio meccanismo di backoff, senza l’utilizzo del CCA durante la fase di backoff iniziale e con l’utilizzo del contatore solo nella fase del backoff di congestione è caratteristica del B-MAC, e lo distingue da numerosi altri protocolli di contesa del canale [48].

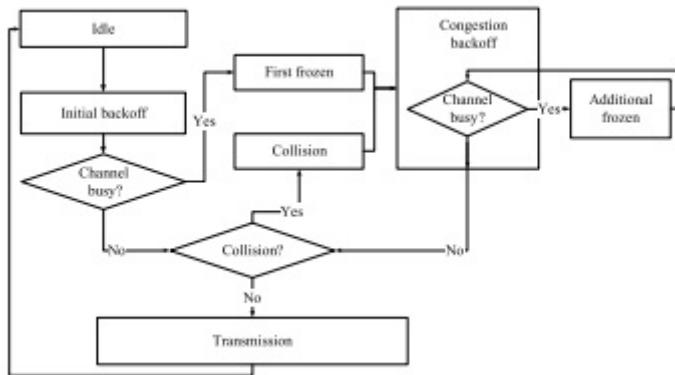


Figura 2.32. Diagramma di transizione tra gli stati di un nodo attivo in B-MAC.

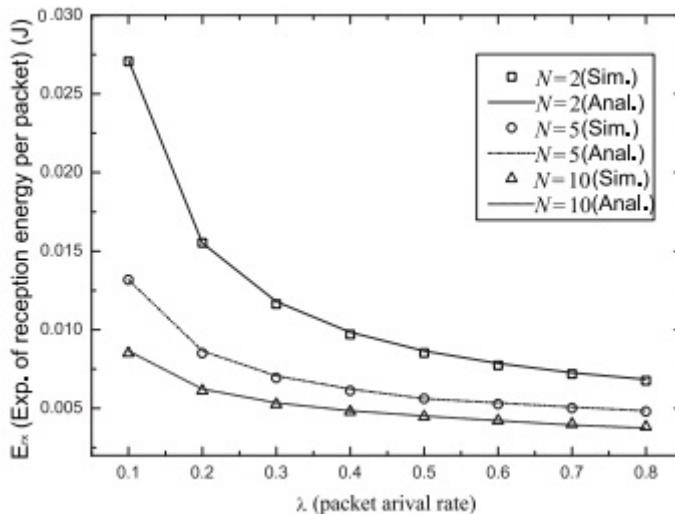


Figura 2.33. Aspettative di consumo energetico per la ricezione di un pacchetto in B-MAC.

Vengono riportati brevemente i consumi del B-MAC in due situazioni tipiche: trasmissione e ricezione. La Figura 2.33 mostra le aspettative di consumo energetico di un nodo che deve inviare un pacchetto secondo il carico di traffico e il numero di contendenti del canale.

In Figura 2.34 vengono presentati i medesimi valori per la ricezione di un pacchetto da parte di un nodo.

## 2.9 Valutazioni generali e scelte implementative

Alcuni degli algoritmi presentati possono essere visti come un buon punto di partenza, ma non centrano in pieno i nostri requisiti non concentrandosi a fondo sul problema principale che è l’energy aware (anche a discapito del throughput), e aggiungendo una serie di complicazioni inutili per i nostri fini che non fanno altro che appesantire e complicare solamente l’algoritmo stesso facendo consumare inutilmente risorse al microcontrollore, sia dal punto di vista computazionale che energetico.

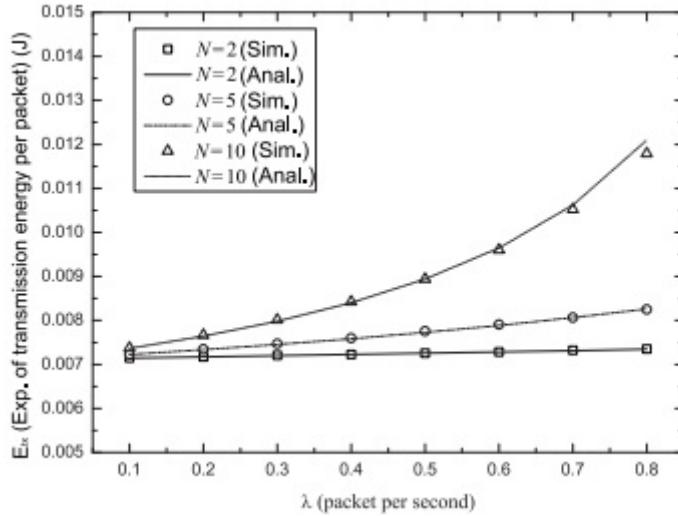


Figura 2.34. Aspettative di consumo energetico per la trasmissione di un pacchetto in B-MAC.

L’algoritmo che presenta delle ottime caratteristiche e meglio risponde alle nostre necessità è lo Z-MAC: analogamente a questo algoritmo si è cercato di prendere gli aspetti migliori del CSMA e del TDMA, compensando i difetti dell’uno con i pregi dell’altro. Tuttavia questa è stata solo la base di partenza: per garantire un’efficienza energetica si sono studiati degli ulteriori accorgimenti per permettere ai nodi di preservare al massimo la loro energia. I risultati ottenuti, illustrati nel capitolo finale, dimostrano che il consumo della wsn progettata è minore di quello previsto dallo Z-MAC: una serie di caratteristiche di tale algoritmo non sono infatti la scelta ottimale per i nostri obiettivi. La gestione dell’attivazione dell’antenna per la fase di ricezione, per esempio, non risponde ai requisiti richiesti, dato che ogni nodo invia periodicamente delle informazioni, perciò non è necessario inviare un preamble per segnalare agli altri nodi l’imminente invio dei dati: i vicini saranno sempre pronti per ricevere senza bisogno di essere “avvisati”. Analogamente, la gestione degli slot assegnati ai nodi può essere interessante per dimensiosare le finestre dinamicamente in base ai dati da trasmettere: tuttavia, difficilmente succederà che un nodo abbia bisogno di uno slot per inviare una quantità maggiore di dati di quella prevista, perciò la contesa per potersi assegnare lo slot di un altro nodo risulta inutile. ZigBee dal canto suo è basato su una serie di algoritmi interessanti che hanno come obiettivo il risparmio energetico. L’architettura progettata ha preso spunto in alcune caratteristiche elementari da ZigBee (come ad esempio tenere la radio spenta il maggior tempo possibile), ma si discosta da ZigBee in molti aspetti, poiché quest’ultimo prevede una serie di funzioni inutili ai nostri scopi. Bisogna anche dire che in commercio esistono tanti dispositivi low power che promettono bassissimi consumi e grandi durate in termini di vita delle batterie, come ad esempio i dispositivi Wasp mote prodotti dalla Libelium, sensori a basso consumo energetico basati sullo standard ZigBee. Non esistono però dispositivi perenni, basati su un algoritmo energy aware che permetta loro di consumare un quantitativo di energia così ridotto da potersi mantenere per sempre grazie a un limitato supporto energetico. L’algoritmo progettato viene presentato all’interno del prossimo capitolo.

# Capitolo 3

## Architettura

### 3.1 Architettura

#### 3.1.1 Descrizione generale

L’architettura del sistema prende spunto da vari algoritmi già ampiamente utilizzati in letteratura, tra i quali si possono notare una divisione in finestre temporali che ricorda l’Aloha e alcune analogie con il distance vector.

L’idea di base è il TDMA, acronimo di Time Division Multiple Access: è una tecnica di multiplazione numerica in cui la condivisione del canale è realizzata mediante ripartizione del tempo di accesso allo stesso da parte degli utenti. Vi sono due tipi di multiplazione a divisione di tempo:

- multiplexing a divisione di tempo sincrono (STDM), noto anche come multiplexing a divisione di tempo quantizzato: prevede che ogni dispositivo abbia a disposizione un’identica porzione di tempo (slot) e che questi ne vengano in possesso attraverso uno schema di tipo round robin (o tramite un altro tipo di scheduler). Questa tecnica di gestione del canale è detta anche multiplexing a divisione di tempo (TDM);
- multiplexing statistico, simile a STDM, con la differenza che ai dispositivi che non devono trasmettere dati non viene assegnato il controllo del canale di trasmissione.

Poichè nella wireless sensor network progettata i nodi effettuano delle misurazioni a intervalli regolari e tutti comunicano i dati rilevati al nodo centrale, l’STDM è senz’altro più appropriato, dato che difficilmente succederà che un nodo sia sprovvisto di informazioni da inviare e non abbia bisogno quindi di occupare il canale.

Il tempo viene quindi suddiviso in intervalli discreti di lunghezza fissa chiamati slot, analogamente come avviene nel protocollo slotted Aloha. Ogni nodo è vincolato a cominciare la propria trasmissione necessariamente all’inizio di uno slot temporale, come è illustrato in Figura 3.1. Se una stazione ad un certo istante è pronta a trasmettere deve attendere necessariamente l’inizio del successivo slot, non può

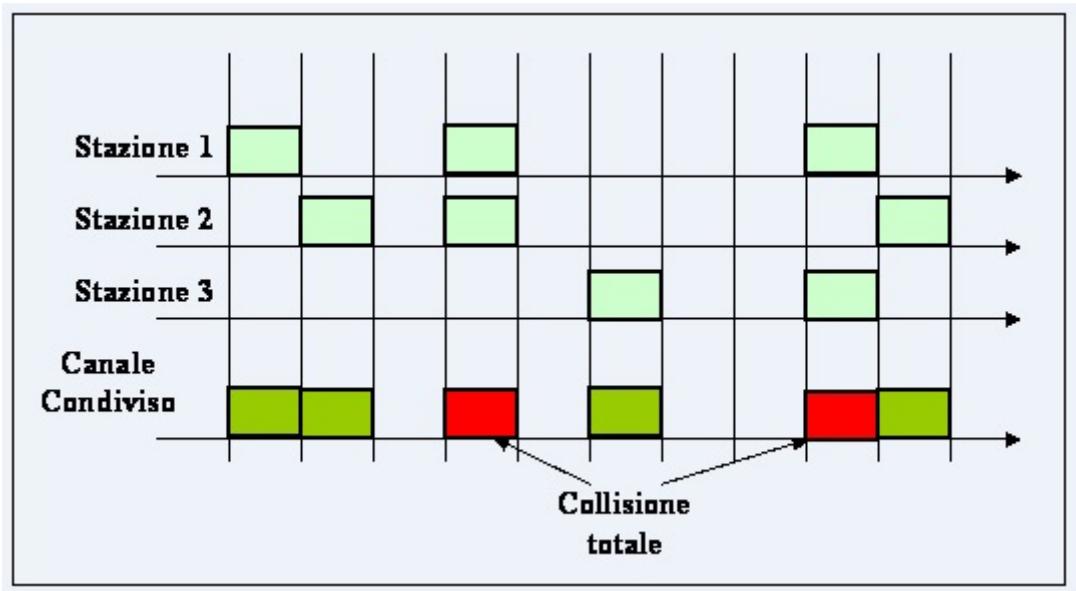


Figura 3.1. Esempio di suddivisione del tempo in slot.

iniziare la trasmissione in un generico istante di tempo  $t$ .

La conseguenza di tale caratteristica è che due trasmissioni o collidono completamente poiché stanno utilizzando lo stesso slot (se due nodi iniziano la trasmissione nel medesimo istante) oppure non collidono affatto; ovvero il periodo di vulnerabilità è pari al tempo dell'intero slot (Figura 3.2).

Per risolvere il problema delle collisioni è necessaria una fase iniziale di discovery

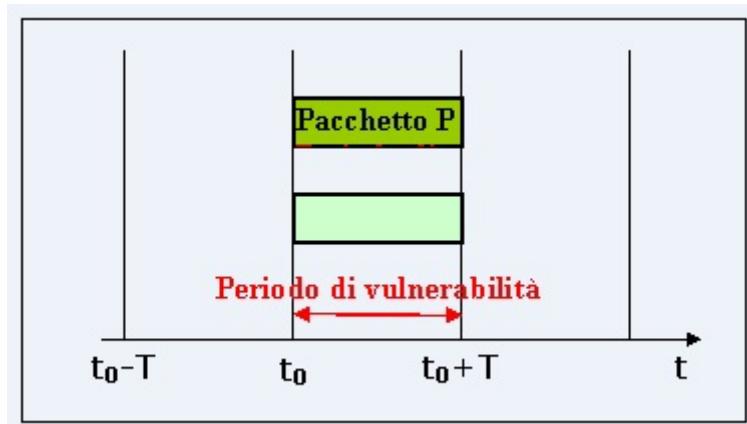


Figura 3.2. Periodo di vulnerabilità di una collisione durante l'invio di un pacchetto.

dei nodi e di setup, in cui il nodo master (il sink) assegna a ogni nodo della rete uno slot fisso. Se lo slot che inizia all'istante  $t_0$  viene assegnato al nodo  $N_1$  nella fase iniziale di setup, l'intervallo di tempo compreso tra  $t_0$  e  $t_0 + T$  (come illustrato in Figura 3.2) sarà di proprietà esclusiva del nodo  $N_1$  per tutta la sua esistenza: nessun altro nodo tenterà di trasmettere nell'intervallo di tempo  $t_0 \rightarrow t_0 + T$ , eliminando il problema delle collisioni del protocollo slotted Aloha.

### 3.1.2 Divisione degli slot temporali

Le assunzioni fatte nella sezione precedente implicano che parte predominante dell'algoritmo è la divisione in slot temporali. Ogni nodo, dopo la fase iniziale di setup, ha uno slot tutto suo che gli viene assegnato dal sink in cui può trasmettere senza il rischio di collisioni con altri sensori. La divisione di un certo intervallo di tempo in slot assegnati ai vari nodi viene fatta all'accensione della rete, per assegnazione di uno slot da parte del nodo principale. Affinchè ogni nodo rispetti tale suddivisione in slot, è necessario che essi siano tutti sincronizzati in modo da trasmettere solo nello slot che gli compete, senza “sforare” da tale slot. Potrebbe infatti capitare una situazione illustrata in Figura 3.3: il nodo  $N_2$  trasmette nello slot che inizia all'istante di tempo  $t_0 - T$  il pacchetto  $P_2$ , mentre il nodo  $N_1$  inizia la sua trasmissione del pacchetto  $P_1$  nello slot successivo (tempo  $t_0$ ). Tuttavia il nodo  $N_1$  ha perso il sincronismo, perciò inizia a trasmettere un istante prima dell'effettivo inizio del suo slot. Ciò provoca una collisione tra i pacchetti  $P_1$  e  $P_2$  dei due nodi: entrambi i pacchetti andranno persi e dovranno essere ritrasmessi, causando un notevole overhead.

Analogamente, si potrebbe verificare una situazione simile, in cui il timer di un

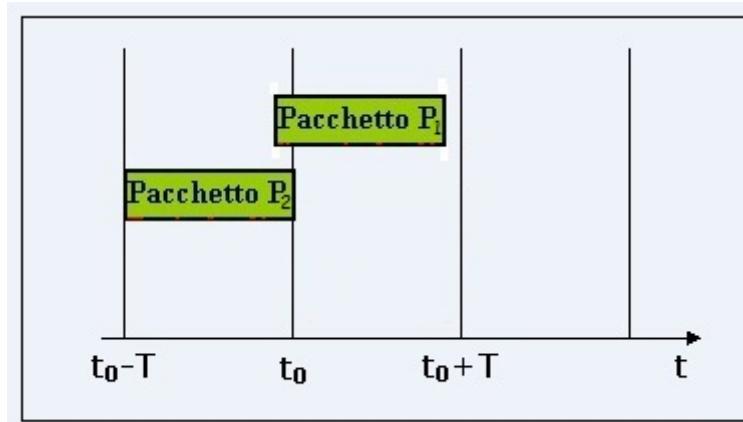


Figura 3.3. Esempio di collisione dovuta alla perdita di sincronismo.

nodo è sfasato “in avanti”, ovvero il nodo inizia la trasmissione in ritardo rispetto all'effettivo inizio del suo slot, terminando quindi più tardi l'invio del pacchetto e causando una collisione con il nodo che trasmette nello slot successivo, come illustrato in Figura 3.4.

La sincronizzazione tra i nodi è quindi una parte molto delicata, in quanto bisogna tenere conto di vari fattori per non rischiare che i nodi perdano il sincronismo tra loro.

Per la parte di sincronizzazione è stato sviluppato un algoritmo ad hoc che permettesse di agire sui timer interni del microcontrollore per poterne resettare valore e prescaler. Vengono presentate di seguito le considerazioni più importanti sulla sincronizzazione dei nodi.

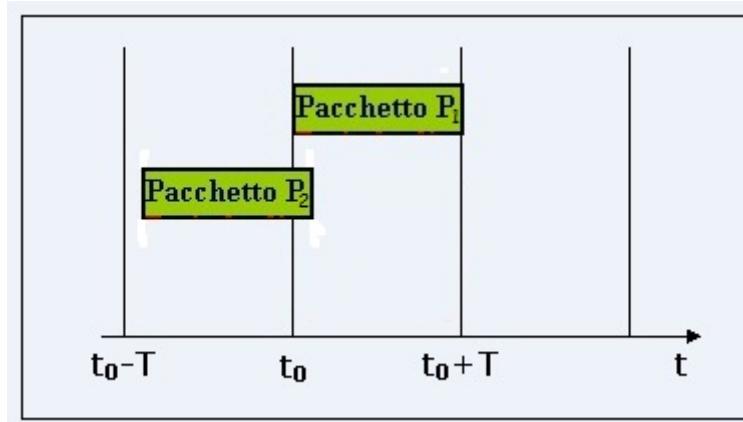


Figura 3.4. Esempio di collisione dovuta alla perdita di sincronismo.

### 3.1.3 Sincronizzazione temporale

Nei sistemi distribuiti, ciascun sensore è dotato del proprio clock ed ha la propria nozione di tempo, tuttavia, è importante avere una scala comune dei tempi tra i diversi sensori per identificare relazioni causa-effetto tra eventi fisici, per supportare l'eliminazione di dati ridondanti e, in generale, per facilitare le operazioni di rete fra i sensori. Dato che ciascun nodo in una rete di sensori opera indipendentemente e fa riferimento al proprio clock, la lettura di esso, effettuata da sensori diversi, può differire. In aggiunta a queste differenze casuali (spostamento di fase), la differenza fra i clock di sensori diversi viene influenzata anche dai tassi variabili di deriva degli oscillatori. Quindi è necessario un meccanismo di sincronizzazione dei clock per assicurare che le misurazioni possano essere confrontate in modo significativo. Diverse tecniche di sincronizzazione per le reti cablate sono state proposte, ma questi approcci non possono essere utilizzati per reti di sensori wireless a causa delle sfide poste dagli ambienti in cui viene effettuato il monitoraggio wireless. Queste sfide includono il grande numero di sensori che potenzialmente possono essere dispiegati, la necessità di auto-configurazione e robustezza, la mobilità dei nodi e la necessità di conservare l'energia. In questa sezione, verranno analizzate alcune tecniche di sincronizzazione temporale che tengono conto di questi vincoli.

I clock degli elaboratori sono basati su oscillatori hardware e sono componenti essenziali di ciascun dispositivo di elaborazione. Un clock tipico consiste di un oscillatore a quarzo stabilizzato e un contatore che è decrementato ad ogni oscillazione del quarzo. Tutte le volte che il contatore raggiunge lo 0, esso viene reinizializzato al proprio valore originario e viene generato un interrupt. Tale interrupt, detto anche tick del clock, incrementa un clock software (anch'esso realizzato in forma di contatore), che può essere letto e utilizzato dalle applicazioni utilizzando delle opportune API (Application Programming Interface). Il clock software fornisce un tempo locale per ciascun nodo e la distanza fra due incrementi del clock viene definita risoluzione del timer. La differenza fra due letture effettuate nello stesso istante da due sensori viene detta offset o scostamento. Se i due clock hanno la stessa frequenza, l'offset si mantiene costante nel tempo e si ottiene la sincronizzazione tarando uno dei due, in modo da annullare l'offset e far sì che le misure coincidano. Si definisce skew la

differenza fra le frequenze di due clock. Clock ideali mantengono costante nel tempo la propria frequenza, ovvero  $dC/dt = 1$  in qualunque istante; nei clock reali il tasso effettivo dipende da diversi parametri, come ad esempio la temperatura, l'umidità dell'ambiente, l'energia fornita e l'età del quarzo. Questa deviazione viene detta deriva (drift rate) ed esprime il tasso con cui due clock si disallineano; si esprime tale tasso con la formula  $\rho = dC/dt - 1$ .

Il massimo drift rate ( $\rho$ ) di un clock basato su quarzo ha valori tipici compresi tra 1 ppm e 100 ppm ed è normalmente indicato dal produttore del quarzo stesso. La

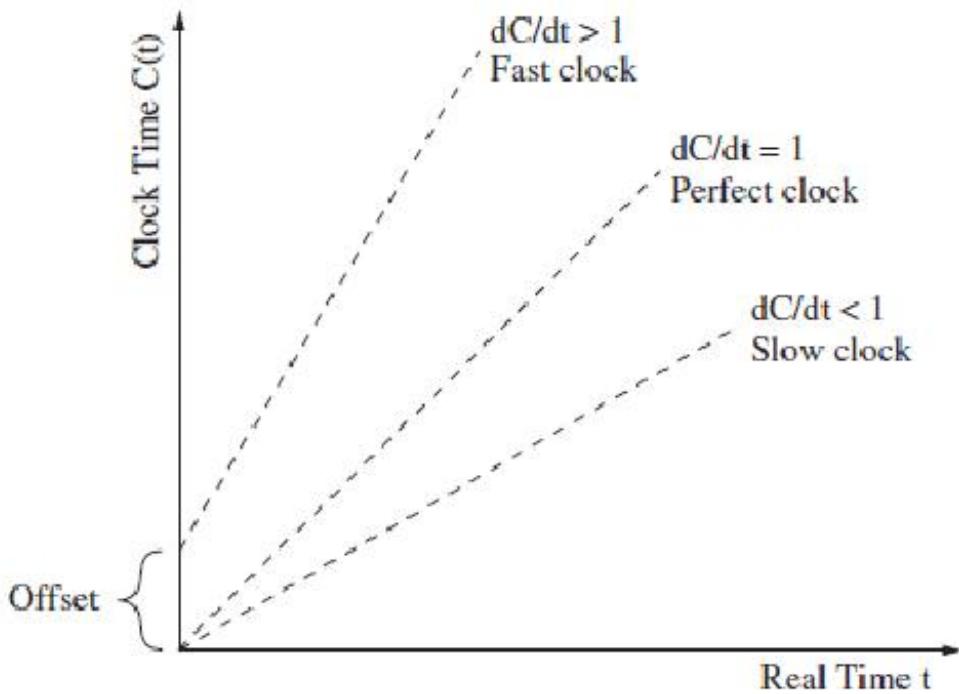


Figura 3.5. Relazioni tra il tempo di clock locale e quello reale.

Figura 3.5 illustra le relazioni tra il tempo di clock locale e quello reale. La presenza di derive origina inconsistenze tra letture del clock di sensori differenti anche a seguito di una sincronizzazione, rendendo necessario ripetere periodicamente tale processo. Assumendo dei clock identici, una coppia di clock sincronizzati tra di loro può derivare al più di una quantità  $\rho_{max}$ . Per limitare l'offset a  $\delta$  secondi, occorre ripetere il processo di sincronizzazione ad intervalli  $\tau_{sync}$  inferiori a  $\frac{\delta}{2\rho_{max}}$ . La funzione  $C(t)$  deve essere continua a tratti e strettamente monotona. Pertanto è necessario applicare in modo graduale le correzioni sul clock, utilizzando una funzione di compensazione lineare che altera la pendenza del tempo locale. Le conseguenze di far variare avanti e indietro il clock potrebbero infatti presentare gravi inconvenienti per le applicazioni software legate allo scorrere del tempo: ad esempio, un interrupt fissato a una determinata ora potrebbe non scattare mai se il clock venisse incrementato di un valore tale da portare l'orologio a superare tale istante.

Possono essere definite due forme di sincronizzazione: quella interna e quella

esterna. Nella sincronizzazione esterna, i clock di tutti i nodi vengono sincronizzati con una sorgente temporale esterna o clock di riferimento. Questo di solito è un riferimento accurato fornito da un dispositivo ad hoc e conforme alle specifiche UTC (Coordinated Universal Time).

La sincronizzazione interna si ottiene, invece, quando i clock di tutti i nodi vengono sincronizzati gli uni con gli altri senza supporto di un clock di riferimento esterno. L'obiettivo di tale forma di sincronizzazione è ottenere una vista consistente del tempo tra tutti i nodi che formano la rete anche se tale valore risulta differente dal tempo reale. La sincronizzazione esterna, invece, garantisce anche la coerenza con il tempo reale. Quando i nodi di una rete sono sincronizzati ad un clock esterno, l'accuratezza di un clock descrive il valore massimo che l'offset di tale clock può assumere rispetto all'orologio di riferimento. Viceversa, in presenza di sincronizzazione interna, la precisione indica il massimo offset tra una qualsiasi coppia di nodi presenti nella rete. Se due nodi sono sincronizzati esternamente con un'accuratezza di  $\delta$ , essi sono anche sincronizzati internamente con una precisione di  $2\delta$ .

### 3.1.4 Sincronizzazione nelle reti wireless

La sincronizzazione temporale è un servizio richiesto da molte applicazioni distribuite. Sono stati proposti molti protocolli di sincronizzazione temporale, sia per reti cablate che wireless; ad esempio il Network Time Protocol (NTP) è un approccio molto diffuso, scalabile, robusto e auto-configurante che viene spesso usato insieme al sistema GPS (Global Positioning System) per ottenere accuratezze dell'ordine di pochi microsecondi. Tuttavia, l'approccio di tale protocollo non è adatto alle reti di sensori per le caratteristiche e i vincoli unici di tale tipo di reti. Ad esempio, in una rete i cui nodi sono dotati di sensori di prossimità, ciascuno di essi può determinare un evento quando un oggetto in movimento transita nei suoi pressi. La correlazione temporale accurata di tali eventi consente di rispondere a domande del tipo: *“Quanti oggetti sono stati rilevati? Qual è la direzione dell'oggetto e qual è la sua velocità?”* Come conseguenza è necessario che un osservatore sia in grado di stabilire il corretto ordine degli eventi, ciascuno dei quali è etichettato con la lettura dell'orologio locale del dispositivo che lo ha rilevato. La sincronizzazione temporale è necessaria per una varietà di applicazioni distribuite, quali protocolli di comunicazione, sicurezza, consistenza dei dati e controllo della concorrenza. Dal punto di vista energetico, molte reti di sensori si basano su protocolli che alternano fasi di attività a fasi di riposo, consentendo alla rete di spegnere in modo selettivo gruppi di nodi al fine di ridurne il consumo. In questo caso, il coordinamento temporale tra nodi è essenziale al fine di permettere a ciascuno di determinare quando può entrare nella fase di riposo e quando invece deve riattivarsi così da garantire la possibilità di scambiare informazioni con i nodi adiacenti.

La deriva dei clock è influenzata da parametri ambientali quali la temperatura, la pressione e l'umidità.

A differenza di quanto succede nelle reti cablate, i cui nodi si trovano ad operare all'interno di ambienti controllati, i sensori wireless sono spesso dispiegati all'esterno,

in ambienti ostili, dove le fluttuazioni nelle caratteristiche ambientali sono frequenti. Negli ambienti controllati, le variazioni tipiche degli oscillatori sono dell'ordine di 1-3 ppm, corrispondenti a errori di circa un secondo ogni 4-12 giorni. Sensori a basso costo per ambienti esterni possono avere prestazioni molto peggiori, richiedendo frequenti sincronizzazioni. Tuttavia, occorre che tali operazioni non contribuiscano significativamente al consumo energetico dei nodi, visto che la sincronizzazione deve essere effettuata indipendentemente dalla disponibilità di dati da trasmettere. Conseguentemente, i protocolli di sincronizzazione per le reti wireless richiedono la minimizzazione sia del numero di pacchetti scambiati che delle loro dimensioni. L'asimmetria delle reti wireless contribuisce ulteriormente a complicare i meccanismi di sincronizzazione: le misure degli offset e dei ritardi di propagazione si possono basare infatti sullo scambio di comunicazioni nelle due direzioni tra coppie di nodi. Tuttavia, quando lo scambio nella direzione contraria non è possibile a causa di fattori variabili e non predibili legati alla mobilità dei nodi, alla presenza di interferenze nei pressi di uno dei dispositivi e ai diversi livelli energetici, esse non possono essere effettuate. Ne consegue che i protocolli di sincronizzazione temporale devono essere progettati tenendo conto delle esigenze di robustezza e di riconfigurabilità delle reti.

Ulteriori vincoli sono introdotti dalle limitate disponibilità di risorse in termini di capacità di calcolo e memoria, imponendo l'adozione di protocolli semplici in grado di scalare sia con la dimensione della rete che con la densità dei dispositivi. L'effettiva accuratezza o precisione della sincronizzazione temporale dipende in ogni caso dall'applicazione finale.

### 3.1.5 Meccanismi elementari di sincronizzazione

La maggior parte dei protocolli di sincronizzazione esistenti sono basati sull'allineamento di singole coppie, dove due nodi sincronizzano i propri clock inviandosi almeno un messaggio. La sincronizzazione a livello rete può essere ottenuta ripetendo tale processo tra più coppie di nodi fino a che tutti quanti non hanno aggiornato il proprio clock.

L'approccio più semplice per sincronizzare una singola coppia si basa sull'invio di un singolo messaggio: al tempo  $t_1$ , il nodo  $i$  invia al nodo  $j$  un messaggio contenente il valore  $t_1$ . Quando lo riceve, il nodo  $j$  provvede a leggere il proprio clock determinando il valore  $t_2$ . La differenza fra queste due marche temporali è pari alla somma del tempo di invio  $D$  e dell'offset  $\delta$ . Il tempo di invio tiene conto sia del tempo di propagazione nel mezzo wireless, sia dei ritardi dovuti all'elaborazione del pacchetto ed al suo accodamento nel buffer di trasmissione. Il primo addendo può essere trascurato, in quanto due nodi riescono a comunicare se sono in prossimità l'uno dell'altro (il tempo di propagazione risulta inferiore al microsecondo). Il tempo di accodamento, invece, può essere variabile ed avere uno o due ordini di grandezza in più. Per compensare tale effetto, spesso si usa il protocollo raffigurato nella Figura 3.6: in questo caso, il nodo  $j$  all'atto della ricezione del pacchetto, non solo determina il tempo corrente  $t_2$ , ma prepara un pacchetto che riporta la tripla  $[t_1, t_2, t_3]$ , essendo  $t_3$  l'istante in cui tale pacchetto viene accodato. Assumendo che i tempi di accodamento nei due nodi siano in media uguali, il nodo  $i$  che riceve tale risposta

al tempo  $t_4$  può determinare sia il tempo di trasmissione che l'offset, secondo le seguenti formule:

$$D = \frac{(t_2 - t_1) + (t_4 - t_3)}{2}$$

$$\text{offset} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}$$

Si noti che in questo caso solamente il nodo  $i$  ha tutte le informazioni necessarie per il calcolo dell'offset, quindi è necessario che esso trasmetta a  $j$  il valore di offset calcolato, utilizzando un terzo messaggio.

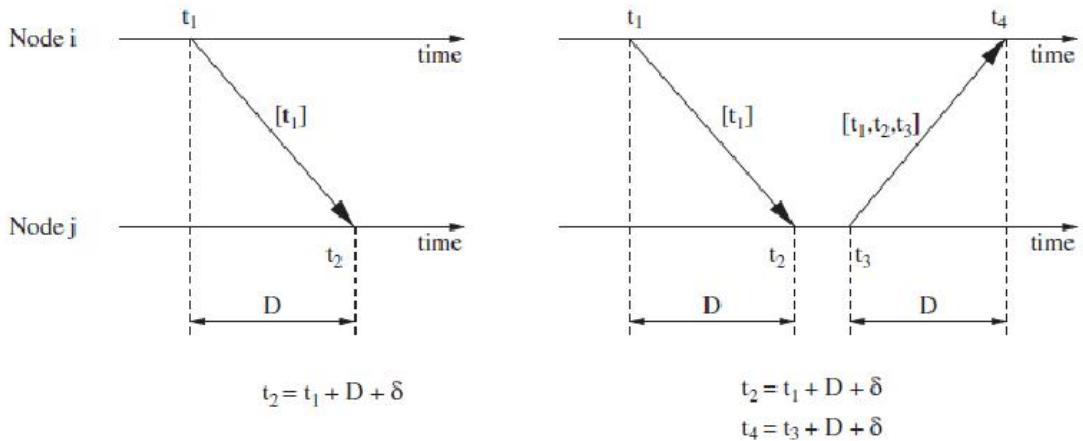


Figura 3.6. Sincronizzazione a coppie.

Un meccanismo alternativo è la sincronizzazione ricevitore-ricevitore che si basa sulla misura del tempo in cui lo stesso messaggio giunge a ricevitori diversi e può essere utilizzato quando la trasmissione avviene in broadcast. Tale approccio è rappresentato in Figura 3.7: il nodo  $i$  trasmette un messaggio, dal contenuto arbitrario, che viene ricevuto da tutti gli altri nodi all'incirca nello stesso istante. I nodi  $j$  e  $k$  scambiano le relative informazioni sul tempo di arrivo e calcolano l'offset come differenza fra i tempi di ricezione. Nel caso in cui ci siano due ricevitori, sono necessari tre messaggi per la sincronizzazione.

Il non determinismo della latenza di comunicazione contribuisce significativamente alla precisione che si può ottenere. In generale, la latenza con cui un messaggio viene consegnato è il risultato della somma di diverse componenti (Figura 3.8):

- ritardo di trasmissione, che è il tempo impiegato dal trasmettitore per generare il messaggio di sincronizzazione e consegnarlo all'interfaccia di rete. Tale ritardo include i tempi introdotti dal sistema operativo (chiamate di sistema, cambi di contesto), dalla pila protocolare dello strato di rete e dal driver dell'interfaccia di rete;

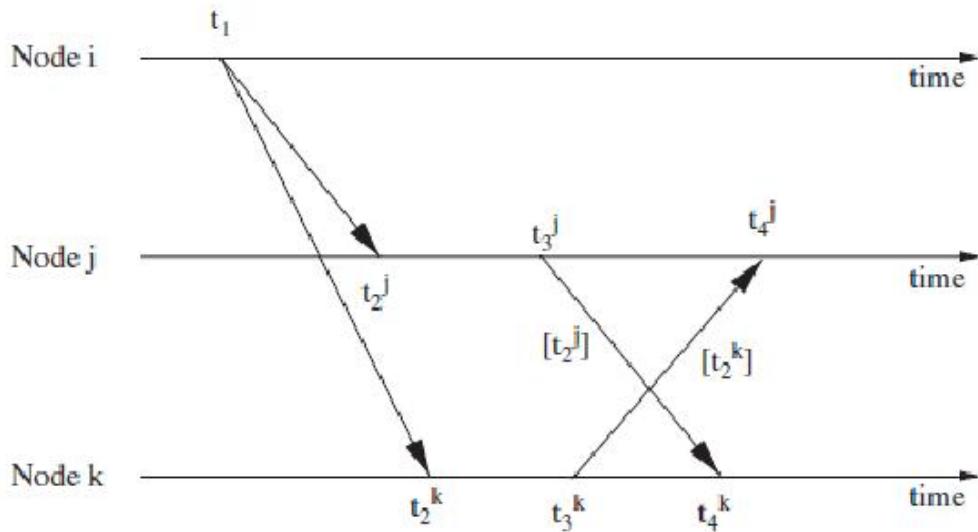


Figura 3.7. Schema di sincronizzazione ricevitore-ricevitore.

- ritardo di accesso, che indica il tempo impiegato dal trasmettitore per accedere al canale fisico e dipende principalmente dal protocollo MAC (medium access control) utilizzato. I protocolli basati sulla contesa come il CSMA/CA devono aspettare che il canale sia libero prima di poter accedere al canale stesso. Quando più dispositivi accedono al canale nello stesso istante, avviene una collisione che causa ulteriori ritardi (dovuti ad esempio al meccanismo di backoff esponenziale utilizzato in molti protocolli MAC);
- ritardo di propagazione, che indica il tempo necessario affinché il messaggio venga propagato al ricevitore. Quando i nodi condividono il mezzo fisico, i ritardi di propagazione sono molto brevi e, solitamente, possono essere trascurati;
- ritardo di ricezione, che è il tempo impiegato dal ricevitore per ottenere il messaggio dal mezzo fisico, elaborarlo e notificare all'applicazione la ricezione. La notifica solitamente avviene tramite interrupt, allo scattare del quale è possibile leggere il valore del clock. Solitamente il ritardo di ricezione è più breve di quello di trasmissione.

Molti schemi di sincronizzazione per le reti di sensori wireless applicano tecniche volte a ridurre la durata o la variazione di alcune di queste componenti, ad esempio se si etichetta il messaggio a livello MAC, il ritardo di trasmissione e di ricezione possono essere ridotti.

### 3.1.6 Protocolli di sincronizzazione

Un meccanismo di sincronizzazione molto semplice prevede l'utilizzo di un riferimento temporale globale, come ad esempio quello ottenuto da un ricevitore GPS

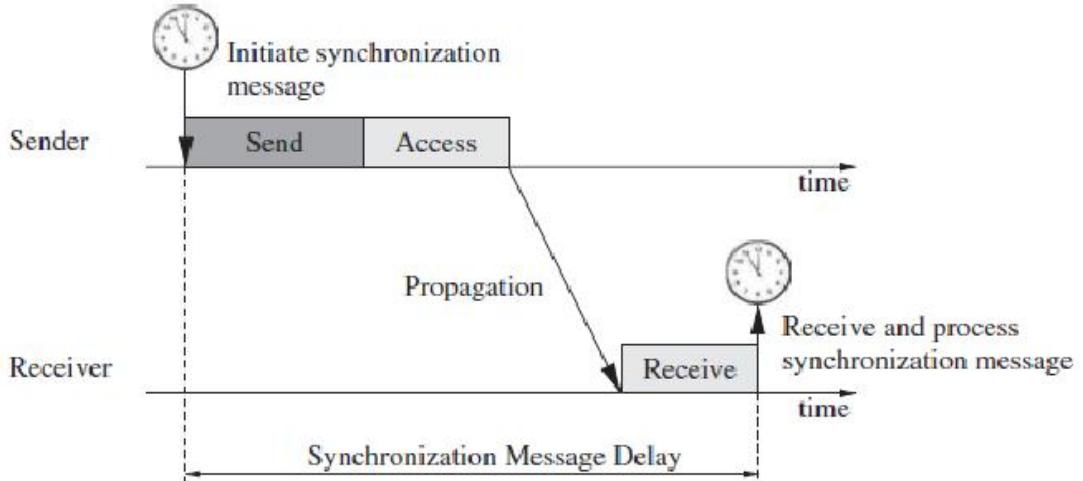


Figura 3.8. Ritardo subito da un messaggio di sincronizzazione.

(Global Positioning System) che invia continuamente messaggi contenenti una marca temporale. I ricevitori GPS, compresi quelli a basso costo, hanno solitamente un'accuratezza superiore a  $200\text{ns}$ . Tuttavia, questi approcci non sono sempre utilizzabili nelle reti di sensori: ad esempio, il segnale GPS non viene ricevuto ovunque (in acqua, negli ambienti interni, in aree coperte da fronde, ecc) e, inoltre, richiede un livello di segnale ricevuto relativamente alto, che potrebbe non essere disponibile per sensori a basso costo o molto piccoli. Se la rete è organizzata in modo gerarchico, i centri dei cluster potrebbero essere sensori con capacità maggiori e che, quindi, potrebbero essere dotati di ricevitori GPS. In tal caso, tali nodi possono essere usati come riferimenti per la sincronizzazione dei clock dei rimanenti sensori.

Il protocollo Lightweight Tree-based Synchronization (LTS) [41] è in grado di garantire una determinata precisione (anziché la massima possibile) con un overhead minimo. Tale protocollo può essere utilizzato con algoritmi multi-hop centralizzati o distribuiti per la sincronizzazione. La Figura 3.9 rappresenta lo scambio di messaggi fra una coppia di nodi utilizzando LTS: il nodo  $j$  invia a  $k$  un messaggio contenente la marca temporale che indica il tempo di trasmissione ( $t_1$ ). Al ricevere di tale messaggio, il nodo  $k$  invia al nodo  $j$  un messaggio contenente la marca temporale dell'istante di ricezione del pacchetto ( $t_2$ ), la marca temporale dell'istante di invio ( $t_3$ ) e  $t_1$ . Questo messaggio viene ricevuto da  $j$  nell'istante  $t_4$ . Si noti che i tempi  $t_1$  e  $t_4$  sono rilevati con il clock del nodo  $j$ , mentre  $t_2$  e  $t_3$  con quello di  $k$ . Assumendo un ritardo di trasmissione pari a  $D$ , uguale in entrambe le direzioni, e un offset fra i clock dei due nodi ignoto, il valore di  $t_2$  è pari a  $t_1 + D + \text{offset}$ . Analogamente,  $t_4$  è pari a  $t_3 + D - \text{offset}$ . L'offset può essere, quindi calcolato come:

$$\text{offset} = \frac{t_2 - t_4 - t_1 + t_3}{2}$$

La versione multihop centralizzata di LTS si basa sull'utilizzo di un unico nodo di riferimento, che è la radice dello spanning tree che include tutti i nodi della rete. Per massimizzare l'accuratezza della sincronizzazione, è necessario minimizzare la profondità di tale albero, in quanto gli errori che risultano dalla sincronizzazione a coppie sono additivi e quindi aumentano con il numero di hop. Il protocollo

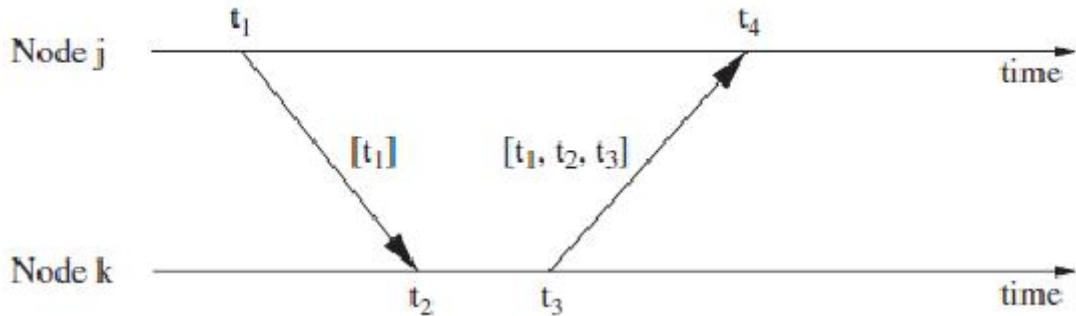


Figura 3.9. Sincronizzazione di una coppia tramite LTS.

LTS prevede che un algoritmo di costruzione dell’albero sia eseguito ogni volta che è necessaria la sincronizzazione. Terminata tale fase, il nodo riferimento inizia il processo di sincronizzazione eseguendo aggiornamenti di coppie con ciascuno dei propri figli. Una volta che un nodo è sincronizzato, può a sua volta eseguire tale processo con i propri figli. Tutto questo continua fino a quando tutti i nodi della rete non sono stati sincronizzati. La sincronizzazione di coppia ha un overhead fisso di 3 messaggi, quindi se un albero ha  $n$  archi, il numero di messaggi di overhead sarà  $3n - 3$ .

La versione multihop distribuita non richiede la costruzione dello spanning tree e la responsabilità del processo di sincronizzazione viene data ai singoli nodi anziché al nodo di riferimento. In questa versione, si assume la presenza di uno o più nodi di riferimento che sono contattati dai sensori ogni volta che uno di questi necessita di sincronizzazione. Tale approccio permette a ciascun nodo di definire il proprio periodo di risincronizzazione, ad esempio basandosi sull’accuratezza di cui ha bisogno, la distanza (in termini di numero di hop) dal nodo di riferimento più vicino, dalla deriva dei clock e dal tempo trascorso dall’ultima sincronizzazione. Per eliminare potenziali inefficienze, la versione distribuita di LTS tenta di eliminare richieste duplicate: un nodo può interrogare i propri vicini per sapere se hanno richieste di sincronizzazione pendenti e, in caso affermativo, il nodo si sincronizza con uno di essi (a distanza quindi di 1 hop) anziché con il nodo di riferimento.

Il protocollo Timing-sync Protocol for Sensor Networks (TPSN) [14] è basato sulla sincronizzazione trasmettitore-ricevitore che utilizza un albero per organizzare la rete. Questo approccio è costituito da due fasi: la fase di level discovery (eseguita durante il dispiegamento della rete) e la fase di sincronizzazione.

L’obiettivo della fase di level discovery è la costruzione di una topologia gerarchica della rete, dove a ciascun nodo viene assegnato un livello (il nodo radice, ad esempio quello dotato di ricevitore GPS, risiede a livello 0). La radice della gerarchia inizia tale fase inviando in broadcast un messaggio di level\_discovery che contiene il livello e l’identificativo univoco del trasmettitore. I nodi, che si trovano nelle vicinanze della radice e che ricevono tale messaggio, lo utilizzano per definire il proprio livello (livello 1) e inviano a loro volta un messaggio di level\_discovery. Se un nodo riceve più messaggi e ha già stabilito il proprio livello, li scarta. Vi sono situazioni in cui un nodo non sa quale sia il proprio livello ad esempio perché a causa di collisioni non ha

ricevuto un messaggio di level\_discovery o perché il nodo si è aggregato ad una rete che ha già completato la fase di discovery. In questo caso, il nodo può trasmettere una richiesta di level\_request ai propri vicini, che a loro volta rispondono inviando il proprio livello. Il nodo, quindi, si assegna un livello pari al valore minore dei livelli dei propri vicini, incrementato di un'unità. La caduta di un nodo può essere gestita in modo analogo: quando un nodo di livello  $i$  rileva di non avere vicini di livello  $i - 1$  (nello scambio di messaggi della fase di sincronizzazione), invia un messaggio di level\_request per potersi reinserire nella gerarchia. Se avviene la caduta del nodo radice, i nodi di livello 1 eseguono un algoritmo di selezione del leader. Al termine di tale fase, il nodo scelto darà inizio a una nuova fase di level discovery per la creazione di una nuova gerarchia.

Durante la fase di sincronizzazione, il protocollo TPSN utilizza l'allineamento a coppie lungo gli archi della struttura gerarchica stabilita nella fase precedente, ovvero ogni nodo di livello  $i$  sincronizza il proprio orologio con i nodi di livello  $i - 1$ . Questa sincronizzazione a coppie risulta sostanzialmente identica a quella dell'approccio LTS, fatto salvo che i messaggi scambiati riportano anche il livello del nodo mittente. La fase di sincronizzazione viene iniziata dal nodo radice che invia una richiesta apposita. Dopo un intervallo casuale, i nodi a livello 1 iniziano il proprio scambio di messaggi bidirezionale con il nodo radice. Questo processo viene poi ripetuto per tutti i livelli successivi. Come nel caso di LTS, l'errore di sincronizzazione di TPSN dipende dalla profondità della struttura gerarchica e dai ritardi end-to-end subiti dai singoli messaggi durante la sincronizzazione a coppie. Al fine di ridurre tale latenze e limitare l'errore, TPSN assume che i pacchetti ricevuti incorporino una marca temporale fornita a livello MAC.

Il protocollo Flooding Time Synchronization Protocol (FTSP) [31] mira ad ottenere una sincronizzazione a livello rete con errori dell'ordine del microsecondo, possibilità di scalare a centinaia di nodi e robustezza ai cambiamenti nella topologia di rete dovuti al malfunzionamento di nodi o di collegamenti. Esso differisce dalle precedenti soluzioni in quanto utilizza un singolo messaggio broadcast per creare dei punti di sincronizzazione tra un mittente e più destinatari, eliminando in tal modo la maggior parte delle sorgenti degli errori di sincronizzazione. A tal fine, viene affinata l'analisi dei ritardi end-to-end nell'invio e nella ricezione dei messaggi come mostrato in Figura 3.10.

Quando il buffer di trasmissione del dispositivo radio del nodo  $i$  si svuota, viene generato un interrupt al microcontrollore relativo (nell'istante  $t_1$ ) che viene servito dopo un intervallo  $d_1$ . Come prima operazione, la CPU determina la marca temporale corrente ( $t_2$ ) e provvede a riempire nuovamente tale buffer che verrà trasmesso al tempo  $t_3$ . Il messaggio arriverà al nodo  $j$  al tempo  $t_4$ , qui verrà decodificato ( $d_4$ ), eventualmente allineato a livello di byte ( $d_5$ ) e verrà generato un interrupt al tempo  $t_6$ , che sarà servito al tempo  $t_7$ , in cui verrà calcolata la marca temporale di ricezione. L'impatto dei diversi ritardi sul processo complessivo varia in modo significativo: il ritardo di propagazione ( $d_3$ ) è breve e deterministico ( $< 1\mu s$ ),  $d_2$  e  $d_4$  sono parimenti deterministic, ma la loro durata è di qualche centinaia di microsecondi. La durata della fase di allineamento dipende dal bit offset e può richiedere anche esso centinaia di microsecondi, mentre la latenza di risposta all'interrupt ( $d_1$  e  $d_6$ ) non è né deterministica né gaussiana e dura tipicamente alcuni microsecondi.

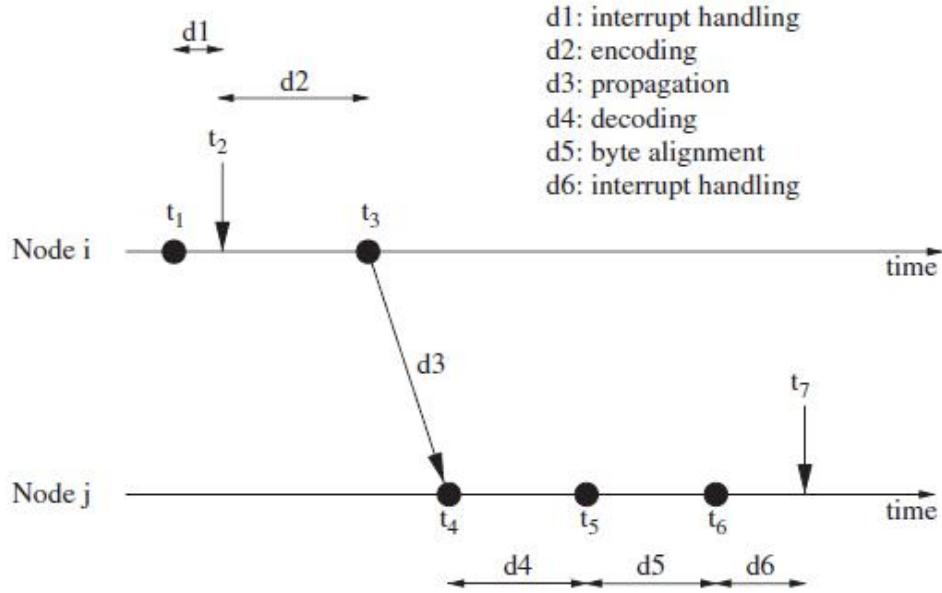


Figura 3.10. Ritardo end-to-end nei messaggi di sincronizzazione.

In questo protocollo, un mittente sincronizza uno o più ricevitori inviando un unico messaggio in broadcast, contenente la marca temporale di invio  $t_2$ . All'atto della ricezione, il destinatario estrae tale valore e determina la propria marca temporale  $t_7$ . La coppia  $t_2 - t_7$  fornisce un punto di sincronizzazione. Per ridurre il jitter introdotto dai ritardi non deterministici legati alla risposta dell'interrupt, il processo viene ripetuto più volte di seguito e viene scelto il punto di sincronizzazione che presenta la differenza minima tra i due valori (corrispondente al tempo di risposta all'interrupt più veloce).

Come in TPSN, FTSP si basa sull'elezione di un nodo che si occupi della sincronizzazione della rete (ad esempio, viene scelto il nodo con l'identificativo minore oppure il collettore della rete), che è responsabile di mantenere il tempo globale e a cui tutti i nodi della rete devono allinearsi. La sincronizzazione viene scatenata quando il nodo radice invia un determinato messaggio, contenente una marca temporale: tutti i nodi che lo ricevono direttamente si sincronizzano con la radice e, successivamente, inviano un messaggio di sincronizzazione ai loro vicini, permettendo ad essi di sincronizzarsi a loro volta. Il nodo radice deve essere unico nella rete: tutti i messaggi di sincronizzazione contengono l'identificativo del nodo radice e un numero di sequenza. Se un nodo non riceve alcun messaggio di sincronizzazione per un determinato intervallo temporale, si dichiara essere la nuova radice dell'albero. Se riceve un messaggio di sincronizzazione contenente un identificativo minore del proprio, esso smette di dichiararsi radice dell'albero. Se un nodo entra a far parte di una rete già organizzata, attende un determinato periodo di tempo per poter raccogliere un numero sufficiente di messaggi di sincronizzazione, prima di dichiararsi radice dell'albero. Tali tecniche fanno sì che TPSN possa gestire cambiamenti di topologia, includendo anche nodi mobili.

Il protocollo Reference-Broadcast Synchronization (RBS) [12] si basa sullo scambio di messaggi tra un insieme di ricevitori per la relativa sincronizzazione. Se viene utilizzata una comunicazione di tipo broadcast, un messaggio inviato da una sorgente arriva approssimativamente nello stesso istante a tutti i ricevitori. Il ritardo subito dal messaggio sarà dovuto soprattutto ai ritardi di propagazione (che sono influenzati dal fenomeno delle riflessioni multiple) e al tempo necessario al ricevitore per ottenere ed elaborare il pacchetto in arrivo. Il protocollo RBS mira a rimuovere gli errori non deterministici causati dal trasmettitore. Dato che tutti i metodi di sincronizzazione si basano su una qualche forma di scambio di messaggi, i ritardi non deterministici subiti da questi pacchetti limitano la granularità del tempo di sincronizzazione che può essere ottenuta. La Figura 3.11 confronta i percorsi critici di un protocollo di sincronizzazione tradizionale con RBS. Sfruttando la natura broadcast del mezzo, il ritardo di trasmissione e di accesso al mezzo sono uguali per entrambi i ricevitori, quindi l'istante di ricezione dipenderà solamente dalle variazioni della propagazione e dal ritardo di ricezione. Di conseguenza il percorso critico sarà molto più breve rispetto a quello di un sistema di sincronizzazione tradizionale.

Se esistono più di due ricevitori, l'errore di fase massimo di tutte le coppie viene

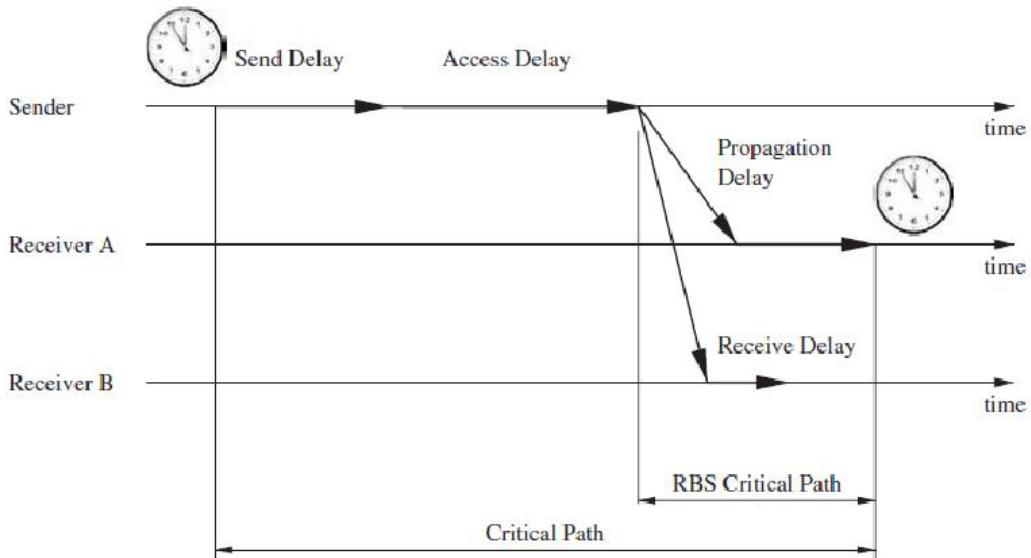


Figura 3.11. Percorso critico di un messaggio di sincronizzazione.

detto dispersione di gruppo. Aumentando il numero di ricevitori, viene incrementata la probabilità che almeno uno di essi sia sincronizzato in modo scarso, portando a valori di dispersione di gruppo maggiori. Per limitare questo fenomeno, è possibile inviare più messaggi di riferimento per aumentare la precisione di sincronizzazione. Infatti, un ricevitore  $j$  può calcolare il proprio offset rispetto a un altro ricevitore  $i$  come la media degli offset di fase di tutti gli  $m$  pacchetti ricevuti da  $i$  e  $j$ :

$$offset[i, j] = \frac{1}{m} \sum_{k=1}^m (T_{j,k} - T_{i,k})$$

Il protocollo RBS può essere esteso per scenari multi-hop prevedendo la presenza di più nodi di riferimento, ciascuno con il proprio dominio di broadcast. Questi domini possono sovrapporsi: i nodi nelle regioni di sovrapposizione si comportano da bridge e permettono la sincronizzazione tra domini differenti. Ad esempio, se i nodi  $A$  e  $B$  sono nell'area di copertura del nodo di riferimento  $C$  e i nodi  $C$  e  $D$  sono nell'area di copertura del nodo di riferimento  $E$ ,  $C$  è il collegamento fra i due domini di broadcast.

Il protocollo RBS necessita di un gran numero di messaggi di sincronizzazione ed è utilizzato nelle situazioni in cui è possibile la sincronizzazione post-facto. In tali scenari, i nodi non rimangono sincronizzati, ma si allineano solamente quando viene rilevato un determinato evento, quindi la sincronizzazione avviene solamente quando necessario, evitando il consumo di energia per la trasmissione di messaggi non necessari.

Il protocollo Time Diffusion Synchronization (TDP) [39] permette a un nodo della rete di raggiungere un tempo di equilibrio, cioè viene concordato un tempo a livello di rete e ciascun nodo si occupa di mantenere il proprio clock con una deviazione piccola e limitata rispetto a tale equilibrio. I nodi in una rete si auto-organizzano in una configurazione simile ad un albero, utilizzando due tipologie di ruoli: i nodi master e i nodi leader di diffusione. La procedura di Time Diffusion (TP) permette ai nodi master di inviare messaggi contenenti informazioni temporali ai loro vicini, alcuni dei quali diventano leader di diffusione e si occupano di inoltrare ulteriormente i messaggi del master. TDP prevede due fasi operative: durante la fase di attività i nodi master vengono eletti (con la procedura ERP - Election/Reelection Procedure) ogni  $t$  secondi in modo da bilanciare il carico sulla rete e di concordare il tempo di equilibrio. Ogni fase di attività viene seguita da una fase di inattività, in cui non avviene alcun processo di sincronizzazione. Ogni intervallo di  $t$  secondi viene a sua volta diviso in intervalli di durata  $d$ , che iniziano con l'elezione dei nodi leader di diffusione. La procedura ERP elimina i nodi foglia e i nodi i cui clock deviano da quelli dei loro vicini di un valore superiore ad una data soglia (questa situazione viene identificata grazie a uno scambio di messaggi fra nodi vicini che permette il confronto delle letture dei clock). Inoltre, tale procedura sceglie i nodi master e leader di diffusione tenendo conto dello stato energetico dei nodi.

La Figura 3.12 illustra il concetto di sincronizzazione in TDP. Un nodo master ( $A$ ) invia le informazioni temporali ai propri vicini. Tutti i nodi leader di diffusione che ricevono tale messaggio ( $C$  e  $D$ ) rispondono con un messaggio di conferma, permettendo al master di determinare il ritardo di andata e ritorno  $\Delta_j$  per ciascun nodo  $j$ , di stimare il ritardo di sola andata come  $\Delta/2$  e ottenerne la deviazione standard. La deviazione standard viene inviata in un altro messaggio etichettato con una marca temporale dal nodo master ai nodi leader. Questo processo continua per un numero di volte pari a  $n$  (in Figura 3.12 per  $n=2$ ). I nodi che ricevono messaggi di sincronizzazione da più master ( $E$ ,  $F$ ) utilizzano la deviazione standard come rapporto pesato del loro contributo all'allineamento temporale.

I protocolli Mini-sync e Tiny-sync [44] forniscono la sincronizzazione a coppie con pochi vincoli riguardanti le risorse di elaborazione, di memorizzazione e di banda.

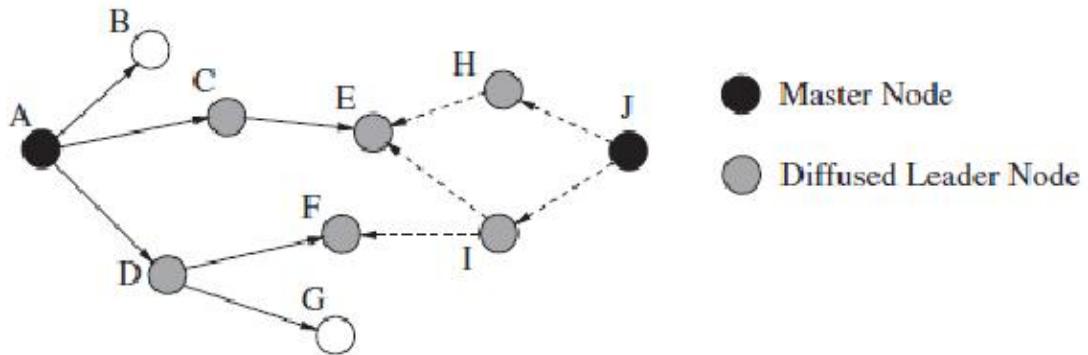


Figura 3.12. Sincronizzazione in TDP.

La relazione fra i clock di due nodi può essere espressa come  $C_i(t) = a_{ij}C_j(t) + b_{ij}$  dove  $a_{ij}$  indica la deriva relativa e  $b_{ij}$  l'offset relativo tra i clock dei nodi  $i$  e  $j$ . Per determinare questa relazione, ad esempio, il nodo  $i$  invia al nodo  $j$  un messaggio contenente una marca temporale all'istante  $t_0$  e il nodo  $j$  risponde inviando un messaggio con la marca temporale al tempo  $t_1$ . Il nodo  $i$  memorizza il tempo di arrivo del secondo messaggio ( $t_2$ ) ottenendo la tripla  $t_0, t_1, t_2$ , che viene detta data-point. Dal momento che  $t_0$  deve essere antecedente a  $t_1$  e  $t_1$  deve essere antecedente a  $t_2$ , si ottiene che:

$$\begin{aligned} t_0 &< a_{i,j}t_1 + b_{ij} \\ t_2 &< a_{i,j}t_1 + b_{ij} \end{aligned}$$

Questa procedura viene ripetuta più volte, in modo da ottenere molti data-point e nuovi vincoli sui valori ammissibili dei parametri  $a_{ij}$  e  $b_{ij}$  per aumentare la precisione dell'algoritmo. Le due versioni del protocollo si basano sull'osservazione che non tutti i data-point sono utili. Ciascun data-point genera due vincoli sulla deriva e offset relativi. L'algoritmo Tiny-sync mantiene solo quattro di questi vincoli: tutte le volte che si ottiene un nuovo data-point, i nuovi vincoli vengono confrontati con i quattro precedenti, in modo da tenere in considerazione solo quelli che portano alla stima migliore di deriva e offset. Uno svantaggio di questo protocollo è però l'eliminazione di vincoli che fornirebbero stime migliori con data-point successivi. Per risolvere tale problema, il protocollo Mini-sync scarta un data-point solamente quando i vincoli risultanti diventano inutili. Tale approccio necessita di maggiori capacità di elaborazione e di memorizzazione rispetto al protocollo Mini-sync, ma ha prestazioni migliori nella precisione del calcolo della deriva e dell'offset.

## 3.2 L'algoritmo di routing

### 3.2.1 Modelli di consegna dei dati

La trasmissione dei dati dai nodi sensori ai nodi collettori può avvenire in modalità differenti:

- trasmissione continua: ciascun sensore invia periodicamente le proprie misure ai nodi collettori;
- trasmissione dovuta a eventi: l'invio dei dati avviene quando il nodo sorgente rileva l'evento d'interesse (ad esempio, quando la temperatura supera una determinata soglia);
- trasmissione causata dalla ricezione di una richiesta: l'invio dei dati avviene a fronte della ricezione da parte del nodo sensore di una richiesta esplicita da parte del collettore;
- modello ibrido: l'invio dei dati avviene con una modalità che è una combinazione delle precedenti.

La tipologia di consegna dei dati usata influenza il protocollo di instradamento per quanto riguarda la minimizzazione del consumo di energia e la stabilità dei percorsi. Ad esempio in un'applicazione per il monitoraggio di un ambiente, dove i dati sono inviati periodicamente al collettore, l'utilizzo di un protocollo gerarchico porta ad avere il minimo consumo di energia. Infatti, i dati ridondanti possono essere aggregati lungo il percorso dai sensori al collettore, riducendo il traffico generato e l'energia consumata.

Le reti in cui la trasmissione dei dati è causata dalla ricezione di una richiesta possono essere considerate come un database distribuito, in cui uno o più nodi (collettori) inviano richieste ai nodi sensore. Le richieste possono essere suddivise in diverse categorie:

- richieste continue, da cui deriva un flusso di dati (ad esempio, “*invia la temperatura misurata per i prossimi 7 giorni, con una frequenza di 1 misura per ora*” ) o richieste singole, che richiedono una misura puntuale (ad esempio, “*la temperatura è maggiore di x?*”);
- richieste aggregate, che consistono di più richieste annidate (ad esempio, “*quali sono i valori di x, y, z?*” ) o richieste semplici (ad esempio, “*qual è il valore della variabile x?*”);
- richieste per dati replicati che possono essere evase da più nodi (ad esempio, “*Esiste almeno un nodo target nell'area x?*” ) o richieste per dati unici che possono, invece, essere soddisfatte da un solo nodo.

La trasmissione dei dati nella wsn progettata è di tipo ibrido: i nodi sono impostati per inviare a intervalli regolari le misure effettuate tramite i propri sensori ma, a seconda dei casi e delle necessità, è possibile che il sink interroghi un nodo per ottenere i dati disponibile. Per ridurre il consumo energetico dovuto alla trasmissione dei dati, è possibile utilizzare meccanismi di aggregazione, come l'eliminazione dei duplicati oppure l'utilizzo di funzioni quali, ad esempio, valore minimo, massimo o medio. L'aggregazione può essere effettuata da tutti i nodi della rete oppure da nodi selezionati, ad esempio quelli con prestazioni maggiori.

### 3.2.2 Presupposti

Come assunzioni generali si ipotizza che i nodi siano fissi, alimentati da una sorgente limitata eventualmente rinnovabile e abbiano capacità di elaborazione e memorizzazione limitata. La rete prevede la presenza di almeno un nodo (il sink) che dispone di energia illimitata e funge da punto di contatto tra la rete ed il mondo esterno. Poichè non risulta possibile definire un algoritmo generale adatto a tutti i possibili scenari di messa in campo, si sono prese in considerazioni una serie di specifiche ulteriori, tipiche del contesto particolare delle reti di sensori, sulla base delle quali è stato possibile definire l'architettura generale della rete e specificare i relativi protocolli e funzionamenti, nell'ottica di produrre un algoritmo che risulti non solo consci del livello energetico complessivamente presente all'interno della rete, ma anche efficiente, ovvero in grado di massimizzare la vita della rete sfruttando al meglio le caratteristiche peculiari della stessa. Di seguito vengono elencati tali presupposti, evidenziando il motivo che ha portato alla loro enunciazione.

- La rete gestisce un throughput effettivo molto inferiore al data rate del singolo link. Tale assunzione risulta ragionevole in quanto, nella maggior parte dei casi, la grandezza fisica oggetto della misura varia molto lentamente nel tempo e può essere sintetizzata con un semplice numero o, al più, una tupla di numeri. Questo porta a esigenze di trasmissione molto ridotte.
- È tollerata una certa latenza nella consegna (dell'ordine dei minuti). In molti casi, i dati raccolti servono per abilitare ragionamenti di ordine superiore dopo essere stati fusi con informazioni provenienti da sorgenti completamente diverse (algoritmi di simulazione, dati inseriti dall'utente, ecc): pertanto non risulta necessario disporre dei dati nel preciso momento in cui essi sono stati raccolti. Al più, per evitare deduzioni sbagliate, è necessario associare a ciascun dato la marca temporale relativa al momento della sua acquisizione.
- La rete è sostanzialmente di tipo data collection (monitoraggio di molte sorgenti distribuite nello spazio che raccolgono informazioni a seguito di eventi rilevati dai propri sensori ed eventuale controllo a banda molto bassa). Questo significa che, nella maggior parte dei casi, la comunicazione è prevalentemente di tipo many-to-one, attraverso percorsi che possono essere mantenuti con poco sforzo all'interno dei nodi stessi. Ogni tanto esiste la necessità di comunicazioni one-to-many, per la diffusione di informazioni di configurazione e, con estrema rarità, possono essere necessarie comunicazioni one-to-one tra i singoli nodi della rete. Ciò significa che è possibile progettare l'architettura generale della rete in modo da rendere energeticamente efficienti i primi due meccanismi ed accettare che, qualora occorra, il terzo meccanismo sia possibile anche se comporta un consumo energetico elevato.
- Esiste almeno un nodo principale (sink) che ha il compito di raccogliere tutti i dati e inviarli da qualche parte; non ha problemi energetici (può essere sempre in ascolto); costituisce la fonte temporale di riferimento. Nel caso in cui siano presenti più nodi sink, essi vengono a creare un partizionamento implicito della rete in più sottogruppi (ciascuno dei quali coordinato dal proprio sink)

che permette di ridurre ulteriormente i cammini presenti nella rete e costituisce una fonte implicita di ridondanza per evitare single-point-of-failure.

- Tutti gli altri nodi sono semplici, hanno risorse energetiche limitate, variabili nel tempo e osservabili (o stimabili); tali nodi svolgono due funzioni:
  - rilevano, attraverso propri sensori, uno o più dati da trasmettere;
  - inoltrano le informazioni ricevute da nodi più lontani verso il sink stesso, se raggiungibile, o verso nodi a loro volta più vicini al sink, utilizzando come metrica per valutare la distanza anche la disponibilità energetica residua di tali nodi.

La rete che si viene a formare deve essere auto-configurante e resiliente: questo richiede che la topologia della rete consenta un certo numero di percorsi multipli. Il risparmio energetico viene ottenuto mantenendo spenta la parte di ricezione il maggior tempo possibile ed utilizzando una suddivisione ciclica del tempo in periodi.

### 3.2.3 Descrizione dell'algoritmo

Ogni nodo ha un identificatore univoco e numera i propri pacchetti in modo consecutivo modulo  $N$ .

Il nodo sink, periodicamente (ogni  $T_{ciclo}$ ), invia un pacchetto di beacon nel quale comunica le seguenti informazioni:

- il proprio identificatore;
- numero di hop necessari per raggiungere il sink stesso (0 in questo caso);
- riserva energetica minima lungo il cammino per raggiungere il sink (infinito in questo caso);
- la finestra temporale di ascolto per messaggi entranti e il proprio MAC con cui può essere indirizzato dagli altri nodi (0 è l'indirizzo MAC riservato al sink);
- un eventuale payload per la diffusione dei dati nella rete. Tale payload può contenere messaggi destinati a tutti i nodi, messaggi destinati ad un singolo nodo, ricevuti di ritorno per pacchetti precedentemente trasmessi (ACK, ovvero Acknowledge), ulteriori informazioni utili per gli altri nodi (RSSI, ecc).

L'invio di dati e beacon avviene durante una finestra temporale (che si ripete ogni  $T_{ciclo}$ ): questa finestra è divisa in  $N_{sensori}$  slot, ognuno dei quali è assegnato a un preciso nodo. La finestra ha una durata pari  $T_{window}$ , perciò ogni nodo ha a propria disposizione  $\frac{1}{N_{sensori}} \cdot T_{window}$ , chiamato  $T_{slot}$ .

Quando un nodo semplice si accende, esso si trova in uno stato di panico poichè non conosce nessun altro nodo: inizia perciò una fase di apprendimento, in attesa di ricevere un beacon da qualcuno (che potrebbe non essere il sink se si trova distante da esso). La fase di apprendimento dura un periodo intero ( $T_{ciclo}$ ), durante tale

periodo vengono acquisiti tutti i beacon e si sceglie il più conveniente: quello che presenta il minor numero di hop per raggiungere il sink e, in caso di parità, la minore energia residua totale del cammino. Il primo parametro preso in considerazione è la distanza dal sink perché coinvolgere meno nodi possibili nell'invio di un dato comporta un consumo energetico minore a livello generale della rete.

Da questo punto in avanti, il nodo è in grado di inviare dati alla rete se riesce a mantenersi in sincronia con il nodo scelto e cerca di comunicare con esso durante la propria finestra temporale.

Al fine di consentire alla rete di estendersi per più hop, quando un nodo ha raggiunto lo stato precedente, può cominciare a trasmettere, con la stessa periodicità del proprio predecessore (e con uno scostamento temporale che dipende dalla propria finestra), un proprio beacon, che ha la stessa struttura del pacchetto beacon del sink, ma dati differenti.

In tale pacchetto il numero di hop viene incrementato di un unità rispetto al beacon scelto e la riserva energetica viene posta al valore minimo tra la propria energia residua e quella annunciata dal predecessore.

Il nodo entra a questo punto nello stato di instradamento e deve preoccuparsi di ricevere e inviare pacchetti. Un timer, con periodicità pari al periodo della rete ( $T_{ciclo}$ ), indica al nodo le operazioni fondamentali che deve compiere: all'inizio della finestra temporale di ricezione verrà accesa la radio per ricevere eventuali pacchetti, ed essi verranno memorizzati nel nodo. Al termine della finestra temporale, la radio viene spenta per risparmiare energia.

All'interno di questa finestra temporale sono presenti  $N_{sensori}$  slot, uno per ogni sensore: ogni nodo deve stare attento all'arrivo del proprio slot di trasmissione. All'arrivo del proprio slot, il nodo deve innanzitutto inviare un nuovo beacon (ogni  $N_{beacon} \cdot T_{ciclo}$ , con le relative informazioni aggiornate, a beneficio dei propri figli. Inoltre, se allo scattare di questo istante sono presenti messaggi da inoltrare, vengono trasmessi anche questi utilizzando sempre un meccanismo di tipo CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance).

Un nodo deve anche preoccuparsi della ricezione, da parte del nodo predecessore, del proprio beacon. Per mantenere la sincronizzazione e reagire all'evoluzione energetica lungo il cammino, occorre ricevere tale messaggio ed aggiornare i propri parametri: l'energia residua che dovrà essere annunciata deve essere ricalcolata. Se l'energia del nodo stesso è troppo bassa, esso smetterà di annunciare i propri beacon e si limiterà a far parte della rete per inviare i propri dati. Se, viceversa, l'energia del cammino precedente è inferiore ad una data soglia, il nodo re-inizierà una fase di apprendimento, così da cercare un predecessore differente. Durante tale fase, può continuare a ricevere pacchetti entranti, ma non li trasmetterà: essi saranno accodati in un buffer in attesa di trasmetterli appena sarà ripristinata la connettività di rete. Al termine della fase, se è stato individuato un nodo con metrica migliore, questo sarà scelto come nuovo predecessore, rifasando i timer sulle sue informazioni, altrimenti il nodo dovrà smettere di annunciare il proprio beacon, al fine di evitare di prendere in carico dati ulteriori che esso non potrà più consegnare. In questo caso, sarà attivato un algoritmo di back-off che prevede attese progressivamente più lunghe fino ad un valore massimo ragionevole: se tutte quante dovessero scadere senza che l'energia residua lungo il cammino torni sopra la soglia, il nodo si riporterà nella

fase di apprendimento.

Inoltre, alla ricezione del beacon devono essere ricalcolati gli offset dei diversi timer per mantenere la sincronizzazione con la fonte principale (il sink).

Se non viene ricevuto il beacon atteso, e questo si ripete per due o tre cicli successivi, il nodo sceglierà qualcun’altro come suo next hop (anche se questo comporta il peggioramento della strada per raggiungere il sink). Se non è possibile scegliere qualcun altro come next hop si rientra nella fase di apprendimento (panico), inviando nel proprio beacon infinito ( $N_{sensori} + 1$ ) come distanza dal sink in termini di numero di hop. Così i nodi a valle sanno che a monte si è persa la connettività verso il sink, e possono iniziare a cercare un percorso alternativo.

In caso di corretta ricezione del beacon, sarà valutato anche il contenuto del suo payload. Se esso contiene dati di configurazione (comunicazioni di tipo many-to-one) essi saranno elaborati opportunamente ed inclusi nel payload del proprio beacon da trasmettersi al momento opportuno. Poiché la comunicazione dei dati di configurazione è probabilistica, il nodo sink può ripetere tale messaggio più volte consecutive (in cicli differenti) ed è possibile chiedere ai singoli nodi di dare conferma (attraverso un pacchetto di tipo dati) dell’avvenuta ricezione. Tali comportamenti, tuttavia, sono implementati a livello applicativo: il protocollo non fornisce supporto diretto per tale eventualità.

Anche la comunicazione one-to-one all’interno della rete deve essere supportata a livello applicativo: se un nodo della rete deve comunicare qualcosa ad un altro nodo della rete esso invia un pacchetto di tipo dati al sink che provvede, sulla base delle informazioni contenute al suo interno, a ritrasmetterlo all’interno del payload del proprio beacon. Sebbene tale comportamento sia fortemente inefficiente dal punto di vista energetico, i casi d’uso normali in generale non lo prevedono e rendono tale eventualità molto remota.

## Stato dei nodi

Un nodo quando viene acceso per la prima volta si trova in una situazione in cui non conosce nessuno. La Figura 3.13 mostra il comportamento di un nodo alla prima accensione.

Il nodo entra in una fase di discovery, in cui tiene accesa la sua antenna finché non riceve un beacon, dal sink o da un altro nodo. Questa fase non dura per sempre, poiché il nodo potrebbe trovarsi in una zona in cui non riceve il segnale di nessun altro nodo (eventualità poco probabile perché indicherebbe una disposizione dei nodi poco intelligente), perciò sono alternate delle fasi in cui l’antenna è costantemente accesa a fasi in cui l’antenna è spenta per non consumare inutilmente energia, questo finché non riceve un beacon e si sincronizza con il nodo mittente, settandolo come next hop (il primo vicino da cui arriva un beacon viene scelto come next hop, in seguito quando si ricevono altri beacon si sceglierà il percorso migliore). A questo punto deve attendere che arrivi  $T_{ciclo}$ , ovvero l’istante in cui inizia la finestra di trasmissione.

Bisogna ricordare che a ogni nodo viene assegnato un indirizzo con cui può essere identificato in maniera univoca e uno slot in cui può trasmettere: il nodo quando

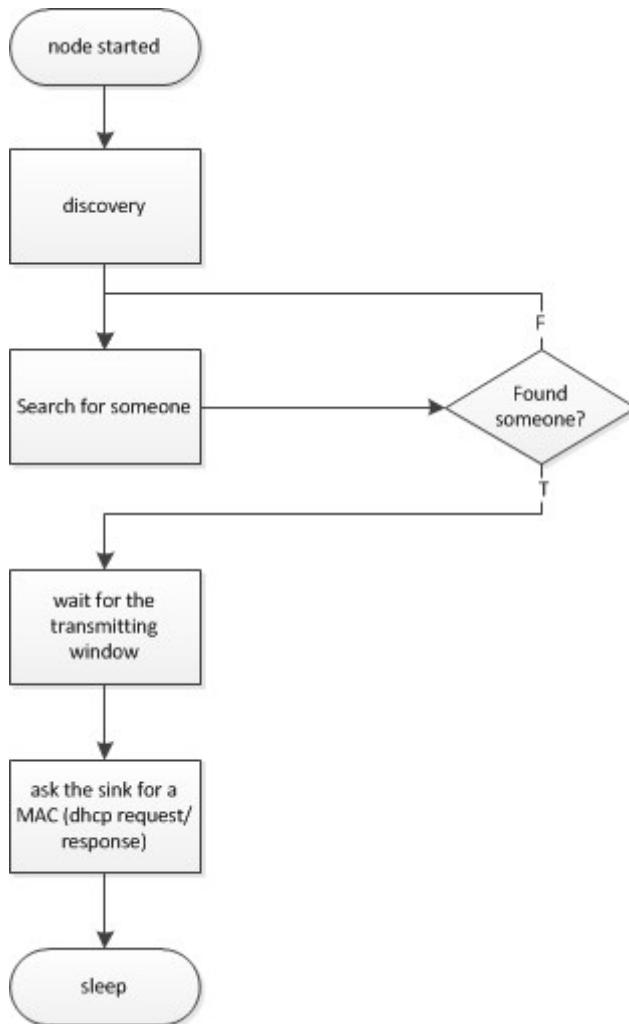


Figura 3.13. Stato di un nodo alla sua accensione.

arriva in questa fase tuttavia non ha ancora uno slot a se riservato e non possiede neanche un indirizzo (MAC): deve mandare un pacchetto particolare in cui chiede al sink che gli vengano assegnati entrambi. Questo pacchetto, che possiamo chiamare *dhcp request*, viene inviato al sink in un’istante di tempo casuale nello slot riservato al sink (ovvero lo slot 0). Si è scelto di inviare la *dhcp request* in questo slot poichè il sink tipicamente manda solo il proprio beacon e non altri pacchetti, quindi la probabilità di collisione è minore.

Il sink risponderà con una *dhcp response* in cui comunica al nodo il proprio indirizzo e lo slot assegnatogli. Il sink parte ad assegnare gli slot dall’ultimo disponibile in ordine descrescente, per motivi che saranno chiariti in seguito. Lo slot in cui trasmettere coincide con l’indirizzo che viene assegnato dal sink, per usare un identificativo in meno, perciò se al nodo  $N$  viene assegnato MAC 0x01 trasmetterà nel primo slot, se ha ricevuto 0x07 come indirizzo MAC il suo slot sarà il settimo e così via. In seguito viene analizzato più nel dettaglio questo scambio di messaggi.

A questo punto il nodo ha terminato la fase di setup, può quindi entrare in uno stato di sleep per poi riattivarsi e iniziare il suo ciclo di attività periodico.

La struttura temporale ciclica dello stato dei nodi quando la rete è a regime, ovvero esclusa la fase di setup iniziale, è descritta nella Figura 3.14: vengono illustrate le operazioni svolte da un nodo dal wake up a quando torna nuovamente nella fase di sleep.

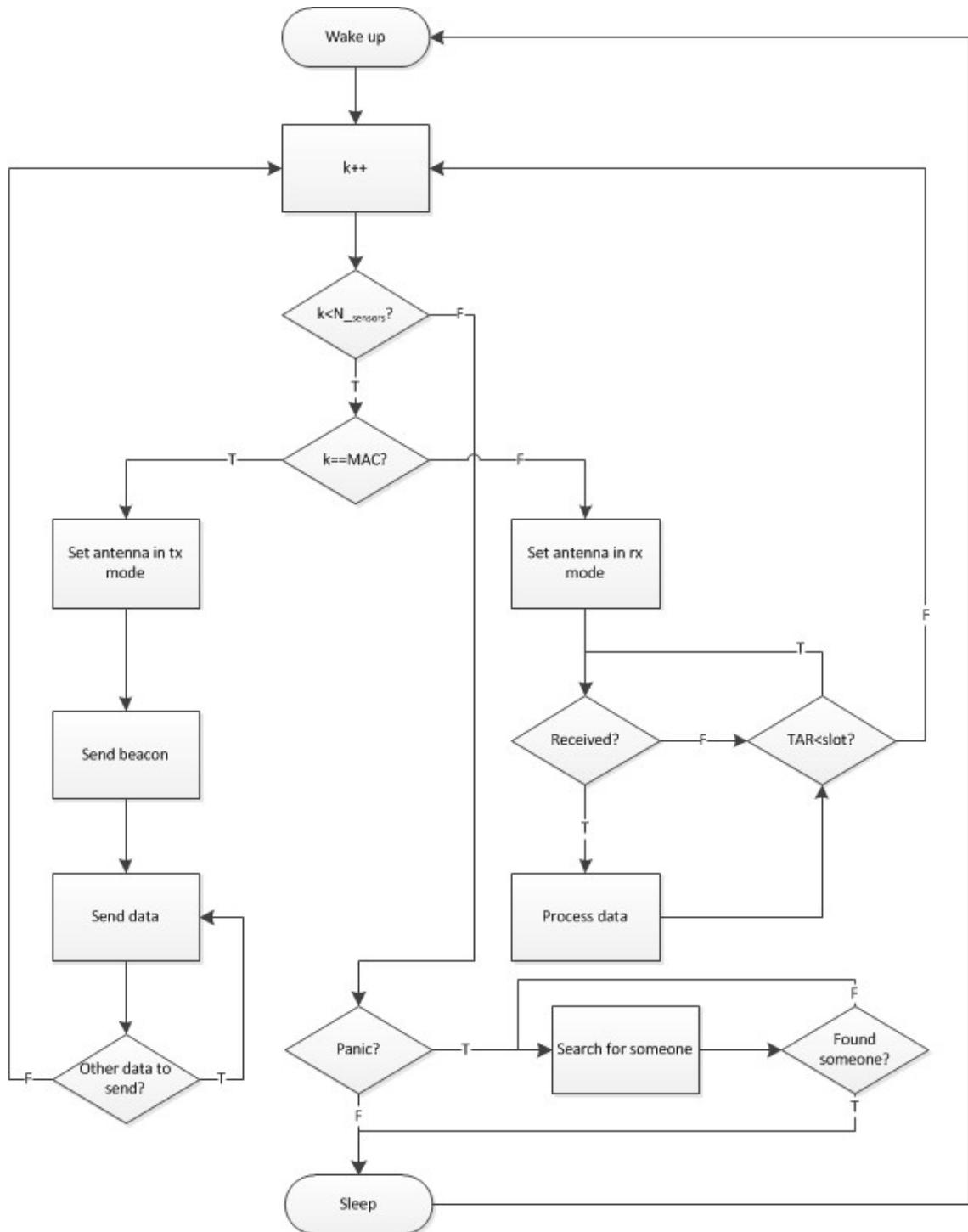


Figura 3.14. Stato di un nodo quando la rete è a regime.

Analizziamo brevemente il grafo illustrato per scendere maggiormente nel dettaglio delle attività svolte da un nodo.

La fase di wake-up avviene allo scattare del timer che riattiva il nodo dalla fase di sleep (fase a basso consumo energetico in cui il microcontrollore è in deep sleep mode e l'antenna radio è disattivata).

Come spiegato in precedenza, ogni nodo ha a sua disposizione uno slot fisso in cui solo lui trasmette, questi  $N_{sensori}$  slot vengono contati tramite  $k$ .

Quando  $k$  corrisponde al proprio MAC, il nodo entra nella fase di trasmissione e invia il proprio beacon e tutti i dati che ha nella propria coda (ovvero misure rilevate dal suo sensore o dati ricevuti da altri nodi che lo considerano come next hop e che quindi vanno instradati verso il sink), mentre in tutti gli altri  $N_{sensori} - 1$  casi in cui il valore di  $k$  non indica il proprio slot, essendo differente dal MAC, resta in ascolto di eventuali pacchetti.

Al termine di tutti gli  $N_{sensori}$  slot si possono verificare due casi: il generico nodo  $N$  ha ricevuto almeno un beacon, quindi conosce i propri vicini: è stato perciò in grado di scegliere un next hop opportuno (come spiegato in precedenza); il nodo torna nello stato a basso consumo energetico (modalità di sleep) in cui resterà fino a essere riattivato dal timer e iniziare nuovamente il ciclo. Oppure,  $N$  non sa quali altri nodi siano presenti nel suo vicinato. Questo secondo stato è quello in cui si trova un nodo alla sua accensione iniziale dopo la fase di wake-up, chiamato stato di panico. Un nodo altrimenti può essere nella fase di panico perché ha perso la connettività con il suo next hop, cioè non ha ricevuto un beacon per  $N_{beacon} \cdot T_{ciclo}$ . Quando un nodo si trova nello stato di panico, non può raggiungere il sink, perciò è necessario che trovi un next hop nel minor tempo possibile. Per questo motivo tiene accesa la sua antenna finché non riceve un beacon, dal sink stesso o da un altro nodo. Tuttavia, potrebbe succedere che il nodo è rimasto isolato e non è in grado di ricevere alcun pacchetto da nessun altro nodo della rete: per evitare di consumare inutilmente la propria carica residua nell'attesa di un beacon che non arriverà mai, si alternano una fase in cui l'antenna è accesa nella speranza di captare un nodo nelle vicinanze a una fase in cui l'antenna viene spenta. La prima fase è ovviamente più lunga dell'intervallo di tempo che trascorre tra l'invio di un beacon e il successivo. La seconda fase invece, con il trascorrere del tempo, tende a essere predominante rispetto alla prima.

Quando un nodo, durante la fase di panico, riceve un beacon esce dallo stato di panico, setta quel vicino come next hop, sincronizza il proprio timer con quello del nodo da cui ha appena ricevuto il pacchetto e si pone in sleep.

Il nodo sarà riattivato dal timer trovandosi nella fase di wake-up: questa volta non essendo più in uno stato di panico può entrare tranquillamente nella fase successiva, in cui può inviare e ricevere dati.

### Assegnazione di uno slot a un nodo

Ogni nodo ha bisogno che gli venga assegnato un indirizzo MAC dal sink. Il MAC differisce dall'identificativo poiché quest'ultimo è settato a mano dallo sviluppatore in ogni nodo (può essere una qualsiasi stringa) e viene utilizzato solo quando non si dispone ancora di un indirizzo MAC. Deve perciò essere scelto in modo univoco.

Quando un nodo invia un pacchetto di tipo *dhcp response* il sink capisce che il nodo non ha ancora un indirizzo MAC e perciò uno slot in cui trasmettere, ricordiamo infatti che lo slot riservato a ogni nodo corrisponde numericamente all'indirizzo MAC ricevuto: se al nodo  $N$  viene assegnato il MAC  $0x09$  esso potrà trasmettere nel nono slot, partendo a contare dallo slot 0 che è riservato per il sink stesso.

La *dhcp response* ovviamente non è detto che sia inviata direttamente al sink, poichè un nodo potrebbe non riuscire a raggiungerlo direttamente, in caso negativo viene inviata al proprio next hop che provvederà a effettuarne il forward verso il nodo centrale.

Quando il sink riceve una *dhcp response* ed elabora una risposta, assegna al nodo  $N$  l'indirizzo MAC più alto disponibile (partendo da  $N_{sensori}$ ). I MAC vengono assegnati in ordine decrescente affinchè i pacchetti inviati da un nodo foglia verso il sink abbiano la latenza minore possibile. Se il nodo foglia ha un MAC più basso dei nodi che i suoi pacchetti devono attraversare per arrivare al sink, il suo slot sarà prima in ordine temporale di quello dei suoi next hop. Perciò, il pacchetto del nodo foglia sarà forwardato nella stessa finestra in cui è stato inviato, senza bisogno di aspettare cicli di trasmissione successivi.

Dalla Figura 3.15 in cui è illustrata una rete con 32 nodi disposti linearmente, si può capire facilmente questo ragionamento. Il nodo foglia ( $N_1$ ) ha indirizzo MAC uguale a 1, mentre il suo next hop ha indirizzo 2, il nodo seguente ha indirizzo 3 e così via fino al nodo più vicino al sink che ha indirizzo 32. Perciò, quando tutti i nodi saranno in ascolto, pronti per ricevere dati,  $N_1$  sarà il primo nodo a inviare i propri pacchetti, avendo a disposizione lo slot numero uno per trasmettere: i dati saranno ricevuti da  $N_2$ , il quale ne effettuerà il forward nello slot successivo; il nodo seguente  $N_3$  riceverà i dati inviati da  $N_2$  nel secondo slot e li invierà nel suo slot (il numero tre) e così via fino al sink. In un'unica finestra temporale (composta da 32 slot) i dati sono quindi arrivati dal nodo foglia al nodo centrale.

Se invece l'assegnazione dei MAC fosse casuale, ci potrebbero volere fino a  $N$  finestre per permettere che i dati del nodo foglia arrivino fino al sink, dove  $N$  è il numero dei nodi che i dati devono attraversare, nell'esempio illustrato in Figura 5.10 32. Ciò porterebbe a un'elevata latenza della rete, dato che il sink conoscerebbe un valore rilevato all'istante di tempo  $t$  dal nodo  $N$  solo all'istante  $t + 32 T_{ciclo}$ .

Come si può osservare in Figura 3.16 la *dhcp request* inviata da un nodo passa per i vari next hop e arriva al nodo centrale senza accorgimenti particolari.

Il pacchetto di risposta, invece, non può essere inviato dal sink in unicast al nodo che ha effettuato la richiesta poichè nessun nodo della rete conosce la topologia della rete stessa e quindi la disposizione dei nodi all'interno di essa. Questo è un problema tipico di tutti gli algoritmi basati su distance vector, che si differenziano dagli algoritmi di tipo link state, nei quali tutti hanno a disposizione una mappa completa della rete. Per superare questa limitazione si potrebbero memorizzare in ogni nodo i nodi traversati, ovvero utilizzare una modalità stateful. Mantenere lo stato di una connessione porterebbe una serie di complicazioni innanzitutto nei nodi stessi, i quali dovrebbero tenere traccia, per tutti i pacchetti che inoltrano, di una terna di informazioni (*packet\_id, sender, receiver*) per identificare il percorso che sta facendo ciascun pacchetto che passa tramite loro. Inoltre bisognerebbe preoccuparsi dell'eventuale spostamento o interruzione improvvisa (guasto, intervento esterno,

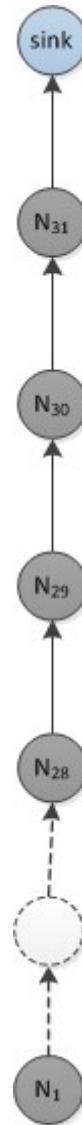


Figura 3.15. Esempio di assegnazione dei MAC ai nodi in una disposizione lineare in una rete di 32 sensori.

ecc) di un nodo poichè il pacchetto di ritorno non sarebbe più in grado di tornare al mittente se venisse a mancare un nodo intermedio.

Il pacchetto di risposta segue quindi il percorso illustrato in Figura 3.17. Si può notare che solamente il sink memorizza le informazioni sui pacchetti ricevuti, in modo da poter fare un pruning sui nodi direttamente collegati ad esso e non inviare la risposta in broadcast. Gli altri nodi non sono in possesso di alcuna informazione, perciò inoltrano il pacchetto in broadcast. Notare che un nodo accetta un pacchetto e lo inoltra a sua volta solo se proviene dal suo next hop. Perciò, quando  $N_{27}$  invia la *dhcp response* in broadcast, il pacchetto sarà processato solo da  $N_{21}$ : viene scartato da  $N_{28}$  e  $N_{22}$ .

Questo esempio, illustrato nel caso particolare di una *dhcp response*, può essere generalizzato per ogni pacchetto dal sink verso qualsiasi nodo.

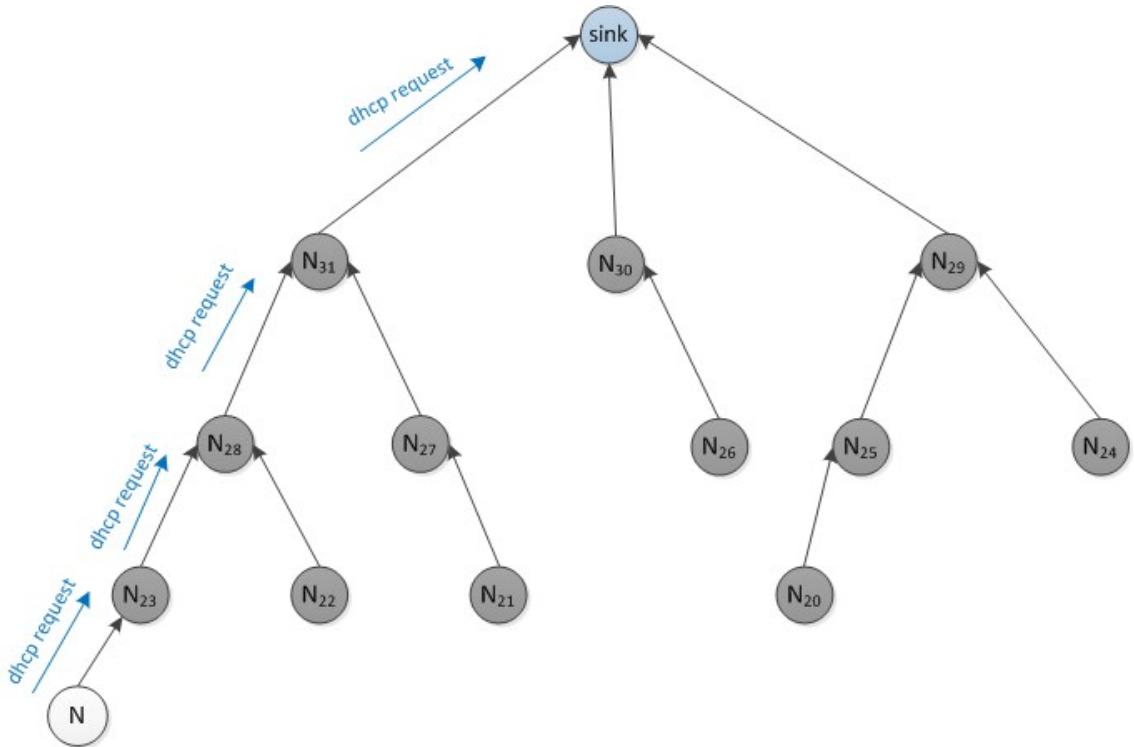


Figura 3.16. Sequenza di invio del pacchetto dhcp request da un nodo foglia al sink.

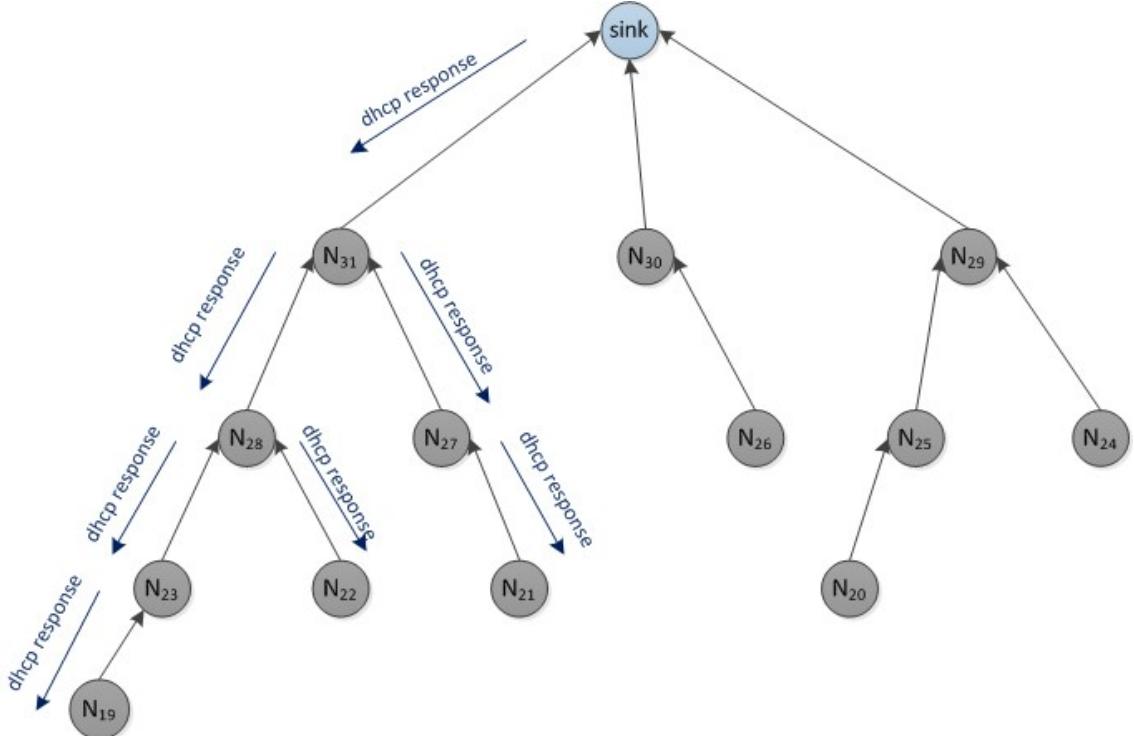


Figura 3.17. Sequenza di invio del pacchetto dhcp response dal sink a un nodo foglia.

## I pacchetti

Entriamo nel dettaglio delle informazioni contenute nei vari pacchetti che attraversano la rete.

I pacchetti sono di quattro tipi: *dhcp request*, *dhcp response*, *beacon*, *dati*. I primi due sono quelli utilizzati nella fase di setup all'accensione della rete, il beacon è senz'altro il pacchetto più importante perchè permette di scoprire i nodi presenti e di sincronizzarsi con essi. L'ultimo pacchetto è quello che permette ai nodi di scambiarsi le misure rilevate tramite i propri sensori.

I campi contenuti nei pacchetti sono i seguenti:

*dhcp\_request* [0] → *packet\_length*;

*dhcp\_request* [1] → *node\_id*;

*dhcp\_request* [2] → *packet\_type*;

*dhcp\_request* [3] → *next\_hop*;

*dhcp\_response* [0] → *packet\_length*;

*dhcp\_response* [1] → *node\_id*;

*dhcp\_response* [2] → *packet\_type*;

*dhcp\_response* [3] → *next\_hop*;

*dhcp\_response* [4] → *MAC*;

*beacon* [0] → *packet\_length*;

*beacon* [1] → *sender\_MAC\_address*;

*beacon* [2] → *packet\_type*;

*beacon* [3] → *SOC(state\_of\_charge)*;

*beacon* [4] → *hop\_number*;

*beacon* [5] → *power*;

*beacon* [6.. $N_{sensori}$ ] → *acknowledgement*;

*data* [0] → *packet\_length*;

*data* [1] → *sender\_MAC\_address*;

*data* [2] → *packet\_type*;

*data* [3] → *receiver\_MAC\_address*;

*data* [4] → *packet\_id*;

*data* [5] → *next\_hop*;

*data* [6] → *data*;

*data* [7] → *packet\_id*;

Nella *dhcp\_request* vengono inviati:

- la lunghezza del pacchetto, poichè serve nella callback di ricezione al microcontrollore per sapere quanti byte leggere dall'antenna tramite spi;
- l'id del nodo (poichè non ha ancora un MAC) in modo che si possa identificare il nodo che ha effettuato la richiesta e il mittente possa riconoscere la *dhcp\_response* di risposta indirizzata a lui;
- il tipo di pacchetto, per identificare che pacchetto è stato inviato (per la *dhcp\_request* 2);

- il campo next hop, che viene aggiornato da ogni nodo intermedio nel percorso verso il sink inserendovi il proprio next hop, in modo tale che la *dhcp\_request* venga inoltrata solo dal next hop del mittente (per evitare che inutili pacchetti doppi viaggino per la rete).

La *dhcp\_response* contiene gli stessi parametri della *dhcp\_request* (il *packet\_type* vale 3) con l'aggiunta del MAC che il sink assegna al nodo che ha effettuato la richiesta (identificabile tramite l'id presente nel pacchetto).

Il *beacon* contiene:

- la lunghezza del pacchetto;
- il MAC del mittente (non più l'id, perchè quando un nodo manda un beacon è già avvenuta la fase di negoziazione di un indirizzo e quindi gli è stato assegnato un MAC);
- il tipo di pacchetto, ovvero 0;
- lo stato di carica (state of charge), che corrisponde alla carica residua minima lungo il cammino: ogni nodo prima di inviare un beacon effettua una lettura della propria carica e se è inferiore a quella attualmente presente nel beacon ricevuto dal proprio next hop la aggiorna;
- il numero di hop che separano il nodo dal sink (ogni nodo quando riceve un beacon da un vicino incrementa il numero di hop che gli è stato comunicato e, se lo sceglie come next hop, inserisce tale numero nel proprio beacon);
- la potenza con cui il mio next hop vorrebbe che gli inviassi i dati. Questo calcolo è basato sull'RSSI (Received Signal Strength Indicator) che è una misura della potenza con cui si riceve un segnale radio. Quando un nodo riceve un pacchetto dati che deve instradare, analizza l'RSSI del segnale ricevuto e in base a questo decide se il mittente al prossimo invio può usare una potenza minore o maggiore. Ogni nodo, ricevendo questa sorta di “feedback” dal proprio next hop su come vengono ricevuti i suoi pacchetti, può modulare la propria potenza di invio, abbassandola se l'RSSI ricevuto era alto, o abbassandola in caso contrario. La modulazione della potenza di invio avviene solo per i pacchetti di tipo dati, il *beacon* viene inviato sempre con la potenza massima;
- gli ack dei pacchetti ricevuti. Ogni nodo invia infatti al mittente, tramite il proprio beacon, un ack per ogni pacchetto dati ricevuto. Il next hop del nodo  $N_i$  inserirà nel suo beacon alla posizione  $6+i$  l'id dell'ultimo pacchetto ricevuto dal nodo con MAC  $i$ . Il nodo  $N_i$ , quando riceve il beacon dal suo next hop, cancella dalla sua coda tutti i pacchetti che hanno id inferiore a quello indicato nell'ack, essendo sicuro che sono stati correttamente ricevuti.

Il pacchetto *dati* ha i seguenti campi:

- la lunghezza del pacchetto;

- il MAC del mittente, in cui il primo bit viene settato a 0 se il pacchetto viaggia verso il sink, 1 in caso contrario. Dato che si usa un byte, si hanno a disposizione ulteriori 7 bit con cui identificare ben  $2^7 = 128$  sensori. Questo accorgimento è indispensabile per permettere la comunicazione bidirezionale nodo  $N$  - sink: quando un pacchetto viaggia verso il sink viene preso in consegna solo dal next hop del nodo  $N$ , quando il pacchetto proviene dalla direzione opposta viene inoltrato solo se proviene dal mio next hop.
- il tipo di pacchetto, cioè 1;
- il MAC del nodo destinatario (che generalmente è quello del sink, *0x00*);
- l'id del pacchetto, un intero usato per identificare un pacchetto e potergli inviare un ack di risposta. Questo id viene incrementato modulo 32768 da ogni nodo all'invio di un nuovo dato;
- il campo next hop, che viene aggiornato da ogni nodo intermedio nel percorso verso il sink inserendovi il proprio next hop, mentre nel percorso dal sink viene aggiornato con il proprio indirizzo;
- i dati veri e propri.

Si può osservare che il *beacon* e la *dhcp request* non presentano il campo destinatario, in quanto il primo è sempre inviato in broadcast e il secondo verso il sink.

## Finestre dinamiche

La disposizione dei nodi all'interno della rete in genere è effettuata in modo tale da coprire la maggior superficie possibile. Ciò comporta una certa distanza tra i nodi, con la conseguenza che ogni nodo non ha molti vicini con cui riesce a comunicare. Di conseguenza è uno spreco energetico restare in ascolto degli slot dei nodi da cui non si ricevono pacchetti, contando che il risparmio energetico si ottiene soprattutto mantenendo l'antenna spenta per il maggior tempo possibile.

La suddivisione della finestra di invio dei pacchetti in  $N_{sensori}$  slot ci permette una notevole flessibilità nell'accendere e spegnere l'antenna solo in corrispondenza dell'arrivo degli slot effettivamente utili a un determinato nodo.

Il nodo  $N$  tiene traccia dei nodi da cui riceve dei pacchetti segnandosi il loro MAC, sia quando riceve *beacon* che *dati*. Nell'istante in cui arriva la finestra di trasmissione,  $N$  cerca nella sua base dati (un array di unsigned int), da chi può ricevere dei pacchetti: i loro MAC, insieme al proprio, sono gli unici slot in cui servirà tenere l'antenna accesa. Dato che l'accensione e lo spegnimento dell'antenna impiega un certo lasso di tempo, si inizia l'accensione del CC2500 lo slot precedente a quello in cui il nodo deve essere pronto per ricevere dati. Perciò, se si possono ricevere pacchetti in due slot abbastanza vicini (con un solo slot di "buco" in mezzo) l'antenna non sarà spenta: l'overhead di tempo di spegnimento e accensione sarebbe tale da far perdere il sincronismo e non essere pronti per la ricezione all'arrivo del secondo slot. Quindi se i vicini del nodo  $N$  hanno MAC *0x09* e *0x11*, l'antenna non sarà

spenta durante lo slot  $0x10$  per non rischiare di non riuscire a riaccenderla in tempo per l'inizio dello slot dell'undicesimo nodo.

Tuttavia, contando che l'assegnazione dei MAC avviene in modo dinamico da parte del sink, e ricordando che è un'assegnazione “intelligente”, cioè per livelli in base alla distanza dal nodo principale, in cui i nodi più lontani dal sink hanno i MAC più bassi, questa situazione raramente si presenta.

Questo miglioramento, porta ad un massimo di  $(N_{sensori} - 2)/N_{sensori}$  il risparmio di energia (in genere nei nodi foglia, i quali dovranno essere attivi solo nel proprio slot e in quello del proprio next hop).

A intervalli regolari, è opportuno fare dei “discovery”, cioè tenere l'antenna accesa per tutti gli  $N_{sensori}$  slot, in modo tale da accorgersi dell'accensione di nuovi nodi nei paraggi.

# Capitolo 4

## Materiali e metodi

### 4.1 Configurazione hardware

#### 4.1.1 Microcontrollore

##### Scelta del microcontrollore migliore

Il microcontrollore [10] è un sistema a microprocessore completo, integrato in un solo chip e progettato per ottenere la massima autosufficienza funzionale. Il suo rapporto prezzo-prestazioni è ottimizzato per una specifica applicazione, a differenza, ad esempio, dei microprocessori impiegati nei personal computer, adatti per un uso più generale.

Il microcontrollore è la forma più diffusa e più invisibile di computer. Comprende la CPU, un certo quantitativo di memoria RAM e ROM (può essere PROM, EPROM, EEPROM o FlashROM) e una serie di interfacce di I/O (input/output) standard, fra cui molto spesso uno o più bus ( $I^2C$ , SPI, CAN, LIN), che permettono l'interfacciamento con altri dispositivi. Le periferiche integrate sono la vera forza del microcontrollore e possono essere: convertitori ADC e DAC multicanale, timer e counter, USART, numerose porte esterne bidirezionali bufferizzate, comparatori, PWM.

I microcontrollori sono contenuti nella quasi totalità di elettrodomestici e di apparecchi di uso quotidiano.

Questi sistemi hanno una capacità di calcolo molto limitata e di solito eseguono lo stesso programma (firmware) per tutta la durata del loro funzionamento.

Nel caso analizzato i requisiti hardware richiesti sono:

- presenza di timer interni per la sincronizzazione delle finestre di trasmissione,
- basso consumo in modalità low power,
- bassi tempi di wake up,
- presenza di almeno una porta  $I^2C$  e di una SPI, per collegare il battery meter e l'antenna,

- possibilità di un RTC (Real-Time Clock) interno, al fine di evitare l'utilizzo di uno esterno.

A causa di questi vincoli la scelta si è focalizzata su due dispositivi:

- i PIC della Microchip Technology,
- gli MSP430 della Texas Instruments.

PIC, il cui nome aziendale è “PICmicro”, è una famiglia di circuiti integrati a semiconduttore con funzioni di microcontrollore ed è stata sviluppata nel 1975 dalla General Instrument con il nome di “Programmable Intelligent Computer”. Quando, nel 1987, la sussidiaria General Instrument Microelectronics è stata separata dal resto dell'azienda per formare la Microchip Technology, il termine PIC ha assunto il significato di “Peripheral Interface Controller”, che mantiene ancora oggi. Il PIC è in grado di svolgere un set di istruzioni ridotto (RISC).

Sono stati presi in considerazione PIC delle famiglie 16, 18, 24, 30, 32. Si analizzano di seguito i pregi e difetti di ciascuna di esse:

- la famiglia 16, nonostante tra i suoi pregi ci sia un consumo estremamente basso in fase di sleep (nell'ordine della decina di  $nA$ ) e active, è stata esclusa a causa delle sue limitate possibilità, dovute al ridotto numero di timers a 16 bit;
- la famiglia 18 è molto simile alla famiglia 16 e presenta gli stessi pregi (consumi ridotti), un numero maggiore di timers e un maggior numero di comparatori A/D, ma non è ancora sufficiente;
- la famiglia 24 è un buon compromesso tra le caratteristiche del microcontrollore e i consumi dello stesso, e per questo motivo è stata scelta come riferimento per le prime implementazioni dell'algoritmo;
- le famiglie 30 e 32, che hanno a disposizione maggiore potenza elaborativa e memoria, sono state scartate a causa del loro eccessivo consumo, che mal si adattava al tipo di algoritmo energy-aware richiesto.

Il PIC che risponde maggiormente ai requisiti è il PIC24F16KA101, con tecnologia XLP (eXtreme Low Power), 3 timers a 16 bit, comparatori A/D, la cui piedinatura è illustrata in [4.1](#).

La programmazione del dispositivo è stata effettuata tramite l'ambiente di sviluppo MPLAB e il programmatore utilizzato è il PICkit 2, illustrato in [4.2](#).

Questo microcontrollore, tuttavia, ha portato nel corso dello sviluppo della piattaforma alcuni problemi, che hanno causato l'utilizzo di un altro dispositivo.

Alcuni di questi problemi sono stati:

- difficoltà di programmazione e debugging,

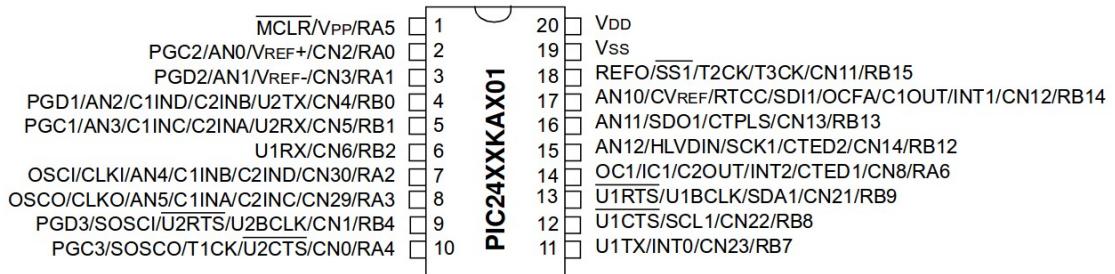
**20-Pin PDIP, SSOP, SOIC<sup>(2)</sup>**

Figura 4.1. La piedinatura del PIC 24F16KA101.



Figura 4.2. Il programmatore PICkit 2.

- alti costi di sviluppo dovuti principalmente alla mancanza di una board a basso costo (una delle meno costose è la “startusb for PIC” della MikroElektronika che costa circa 19\$),
- consumo eccessivo in modalità sleep e active (comparato con altri microcontrollori), discusso in [5.1.2](#).

MSP430 è una famiglia di micrонтrollori prodotta dalla Texas Instruments. Possiedono una CPU a 16 bit e sono particolarmente indicati per le applicazioni embedded low cost e soprattutto in ambito low power. A differenza dei PIC, gli MSP430 si sono rivelati migliori sotto molti punti di vista:

- facilità di programmazione tramite l’ambiente di sviluppo CCSTUDIO 5 (una versione modificata e brandizzata TI di eclipse che permette l’inserimento di breakpoints durante l’esecuzione del codice grazie alla tecnologia Spy-Bi-Wire),
- una migliore gestione energetica (i consumi in modalità sleep e active sono stati significativamente inferiori al PIC),
- la possibilità di avere una scheda di sviluppo a basso costo (la Launchpad MSP-EXP430G2 costa 5\$ circa),
- la presenza di librerie fornite dalla TI per l’interfacciamento con l’RF antenna.

La scheda di sviluppo scelta è stata quindi la Launchpad MSP-EXP430G2, illustrata in 4.3, che comprende un clock di precisione che può essere usato come sorgente per l’RTC interno, due led per debug, l’interfaccia JTAG, due pulsanti, uno programmabile e l’altro di reset, e alcuni jumper per il disaccoppiamento della parte di interfaccia usb con la scheda vera e propria.

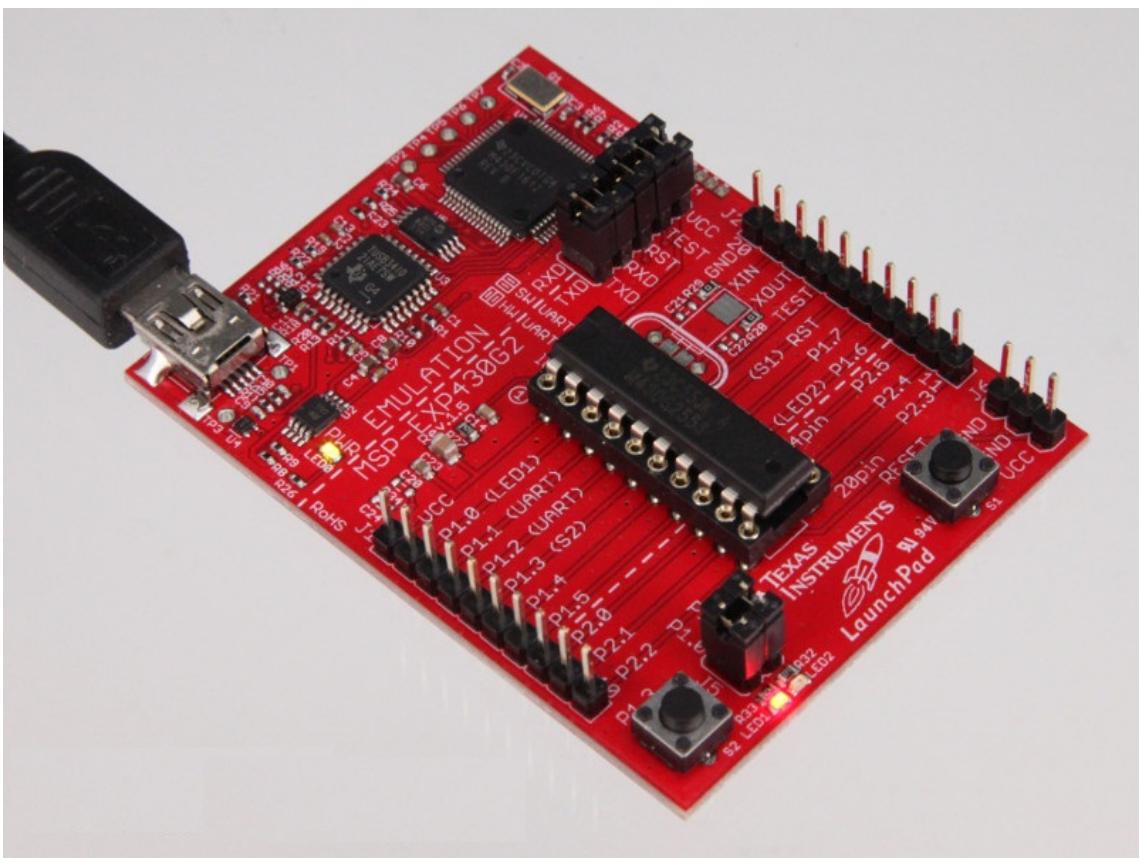


Figura 4.3. La scheda launchpad MSP-EXP430G2 (65 x 50) mm.

Per decidere quale MSP430 si adattasse meglio alle specifiche richieste, la scelta si è focalizzata su quei microcontrollori che presentano sia un’interfaccia  $I^2C$  sia SPI. Le diverse sottofamiglie che sono state analizzate sono:

- MSP430G2x33 e MSP430G2x03,
- MSP430G2x53 e MSP430G2x13,
- MSP430G2x33 e MSP430G2x03.

Questi insiemi di modelli, secondo i datasheet forniti, presentano gli stessi consumi in modalità active e sleep (con RTC attivo).

Perciò si sono utilizzati gli MSP430G2553 perché forniti insieme alla scheda Launchpad.

### Low Power Modes

La possibilità di impostare nel microcontrollore una modalità a basso consumo energetico si è rivelata indispensabile per minimizzare i consumi stessi.

Essenzialmente gli MSP430 hanno 4 modalità LPM (Low Power Modes) diverse, come si può vedere nella Figura 4.4

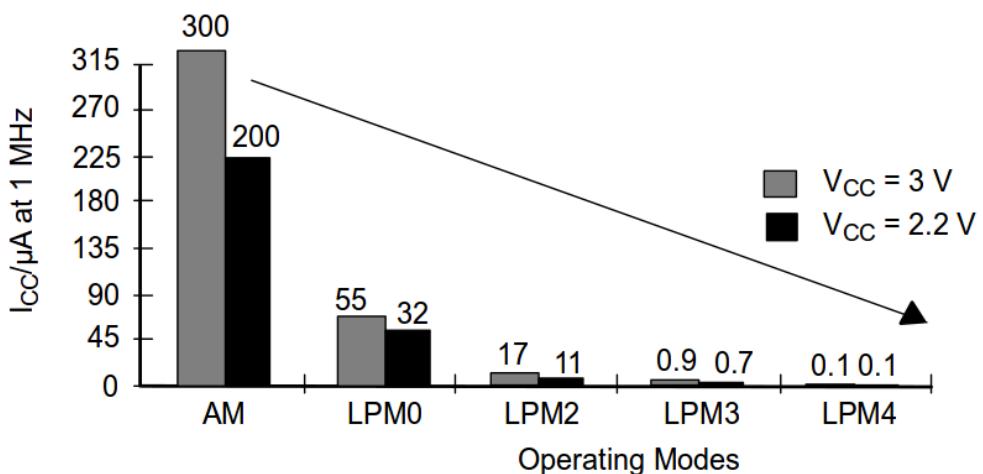
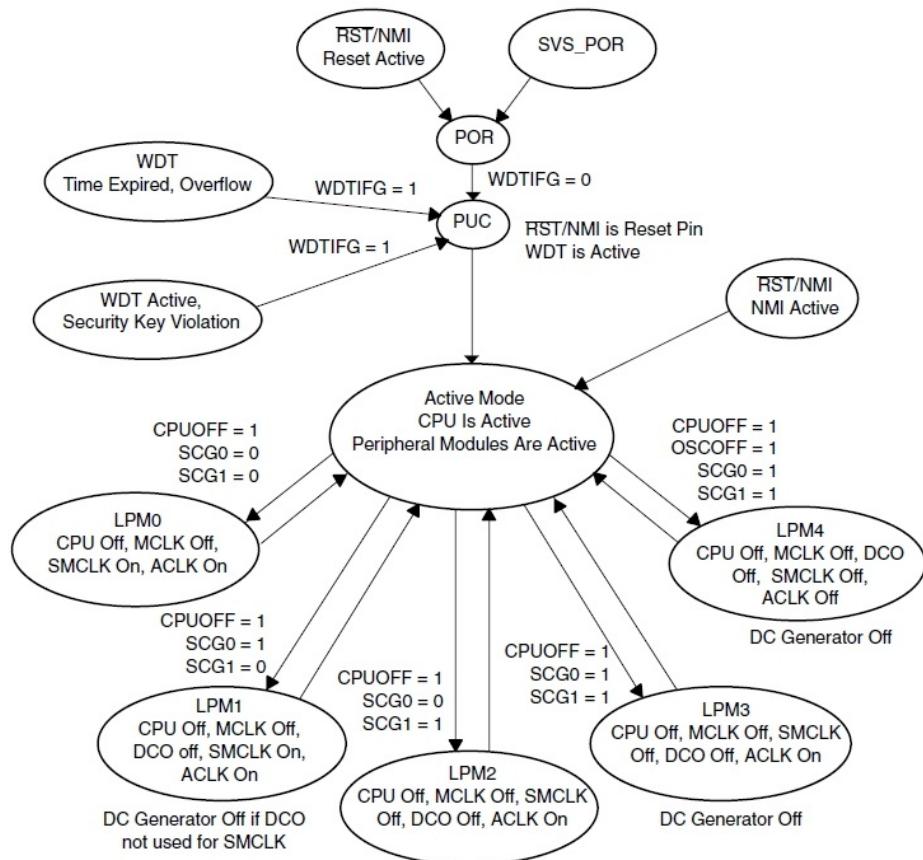


Figura 4.4. Grafico dei consumi di un MSP430 nelle varie modalità LPM.

Nella Figura 4.5 invece si può osservare uno schema più dettagliato sulle varie LPM. Si può osservare che la modalità LPM4 è quella che permette un consumo minore di energia, segue la LPM3 e così via, fino alla modalità che consuma maggiormente energia, ovvero LPM0. Il risparmio di energia si ottiene disabilitando sempre più funzionalità del microcontrollore.

La modalità LPM4, come detto in precedenza, richiede meno energia di funzionamento, ma ha lo svantaggio di non mantenere clock attivi, quindi l'uscita da questa modalità può avvenire solo tramite lo scatenarsi di un interrupt esterno. L'utilizzo di questa modalità avrebbe imposto l'aggiunta di un RTC esterno: la combinazione LPM4 e RTC esterno, tuttavia, si è rivelata nelle prove sperimentali molto più dispendiosa a livello energetico dell'utilizzo della LPM3, la quale permette di utilizzare un RTC interno.



SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled SMCLK , ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled, DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

Figura 4.5. Modalità low power di un MSP430.

L'RTC esterno utilizzato (comunicante con il microcontrollore tramite bus  $I^2C$ ) consumava durante il suo funzionamento circa  $40\mu A$ , un valore decisamente maggiore della differenza di consumi tra la LPM3 e la LPM4 (valore compreso tra 0.6 e i  $0.8\mu A$ ) in base alla tensione di alimentazione del microcontrollore, come mostrato in Figura 4.4.

### 4.1.2 Battery meter

Il Battery meter è un componente fondamentale per il buon funzionamento dell'algoritmo, in quanto permette a un dispositivo di valutare la propria energia residua. Per questo motivo la scelta si è focalizzata su una serie di apparecchi diversi sviluppati dalla MAXIM e dalla TI, riassunti in tabella 4.1.

La scelta del miglior battery meter ha presentato numerosi problemi, in quanto le modalità di consumo dichiarate dai costruttori non erano spesso paragonabili tra di loro. A causa di questo problema si è dovuta controllare ogni singola modalità cercandone il suo reale funzionamento e consumo (ad esempio se nella modalità di sleep o hibernate venisse comunque aggiornato lo stato di carica o meno).

I risultati della ricerca hanno condotto a due principali battery meter: la serie MAX17048/MAX17049 (nella versione a una o due celle) prodotta dalla Maxim e il BQ27510-G2 fornito dalla Texas Instruments.

Il primo battery meter garantisce bassi consumi sia nella modalità di sleep sia in active, ma non aggiorna lo stato di carica durante la prima modalità.

Invece, il BQ27510-G2 permette una modalità active molto meno onerosa in termini energetici a fronte di una modalità di sleep (sempre senza aggiornamento dello stato di carica) paragonabile a quella del MAX17048/MAX17049.

In caso la corrente di carico assorbita (chiamata  $I_{load}$ ), superi però una certa soglia (*Sleep Current*), il consumo del battery meter della Texas Instruments raggiunge i  $103\mu A$  contro i  $40\mu A$  dell'altro dispositivo. Dato che il duty cycle tra la fase di sleep e quella di active è molto elevato, questa situazione si verifica molto raramente, più precisamente solo quando l'antenna è attiva e comporta un assorbimento di corrente elevato.

Perciò, si può concludere che il BQ27510-G2 è il dispositivo più adatto alle specifiche richieste.

### BQ27510

Il BQ27510-G2 misura la tensione della cella, la temperatura e la corrente per determinare lo stato di carica (State of Charge, abbreviato in SOC) della batteria. Il BQ27510-G2 monitora l'attività di carica e scarica rilevando la tensione ai capi di un piccolo valore di resistenza (che varia da  $5m\Omega$  a  $20m\Omega$ ) tra i pin SRP e SRN e in serie con la cella. Lo stato di carica della batteria viene regolato grazie all'integrale della carica che vi passa attraverso durante il funzionamento del sistema stesso. La capacità totale della batteria si trova confrontando gli stati di carica prima e dopo l'applicazione del carico con la quantità di carica passata. Quando un carico viene applicato, l'impedenza della cella è misurata confrontando la OCV (Open Circuit

Componente	Active	Sleep/Hibernate	Precisione	Note
MAX17040/MAX17041	40 – 65 $\mu$ A	1 – 3 $\mu$ A	$\pm$ 12.5mV	-
MAX17043/MAX17044	50 – 75 $\mu$ A	1 – 3 $\mu$ A	$\pm$ 12.5mV	-
MAX17047/MAX17050	25 – 42 $\mu$ A	< 0.5 $\mu$ A (Shutdown)	$\pm$ 12.5mV	<i>a</i>
MAX17048/MAX17049	23 – 40 $\mu$ A	4 $\mu$ A	$\pm$ 7.5mV	<i>b</i>
MAX17058/MAX17059	23 – 40 $\mu$ A	0.5 – 2 $\mu$ A	$\pm$ 7.5mV	<i>c</i>
BQ27000/BQ27200	52 – 90 $\mu$ A	0.6 – 1.5 $\mu$ A	-	-
BQ27010/BQ27210	52 – 90 $\mu$ A	0.6 – 1.5 $\mu$ A	-	-
BQ27501/BQ27505-J5/BQ27500-V130	58 – 114 $\mu$ A	4 $\mu$ A	-	-
BQ27520-G3	118 $\mu$ A	8 $\mu$ A	-	-
BQ27541-{G1-V200}	60 – 131 $\mu$ A	6 $\mu$ A	-	-
BQ33100	< 450 $\mu$ A	< 4 $\mu$ A	-	-
BQ27410-G1	60 – 103 $\mu$ A	6 $\mu$ A	-	-
BQ27425-G1	23 – 114 $\mu$ A	6 $\mu$ A	-	-
BQ27510-G2	18 – 103 $\mu$ A	4 $\mu$ A	-	-

Tabella 4.1. Battery meters

<sup>a</sup>Calibrazione non richiesta, adattamento alle caratteristiche della cella, adattamento alle derive a lungo termine<sup>b</sup>No fase di learning, compatibile solo con accumulatori agli ioni di Litio<sup>c</sup>No fase di learning, compatibile solo con accumulatori agli ioni di Litio<sup>d</sup>Con tensione di ingresso compresa tra 0 e 1.5V<sup>e</sup>Con tensione di ingresso compresa tra 0 e 1.5V<sup>f</sup>58 con valore di corrente assorbita programmabile e compreso tra 0 e 100mA, 114 altrimenti  
<sup>g</sup>Con tensione di ingresso compresa tra 0 e 1.5V<sup>h</sup>60 con valore di corrente assorbita programmabile e compreso tra 0 e 100mA, 131 altrimenti<sup>i</sup>60 con valore di corrente assorbita programmabile e compreso tra 0 e 100mA, 103 altrimenti<sup>j</sup>23 con valore di corrente assorbita programmabile e compreso tra 0 e 100mA, 118 altrimenti<sup>k</sup>18 con valore di corrente assorbita programmabile e compreso tra 0 e 100mA, 103 altrimenti

Voltage, ovvero tensione a vuoto) ottenuta da una funzione predefinita per l’attuale stato di carica con la tensione durante il carico. Misure di OCV e di integrazione della carica determinano la capacità di carica e la capacità generica Qmax. I valori iniziali di Qmax sono ricavati dal datasheet del produttore della cella e moltiplicati per il numero di celle parallele. Qmax viene anche utilizzato per ottenere il valore di capacità iniziale. Il BQ27510-G2 acquisisce e aggiorna il profilo di impedenza della batteria durante il normale uso della batteria stessa. Esso utilizza questo profilo insieme al SOC e al valore Qmax, per determinare FullChargeCapacity() e lo StateOfCharge(), in particolare per il carico e la temperatura attuale. FullChargeCapacity() è la capacità disponibile di una batteria completamente carica con il carico e la temperatura attuali fino a quando Voltage() raggiunge la tensione “Term” (regolabile su un registro con valori da 0 a 32768mV). NominalAvailableCapacity() e FullAvailableCapacity() sono le versioni non compensate (senza o con carico minimo) rispettivamente di RemainingCapacity() e FullChargeCapacity(). Il BQ27510-G2 ha due flag ai quali si può accedere tramite la funzione Flags() la quale avvisa quando lo stato di carica della batteria è sceso a livelli critici. Quando RemainingCapacity() scende al di sotto la soglia di capacità minima, specificata nel registro SOC1, il flag [SOC1] (Stato di carica iniziale) viene settato. Il flag viene cancellato quando RemainingCapacity() supera la soglia SOC1. Il pin BAT\_LOW del BQ27510-G2 riflette automaticamente lo stato del flag [SOC1]. Quando RemainingCapacity() scende al di sotto della seconda soglia di capacità, chiamata SOCF, il flag [SOFC] (Stato di carica finale) viene settato, il quale serve come un avvertimento finale dello stato di carica. Se la soglia SOCF vale -1, il flag non è operativo durante la scarica. Allo stesso modo, quando RemainingCapacity() supera la soglia SOCF e il flag [SOFC] è già stato impostato, il flag [SOFC] viene resettato.

Le figure 4.6 e 4.7 mostrano, rispettivamente, un esempio di flow chart del dispositivo e i suoi stati di consumo, mentre la Figura 4.8 illustra lo schema elettrico di un BQ27510-G2.

### 4.1.3 RF antenna

L’RF antenna è un altro componente chiave dell’algoritmo, in quanto permette la comunicazione tra i diversi nodi della rete. Le caratteristiche più importanti sono:

- consumi estremamente ridotti in modalità sleep e active,
- facilità e velocità di switch tra la modalità di ricezione e trasmissione,
- bassi tempi di latenza nei passaggi di stato (active e sleep),
- costi contenuti,
- possibilità di variare la potenza di trasmissione e disponibilità d’informazione sulla qualità del segnale ricevuto (RSSI).

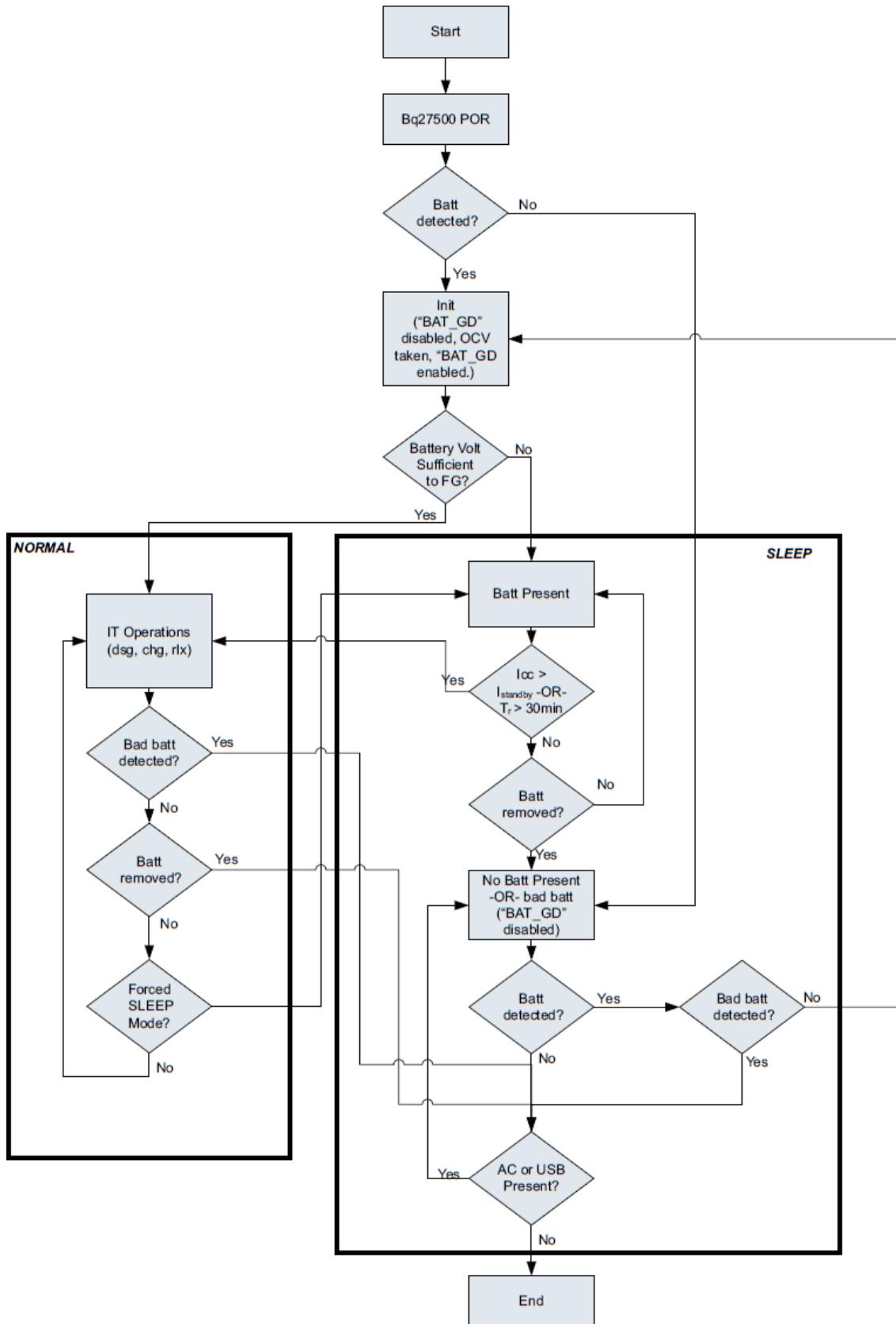


Figura 4.6. Flow chart del BQ27510-G2.

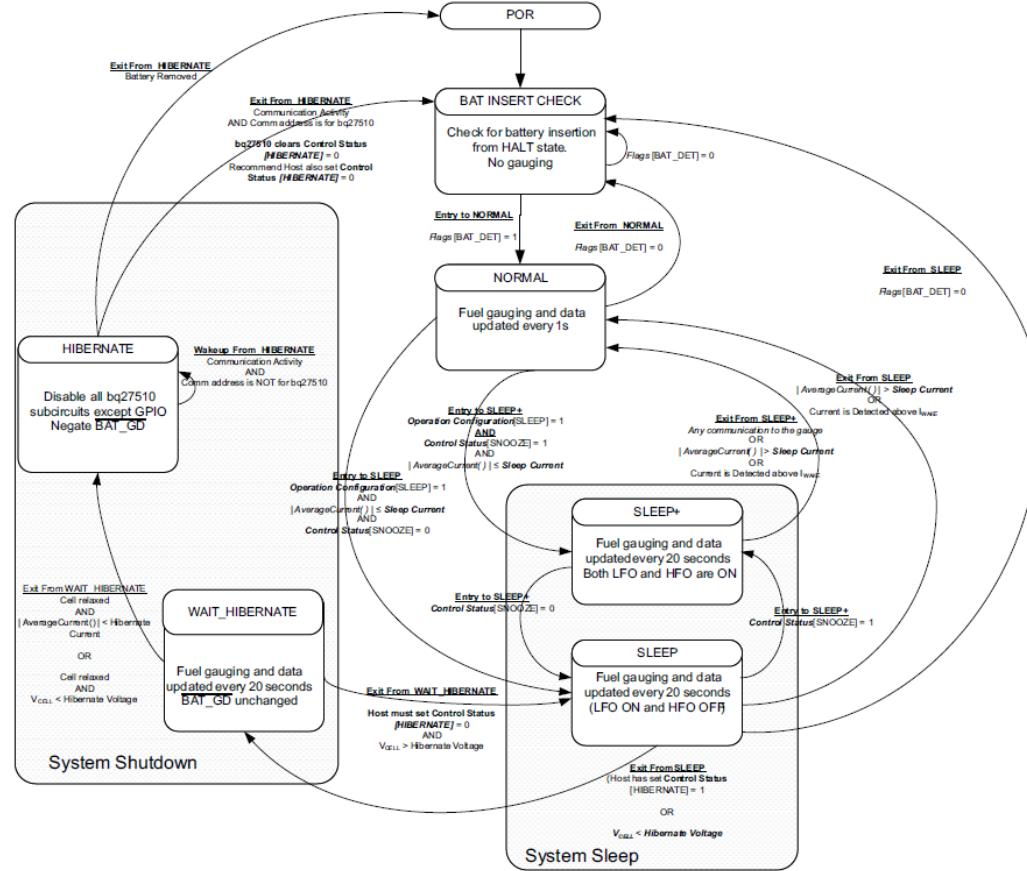


Figura 4.7. Stati del BQ27510-G2.

## REFERENCE SCHEMATIC

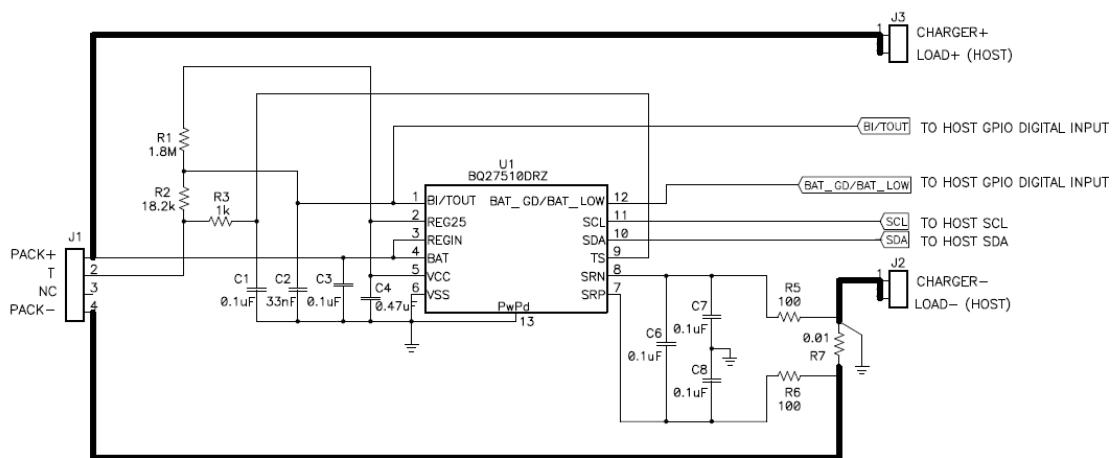


Figura 4.8. Schema di un BQ27510-G2.

## CC2500

Il CC2500 è un'antenna molto utilizzata nelle reti di sensori wireless per via del suo costo molto contenuto e delle buone prestazioni del chip.

Si è scelto di utilizzare due versioni dell'FM-TRX2-24G della Quasar, un chip che integra un CC2500 al suo interno. Questo chip permette la comunicazione tramite SPI tra l'antenna e il microcontrollore, ha a disposizione un controllo di integrità (CRC) on chip e un feedback sulla qualità della ricezione del segnale tramite RSSI. La frequenza utilizzata dal CC2500 varia nel range 2400 – 2483.5 MHz. L'utilizzo di una frequenza inferiore avrebbe permesso un maggior raggio d'azione, che esula però dagli obiettivi iniziali.

Nel nostro caso alcuni sensori sono stati realizzati con la versione 1 della scheda (oramai non più in produzione), mostrata in Figura 4.9, mentre i successivi prototipi utilizzano il nuovo chip (versione2), mostrato in Figura 4.10 che risolve alcuni bug di stabilità della versione precedente e fornisce una strip line che consente un montaggio più semplice sulle schede.



Figura 4.9. RF Antenna con chip CC2500 versione 1 (16.3 x 19.6) mm.

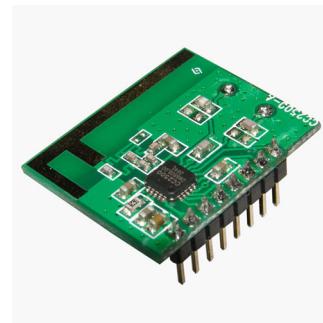


Figura 4.10. RF Antenna con chip CC2500 versione 2 (24 x 19) mm.

### 4.1.4 RTC

L'RTC è un componente fondamentale per mantenere la sincronizzazione tra i nodi della rete.

È stata utilizzata una scheda contenente un RTC isl1209 della Intersil che si interfaccia con gli altri dispositivi tramite  $I^2C$ . L'utilizzo di questa scheda, al posto di un RTC interno al microcontrollore, consente di spostare la parte di sincronizzazione dei sensori verso un chip esterno ad alta precisione. Le prove effettuate con tale scheda, tuttavia, hanno dimostrato che i consumi con l'RTC esterno sono superiori a quelli ottenuti con un RTC interno, quindi si è abbandonata la scheda esterna per contenere i consumi, semplificare l'architettura e diminuire i costi.

L'RTC scelto è stato quindi quello interno al microcontrollore, e per aumentarne la precisione è stato aggiunto manualmente un cristallo di quarzo a 32768Hz.

#### 4.1.5 Architettura scelta

L'architettura scelta comprende quindi:

- MSP430G2553,
- BQ27510-G2,
- CC2500 su scheda QFM-TRX1-24G (tre prototipi), oppure CC2500 su scheda FM-TRX2-24G (versione obsoleta utilizzata per due prototipi).

I motivi delle scelte sono stati trattati nelle sezioni corrispondenti.

#### 4.1.6 La scheda su eagle

Per il deploy della rete di sensori è stata progettata una scheda su eagle. Lo schema si può vedere in Figura 4.12, mentre la board si può vedere in Figura 4.12.

Nella scheda è previsto un jumper e la possibilità di aggiungere un led per il debug, come è altresì prevista la possibilità di aggiungere un transistor per spegnere la RF antenna interrompendo l'arrivo di corrente, ed evitare così il consumo della stessa quando si trova in modalità sleep (benchè contenuto), ciò a discapito di un aumento dei costi (il transistor) e di un maggiore tempo di wakeup dell'antenna ad ogni ciclo.

La scheda prevede anche la possibilità di programmare il chip BQ27510 direttamente onboard, tramite appositi switch SDA e SDL. L'elenco dei componenti utilizzati è visualizzato nelle tabelle 4.2, 4.3, e 4.4

### 4.2 Comunicazione tra i componenti

#### 4.2.1 $I^2C$

$I^2C$  [25], acronimo di Inter Integrated Circuit è un sistema di comunicazione seriale bifilare utilizzato tra circuiti integrati, sviluppato dalla Philips nel 1982.

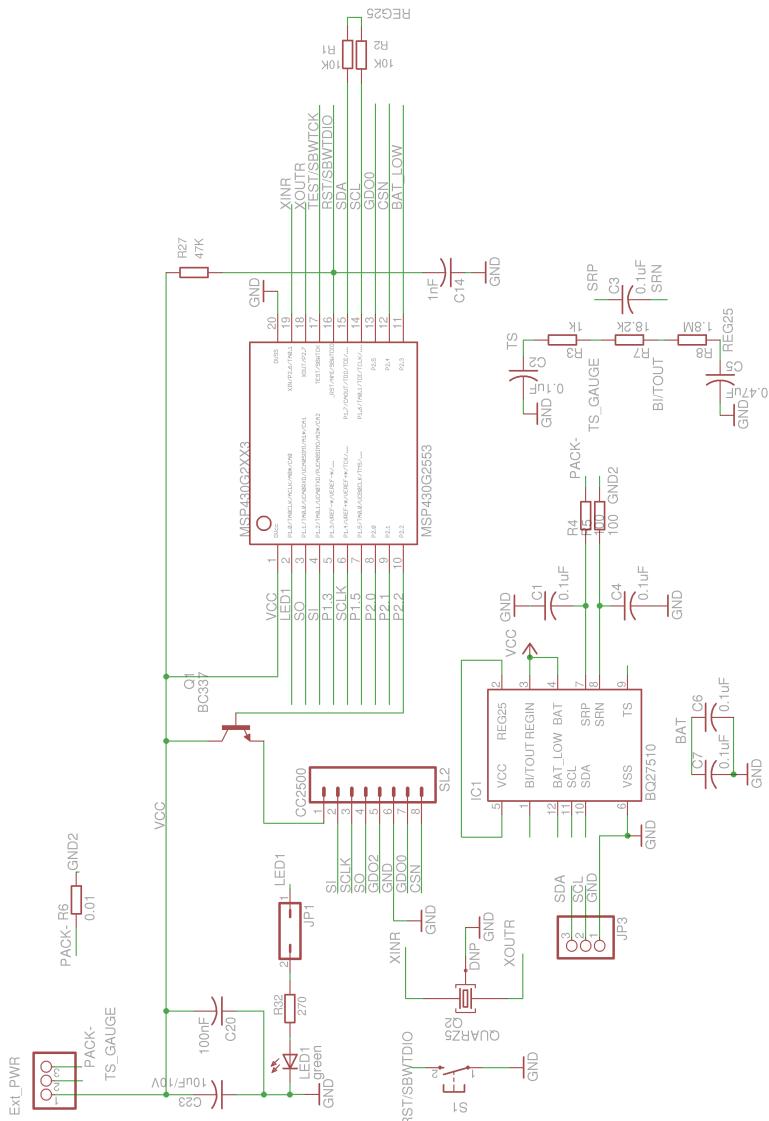


Figura 4.11. Lo schema di eagle.

Componente	Valore
C1	0.1 $\mu F$
C2	0.1 $\mu F$
C3	0.1 $\mu F$
C4	0.1 $\mu F$
C5	0.47 $\mu F$
C6	0.1 $\mu F$
C7	0.1 $\mu F$
C14	1 nF
C20	100 nF
C23	10 $\mu F$ /10V

Tabella 4.2. Elenco dei Condensatori utilizzati

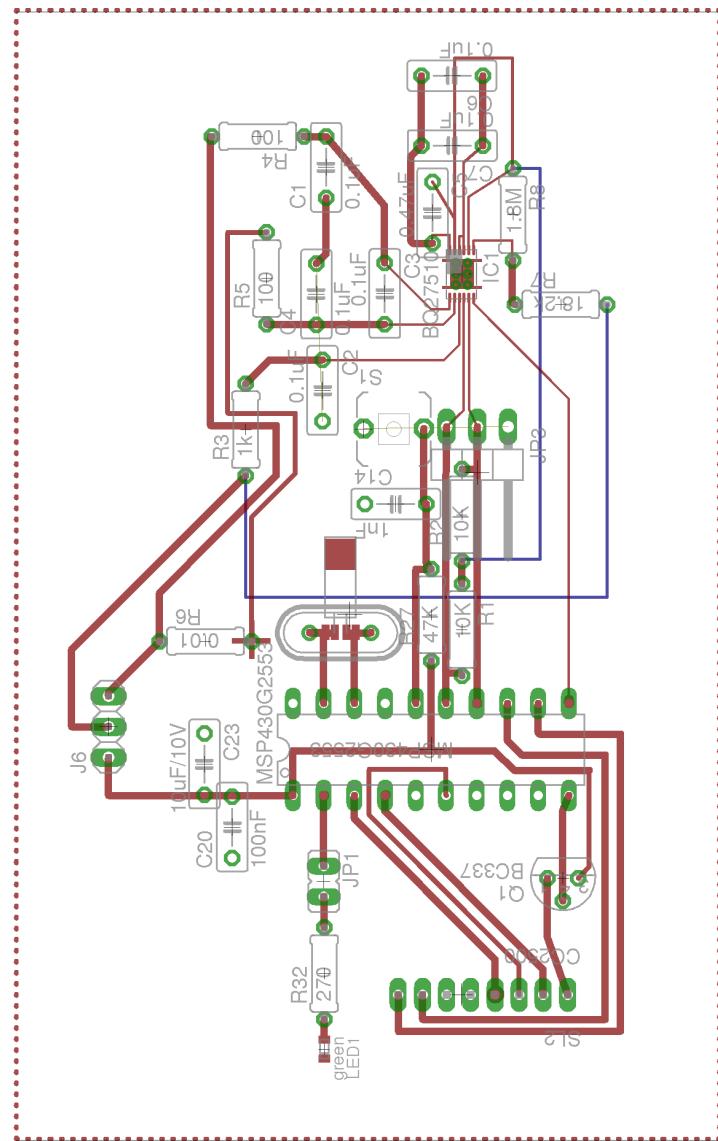


Figura 4.12. La board su eagle (90 x 60) mm.

Componente	Valore
R1	10 $k\Omega$
R2	10 $k\Omega$
R3	1 $k\Omega$
R4	100 $\Omega$
R5	100 $\Omega$
R6	0.01 $\Omega$
R7	18.2 $k\Omega$
R8	1.8 $M\Omega$
R27	47 $k\Omega$
R32	270 $\Omega$

Tabella 4.3. Elenco dei resistori utilizzati

Componente	Valore/Tipo
IC1	BQ27510
J6	1X03
JP1	JP1
JP3	1X03/90
LED1	DIODE0603
MSP430G2553	MSP430G2553
Q1	BC337
Q2	QUARZ5
S1	PBTH
SL2	CC2500

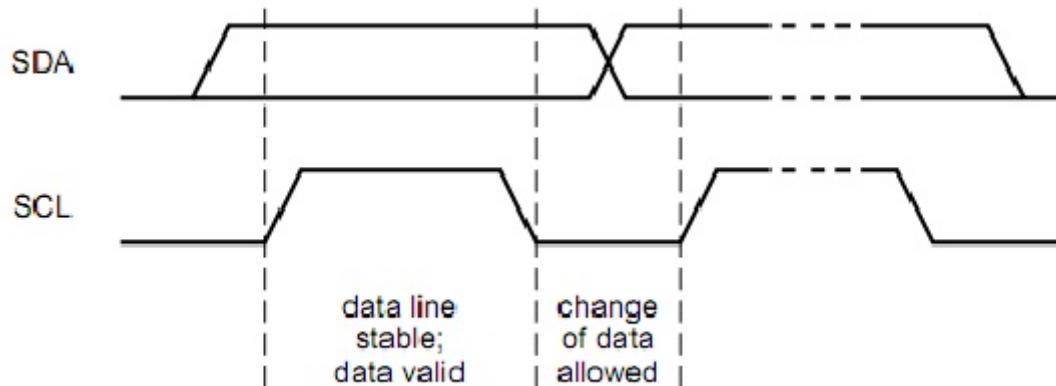
Tabella 4.4. Elenco dei chip, jumper e led

Il classico bus  $I^2C$  è composto da almeno un master e uno slave. Trattandosi di un protocollo seriale i vantaggi che offre sono quelli di impegnare solo due linee (e quindi due pin dei dispositivi che lo usano).

Il protocollo hardware dell' $I^2C$  richiede due linee seriali di comunicazione:

- SDA (Serial Data Line), linea utilizzata per i dati,
- SCL (Serial Clock Line), linea utilizzata per il clock (per la presenza di questo segnale l' $I^2C$  è considerato un bus sincrono).

La linea SCL fornisce la temporizzazione per il trasferimento ed è sempre pilotata dal master. Normalmente, mentre SCL vale 1, la linea SDA rimane stabile. Quando, invece SCL scende a 0, è possibile modificare il valore presente sulla linea SDA, come illustrato in 4.13.

Figura 4.13. I segnali SDA e SCL sul bus  $I^2C$ .

Come accennato in precedenza l' $I^2C$  è un bus con un clock (SCL) e una linea dati (SDA) e 7 bit possibili di indirizzamento. Un bus ha due tipi di nodi:

- almeno un nodo master, ovvero il dispositivo che emette il segnale di clock,
- uno o più nodi slave, il nodo che si sincronizza sul segnale di clock senza poterlo controllare.

Nel bus possono essere presenti più dispositivi che svolgono la funzione master, ma solo uno alla volta può svolgere tale compito. Un esempio di bus  $I^2C$  con alcuni dispositivi è illustrato in Figura 4.14.

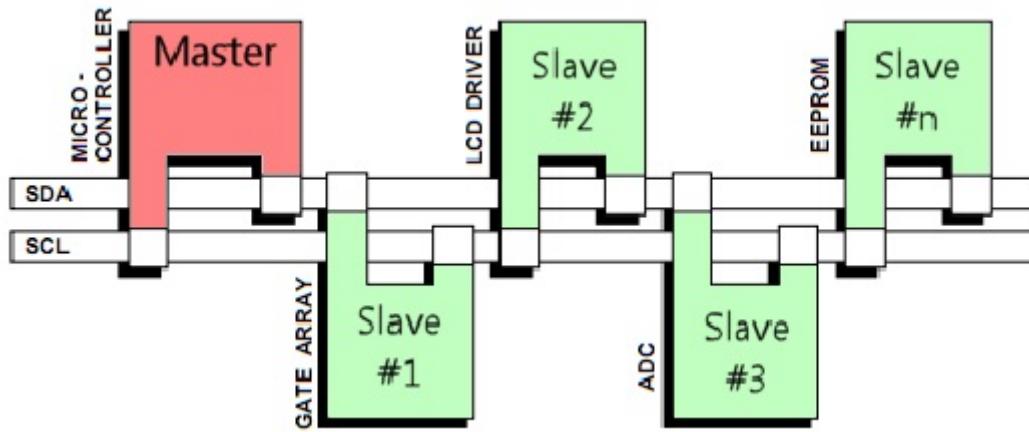


Figura 4.14. Esempio di bus  $I^2C$ .

In generale ci sono 4 modi distinti di operare:

- un master trasmette, controlla il clock e invia dati a uno slave,
- un master riceve, emette il clock ma riceve dati da uno slave,
- uno slave trasmette, non può pilotare il clock ma riceve il permesso dal master di scrivere sul bus e gli invia dati,
- uno slave riceve, il dispositivo non controlla il clock ma viene indirizzato dal master e riceve dati da esso.

Ogni dispositivo dispone di un indirizzo su 7 bit assegnato dal costruttore (B1, ..., B7). Il numero massimo di dispositivi che possono condividere il bus è 112, poiché 16 indirizzi sono riservati. Il master inizia lo scambio di informazioni inviando lo start bit (S) seguito dall'indirizzo dello slave su 7 bit con cui vuole comunicare. Segue un bit (B8) che indica se vuole trasferire informazioni allo slave (ovvero scrivere, ammesso che il dispositivo permetta questa possibilità) o ricevere informazioni (leggere i dati dallo slave indirizzato in precedenza). Nel primo caso il bit B8 viene tenuto basso dal master, mentre nel caso voglia ricevere informazioni il dispositivo rilascerà la linea dati (che sarà alta per la presenza del pull-up).

Se lo slave indirizzato (B1, ..., B7) esiste, prende il controllo della linea dati sul successivo impulso alto del SCL e la forza bassa, tale comportamento corrisponde

all’invio di un ACK. Il master sa quindi che il dispositivo selezionato ha ricevuto la richiesta ed è in attesa di rispondere.

Gli indirizzi e i dati, per convenzione del bus, sono trasmessi iniziando dal bit

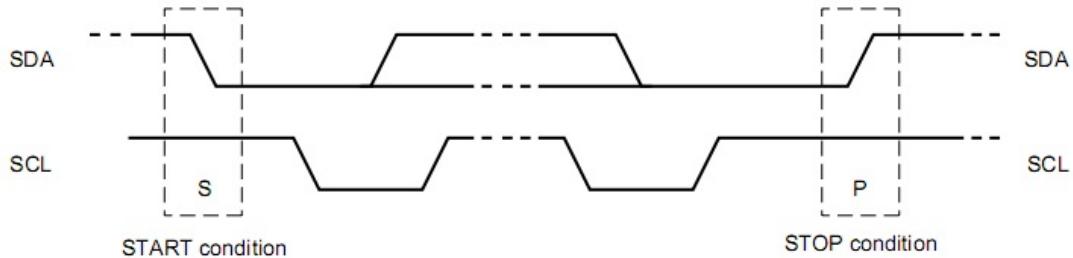


Figura 4.15. Segnali di start e stop condition.

più significativo (B1) e terminando con il bit meno significativo. Lo start e lo stop seguono le indicazioni della Figura 4.15:

- la condizione di start (S) è rappresentata da una transizione da alto a basso della linea SDA mentre la linea SCL è a un livello logico alto,
- il segnale di stop (P) è determinato da una transizione da basso ad alto della linea SDA mentre la linea SCL è a un livello logico alto.

Al termine di una transazione il bus deve restare inattivo per almeno  $4.7 \mu s$ .

Se il master vuole ricevere informazioni da uno slave ad ogni byte ricevuto invia un ACK meno l’ultimo byte. Inviato questo, può mandare uno STOP bit (P) o uno START bit (S) a seconda se voglia o meno mantenere il controllo del bus per un altro trasferimento. Un esempio di utilizzo del bus è illustrato in 4.16.

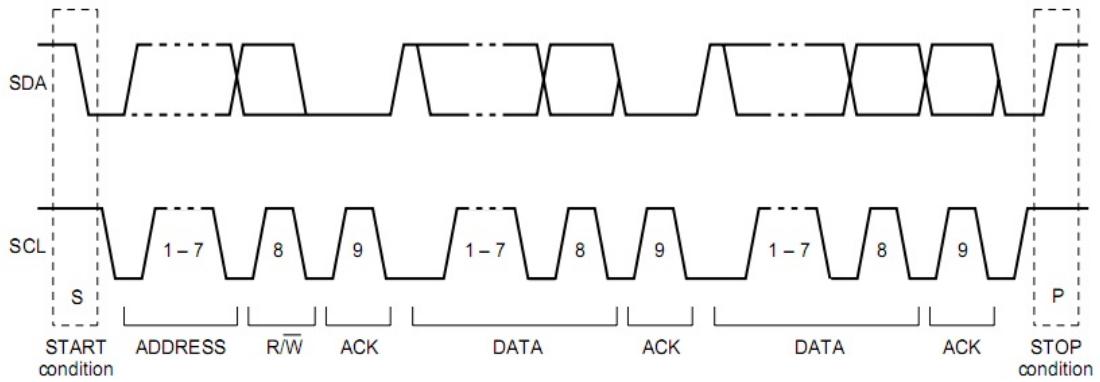


Figura 4.16. Esempio di indirizzamento di un dispositivo e invio dei dati sul bus  $I^2C$ .

A livello hardware, le linee SCL e SDA sono open-drain o open-collector a seconda della tecnologia usata (rispettivamente MOSFET o BJT).

È quindi indispensabile la presenza di un resistore di pull-up. Quando un dispositivo

attiva la sua uscita, forza bassa la linea portandola a livello logico zero, mentre lasciandola libera viene tirata su dal resistore di pull-up ed è considerata a livello logico alto (Vdd). Questo comporta il vantaggio di non avere in nessun caso conflitti hardware (nessun dispositivo può forzare il livello logico alto).

Va anche detto che, per migliorare la lettura in presenza di disturbi, è spesso presente sugli ingressi dei dispositivi un'isteresi. Il bus come detto in precedenza è a due fili poiché la comunicazione necessita di due linee per l'invio dei segnali (clock e dati). In realtà è comunque indispensabile avere un filo di riferimento comune (chiamato GND o Vss), per cui in realtà il numero minimo di connessioni fisiche è tre.

Per quanto riguarda la connessione positiva di alimentazione (indicata come Vdd, Vdd1, Vdd2, Vcc o simili, il cui valore è di solito +5 V o +3,3 V) non è indispensabile che sia comune a tutti i componenti connessi (anche se nella maggior parte delle applicazioni lo è).

### Esempi di invio dati

Come detto in precedenza il master pilota la linea: può richiedere trasferimenti da e verso uno specifico dispositivo. In entrambi i casi, comunque, si occupa di iniziare i trasferimenti di dato. Il master deve quindi generare le condizioni di START e STOP e inviare il segnale di clock durante le fasi di trasferimento dati. In aggiunta

- se è in modalità trasmissione genera la condizione di STOP a fronte della ricezione di un NACK da parte dello slave,
- se è in modalità ricezione genera un ACK dopo ogni byte ricevuto correttamente e un NACK in caso di trasferimento errato o per interrompere il trasferimento.

Lo slave, invece, è costantemente in ascolto sul bus per rilevare una condizione di START. Quando il master indirizza un dispositivo, se lo slave riconosce il proprio indirizzo, risponde con un ACK.

Durante le fasi di trasferimento dati:

- se è in modalità ricezione genera un ACK dopo ogni byte ricevuto correttamente o un NACK in caso di trasferimento errato,
- se è in modalità trasmissione continua a inviare i dati fino a che il master non risponde con un NACK o con una condizione di STOP.

Ogni dato trasferito sulla linea SDA deve essere sempre composto da 8 bit. Nel caso dell'invio dell'indirizzo dello slave, l'ottavo bit indica la direzione del trasferimento, il bit a 1 significa READ, a 0 WRITE. Per ogni transazione possono essere trasferiti un numero illimitato di byte sempre con in modalità MSB (Most Significant Bit). Dopo il trasferimento di ogni byte il destinatario genera obbligatoriamente un bit di acknowledge, che può essere un ACK (livello logico basso) o un NACK (livello logico alto).

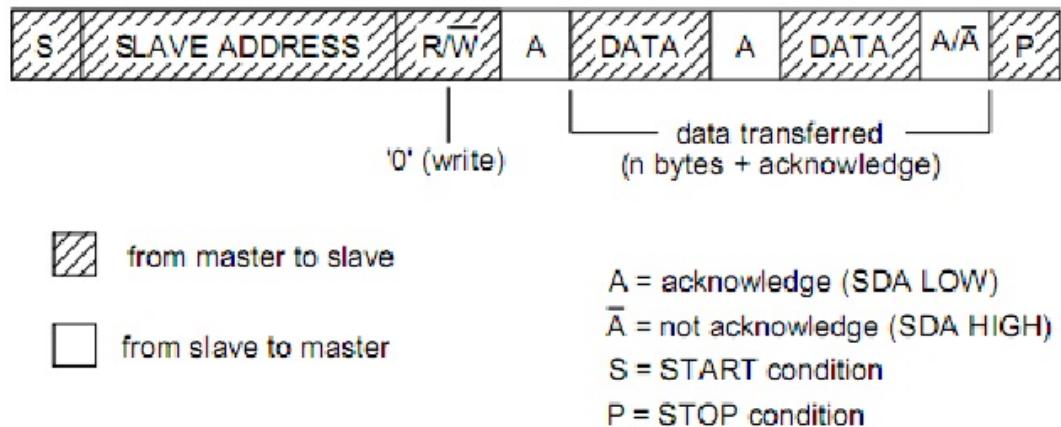


Figura 4.17. Esempio di trasmissione dati da parte del master e ricezione dello slave.

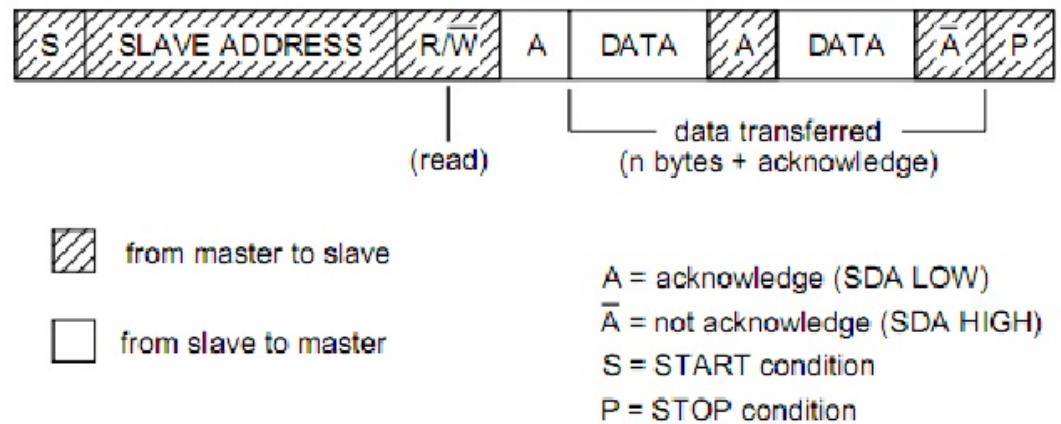


Figura 4.18. Esempio di trasmissione dati da parte dello slave e ricezione del master.

Si possono unire più transazioni segnalando la condizione di START nel primo bit inviato successivo ad un acknowledge, chiamato Repeated START, eliminando così il tempo minimo di attesa. Questa modalità combinata di invio è illustrata in Figura 4.19.

## L'utilizzo del PIC

Si analizza di seguito come sia stato programmato il PIC per comunicare con i componenti illustrati in precedenza (RTC e BQ27510) tramite  $I^2C$ .

Il codice qui presentato è stato testato ed è stato utilizzato per rilevare i consumi del microcontrollore (presentati nel capitolo successivo), decidendo di scartarlo e concentrarsi su un'altra architettura.

Nella sezione principale del programma, il main, come si può vedere nel codice riportato in 4.1, viene innanzitutto inizializzato l'RTC e settate le porte su cui è collegato il battery meter tramite  $I^2C$ . L'RTC esterno, collegato anch'esso tramite

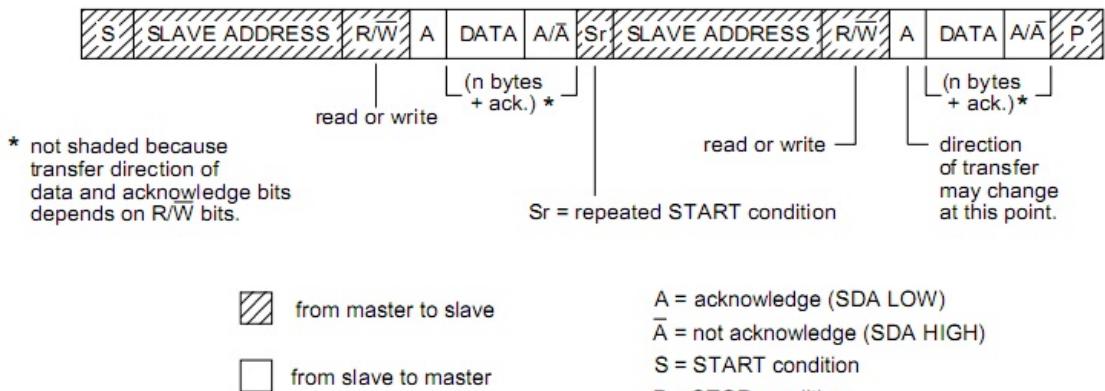


Figura 4.19. Esempio di trasmissione dati con Repeated Start.

$I^2C$ , è configurato per far scattare un interrupt a intervalli regolari. Il microcontrollore entra quindi in un loop infinito in cui viene messo in uno stato a basso consumo energetico (powersave0). Uscirà da questo stato solo all'arrivo dell'interrupt da parte dell'RTC.

Listing 4.1. Main

```

1 int main(void)
2 {
3     long int i,j;
4
5     /***** Inizialize and start RTC *****/
6     RTC_init();
7
8     /***** Setup I2C port *****/
9     PORTBbits.RB12=0;
10    LATBbits.LATB12=0;
11    TRISBbits.TRISB12=0;
12    TRISBbits.TRISB13=0;
13
14    INTCON1bits.NSTDIS=0;
15    INTCON2bits.INT0EP=0;
16    IEC0bits.INT0IE=1; // enable Interrupt 0
17
18    /***** Enter and stay forever in powersave mode *****/
19    while (1)
20    {
21        DSCONbits.DSEN=1;
22        Nop();
23        Nop();
24        Nop();
25        asm volatile("PWRSAV #0");
26    }
27

```

```

29 }  
      return 0;
}

```

L'Interrupt Service Routine è illustrata in 4.2. La routine, cancellati gli opportuni flag dell'interrupt ricevuto, non fa altro che andare a leggere lo stato di carica del battery meter. Fatto ciò setta una porta a 1 e a 0 con un breve delay intermedio, per fare lampeggiare un led collegato a tale porta.

Il microcontrollore, uscito dall'ISR, torna nel loop principale del main, settandosi nuovamente nella modalità powersave.

**Listing 4.2. Interrupt Service Routine dell'interrupt generato dal timer**

```

1 void __attribute__((interrupt, no_auto_psv)) _INT0Interrupt(void)
{
3   /*When the Timer1 expires, Interrupt INT0 will fire*/
4   /*This is the ISR that will be executed*/
5
6   long int i;
7
8   MI2C1_Clear_Intr_Status_Bit; //Clear interrupt flag
9   IFS0bits.INT0IF=0;
10  MI2C1_Clear_Intr_Status_Bit; //Clear interrupt flag
11
12  /***** Read bq27510 SOC *****/
13  battery=myI2CRead(BATTERY_ADDRESS, BATTERY_SOC);
14
15  //IEC0bits.INT0IE=0; // enable Interrupt 0
16
17  /***** Blink a led for debug *****/
18  PORTBbits.RB13=1;
19  for(i=0;i<150000;i++);
20  PORTBbits.RB13=0;
21  for(i=0;i<150000;i++);
22
23  IFS0bits.INT0IF=0;
}

```

**Listing 4.3. Lettura dello stato di carica del bq27510**

```

1 unsigned char myI2CRead(unsigned char slave_address, unsigned char command)
2 {
3   int i;
4   unsigned char battery;
5
6   myI2CInit(); // set up and open I2C
7
8   StartI2C1();
9   while(I2C1CONbits.SEN ); //Wait until Start sequence is completed
10  MI2C1_Clear_Intr_Status_Bit; //Clear interrupt flag

```

```

12 //Write Slave address and set master for transmission
13 MasterWriteI2C1((slave_address<<1)|0);
14 while(I2C1STATbits.TBF); //Wait until address is transmitted
15 while(!IFS1bits.MI2C1IF); //Wait for ninth clock cycle
16 MI2C1_Clear_Intr_Status_Bit; //Clear interrupt flag
17 while(I2C1STATbits.ACKSTAT);

18 for(i=0;i<1000;i++);
19 **** Master Data transmission after slave ACK ****
20 MasterWriteI2C1(command); //Transmit string of data

21 **** Restart condition for further reception from master ****
22

24 IdleI2C1(); //wait for the I2C to be Idle
25 RestartI2C1(); //Restart signal
26 while(I2C1CONbits.RSEN ); //Wait until Restart sequence is completed
27 for(i=0;i<1000;i++);

29 **** Master sends Slave Address with read signal ****
30 Nop();
31 MI2C1_Clear_Intr_Status_Bit;
32 //Write Slave address and set master for reception
33 MasterWriteI2C1((slave_address<<1)|1);
34 while(!IFS1bits.MI2C1IF); //Wait for ninth clock cycle
35 while(I2C1STATbits.ACKSTAT); //check for ACK from slave
36 MI2C1_Clear_Intr_Status_Bit;

38 **** Master Recieves from slave ****
39 battery = MasterReadI2C1(); //Master recieves from Slave upto 10 bytes

42 **** Stop condition ****
43 IdleI2C1(); //Wait for the I2C to be Idle
44 StopI2C1(); //Terminate communication protocol with stop signal
45 while(I2C1CONbits.PEN); //Wait until stop sequence is completed

47 CloseI2C1(); //Disable I2C
48 return battery;
}

```

La funzione *myI2CRead*, chiamata dall'ISR precedente e riportata in 4.3, illustra la lettura dello State of Charge del BQ27510 tramite  $I^2C$ . Alla funzione viene passato l'indirizzo del battery meter, definito pari a 0x55, ovvero 1010101 in binario su 7 bit. Per quel che riguarda l'ottavo bit, bisogna differenziare i casi in cui il dispositivo è interrogato in lettura dai casi in cui è interrogato in scrittura da parte del master del canale, ovvero il microcontrollore. Come spiegato nella sezione riguardante lo standard  $I^2C$ , il primo bit deve essere a 1 se il master dà allo slave la possibilità di scrittura e 0 in caso di lettura. Per questo, alle righe 13 e 34 della *myI2CRead*, l'indirizzo dello slave viene shiftato verso sinistra e il primo bit settato in maniera opportuna.

L'inizializzazione del canale  $I^2C$  avviene con la funzione *myI2CInit*, illustrata in [4.4](#), che setta gli indirizzi su 7 bit, disabilita lo slew rate e l'IPMI e setta il baudrate a 100 khz.

**Listing 4.4. Inizializzazione dell' $I^2C$**

```

1 void myI2CInit()
{
3     unsigned int config1 = 0;
4     unsigned int config2 = 0;

5     /* Turn off I2C modules */
6     CloseI2C1(); //Disable I2C1 module if enabled previously

9     /***** I2C interrupt configuration *****/
10    ConfigIntI2C1(MI2C_INT_OFF); //Disable I2C interrupt

11   /***** I2C1 configuration *****/
12   /*****
13   *
14   * I2C1 enabled
15   * continue I2C module in Idle mode
16   * IPMI mode not enabled
17   * I2CADD is 7-bit address
18   * Disable Slew Rate Control for 100KHz
19   * Enable SM bus specification
20   * Disable General call address
21   *****/
22   config1 = (I2C_ON | I2C_7BIT_ADD );
23   config2 = BAUD_RATE; // BAUD_RATE defined as 100 khz
24   OpenI2C1(config1,config2); //Configure I2C1

26   IdleI2C1();
27 }
```

Per le operazioni di debug dell' $I^2C$  è stato utilizzato un oscilloscopio (un Tektronik serie DPO) in modalità  $I^2C$ . Tramite questo strumento è stato possibile analizzare i segnali inviati sul bus per verificare che venissero scritti gli indirizzi giusti e il valore di SOC letto fosse quello effettivamente inviato.

Nelle Figure [4.20](#) e [4.21](#) viene mostrato uno screenshot dell'oscilloscopio durante una lettura. In azzurro è visualizzato il segnale su SCL mentre in verde quello su SDA. I dati visualizzati sullo schermo corrispondono all'indirizzo del dispositivo indirizzato, il comando inviato e il valore restituito.

## L'utilizzo del MSP430

All'interno del main viene inizializzata l' $I^2C$  tramite la funzione *my\_i2c\_init*, riportata in [4.5](#).

Questa funzione imposta la velocità del canale a 100 khz, setta la modalità sincrona,

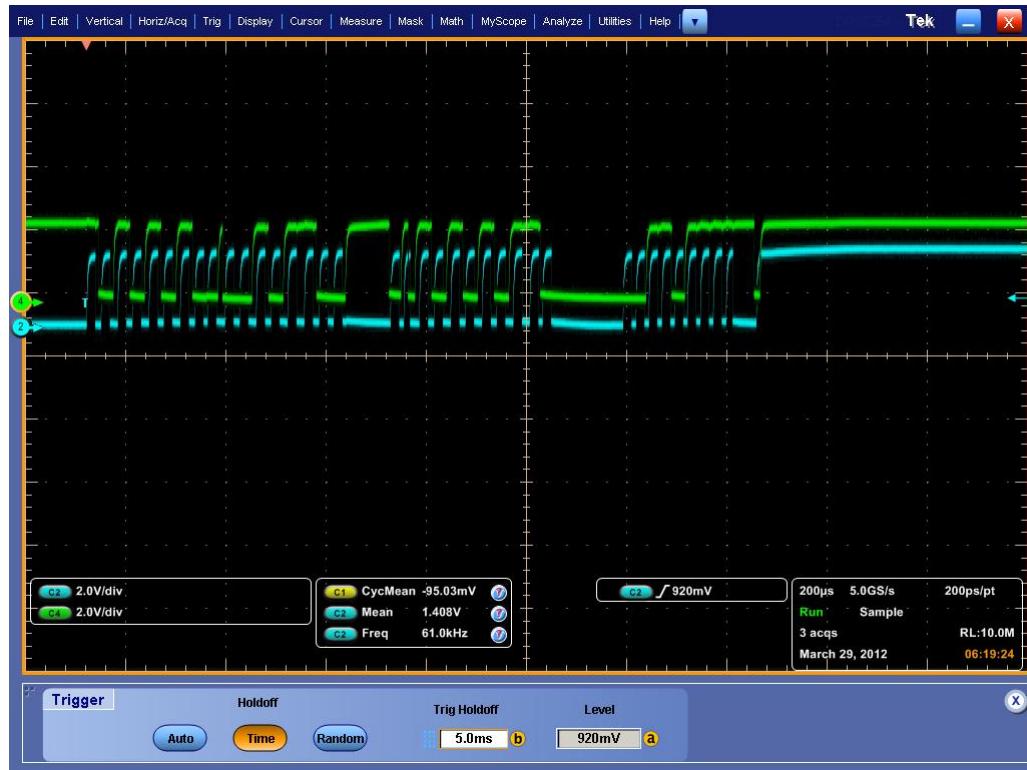


Figura 4.20. Esempio di lettura tramite  $I^2C$ .

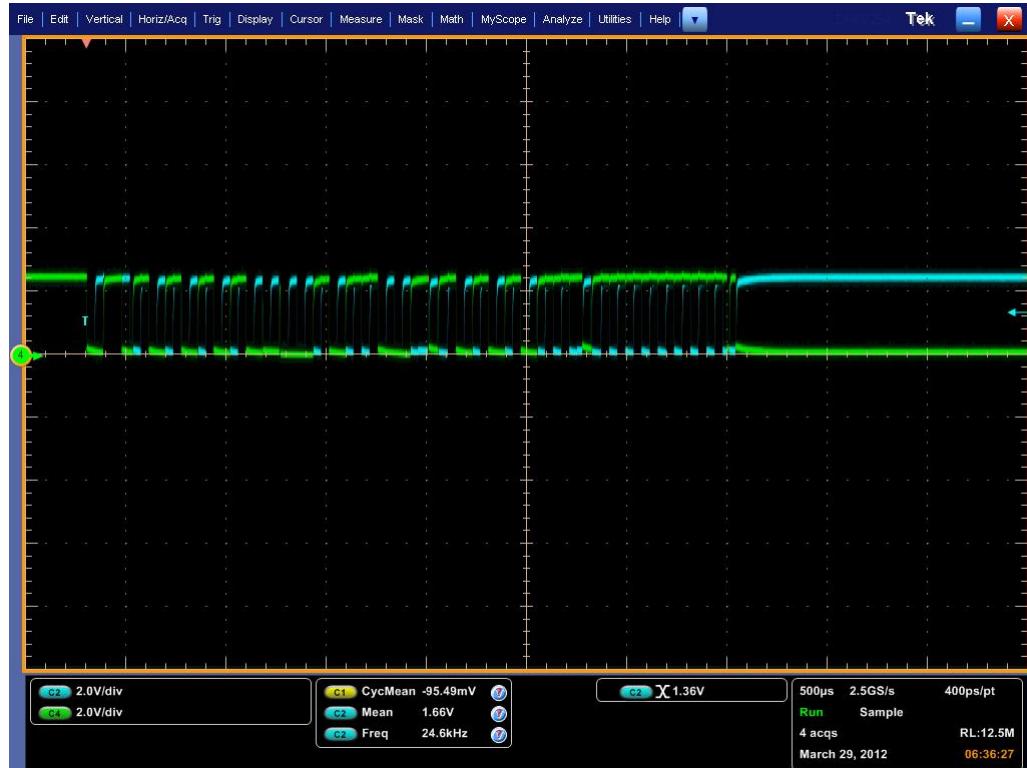


Figura 4.21. Esempio di lettura tramite  $I^2C$ .

assegna i pin su cui sono collegati SDA e SCL e setta 7 bit dell’indirizzo dello slave, ovvero il BQ27510. L’ottavo bit, quello più significativo, è impostato a uno o zero quando viene eseguita una lettura o una scrittura dal parte del battery meter.

**Listing 4.5. Inizializzazione dell’ $I^2C$**

```

void my_i2c_init()
{
    //*****
    //***** I2C configuration *****
    //*****

    UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC; //I2C Master, synchronous mode
    UCB0CTL1 = UCSSEL_2 + UCSWRST; // Use SMCLK, keep SW reset
    UCB0BR0 = 12; // fSCL = SMCLK/12 = ~100kHz
    UCB0BR1 = 0;
    UCB0I2CSA = 0x055; // Slave Address is 055h
    UCB0CTL1 &= ~UCSWRST; // Clear SW reset, resume operation
    IE2 |= UCB0RXIE + UCB0TXIE; // Enable RX, TX interrupt
    P1SEL |= BIT6 + BIT7; // Assign I2C pins to USCI_B0
    P1SEL2|= BIT6 + BIT7; // Assign I2C pins to USCI_B0

    return 0;
}

```

Il microcontrollore viene settato in una modalità low power (LMP3), come spiegato nella sezione precedente.

Quando viene risvegliato dal timer, deve leggere lo stato di carica del battery meter. Viene quindi chiamata la funzione `Read_i2c()`, riportata in 4.6, con cui si inviano i segnali di start sul canale  $I^2C$  per iniziare la comunicazione.

Bisogna poi gestire due interrupt, che si ricevono sui pin assegnati all’ $I^2C$ , ovvero *USCI\_B0*: uno per la ricezione e uno per la trasmissione, illustrati in 4.7. Per leggere lo stato di carica ovviamente è utilizzato il primo interrupt, in quanto bisogna mandare al battery meter il comando per leggere il SOC. Quando il BQ27510 scrive sul canale lo stato di carica, settando il flag *UCB0RXIFG* e riempendo il buffer *UCB0RXBUF*, si procede alla sua lettura.

Il secondo interrupt, *USCIAB0RX*, è utilizzato solamente per gestire l’arrivo di un nack sul canale.

**Listing 4.6. Lettura tramite  $I^2C$**

```

void Read_i2c()
{
    UCB0CTL1 |= UCTXSTT + UCTR; // I2C start condition
    while (UCB0CTL1 & UCTXSTT); // Start condition sent?
    UCB0CTL1 |= UCTXSTP; // I2C stop condition
    while(UCB0CTL1 & UCTXSTP); // Ensure stop condition got sent
    UCB0CTL1 &= ~UCTR; // I2C RX command
    UCB0CTL1 |= UCTXSTT; // I2C start condition
    while (UCB0CTL1 & UCTXSTT); // Start condition sent?
    UCB0CTL1 |= UCTXSTP; // I2C stop condition
}

```

```

12 }  
    while(UCB0CTL1 & UCTXSTP); // Ensure stop condition got sent

```

Listing 4.7. Interrupt generati dal canale dell' $I^2C$ .

```

// USCI_B0 Data ISR  

2 #pragma vector = USCIAB0TX_VECTOR  

 _interrupt void USCIAB0TX_ISR(void)  

4 {  

    if(IFG2 & UCB0TXIFG)  

    {  

        IFG2 &= ~UCB0TXIFG; // Clear USCI_B0 TX int flag  

        UCB0TxBuf = 0x2C; // Send SOC command  

    }  

10   else if(IFG2 & UCB0RXIFG)  

    {  

        IFG2 &= ~UCB0RXIFG; // Clear USCI_B0 RX int flag  

        soc = UCB0RXBuf; // Move RX data to a variable  

14    }  

16 }  

18 // USCI_B0 State ISR  

19 #pragma vector = USCIAB0RX_VECTOR  

20 _interrupt void USCIAB0RX_ISR(void)  

21 {  

    if (UCB0STAT & UCNACKIFG)  

    {  

        UCB0CTL1 |= UCTXSTP;  

        UCB0STAT &= ~UCNACKIFG;  

    }  

26   else if (UCB0STAT & UCSTTIFG)  

        UCB0STAT &= ~UCSTTIFG;  

28 }

```

#### 4.2.2 SPI

Il Serial Peripheral Interface o SPI [25] è un bus standard di comunicazione ideato dalla Motorola: è un sistema di comunicazione tra un microcontrollore e altri circuiti integrati o tra più microcontrollori.

La trasmissione avviene tra un dispositivo detto master e uno o più slave. Il master controlla il bus, emette il segnale di clock, decide quando iniziare e terminare la comunicazione. Il master deve essere unico ma la qualifica di master può essere scambiata tra diversi dispositivi connessi al bus (purché siano in grado di farlo e cioè non siano nati come semplici slaves). Il bus SPI si definisce:

- di tipo seriale,
- sincrono, per la presenza di un clock che coordina la trasmissione e ricezione dei singoli bit e determina la velocità di trasmissione,

- full-duplex, in quanto l'invio dei dati può avvenire contemporaneamente in trasmissione e ricezione.

Per quanto riguarda la velocità di scambio dei dati (in pratica la frequenza del clock) non vi è un limite minimo (in quanto i dispositivi sono statici: possono mantenere, se alimentati correttamente, uno stato logico per un tempo indefinito) ma vi è un limite massimo, che va determinato dai datasheet dei singoli dispositivi connessi e dal loro numero, in quanto ogni dispositivo collegato al bus introduce sulle linee di comunicazione una capacità parassita.

Il sistema di comunicazione di solito serve per lo scambio di dati tra dispositivi

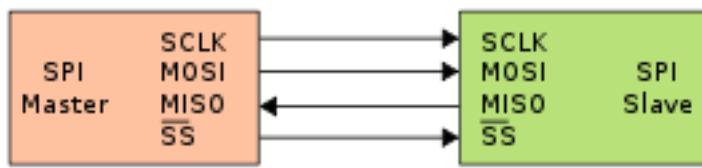


Figura 4.22. Collegamento Master-Slave tramite SPI.

presenti sulla stessa scheda (o comunque tra schede vicine tra di loro) in quanto non prevede particolari accorgimenti hardware per trasferire informazioni tra dispositivi lontani connessi con cavi soggetti a disturbi.

Il sistema è comunemente definito a quattro fili. Con questo si intende che le linee di connessione che portano i segnali sono in genere quattro. Va però tenuto conto che vi deve comunque essere una connessione di riferimento (0 Vdc, comunemente indicata con GND), quindi fisicamente i fili sarebbero cinque. Discorso analogo si può fare su altri bus seriali definiti a due fili ( $I^2C$ ) o ad un filo in quanto si considerano solo le linee di segnale e si sottintende la connessione di riferimento.

Il bus SPI si basa su 4 segnali, i cui nomi possono variare a seconda del costruttore dei dispositivi utilizzati:

- SCLK - SCK: Serial Clock, segnale periodico costantemente emesso dal master,
- SDI – MISO – SIMO – DI - SI: Serial Data Input, Master Input Slave Output, Data Input, Serial Input, ovvero l'ingresso per il master ed uscita per lo slave,
- SDO – MOSI – SOMI – DO – SO: Serial Data Output, Master Output Slave Input, Data Output, Serial Output, cioè dati in uscita dal master,
- CS – SS – nCS – nSS – STE: Chip Select, Slave Select, segnale emesso dal master per scegliere con quale dispositivo slave vuole comunicare (dalla Figura 4.22 si vede che il segnale SS è negato, si comprende perciò che per comunicare con il dispositivo slave deve venire messo a livello logico basso).

Di questi segnali il Chip Select (CS o SS) non è indispensabile in tutte le applicazioni. Il segnale SCLK è il clock seriale che scandisce gli istanti di emissione e di lettura

dei bit sulle linee di dati. È un segnale emesso dal master ed è quindi quest'ultimo a richiedere di volta in volta la trasmissione di una dato. Il segnale SDI/MISO è la linea attraverso cui il dispositivo (master o slave) riceve il dato seriale emesso dalla controparte. Sullo stesso fronte di commutazione del clock, il dispositivo emette, con la stessa cadenza, il suo output ponendo il dato sulla linea SDO/MOSI (linea di output di dato).

### Comunicazione tramite SPI

La trasmissione dei dati sul bus SPI si basa sul funzionamento dei registri a scorrimento (shift register). Ogni dispositivo, sia master che slave, è dotato di un registro a scorrimento interno i cui bit vengono emessi e, contemporaneamente, immessi, rispettivamente, tramite l'uscita SDO/MOSI e l'ingresso SDI/MISO, come illustrato in Figura 4.23. Il registro può avere dimensione arbitraria (ma uguale per i

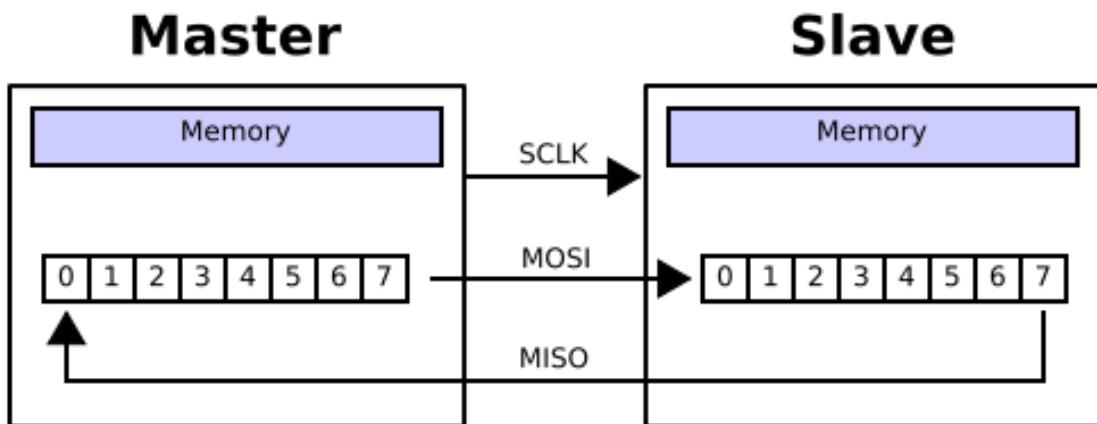


Figura 4.23. Utilizzo degli shift register da parte di Master e Slave del bus SPI.

dispositivi master e slaves) anche se usualmente ha dimensione 8 bit. Il registro a scorrimento è un'interfaccia completa mediante la quale vengono impartiti comandi e trasmessi dati che arrivano in modo seriale ma che internamente sono prelevati, a fine trasmissione, in modo parallelo. Ad ogni impulso di clock i dispositivi che stanno comunicando sulle linee del bus emettono un bit dal loro registro interno rimpiazzandolo con un bit emesso dall'altro interlocutore. La sincronizzazione è fatta sui fronti di clock di salita o di discesa regolata da 2 parametri impostabili: CPOL e CPHA. CPOL regola la polarità del clock ovvero discrimina lo stato normale di riposo cui si porta la linea di clock quando non è attiva. Quando CPOL è impostato a 0, il clock, nel suo stato di riposo, si porta a livello logico basso; viceversa il clock si porta a livello logico alto durante il tempo di inattività se CPOL è impostato ad 1. CPHA regola il fronte di clock in cui il ricevente campiona il segnale in ingresso. Se CPOL=0 allora con CPHA possiamo scegliere di campionare il dato sul fronte di discesa del segnale di clock, impostando CPHA=0, oppure sul fronte di salita, impostando CPHA ad 1. L'inverso accade se CPOL è settato ad 1.

Queste opzioni, in genere, sono impostabili sul dispositivo master e permettono di adattarlo a tutte le possibili varianti di dispositivi slaves che normalmente, invece,

vengono progettati per avere uno dei 4 modi di comunicazione possibili (tutte le combinazioni di CPOL e CPHA). Le modalità di funzionamento più spesso utilizzate dai dispositivi in commercio sono quelle con CPHA=CPOL=0 e con CPHA=CPOL=1. Il dato di output è emesso sempre in corrispondenza della prima transizione del clock. La comunicazione viene intrapresa sempre su iniziativa del dispositivo master, che abilita lo slave tramite CS e successivamente impone il clock sulla linea dedicata. Con questa procedura ha inizio lo scambio dei bit tra i due registri. Alla fine di ogni parola trasmessa il contenuto del registro dello slave sarà passato al master e viceversa. Con opportune parole identificative si possono inviare comandi al dispositivo ricevente che potrà effettuare l'elaborazione assegnata ponendo quindi nel suo shift-register il dato richiesto che al prossimo ciclo di trasmissione verrà trasmesso al richiedente.

Molti microcontrollori dispongono di un hardware dedicato per la gestione del-

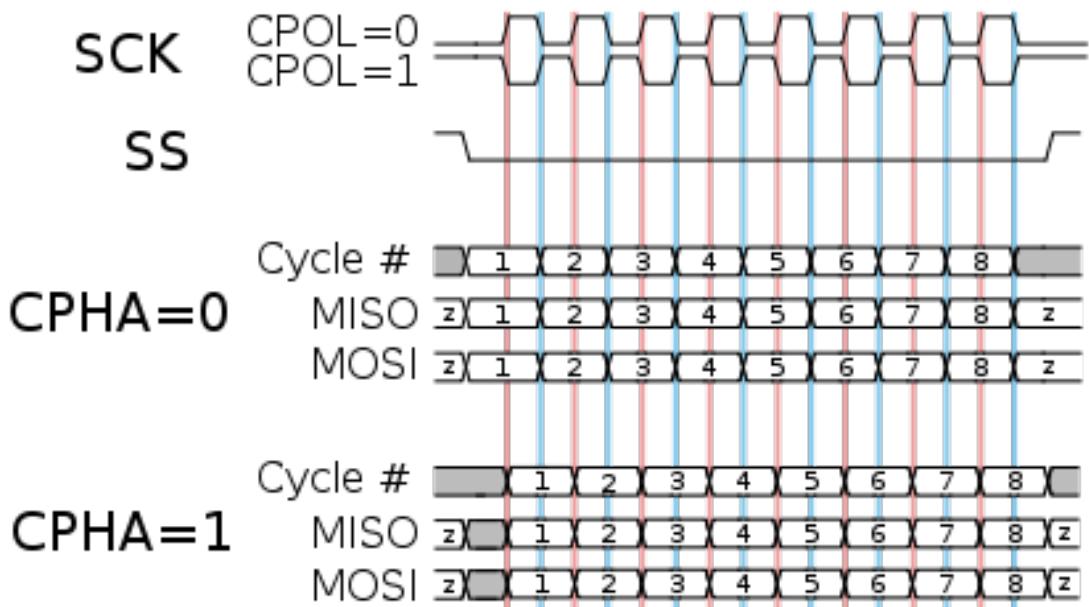


Figura 4.24. Esempio di comunicazione tramite SPI.

l'SPI programmabile nei dettagli. Questo non è strettamente necessario per poter comunicare con un dispositivo slave SPI (come ad esempio una memoria eeprom): in ogni caso si potrà implementare nel firmware delle funzioni dedicate al colloquio (che forse risulteranno più lente e occuperanno più spazio nel firmware di quelle disponibili con un blocco hardware dedicato) ma che comunque permetteranno di comunicare in modo efficiente con la periferica SPI, in quanto non vi è, nella temporizzazione dei dati, un limite di tempo massimo da rispettare (si tratta infatti di dispositivi statici, che possono cioè sospendere a tempo indeterminato la comunicazione senza per questo avere perdita di dati; in altre parole hanno dei limiti per quanto riguarda la velocità massima di trasmissione ma che possono funzionare anche in modo lento quanto si vuole).

### 4.2.3 I timer dell’MSP430

Una corretta gestione dei timer a livello di microcontrollore è indispensabile per la corretta sincronizzazione dell’intera rete. Gli MSP430 presentano generalmente diverse tipologie di timers:

- Basic Timer che di norma arriva fino a 2s, utilizzato per applicazioni semplici quali clock per moduli LCD (per il refresh ad esempio) oppure semplici orologi (si può fare in modo che venga scatenato un interrupt ogni secondo);
- RTC che è un estensione del Basic timer, ma aggiunge registri per Anno, Mese, Giorno, Giorno della settimana, Ore, Minuti, Secondi, non richiedendo la gestione esplicita dell’incremento dei valori dei contatori;
- Watchdog Timer (WTD/WTD+) utilizzato come “cane da guardia” (essenzialmente se non viene rinfrescato almeno ogni 32ms circa, il microcontrollore viene resettato, utilizzato in caso a causa di errori nel codice non si possa resettare manualmente l’MSP430), oppure come timer (invece di un reset viene scatenato un interrupt);
- Timer\_A il più versatile con possibilità di selezione di clock esterni (oppure uno dei 4 interni), modalità di count e registri di compare;
- Timer\_B molto simile al Timer\_A, con possibilità di programmazione della lunghezza in bit dei timer e un numero maggiore di registri CCR.

Nel nostro caso il Basic Timer è stato scartato in quanto non garantiva periodi di sleep maggiori di qualche secondo (il sistema avrebbe richiesto un numero troppo elevato di risvegli).

l’RTC presentava oltre agli svantaggi del Basic Timer un consumo maggiore per la quantità di registri utilizzati (e non è presente nella versione dell’MSP430 utilizzata). Il Watchdog garantisce periodi di sleep minori di quelli del Basic Timer ( 32ms). Il Timer\_A 4.25 invece garantisce periodi di sleep molto elevati (decine di secondi), la possibilità di utilizzo di clock esterni e una granularità minima sufficientemente elevata.

Il Timer\_B non è presente nella versione dell’MSP430 utilizzata.

Nella Figura 4.26 si può vedere un esempio di diverse modalità di utilizzo dei timers mentre nella Figura 4.27 si possono notare le diverse modalità di count dei registri.

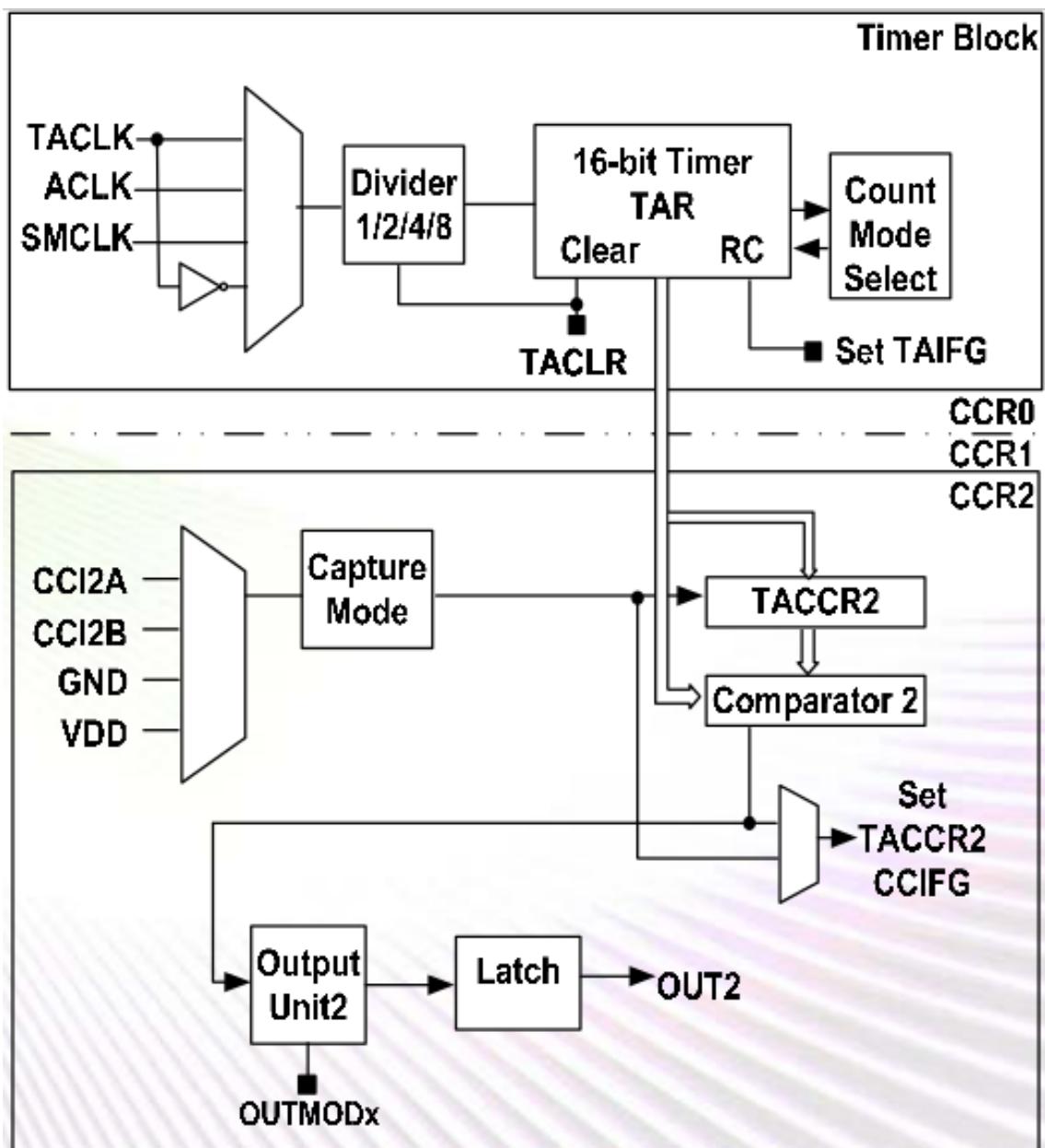


Figura 4.25. Esempio di schema per il Timer\_A.

**Example 1: MCLK = SMCLK = 1.048MHz and ACLK = 32kHz**

	<b>Minimum Period</b>	<b>Maximum Period</b>
<b>Watchdog</b>	61us / 16.4kHz	1sec / 1Hz
<b>Basic / RTC</b>	1.9us / 524kHz	2sec / 0.5Hz
<b>Timer_A/B</b>	0.95us / 1.048MHz	32sec / .031Hz

**Example 2: MCLK = SMCLK = 16MHz and ACLK = VLOCLK = 12kHz**

	<b>Minimum Period</b>	<b>Maximum Period</b>
<b>Watchdog</b>	4us / 250kHz	2.7sec / 0.37Hz
<b>Basic / RTC</b>	125ns / 8MHz	5.5sec / 0.18Hz
<b>Timer_A/B</b>	62.5ns / 16MHz	87.4sec / .011Hz

Figura 4.26. Esempio di periodi di interrupt con varie modalità di utilizzo dei timers.

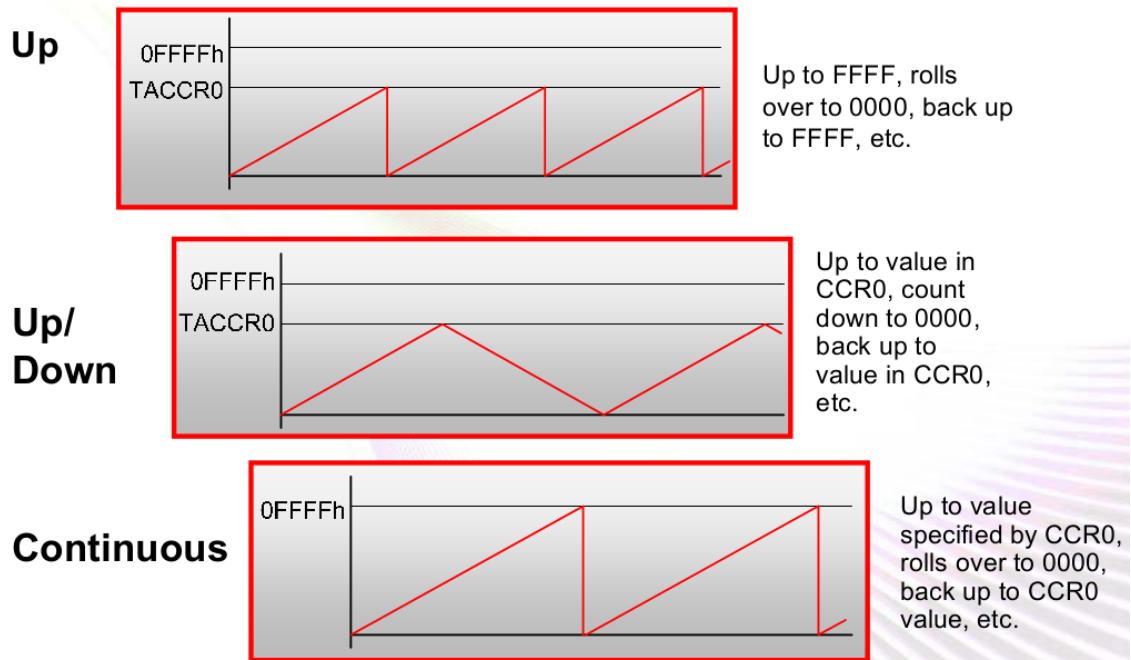


Figura 4.27. Esempio di funzionamento delle modalità di count.

# Capitolo 5

## Conclusioni

### 5.1 Conclusioni

#### 5.1.1 Consumi

Per valutare i consumi della piattaforma progettata si è deciso di utilizzare un condensatore di tipo super cap, il modello GZ215 prodotto dalla Cap-xx con capacità  $0.075F$ . Il motivo dell'utilizzo di un condensatore rispetto ad una batteria è dovuto principalmente alla necessità di misurazioni precise, che una batteria, a causa delle reazioni chimiche e della sua eccessiva capacità, non sarebbe stata in grado di rilevare.

Le misurazioni sono state effettuate caricando il super cap ad una certa tensione, calcolando l'energia immagazzinata (in Joule), tracciando il profilo di scarica (in Volt) e misurando la differenza di energia residua dello stesso dopo certo numero di trasmissioni.

Per le nostre misure si è considerato il range di tensione che va da 3.6V a 1.8V, ovvero la dinamica di tensione accettata dall'elettronica della nostra piattaforma. In seguito sono elencati i profili di scarica in varie situazioni. Ad esempio:

1. profilo di scarica con trasmissione del massimo numero di pacchetti consentiti in uno slot (Figura 5.1),
2. profilo di scarica con trasmissione di un solo pacchetto nel proprio slot (Figura 5.2),
3. profilo di scarica con lettura dell' $I^2C$  ad ogni ciclo di wake up (Figura 5.3),
4. profilo di scarica con lettura dell' $I^2C$  ad ogni ciclo di wake up e trasmissione del massimo numero di pacchetti consentiti in uno slot (Figura 5.4),
5. profilo di scarica con lettura dell' $I^2C$  ogni 3 cicli di wake up (Figura 5.5),
6. profilo di scarica con numero di slot attivi pari a 2, corrispondente a al caso in cui un nodo abbia solo un vicino (Figure 5.6 e 5.7).



Figura 5.1. Scarica del condensatore con trasmissione continua.

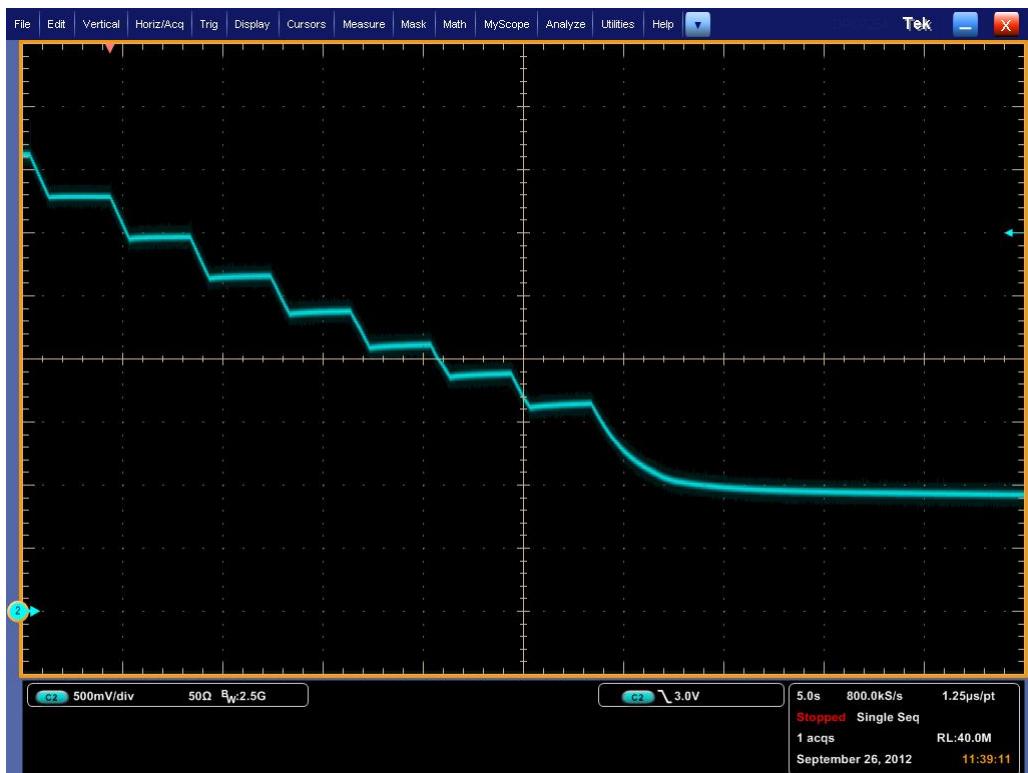


Figura 5.2. Scarica del condensatore con trasmissione di un solo pacchetto.



Figura 5.3. Scarica del condensatore con lettura dell' $I^2C$ .

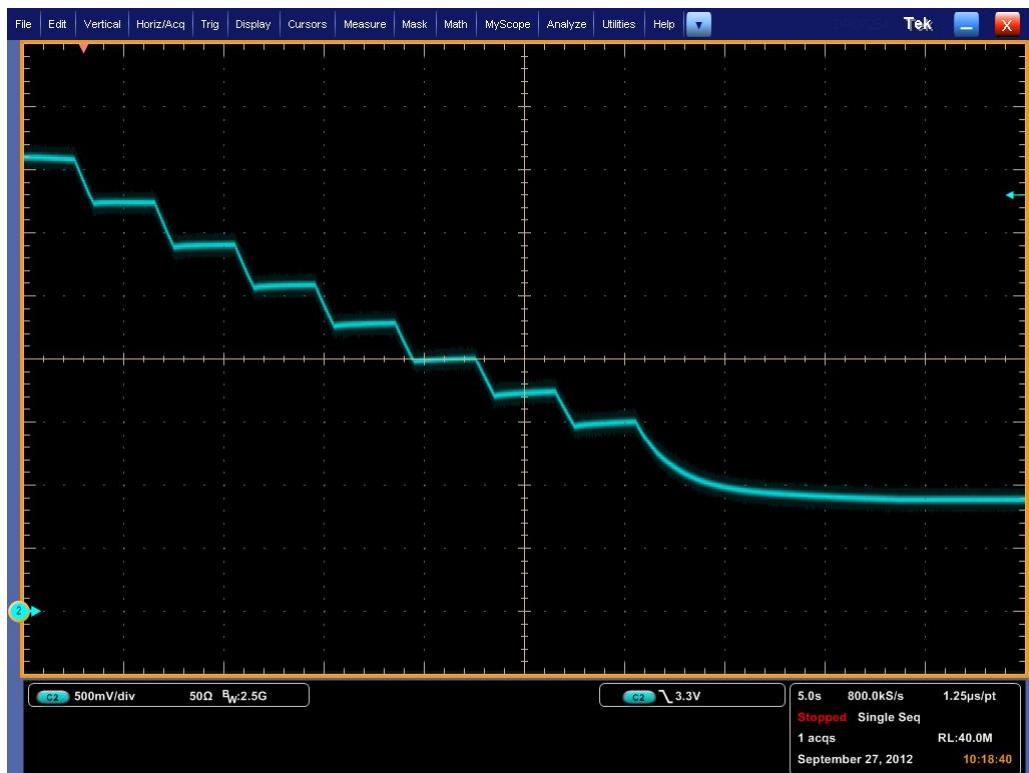


Figura 5.4. Scarica del consensatore con lettura dall' $I^2C$  e trasmissione continua.



Figura 5.5. Scarica del condensatore con lettura saltuaria dell' $I^2C$ .

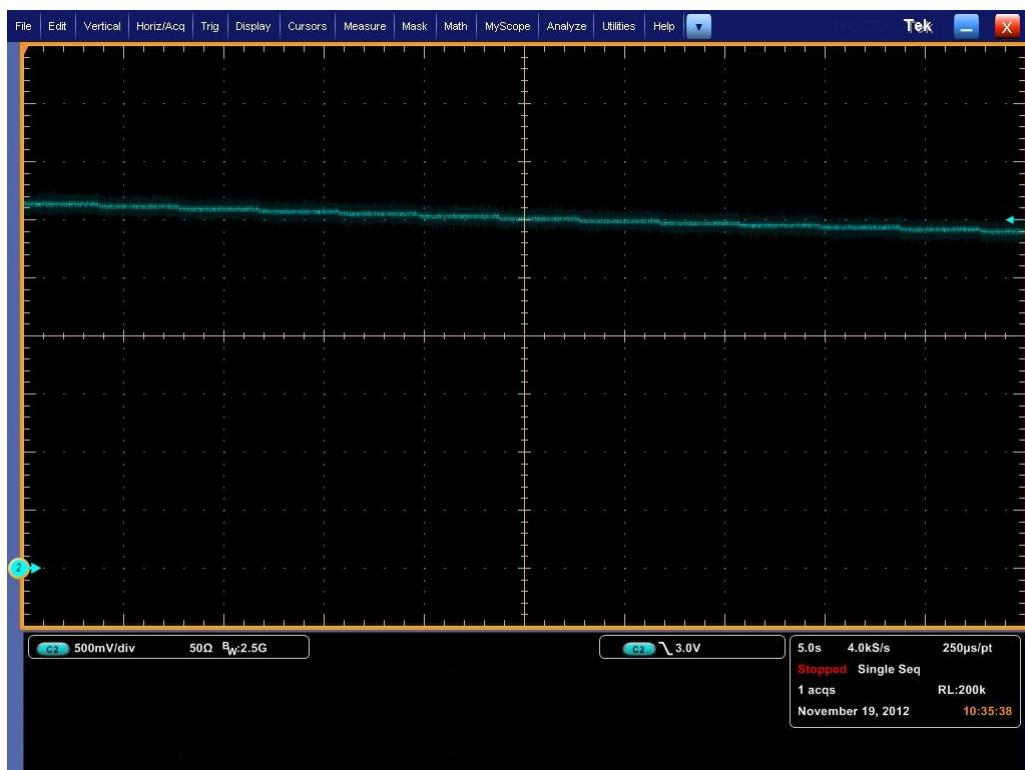


Figura 5.6. Scarica del condensatore con numero di finestre attive uguale a due.

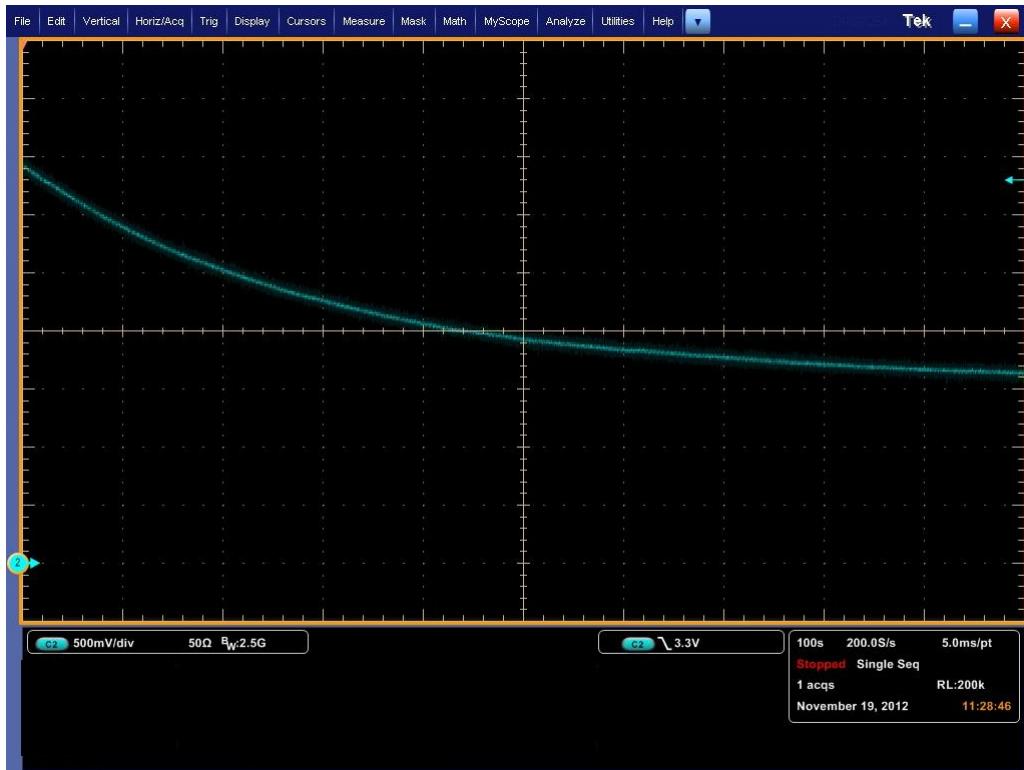


Figura 5.7. Scarica del consensatore con numero di finestre attive uguale a due (scala dei tempi pari a  $100s/div$ ).

Se la tensione tra i due lati del super cap, a causa di leakage interni, non è pari a  $Vcc/2$  si rischia di danneggiarlo irreparabilmente. Per questo motivo sono state inserite due resistenze in parallelo ad entrambi i suoi lati come si può vedere in Figura 5.8, facendo così in modo di fissare la tensione sul singolo lato ad un valore prossimo a  $Vcc/2$ .  $RL1$  ed  $RL2$  indicano le resistenze di interne (leakage) al super cap, mentre  $R1$  ed  $R2$  sono le resistenze inserite.

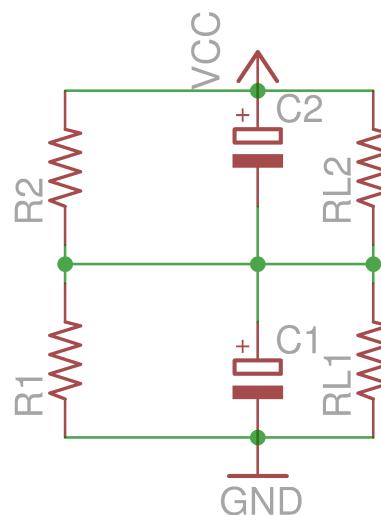


Figura 5.8. Circuito di protezione per il supercap.

Prendendo il leakage massimo dichiarato sul datasheet, pari a  $2\mu A$ , si è deciso di far fluire sulle resistenze una corrente pari a circa 10 volte il leakage. Supponendo inoltre la minima tensione (quindi  $1.8V$ ) si sono inserite due resistenze da  $110K\Omega$  portando così il leakage sulle resistenze ad un valore compreso tra gli 8 e i  $16\mu A$ . A causa di questo leakage tutti i consumi della piattaforma sono quindi approssimati per eccesso.

L'energia del super cap viene così calcolata secondo la formula:  $\frac{1}{2}CV^2$  con  $V_{max} = 3.6V$  e  $V_{min} = 1.8V$  ottenendo così un range di energia compreso tra  $12.15mJ$  e  $48.6mJ$ , e quindi un delta pari  $36.45mJ$ .

Per il power sourcing si suppone di avere a disposizione un generatore di tensione termoelettrico collegato ad un calorifero. Un esempio è il modello MPG-D751 della Micropelt, le cui dimensioni sono pari a  $4248\mu m \cdot 3364\mu m \cdot 1090\mu m$ , con le seguenti caratteristiche:

- produzione un valore di potenza compreso tra  $4mW$  e  $6mW$  con un  $\Delta T$  tra le superfici pari a  $20^\circ K$  come mostrato in figura 5.9,
- raggiungimento di tale  $\Delta T$  per almeno  $\frac{1}{3}$  della giornata,

Con queste caratteristiche di power sourcing si è ottenuto un valore di energia giornaliera disponibile  $U$  per ogni singolo sensore di circa  $144J$ , attraverso la formula (supponendo in media una produzione di  $5mW$ ):  $U = 5mWh * 3600 * 24 * \frac{1}{3}$

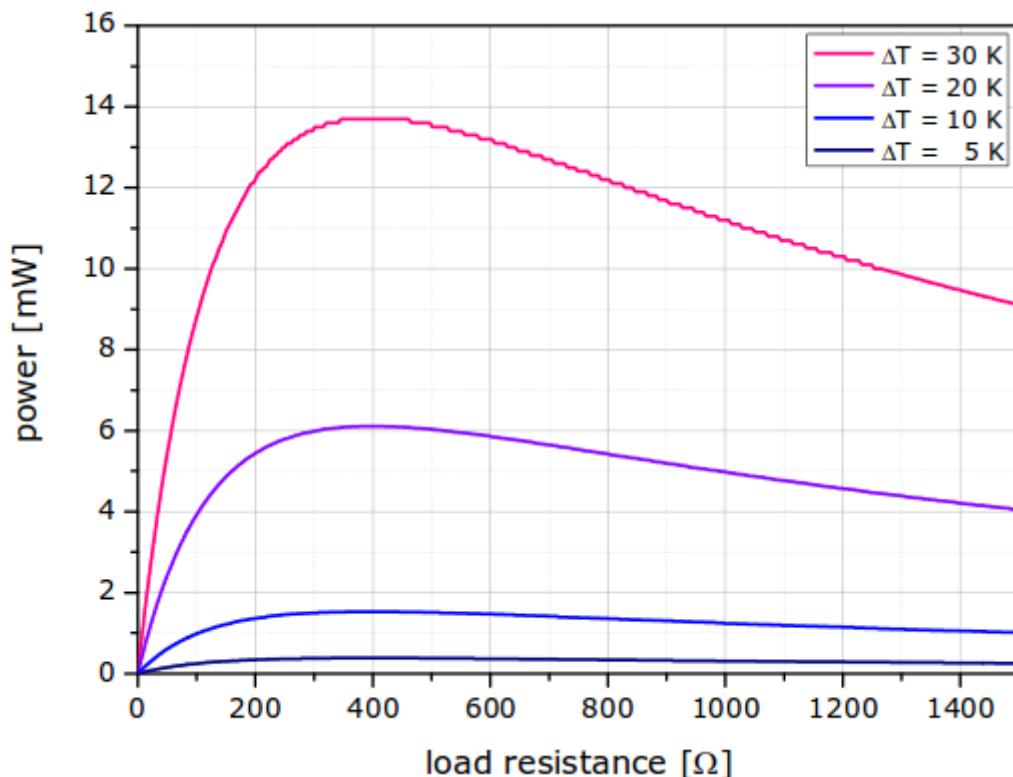


Figura 5.9. Grafico di potenza fornita dal generatore MPG-D751.

Supponendo la seguente situazione:

- trasmissioni a potenza massima ( $1dBm$ ),
- rete di 32 sensori in cui un solo nodo è il next hop dei rimanenti 30 (la topologia di rete mostrata in 5.12),
- la lettura dello stato di carica della batteria tramite  $I^2C$  ad ogni  $T_{ciclo}$ ,
- l'invio del massimo numero di pacchetti consentiti durante il proprio slot,

si ottiene un numero di invii pari a  $6 - 7$  prima della scarica del super cap, che corrisponde al worst case, la situazione di massimo consumo. Dividendo l'energia consumata per il numero di invii effettuati si ottiene un consumo pari a  $36.45/6 = 6.075mJ$  in media per ogni singola trasmissione. In caso si voglia fare un invio al minuto per tutto l'arco della giornata in queste condizioni, nel nodo più congestionato si consumerebbero quindi  $6.075*60*24 = 8748mJ$  che, comparati ai  $144J$  disponibili, sono circa  $1/16.5$ . In sostanza, nel caso peggiore, si potrebbe trasmettere una volta ogni  $3s$  rimanendo sotto la soglia di consumo massimo consentita dal power sourcing.

Nel caso invece di un nodo in ascolto solo di un vicino, il suo next hop, l'algoritmo converge verso il caso migliore, cioè una situazione in cui l'antenna rimane accesa solo per due slot: quello del nodo stesso e quello del next hop.

Il consumo totale del sistema in questa configurazione è pari a  $2/32$  del precedente, dato che si ascoltano solo 2 slot su 32. La Figura 5.6 mostra il grafico dei consumi con la stessa scala dei tempi delle precedenti misure, mentre la Figura 5.7 mostra la stessa situazione nel caso di scala dei tempi di  $100s/div$ . Si può notare l'esponenziale dovuto alla scarica del condensatore per via dei leakage interni e delle resistenze inserite per non danneggiarlo.

Supponendo invece gli stessi scenari di test, ma variando la potenza di invio (massima in precedenza) al valore minimo ammesso, ovvero  $-55dBm$  il consumo di corrente della RF antenna passerebbe da  $21.5mA$  a un valore di  $8.4mA$  permettendo un risparmio ulteriore del sistema di circa 2.5 volte.

La componente di consumo dell'MSP430, essendo ordini di grandezza inferiore rispetto alla RF antenna, è stata considerata trascurabile.

### 5.1.2 Test

Per testare il corretto funzionamento dell'algoritmo sono state prese in considerazione alcune topologie di reti considerate più importanti e rappresentative.

Da alcuni test effettuati in una zona completamente all'aperto, due nodi sono in grado di comunicare fino a una distanza di un centinaio di metri circa. Questa distanza, ovviamente, si riduce notevolmente in caso di disposizione dei nodi all'interno di un edificio, con ostacoli tra di essi e interferenze sulla frequenza di  $2.4\text{ Ghz}$  in cui comunicano (come le numerose reti wi-fi presenti ormai dappertutto).

Contando quindi la difficoltà nel testare una rete con un elevato numero di nodi e avendo a disposizione un numero esiguo di sensori, le reti effettivamente testate sono state leggermente più semplici di quelle qui presentate, pur mantenedo la stessa

topologia. L’obiettivo di testare la convergenza della rete e la sua robustezza è stato comunque raggiunto anche con un numero esiguo di sensori. Gli esempi presentati sono riferiti a una rete con 32 nodi.

La topologia più semplice di rete analizzata, illustrata in Figura 5.10, presenta una disposizione lineare dei nodi. In tale topologia, descritta precedentemente per illustrare l’assegnazione dinamica dei MAC, si suppone che ogni nodo percepisca solo il segnale dei suoi due vicini.

L’ultimo nodo,  $N_{31}$ , presenta la maggiore distanza possibile dal sink: un suo pac-

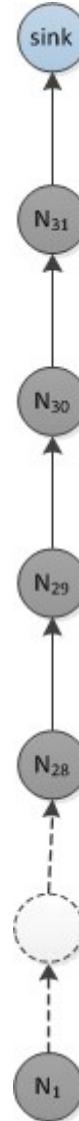


Figura 5.10. Disposizione lineare dei nodi.

chetto inviato verso il nodo centrale deve passare attraverso 30 sensori intermedi per giungere a destinazione. Tuttavia, nonostante la sua elevata distanza dal sink, ovvero 30 hop, la latenza della rete è molto bassa.

Un pacchetto inviato da  $N_{31}$  verso il sink, infatti, arriva a destinazione nell’arco della stessa finestra temporale, grazie a un’attenta assegnazione dei MAC, in valore crescente allontanandosi dal sink.

In caso il nodo  $N_i$  dovesse improvvisamente spegnersi, i suoi vicini,  $N_{i-1}$  e  $N_{i+1}$ , trascorso un tempo pari a  $n \cdot T_{ciclo}$  in cui non ricevono il suo beacon, rileveranno la sua scomparsa dalla rete.

$N_{i+1}$  dovrà modificare gli slot di cui deve stare in ascolto, dato che un nodo che lo considerava come next hop non trasmetterà più dati. Non dovendo, quindi, instradare più i dati di  $N$  verso monte, potrà tenere spenta l'antenna durante lo slot che gli era stato assegnato. Invece,  $N_{i-1}$  entrerà in uno stato di panico e, non percependo alcun percorso migliore per raggiungere il sink, progressivamente capirà di essere rimasto isolato dal nodo centrale. Questa informazione si propagherà verso valle, causando il progressivo disattivarsi dei sensori per non consumare inutilmente energia, come spiegato in precedenza nel funzionamento dell'algoritmo.

La seconda topologia di rete, illustrata in Figura 5.11, prevede che i nodi siano disposti in cerchio attorno al sink. Ogni nodo, anche se fosse in grado di percepire il segnale di un nodo vicino, non cambierebbe l'instradamento dei propri pacchetti, in quanto rileva il sink ad una distanza pari a un hop, perciò predilige sempre questa scelta.

L'eventuale caduta di un nodo non provocherebbe cambiamenti nella topologia della rete.

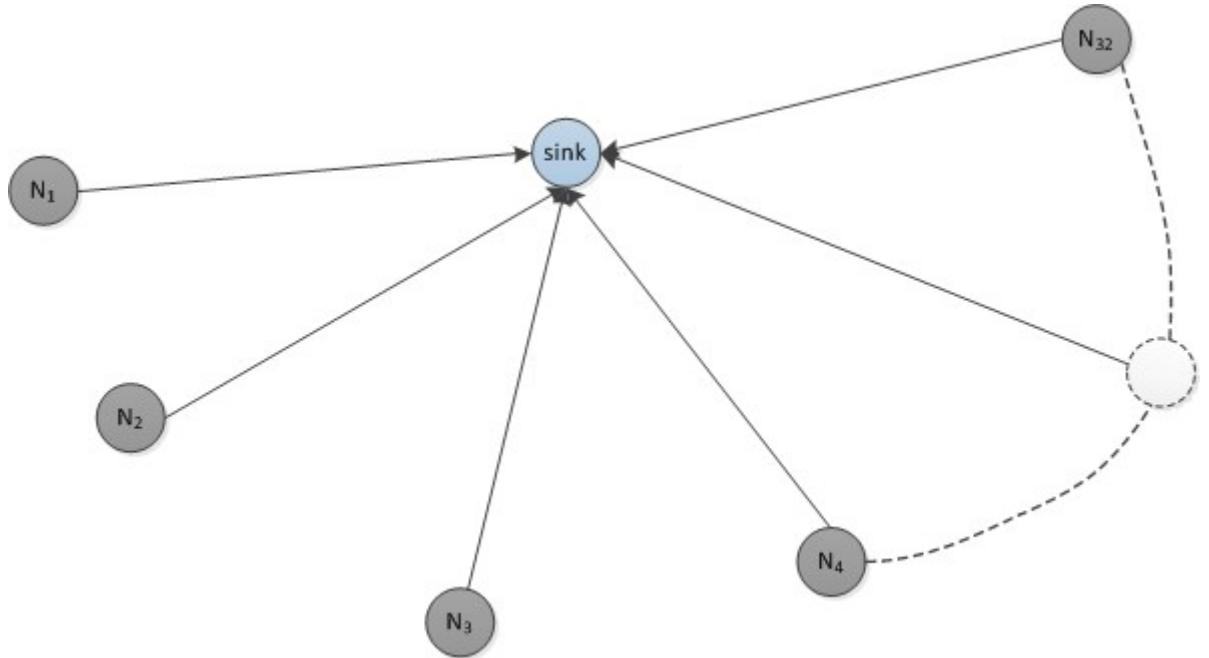


Figura 5.11. Disposizione dei nodi attorno al sink.

Un'ulteriore rete che è stata testata è illustrata in Figura 5.12. Tale topologia presenta il nodo  $N_{31}$  come next hop per tutti gli altri nodi della rete, i quali non sono in grado di percepire il segnale del sink.

Esso avrà il compito di instradare tutti i pacchetti della rete verso il nodo centrale. Tuttavia, il carico a cui è sottoposto non lo porta a essere un collo di bottiglia per quel che riguarda i consumi. Si potrebbe erroneamente pensare che  $N_{31}$  esaurisca la sua energia prima degli altri, lasciando il resto della rete isolata: i test sui consumi illustrati in precedenza dimostrano invece che un nodo, anche trasmettendo

continuamente, riesce a avere un consumo inferiore all’energia disponibile da una semplice fonte rinnovabile.

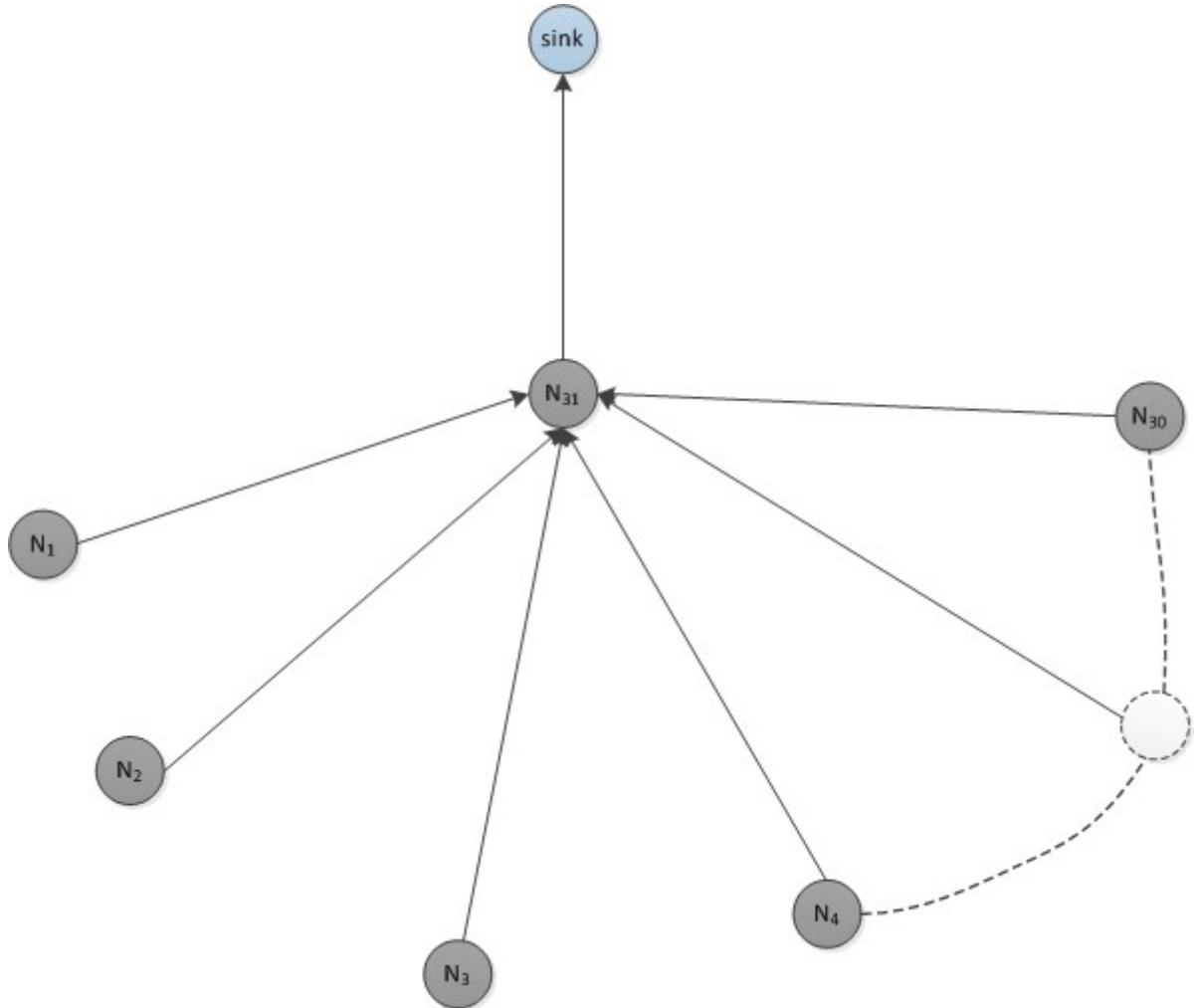


Figura 5.12. Disposizione dei nodi tale che tutti condividono lo stesso next hop.

La topologia di rete illustrata in 5.13 presenta due nodi ( $N_{31}$  e  $N_{30}$ ) che ricevono il segnale del sink e un terzo nodo ( $N_{29}$ ) che deve scegliere uno di essi come next hop.

Dato che entrambi presentano la stessa distanza dal nodo centrale, la scelta di  $N_{29}$  dipenderà dalla valore dello stato di carica di  $N_{31}$  e  $N_{30}$ , che entrambi annunciano nel proprio beacon.

Supponiamo che venga settato  $N_{31}$  come next hop: nel momento in cui la sua carica scenderà sotto il valore di  $N_{30}$ , rendendo il percorso verso il sink che lo attraversa non più ottimale,  $N_{29}$  cambierà immediatamente next hop, dato che continuerà a ricevere i beacon di  $N_{30}$ .

Questa situazione può quindi essere vista come se ci fosse un nodo di “backup” che può essere utilizzato istantaneamente se la carica del proprio next hop diventasse critica.

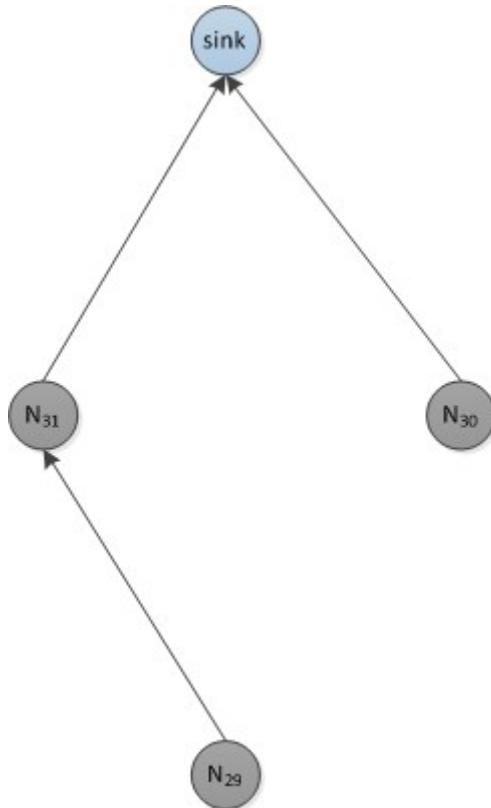


Figura 5.13. Disposizione dei nodi con la presenza di un next hop di “backup”.

L’ultima topologia di rete, illustrata in Figura 5.14, presenta una disposizione casuale dei nodi. Questa topologia può essere vista come l’unione di tutti gli esempi precedenti: quando i nodi vengono collocati in maniera casuale si possono presentare delle situazioni analizzate in precedenza, un esempio è il nodo  $N_{25}$ , il quale sceglie come next hop  $N_{29}$  perché presenta una minore distanza dal sink.

È presente anche un nodo isolato dalla rete, il nodo  $N$ , collocato in alto a destra: esso non percepisce il segnale di nessun altro all’interno della rete, perciò resterà in ascolto a intervalli di tempo decrescenti nell’attesa che vengano accesi dei nodi in sua prossimità, come spiegato nella sezione riguardante l’algoritmo.

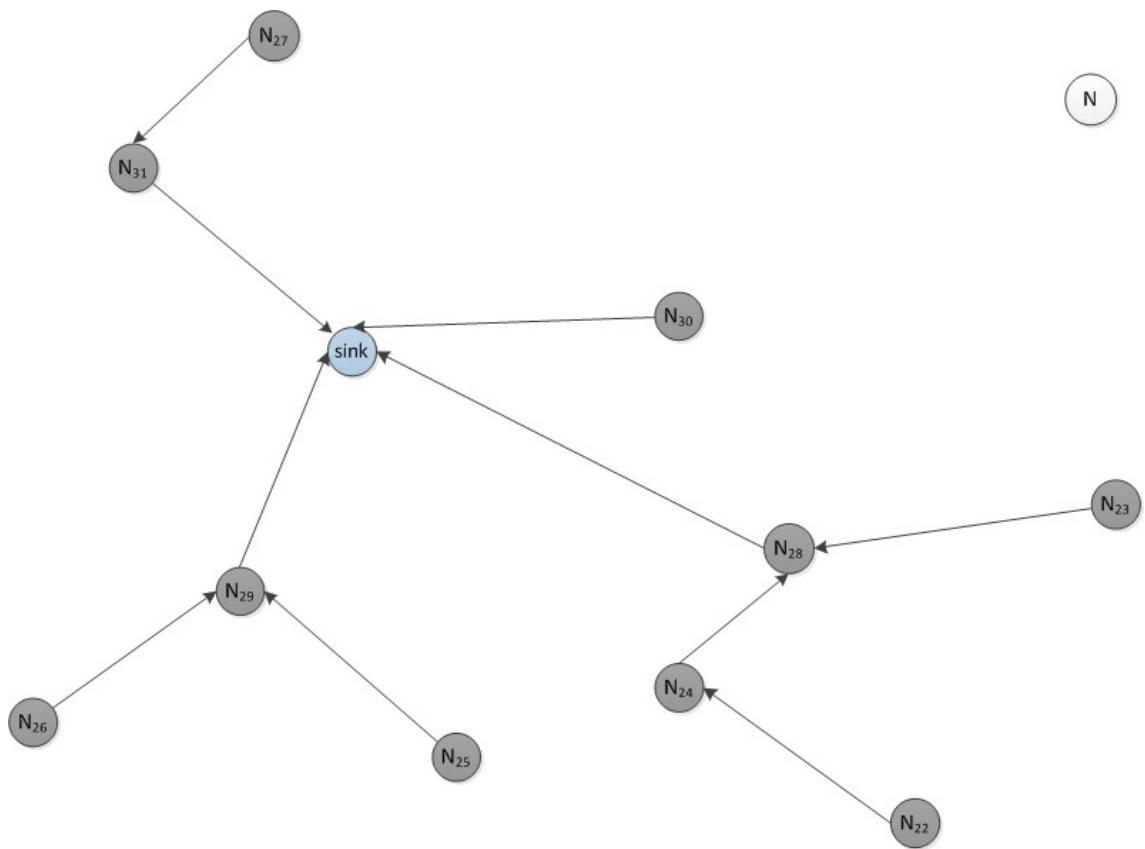


Figura 5.14. Disposizione casuale dei nodi.

# Bibliografia

- [1] Kemal Akkaya and Mohamed Younis. An energy-aware qos routing protocol for wireless sensor networks. In *Proc. of the IEEE Workshop on Mobile and Wireless Networks (MWN 2003)*, pages 710–715, 2003.
- [2] Kemal Akkaya and Mohamed Younis. Energy and qos aware routing in wireless sensor networks. *Cluster Computing*, 8(2-3):179–188, July 2005.
- [3] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325 – 349, 2005.
- [4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.
- [5] A. Boukerche. *Algorithms and Protocols for Wireless, Mobile Ad Hoc Networks*. Wiley Series on Parallel and Distributed Computing. Wiley, 2008.
- [6] David Braginsky and Deborah Estrin. Rumor routing algorithim for sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 22–31, New York, NY, USA, 2002. ACM.
- [7] Jae-Hwan Chang and L. Tassiulas. Maximum lifetime routing in wireless sensor networks. *Networking, IEEE/ACM Transactions on*, 12(4):609 – 619, aug. 2004.
- [8] Maurice Chu, Horst Haussecker, Feng Zhao, Maurice Chu, Horst Haussecker, and Feng Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16, 2002.
- [9] Maurice Chu, Horst Haussecker, Feng Zhao, Maurice Chu, Horst Haussecker, and Feng Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16, 2002.
- [10] Nicola Trivellin Daniele Beninato. Introduzione ai microcontrollori pic. <http://goo.gl/uYp5R>.
- [11] A. Elahi and A. Gschwender. *ZigBee Wireless Sensor and Control Network*. Prentice Hall Communications Engineering and Emerging Technologies Series. Pearson Education, 2009.
- [12] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. pages 147–163, 2002.
- [13] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 263–270, Seattle, Washington, USA, August 1999. ACM.

- [14] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 138–149, New York, NY, USA, 2003. ACM.
- [15] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(4):11–25, October 2001.
- [16] D. Gislason. *ZigBee Wireless Networking*. IT Pro. Elsevier, 2008.
- [17] Tian He, J.A. Stankovic, Chenyang Lu, and T. Abdelzaher. Speed: a stateless protocol for real-time communication in sensor networks. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 46 – 55, may 2003.
- [18] Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.
- [19] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 174–185, New York, NY, USA, 1999. ACM.
- [20] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 174–185, New York, NY, USA, 1999. ACM.
- [21] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, page 10 pp. vol.2, jan. 2000.
- [22] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MOBICOM*, pages 56–67. ACM, 2000.
- [23] Sung-Hwan Jung, Nakjung Choi, and Taekyoung Kwon. An iterative analysis of single-hop b-mac networks under poisson traffic. *Communications and Networks, Journal of*, 14(1):40 –50, feb. 2012.
- [24] Konstantinos Kalpakis, Koustuv Dasgupta, and Parag Namjoshi. Maximum lifetime data gathering and aggregation in wireless sensor networks, 2002.
- [25] F. Leens. An introduction to i2c and spi protocols. *Instrumentation Measurement Magazine, IEEE*, 12(1):8 –13, february 2009.
- [26] L. Li and J.Y. Halpern. Minimum-energy mobile wireless networks revisited. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 1, pages 278 –283 vol.1, jun 2001.
- [27] S. Lindsey and C.S. Raghavendra. Pegasus: Power-efficient gathering in sensor information systems. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 3, pages 3–1125 – 3–1130 vol.3, 2002.

- [28] Stephanie Lindsey, Cauligi Raghavendra, and Krishna Sivalingam. Data gathering in sensor networks using the energy\*delay metric. In *In Proc. of IPDPS Workshop on Issues in Wireless Networks and Mobile Computing*, pages 924–935, 2001.
- [29] A. Manjeshwar and D.P. Agrawal. Teen: a routing protocol for enhanced efficiency in wireless sensor networks. In *Parallel and Distributed Processing Symposium., Proceedings 15th International*, pages 2009 –2015, april 2001.
- [30] A. Manjeshwar and D.P. Agrawal. Apteen: a hybrid protocol for efficient routing and comprehensive information retrieval in wireless sensor networks. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 195 –202, 2002.
- [31] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 39–49, New York, NY, USA, 2004. ACM.
- [32] Injong Rhee, A. Warrier, M. Aia, Jeongki Min, and M.L. Sichitiu. Z-mac: A hybrid mac for wireless sensor networks. *Networking, IEEE/ACM Transactions on*, 16(3):511 –524, june 2008.
- [33] Injong Rhee, Ajit Warrier, Jeongki Min, and Lisong Xu. Drand: distributed randomized tdma scheduling for wireless ad-hoc networks. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '06, pages 190–201, New York, NY, USA, 2006. ACM.
- [34] V. Rodoplu and T.H. Meng. Minimum energy mobile wireless networks. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on*, volume 3, pages 1633 –1639 vol.3, jun 1998.
- [35] Narayanan Sadagopan, Bhaskar Krishnamachari, and Ahmed Helmy. The acquire mechanism for efficient querying in sensor networks. In *In IEEE International Workshop on Sensor Network Protocols and Applications (SNPA '03*, pages 149–155, 2003.
- [36] C. Schurgers and M.B. Srivastava. Energy efficient routing in wireless sensor networks. In *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, volume 1, pages 357 – 361 vol.1, 2001.
- [37] R.C. Shah and J.M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE*, volume 1, pages 350 – 355 vol.1, mar 2002.
- [38] K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie. Protocols for self-organization of a wireless sensor network. *Personal Communications, IEEE*, 7(5):16 –27, oct 2000.
- [39] Weilian Su and Ian F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 13(2):384–397, April 2005.
- [40] L. Subramanian and R.H. Katz. An architecture for building self-configurable systems. In *Mobile and Ad Hoc Networking and Computing, 2000. MobiHOC. 2000 First Annual Workshop on*, pages 63 –73, 2000.
- [41] Jana van Greunen and Jan Rabaey. Lightweight time synchronization for sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless*

- sensor networks and applications*, WSNA '03, pages 11–19, New York, NY, USA, 2003. ACM.
- [42] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *ACM MOBICOM*, pages 70–84, 2001.
  - [43] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31:2002, 2002.
  - [44] Suyoung Yoon, Chanchai Veerarittiphan, and Mihail L. Sichitiu. Tiny-sync: Tight time synchronization for wireless sensor networks. *ACM Trans. Sen. Netw.*, 3(2), June 2007.
  - [45] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, pages 129 – 136, 2002.
  - [46] Moustafa A. Youssef. A constrained shortest-path energy-aware routing algorithm for wireless sensor networks. In *Wireless Commun. and Networking Conference*, pages 794–799, 2002.
  - [47] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks. Technical report, 2001.
  - [48] Hongqiang Zhai, Younggoo Kwon, and Yuguang Fang. Performance analysis of ieee 802.11 mac protocols in wireless lans: Research articles. *Wirel. Commun. Mob. Comput.*, 4(8):917–931, December 2004.