Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Ottimizzazione Energetica di Sensori Strutturali

Relatore: Prof. Giovanni AGOSTA
Correlatore: Ing. Diego MELPIGNANO

Tesi di Laurea di:
Danilo Maria MARTINO Matr. 817684

Anno Accademico 2015–2016

Ringraziamenti

Ringrazio la mia famiglia, che mi ha sostenuto in questo percorso anche nei momenti più difficili, il team di Diego Melpignano e il professor Agosta, che mi hanno guidato verso la laurea, gli amici dell'università, con cui ho passato bei momenti durante e dopo le lezioni, gli amici di una vita, con cui ho trascorso momenti spensierati e sereni.

Sommario

La tesi si propone di creare un sistema che permette di aquisire dati accelerometrici di sensori wireless montati su strutture, minimizzando il consumo energetico, cercando di darne una stima. La strategia utilizzata è di tenere accesa la rete solo nel momento in cui vengono raccolti i dati che sono necessarie solo poche volte al giorno in momenti prestabiliti. ST Microelectronics ha fornito una piattaforma con microprocessore STM32 e radio Spirit1 che supporta Contiki OS, che permette la comunicazione IP tra questi sensori e una centralina. Questa piattaforma è stata revisionata e gli sono stati aggiunti driver per permettere ai sensori di potersi risvegliare risvegliare (RTC) e per entrare in modalità low-power. Poiché la metrica per la creazione di percorsi in multihop non funzionava correttamente, è stata proposta una nuova metrica basata sul valore dell'RSSI dei pacchetti in uscita, basata sul fatto che oltre un certo limite misurato, i pacchetti non vengono più ricevuti. È stato implementato, inoltre, un algoritmo di sincronizzazione dell'RTC, da cui dipende il risveglio dei sensori, da utilizzare per la rete finale, che propaga il timestamp in broadcast. Infine è stato creato l'intero sistema per la raccolta dati: nei sensori sono presenti dei server CoAP che permettono di gestire gli allarmi, che indicano gli istanti di tempo in cui i sensori si risvegliano, per richiedere il valore dell'accelerometro e per spegnere i sensori. Una centralina presenta il client CoAP che si occupa di individuare quando avviene il risveglio dei sensori per poi interrogarli e spegnerli quando tutti questi hanno risposto. Sono state fatte delle prime stime del consumo energetico a partire dal tempo necessario affinché i nodi rispondessero.

Indice

Elenco delle figure

Introduzione

Il lavoro di tesi si colloca nell'ambito del monitoraggio strutturale. Questo è un ambito che consiste nella creazione di un sistema che raccoglie e memorizza dati accelerometrici misurati da sensori montati sulla struttura da monitorare. Negli ultimi anni questo settore ha ricevuto una spinta, grazie all'introduzione di sensori MEMS, che hanno permesso di ridurre i costi dei dispositivi. I dati raccolti da una rete di sensori possono poi essere analizzati da esperti, che possono rilevare in anticipo la presenza di deformazioni dovute al passare del tempo. In questo caso possono essere presi provvedimenti prima che accadano eventi catastrofici o irreparabili, prolungando così la vita della struttura. Queste misure possono essere fatte poche volte al giorno, visto che le deformazioni avvengono su tempi lunghi. La possibilità che si vuole valutare è di utilizzare una rete di sensori dotati di radio alimentati da batteria ricaricabile con celle solari. Questa soluzione prevede costi inferiori di installazione rispetto ad una soluzione cablata, ma necessita attenzione dal lato energetico. L'idea che si vuole realizzare è di avere un sistema che permetta di tenere acceso ciascun sensore solamente per il tempo di raccogliere e trasmettere i dati raccolti, mentre per il resto del tempo tenere la radio spenta e il microcontrollore in modalità low-power. Inoltre, si prevede che i sensori possano essere fuori range rispetto alla centralina, che si occupa della comunicazione con il sistema di raccolta dati. Ciò necessita che i nodi fungano da router per instradare pacchetti provenienti da nodi più lontani dal master. Una volta che si ha un sistema funzionante, si vuole stimare il consumo energetico dei vari nodi che dipende direttamente dal tempo di *on* del sistema, dati il consumo quando i nodi sono accesi e quando in modalità low-power. Per valutare la fattibilità e l'efficacia di questa soluzione su un prototipo reale, si ha a disposizione una piattaforma STM32 Nucleo su cui è

stato portato il sistema operativo Contiki, non ancora completamente consolidata. Questo sistema operativo fornisce lo stack 6LoWPAN, che permette di utilizzare IP e protocolli superiori e protocolli come CoAP, che permettono di interrogare i sensori con pacchetti di pochi byte. Inoltre è presente RPL, algoritmo per l'instardamento di pacchetti in multi-hop.
Dopo aver studiato le varie tecnologie su cui si basa la piattaforma e le soluzioni già esistenti (Capitolo 1) il contributo che viene dato dalla tesi è:

- una revisione della piattaforma, che consiste nella creazione di driver per RTC, modalità low-power e per la EEPROM, e revisione del CSMA/CA (Capitolo 2);
- la creazione di una nuova metrica, che permetta di creare in modo veloce una rete affidabile e la creazione di un meccanismo per la sincronizzazione degli RTC dei vari sensori, da cui dipende il risveglio di ciascuno di essi (Capitolo 3);
- la creazione di un algoritmo per la raccolta e la memorizzazione dei dati nel momento in cui questi si risvegliano, con una prima valutazione del consumo energetico (Capitolo 4).

Capitolo 1

Stato dell'Arte

In questo capitolo sono presentate le tecnologie dello stato dell'arte ormai consolidate e successivamente sono presenti le soluzioni presenti nello stato dell'arte riguardo alla metrica, sincronizzazione dell'RTC e sull'ottimizzazione energetica.

1.1 Tecnologia

In questa sezione sono presentati lo Stack 6LoWPAN e i layer che lo compongono insieme al loro funzionamento nel caso di reti di sensori wireless. È presente inltre una breve presentazione di Contiki OS.

1.1.1 Reti LoWPAN

Le reti LoWPAN sono un argomento caldo di ricerca, viste anche le spinte dall'industria, soprattutto per quanto riguarda il monitoraggio. L'introduzione di nuove tecnologie di connettività low-power come le radio e il bluetooth e tecniche per il risparmio energetico hanno reso possibile l'utilizzo dell'alimentazione a batteria, facilitando l'installazione dei dispositivi. Inoltre l'introduzione della tecnologia MEMS, ha reso economici i sensori permettendone l'uso in grande scala in piattaforme con tecnologie di connettività wireless a basso consumo. I dispositivi di queste reti hanno le seguenti caratteristiche:

1. il costo in denaro è ridotto;

2. reti di questo tipo ne contengono migliaia in uno spazio limitato;
3. offrono connettività wireless;
4. devono consumare poco, poiché pensati per l'alimentazione a batteria, e di conseguenza hanno un range limitato;
5. devono consumare poco, poiché pensati per l'alimentazione a batteria, e di conseguenza hanno un range limitato;
6. si organizzano autonomamente nel creare percorsi per il routing.

1.1.2 IEEE 802.15.4

IEEE 802.15.4 [1] è uno standard per il protocollo che regola il livello fisico e il livello MAC di radio a bassa potenza. I dispositivi possono avere un indirizzo MAC a 16 bit, dinamicamente assegnato dal coordinatore, oppure uno a 64 bit del tipo EUID. Caratterizza i dispositivi in reduced function devices (RFD), dispositivi che non possono trasmettere ad altri RFD e Full Function Device (FFD), dispositivi che possono trasmettere sia RFD che ad altri "peer" FFD. Funziona in due modalità:

beacon-enabled in cui un dispositivo coordinatore di tipo FFD trasmette periodicamente una trama detta beacon, che serve per coordinare la trasmissione dei pacchetti dei nodi RFD adiacenti utilizzando il protocollo CSMA/CA slotted;

beaconless in cui i dispositivi non sono coordinati tra di loro e quindi non c'è distinzione tra FFD e RFD. Questi trasmettono quando necessario utilizzando il protocollo CSMA/CA unslotted.

Trama

La struttura della trama è stata studiata per ridurre al minimo la complessità, ma comunque garantisce comunque una trasmissione su un canale rumoroso. La trama fisica inizia generalmente con un preambolo e da un campo per la sincronizzazione, che servono per il ricevitore a sincronizzarsi con il trasmittente e stabilire se viene trasmessa una trama, che dipendono dal tipo di modulazione scelta. In seguito c'è il payload fisico che corrisponde alla trama al livello MAC. I campi principali della trama sono un campo

di controllo, un campo destinato all'indirizzamento e il payload. La trama di controllo contiene informazioni quali il tipo di trama, il tipo di indirizzo sia della sorgente, sia del destinatario e se richiede l'acknowledgement. Lo

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

Figura 1.1: Trama MAC generica [1].

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

Figura 1.2: Campo di controllo trama MAC [1]

standard definisce le seguenti trame MAC:

trama beacon utilizzata dal coordinatore;

trama dati utilizzata per trasmettere dati;

trama di acknowledgment utilizzata in risposta alla corretta trasmissione di un pacchetto;

trama di controllo utilizzata per la gestione dei trasferimenti tra dispositivi dello stesso tipo.

Layer Fisico

Si occupa di due compiti principali:

il servizio dati che permette di trasmettere la trama fisica;

la gestione del mezzo e della radio che permette di accendere e spegnere la radio, offre informazioni sullo stato del mezzo come il CCA (Clear

Channel Assessment), procedura che verifica che il canale è libero, generazione dei segnali di Energy Detection (ED), potenza del segnale ricevuto (RSSI), indicatore della qualità del link (LQI).

Layer MAC

Il livello MAC abilita la trasmissione al livello fisico e la ricezione di trame MAC (MPDU) passandole ai livelli superiori. Questo livello si occupa della gestione dei beacon, accesso al canale, gestione dei time-slots, validazione delle trame, l'invio di trame di acknowledgment, associazione e dissassociazione dei dispositivi.

1.1.3 IPv6

IP è lo standard del livello di rete che è utilizzato da miliardi di dispositivi che ogni giorno si connettono ad Internet. La versione 4 offre la possibilità di assegnare 2^{32} indirizzi unici a tali dispositivi. Con l'espansione avvenuta del numero di dispositivi connessi ad Internet, gli indirizzi disponibili progressivamente si stannoesaurendo. Nel Dicembre 1998 [7] è uscito lo standard per una nuova versione detta IPv6 che risolve questo problema. Mantenendo l'impostazione generale di IPv4, questa versione introduce delle funzionalità nuove e soprattutto una differente gestione degli indirizzi, che diventano 2^{128}. I campi del pacchetto IPv6 sono i seguenti:

Version di 4 bit, che indica la versione del Protocollo, in questo caso di valore 6;

Traffic Class di 8 bit, utilizzabile per distinguere diversi tipi di traffico nelle reti Differentiated Services;

Flow Label di 20 bit, utilizzabile per identificare un flusso di pacchetti;

Payload Length di 16 bit, che indica la lunghezza del pacchetto;

Next Header di 8 bit che identifica il tipo di header che segue il basic header(può essere di livello superiore come TCP o un extension header);

Hop Limit di 8 bit,che ha la stessa funzione del TTL di IPv4;

Source Address di 128 bit,indirizzo di sorgente;

Destination Address di 128 bit,Indirizzo di destinazione.

Indirizzi IPv6

IPv6 introduce numerose novità riguardo agli indirizzi. Nel pacchetto,il campo diventa da 32 a 128 bit, permettendo l'utilizzo di 2^{128} indirizzi. Nelle applicazioni è rappresentato diviso in 8 parti di 16 bit e ciascun byte è indicato con due cifre esadecimali. Le parti che compongono l'indirizzo sono divise dal simbolo separatore ':'. Inoltre la prima sequenza di 0 che si presenta nell'indirizzo può essere omessa insieme ai separatori che delimitano o sono all'interno della sequenza e sostituiti dai simboli "::". Come gli indirizzi IPv4, la grandezza del prefisso indicante la sottorete è specificato con il simbolo '/' seguito dal numero di bit di cui è composto il prefisso della sottorete. Sono stati creati dei meccanismi come il dual-stack per mantenere la compatibilità con macchine che supportano solo IPv4. Vengono introdotti i seguenti indirizzi speciali:

0:0:0:0:0:0:0:0/128 usato come indirizzo di sorgente quando il nodo non conosce altri suoi indirizzi;

0:0:0:0:0:0:0:1/128 indirizzo di loopback;

IPv4 mapped address (::FFFF:IPv4addr/96) utilizzato da dispositivi dual-stack per far comuncare nodi IPv6 con altri che supportano solo IPv4 ;

IPv4 Compatible Address (::IPv4addr/96) utilizzati per inserire indirizzi IPv4 in indirizzi IPv6 (deprecato);

Link Local Adress (fe80::/10) validi solamente sul link e non routabili, hanno un ruolo nelle operazioni di Neighbor Discovery e Router Discovery.

Indirizzi Site-Local (fc00::/7) che sono indirizzi privati utilizzati solo all'interno di una singola organizzazione;

Indirizzi Multicast (ff00::/8) servono per indirizzare un pacchetto a più destinatari. L'indirizzo è identificato, oltre che dal prefisso comune di

8 bit, dai campi "Scope" (S) di 4 bit, "Flags" (F) di 4 bit e "Group ID" di 112 bit.

ICMPv6

ICMP ha un importanza molto maggiore con IPv6. Questo protocollo è visto come estensione del protocollo IPv6. L'header è composto dai seguenti campi:

ICMP type che indica la versione di ICMP;

ICMP code che indica il tipo di pacchetto ICMP;

checksum campo per il controllo di errori;

message body che contiene il messaggio vero e proprio.

Svolge molte funzioni quali error reporting e diagnostica di rete (come vecchio ICMP), utilizzando per esempio i pacchetti Echo Request ed Echo Reply, risoluzione degli indirizzi di livello link (analogo ad ARP), con i messaggi Neighbor Solicitation e Neighbor Advertisement, e creazione di indirizzi globali con i messaggi Router Solicitation e Router Advertisement che autoconfigurano gli indirizzi IPv6. Svolge anche funzioni di controllo quali l'individuazione del router corretto (simile al redirect di IPv4), il controllo di indirizzi IPv6 duplicati e calcolo del PATH MTU, che serve per calcolare il massimo MTU tra due host, poiché IPv6 permette la frammentazione solo agli endpoint.

Autoconfigurazione indirizzi IPv6

IPv6 permette la configurazione degli indirizzi in modo automatico anche senza l'utilizzo di un server DHCP. Ciò è possibile grazie alla grandezza dell'indirizzo IPv6 che permette l'inserimento a suo interno dell'indirizzo MAC. Possono essere autoconfigurati gli indirizzi Link-Local attraverso il processo di Neighbor Discovery. Un nodo appena collegato alla rete invia il messaggio di Neighbor Solicitation con il suo ipotetico indirizzo Link-Local, per verificare se questo è in conflitto con altri indirizzi sullo stesso link. Se ciò avviene, il nodo con tale indirizzo risponderà con un Neighbor Advertisement e sarà necessaria la configurazione manuale. Per quanto riguarda l'indirizzo

globale, ciascun router periodicamente invia il pacchetto di Router Advertisemet, che contiene il prefisso della sottorete; ciascun nodo può richiederlo attraverso il messaggio di Router Solicitation. Il nodo configurerà così il suo indirizzo IP a partire dal prefisso ricevuto.

1.1.4 Lo stack 6LoWPAN

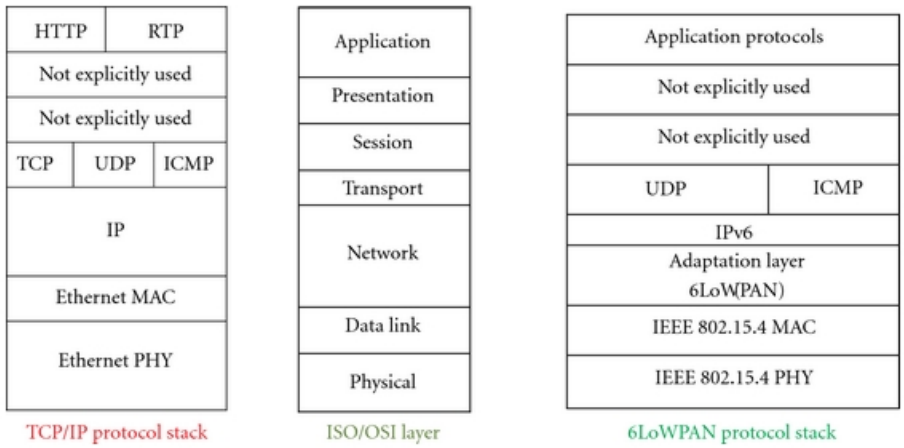

Figura 1.3: Lo stack 6LoWPAN confrontato con lo stack ISO/OSI e TCP/IP [10]

6LoWPAN (IPv6 over Low power Wireless Personal Area Network) [14] sta diventando uno standard per le reti IP su wireless network, studiato per essere utilizzato sul protocollo IEEE 802.15.4. L'uso di IP per questo tipo di reti permette numerosi vantaggi, rispetto ad altre sviluppate su IEEE 802.15.4:

- una connessione semplice con altre reti IP;
- la connessione IP è stata ampiamente testata ed è stato provato che scala. Il socket è usato in tutto il mondo;
- le tecnologie IP sono open e gratuite, con processi documentati e disponibili a tutti;
- Sono presenti tool per la diagnostica per protocollo IP e superiori.

Per connettere ad Internet questo tipo di reti sono sorti alcuni problemi. Il Maximum transmission Unit (MTU) per sodisfare lo standard IPv6 del MTU minimo di 1280 byte e un pacchetto massimo del protocollo IEEE 802.15.4 di 127 byte è necessario che il pacchetto venga compresso e/o frammentato. Le applicazioni dovranno comunque cercare di minimizzare il più possibile la grandezza del pacchetto per evitare il decadimento delle performance dovute alla frammentazione e quelle Web, che utilizzano protocolli come il SOAP, HTTP, XML che hanno payload di lunghezza nell'ordine dei kilobyte e che usano TCP, non sono utilizzabili.In questo contesto, invece, è necessario che i payload siano compatti e che fanno uso di UDP. Inoltre potrebbero sorgere nei problemi nel caso di attraversamenteo di Firewall e NAT: un pacchetto potrebbe avere problemi con questi tipi id entità di rete. Infatti la mancanza di un indirizzo IP statico o il blocco di porte scritte in formato compresso o di applicazioni non standard potrebbero provocare problemi di comunicazione. È necessario anche un layter per la connettività IPv4-IPv6: 6LoWPAN è basato solamente su IPv6. Sono quindi necessarie delle entità che facciano la traduzione degli indirizzi. 6LoWPAN è una tecnologia che si occupa di risolvere alcuni di questi problemi. Viene utilizzato all'interno di queste reti per abilitare l'utilizzo di IPv6, permettendo la frammentazione, la compressione dei pacchetti e dell'autoconfigurazione degli indirizzi.

Trama

Nello stack protocollare, come si può vedere in Figura 1.3, è un layer intermedio tra il protocollo MAC IEEE 802.15.4 e il layer IPv6 e a livello del pacchetto è visto come una serie di header che racchiudono informazioni per le diverse funzioni. Generalmente al livello inferiore viene utilizzata la modalità beaconless, ma non preclude l'utilizzo dei beacon. Il tipo di header 6LoWPAN è identificato dal primo byte e più precisamente dai primi due bit che dividono gli header in quattro gruppi principali:

- Pacchetto non appartenente a LoWPAN (00);
- Header per la compressione (01);
- Mesh header (10),

- Header per la frammentazione (11).

Il Mesh Header è utilizzato nel caso in cui il routing avvenga a livello MAC, memorizza gli indirizzi di origine e destinazione, necessari poiché ad ogni hop gli indirizzi cambiano. I rimanenti bit dell'ottetto di ogni tipologia contengono informazioni aggiuntive che dipendono dal tipo di header.

6LoWPAN: Architettura di rete

Le entità di una rete 6LoWPAN sono le seguenti:

- Border o Edge Router, nodo che ha un'interfaccia comune con gli altri nodi e almeno un'altra di altro tipo (per esempio Ethernet). Questa entità serve per fare da ponte tra una rete 6LoWPAN e si occupa di gestire l'indirizzamento globale dei nodi, diffondendo il prefisso per tutti i nodi della rete. Si occupa anche di rimuovere il layer 6LoWPAN nel caso il pacchetto esca dalla rete LoWPAN, e viceversa. Inoltre ha la possibilità di gestire la traduzione degli indirizzi per il supporto di IPv4;
- Router entità che permette di fare forward di pacchetti che non sono destinati a lui;
- Host che processa pacchetti indirizzati a lui.

Come mostrato in Figura 1.4, tre sono le topologie di reti 6LoWPAN:

- rete semplice, in cui è presente un solo edge router;
- rete estesa, in cui ci sono più edge router, collegati tra loro attraverso un link esterno alla rete 6LoWPAN;
- rete AD-HOC, in cui non è presente un edge router e quindi non è collegata con l'esterno.

6LowPAN: Funzioni

Autoconfigurazione degli indirizzi 6LoWPAN semplifica il modello di costruzione degli indirizzi IPv6: richiede che ciascuna interfaccia 6LoWPAN abbia un un ID a 64 bit derivante dall'indirizzo MAC di 64 o 16 bit a cui

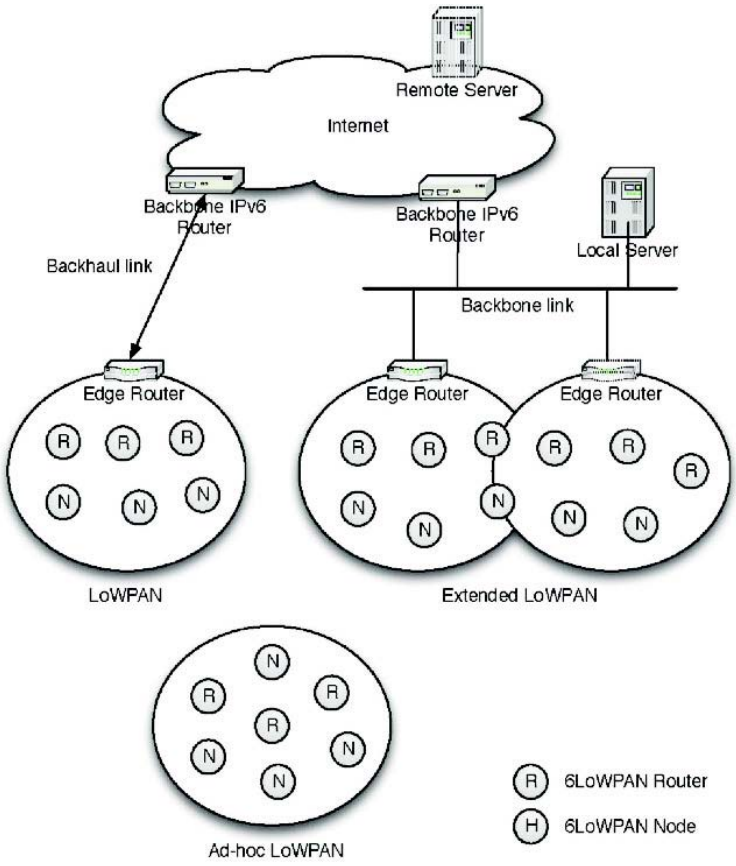

Figura 1.4: Tipi di rete 6LoWPAN [2]

viene aggiunto un prefisso a 64 bit. Per la creazione dell'indirizzo Link-Local viene utilizzato il prefisso fe80::/64, mentre i prefissi che vengono annunciati dai Router devono essere a 64 bit. Inoltre fornisce una versione Neighbor Discovery detta 6LoWPAN-ND, in cui non sono presenti i messaggi Neighbor Solicitation e Neighbor Advertisement, che servono per verificare se è presente un conflitto tra indirizzi, ma ciò è fatto a livello di edge router. Un nodo configura ottimisticamente il suo indirizzo Link-Local. Dopo aver ricevuto il Router Advertisement dall'edge router Invia un messaggio detto Node Registration a cui segue il messaggio di Node Confirmation in caso di successo. In caso l'edge router non sia in vista, il messaggio viene propagato fino ad esso.

Compressione dell'Header siccome per pacchetti piccoli la compressione attraverso algoritmi del tipo gzip risulta poco efficiente, si utilizza il fatto

che il pacchetto racchiude informazioni ridondanti all'interno delle varie componenti. Per esempio la versione IP ha sempre il valore "6" e può non essere inviata, i campi "Traffic Class" e "Flow Label" sono spesso inutilizzato e quindo con il valore 0 potendoli comprimere utilizzando un campo da un solo bit e il campo "Length" può essere inferito dalla lunghezza del rimanente pacchetto IP. Ciò che viene fatto è di ridurre la grandezza delle informazioni facendo corrispondere dei campi dell'header di pochi bit ad informazioni che ricorrono spesso e che in un normale pacchetto IPv6 occuperebbe qualche byte. I campi che non vengono compressi sono scritti successivamente all'header corrispondente. Gli header che vengono compressi sono quelli IP e UDP. La compressione è di due tipi: stateless o context-based. Nel primo caso il campo "Next Header" con i valori che indicano ICMP, UDP o TCP corrispondono a valori del campo "NH". Gli indirizzi, sia di destinazione che sorgente possono essere omessi o elisi della metà superiore o inferiore. Per la compressione dell'header UDP è presente un altro header 6LoWPAN che permette di comprimere porte hanno un valore compreso tra 61616 e 61631, sia destinazione che sorgente omettendo i primi tre byte. Nel caso in cui la compressione sia context-based il campo "Hop Limit" può essere omesso ed indicato da un flag se assume valori rilevanti come 64, 255 o 1, mentre il campo "Next Header" può essere compresso utilizzando il campo di 8 byte LOWPAN_NHC seguito dai campi compressi. Per quanto riguarda gli indirizzi, può essere compresso a 64 bit o a 16 bit oppure omesso, sia quello sorgente sia quello di destinazione. Nei primi due casi può essere fatto riferimento all'indirizzo Data-Link per ricostruirlo. Può essere specificato se questo è da interpretare come stateful, che indica un indirizzo globale che può essere ricostruito con il prefisso annunciato dall'edge router, oppure link-local (stateless). Inoltre permette la compressione degli indirizzi multicast IPv6. La compressione un campo dell'header 6LoWPAN indica il tipo di indirizzo multicast, e in seguito viene scritto la parte variabile. L'header UDP viene compresso quando il campo NH è 1 utilizzando lo specifico LOWPAN_NHC. Il campo compresso ha id '11110', un campo "Checksum" di 1 bit che indica se è eliso oppure no e un campo "Port" di due bit che indica se la porta sorgente o la porta destinazione sono elise ad 8 bit, nel caso in cui abbia il primo byte pari al valore 0xf0 oppure elise entrambe nel caso in cui i primi 12 bit abbiano il valore 0xf0b. Il campo "Length" viene omesso

poiché dedotto dalla lunghezza rimanente. Per entrambe le compressioni, i campi non compressi sono presenti in ordine successivamente a tali header.

Frammentazione e riassemblaggio le informazioni riguardo la frammentazione sono collocate nell'apposito header 6LoWPAN in ciascun frammento anche nel caso in cui i pacchetti arrivano in disordine. In questi vi è sempre specificata la lunghezza totale di 11 bit preceduto dal tipo di frammento ricevuto (5 bit), così da poter allocare al ricevimento del primo frammento un buffer che lo contenga completamente. In seguito è contenuto un tag di 16 bit che identifica l'intero pacchetto riassemblato. Il campo "Datagram Offset" di 8 bit è inserito nel caso dei frammenti successivi al primo.

1.1.5 RPL

RPL (Routing Protocol over Low-Power and Lossy Network) è lo standard proposto da IETF per le reti con perdite di pacchetto significativamente maggiori rispetto alle reti IP tradizionali. È un protocollo di routing di tipo Distance Vector e basato sugli indirizzi del Layer IP. Il protocollo crea uno o più grafi aciclici direzionati orientati verso una destinazione (DODAG), cioè degli alberi, che formano un'istanza. Una stessa rete può contenere più istanze, ciascuna delle quali può creare cammini differenti rispetto alle altre. I nodi 'pozzo', detti anche Radice, possono essere, per esempio, gli edge router 6LoWPAN mentre gli altri nodi sono dei router, oppure nodi foglie. Ogni istanza ha la propria Funzione Obiettivo (OF) con cui si calcola la metrica del link con i propri vicini e il percorso migliore attraverso il Rango (Rank) ricevuto dai propri vicini. La OF deve essere strettamente monotona crescente ed avere un incremento minimo e massimo ad ogni hop stabilito a priori. Ciò serve per evitare che si formino dei cicli. Vengono identificate due tipi di rotte: rotte all'insù, che vanno da un qualsiasi nodo ad un nodo radice e rotte all'ingiù, che vanno dalla radice ad un qualsiasi nodo. Le rotte all'insù sono stabilite da ciascun nodo eleggendo un proprio vicino detto Parent, da cui passa la migliore rotta che risulta dalla funzione obiettivo. Per quanto riguarda le rotte all'ingiù, RPL può agire in due modalità:

Non-Storing in cui il nodo root contiene tutte le tabelle di routing di tutti i nodi verso tutti gli altri. Un pacchetto IPv6 indirizzato dalla radice

a un qualsiasi nodo deve contenere tutti i nodi intermedi da cui passa il pacchetto per raggiungere la destinazione utilizzando un *Extension Header* IPv6 detto "Source Routing";

Storing Mode ciascun nodo mantiene le proprie tabelle di routing verso un qualsiasi nodo.

RPL [4] introduce l'indirizzo multicast "RPL All-Nodes"(ff02::1a), che serve per inviare il messaggio a tutti i nodi nel range del nodo inviante e corrisponde al broadcast radio a livello MAC. I messaggi del protocollo sono del tipo ICMP6 e sono i seguenti:

DODAG Information Object (DIO) contiene le informazioni sul grafo a cui appartiene (Istanza, DODAG, versione del DODAG, Rango, MinInc, MaxInc), e il prefisso del router;

DODAG Information Solicitation solitamente inviato in broadcast radio, serve per scoprire quali sono i propri vicini. Chi lo riceve invia in broadcast il DIO con le proprie informazioni.

DODAG Advertisement Object (DAO) contengono l'indirizzo IP della rotta che il nodo inviante raggiunge utilizzando il campo *Target Information Option*. Nel caso la modalità sia *Non-Storing*, l'informazione viene inviata fino alla radice senza che i nodi ne tengano traccia, mentre nel caso che la modalità sia del tipo *Storing*, viene inviato al proprio parent, che memorizzerà le informazioni, creando una rotta e propagato fino alla radice. Nel primo caso la radice si occupa di creare le tabelle di routing,mentre nel secondo caso ciascun nodo ha le informazioni sul next-hop.

DODAG Advertisement Object Acknowledgment (DAO-ACK) opzionale, inviato in risposta ad un DAO.

1.1.6 CoAP

Una delle necessità che ha una rete di sensori é che le informazioni scambiate a livello applicativo abbiano un payload ridotto e quindi necessitano anche l'uso di UDP. CoAP (Contrained Application Protocol) risponde a

queste esigenze. Un server CoAP [3], che tipicamente è in ascolto sulla porta 5683, utilizza un motore REST per pubblicare una serie di risorse virtuali, che tipicamente corrispondono a delle variabili a cui si fa riferimento attraverso un percorso URI. È possibile accedere, inizializzare, modificare o cancellare una risorsa attraverso i metodi GET, POST, PUT, DELETE, implementati sul server. I messaggi sono identificati da un MessageID, e contengono dei campi che indicano il tipo di richiesta o di risposta che sia ricevuta. Opzionalmente contengono anche campi come il payload o il percorso URI. I messaggi di richiesta possono essere di due tipi: da confermare e da non confermare. Nel primo caso nel caso un server CoAP riceva una richiesta deve inviare un messaggio di acknowledgement al client richiedente. La trama di acknowledgement è utilizzata in alcuni casi anche per inviare la risposta ad un determinato metodo. Inoltre nel caso il server necessiti più tempo per generare una risposta, può inviare un acknowledgement vuoto e poi ricevere la risposta una volta pronta. Nel caso in cui un messaggio necessiti di essere confermato, al messaggio viene associato un timer, allo scadere del quale il messaggio viene inviato nuovamente, mentre il timer raddoppia la sua durata fino ad un massimo di ritrasmissioni stabilito a priori. Inoltre è possibile che un client CoAP possa mettersi in osservazione di una determinata risorsa ed essere informato di ogni sua modifica. Viene seguito il prototipo di tipo Publish-Subscribe: un server CoAP pubblica le risorse che sono osservabili e uno o più client possono sottoscrivere a tale risorsa. Quando questa viene modificata, viene inviato a coloro che hanno sottoscritto a tale risorsa il suo nuovo valore.

1.1.7 Contiki OS

Contiki OS è uno dei più diffusi sistemi operativi per piattaforme embedded con scarse risorse. Le piattaforme con differenti microcontrollori che tuttora lo supportano sono molte. Offre una serie di funzionalità di base come timer, un sistema di processi con kernel non-preemptive. Supporta l'utilizzo del protocollo IEEE 802.15.4 per le radio e su di esso è possibile utilizzare lo stack Rime che ha una serie di primitive a livello MAC per inviare pacchetti, oppure è possibile utilizzare lo stack 6LoWPAN. In questo caso è possibile creare percorsi di routing in due modi: con l'utilizzo del protocollo di Neighbor Discovery di IPv6 oppure attraverso il protocollo RPL.

Inoltre fornisce delle applicazioni che possono essere compilate ed inserite nel file binario come per esempio CoAP, MQTT.

1.2 Metrica

Una caratteristica necessaria delle reti di sensori wireless è la possibilità di comunicare con i vari nodi attraverso un sistema per l'instradamento dei pacchetti che passi attraverso altri nodi intermedi. Sono stati implementati diversi algoritmi di routing, in prevalenza di tipo Distance Vector, ma la vera sfida è la scelta di una metrica, che permette la creazione di percorsi secondo una determinata politica. Una metrica definisce una funzione obiettivo, serve a determinare la distanza tra un nodo ed un altro di un determinato percorso e un modo per determinare il valore del link. Le tipologie di metriche identificate dal ROLL (Routing Over Low power and Lossy networks working group) sono le seguenti:

- basate su caratteristiche del link o del nodo;
- statiche o dinamiche;
- qualitative o quantitative.

Le tipologie di metriche identificate sono le seguenti [14]:

- CPU del nodo;
- Memoria del nodo;
- Energia del nodo;
- Carico sul nodo;
- Throughput del link;
- Latenza del link;
- Affidabilità del link;
- Link coloring.

Una metrica semplice è l'Hop Count: ciascun percorso da un nodo ad un altro che viene scelto è quello che ha il minor numero di hop. Questa scelta riduce il Round Trip Time ma non tiene conto della qualità dei link. Una metrica tra le più utilizzate per l'affidabilità del link è ETX (Expected number of Trasmissions). Questa metrica calcola l'affidabilità del link calcolando il numero medio di trasmissioni che un nodo necessita fare per inviare con successo ad un altro suo vicino. ETX di un link è definito come: $\frac{1}{df \times dr}$, dove *df* indica la probabilità di successo per inviare un pacchetto contentente dati e *dr* è la probabilità che venga ricevuto l'acknowledgement. Il percorso la cui somma degli ETX dei vari link è per raggiungere un certo nodo è minore sarà quello scelto. Questa metrica dà maggior affidabilità, ma può formare percorsi più lunghi. Inoltre necessita maggiore memoria per mantenere una lista dei vicini. Inoltre sono necessari alcuni messaggi unicast tra i vari vicini affinchè i vari nodi calcolino i valori di ETX. Da menzionare inoltre la metrica che riguarda l'energia residua di un nodo: ciascun nodo espone la propria energia residua inviando in broadcast, e il percorso scelto è quello che ha nodi con più energia. In letteratura esistono è un argomento caldo e sono state proposte numerose metriche: per esempio L. H. Chang et al. [6] propongono una metrica che è la somma pesata da un coefficiente α dell'ETX e dell'energia rimanente secondo la formula

$$\alpha \times \frac{ETX}{MaximumETX} + (1 - \alpha)(1 - \frac{RemainingEnergy}{MaximumEnergy})$$

Altri lavori come Michele Rondinone et al. [13] indicano la bontà dell'utilizzo di metriche basate su indicatori come l'RSSI e LQI, evidenziandone la correlazione tra questi e il Packet Reception Rate. Tsung-Han Lee et al. [11] estraggono l'ETX a partire dal valore medio di RSSI ricevuto.

1.3 Sincronizzazione dell'RTC

1.3.1 Perché sincronizzare

La necessità di sincronizzare nasce dal fatto di sapere quando avviene un tale evento registrato da un dispositivo nel mondo reale. Il dispositivo avrà un suo orologio interno con un proprio orario, che differirà dal tempo reale. È definito offset la differenza tra il tempo misurato da un certo orologio ed

un altro nello stesso istante, mentre lo skew e la derivata di questa differenza e sta ad indicare la differenza delle velocità con cui avviene questo offset. La velocità con cui questo avviene dipende dalla precisione dell'orologio di entrambi i nodi. Settare direttamente il tempo presente in un pacchetto per la sincronizzazione genererebbe un delay dovuto a vari fattori quali il tempo di trasmissione, il tempo per la ricezione e il tempo per processare il pacchetto. Sono presenti numerosi protocolli sia per i normali computer che per reti di sensori, wired e wireless, per sincronizzare gli orologi dei vari dispositivi presenti nella rete.

1.3.2 Protocolli

NTP e SNTP

Il protocollo più diffuso per la sincronizzazione dell'RTC dei computer di tutto il mondo è NTP. Questo protocollo si appoggia allo stack TCP/IP e utilizza la porta UDP 123 sia per la sorgente che per la destinazione. La trama NTP contiene le informazioni quali il livello di sincronizzazione, il tipo di clock sorgente, la precisione e vari timestamp che servono per la sincronizzazione. I timestamp sono formati da una parte intera di 32 bit che indica il numero di secondi trascorsi dal 1 Gennaio 1970 e da una parte frazionaria che indica la frazione dei secondi. Per sincronizzare con precisione un orologio sono necessari uno scambio di pacchetti fra un client ed un server NTP per ottenere i seguanti valori:

- Origin Timestamp T_1 : istante di invio della richiesta NTP dal client;
- Recive Timestamp T_2 : istante di arrivo della richiesta al server;
- Transmit Timestamp T_3 : istante di trasmissione della risposta dal server;
- Destination Timestamp T_4 : istante di arrivo della risposta al client.

Questi servono per stimare l'offset del clock attraverso la formula

$$offset = \frac{T_2 - T_1 + T_3 - T_4}{2}$$

. Una versione sempificata del protocollo chiamata SNTP, necessita solo del Transmit Timestamp: l'offset non viene calcolato e viene impostato dal ricevente il Timestamp ricevuto, a scapito, quindi, della precisione. Questo tipo di protocollo permette di sincronizzare i vari nodi uno alla volta attraverso scambi di messaggi unicast tra un server NTP ed il client.

Reference Broadcast Syncronization

Questo protocollo è utilizzato soprattutto nelle reti wireless in cui sono utilizzati i beacon. Il nodo che invia i beacon periodicamente invia il tempo ai propri vicini e questi, una volta ricevuto, si scambiano i tempi in cui è stato ricevuto e viene utilizzata la regressione lineare per calcolare l'offset e lo skew rispetto ad un vicino. In una rete multi-hop, nel caso in cui vengano scambiati dei pacchetti che contengano valori dei clock, questi vengono convertiti utilizzando i valori calcolati.

Timing-Sync Protocol for Sensor Networks

È un protocollo per sincronizzare un'intera rete wireless. Viene creato uno spanning tree cosicchè ciascun nodo interroga il proprio parent, che gli fornisce il timestamp e calcola l'offset attraverso il round trip time tra l'invio della richiesta e l'arrivo della risposta.

Flooding Syncronization Time Protocol

Questo protocollo permette la sincronizzazione dell'intera rete wireless. Ciascun nodo ha un ID e quello con l'ID minore sincronizza l'intera rete, inviando in broadcast il proprio timestamp e ritrasmesso dai riceventi, aggiornando il timestamp. Utilizzando il timestamp di arrivo e la regressione lineare viene sincronizzato il clock. Viene utilizzato il livello MAC per ridurre il delay.

1.3.3 Risparmio energetico e raccolta dati per reti di sensori wireless

Nelle reti di sensori sono essenzialmente utilizzati per risparmiare energia in reti di sensori wireless, ovvero il dutycicling e lo spegnimento a livello applicativo. Il primo è un meccanismo a livello MAC indicato dal protocollo IEEE 802.15.4, dove viene definito un periodo e i sensori rimangono accesi una piccola percentuale di questo periodo. Questo tipo di meccanismo è implementato in Contiki OS ed è chiamato ContikiMAC [9]. La radio, attraverso un interrupt legata al Realtime Timer di Contiki OS, passa da uno stato di low-power allo stato di ricezione all'inizio del periodo e viene eseguito il Clear Channel Assessment, che controlla se il valore di RSSI ha raggiunto una certa soglia. Attraverso la misurazione del tempo di questo segnale si individua se è un pacchetto oppure rumore. Nel caso in cui sia un pacchetto la radio rimane in ricezione e verifica se il pacchetto è indirizzato ad esso e se sì invia un messaggio di Acknowledgement. Altrimenti la radio passa nuovamente nello stato di low-power. Questo è un modo molto efficace per risparmiare energia, adatto soprattutto per registrare eventi provenienti dall'ambiente esterno in modo casuale, a discapito di un maggiore latenza dei pacchetti inviati.
Il secondo metodo è l'accensione dei nodi in momenti prestabiliti a livello applicativo: questo metodo è applicabile solo nel caso in cui gli eventi avvengano in maniera predicibile o comunque se i nodi vengono interrogati in maniera regolare. Un esempio è TASK [5], un modulo di TinyOS che usa TinyDB che permette di interrogare nodi di una rete di sensori attraverso il linguaggio dei database. Aggiunge la parola chiave SAMPLE PERIOD, con cui si specifica ogni quanto raccogliere i dati, permettendo che alla fine di questa operazione di mettere i sensori in modalità low-power.

Capitolo 2

Piattaforma e Nodo Sensore

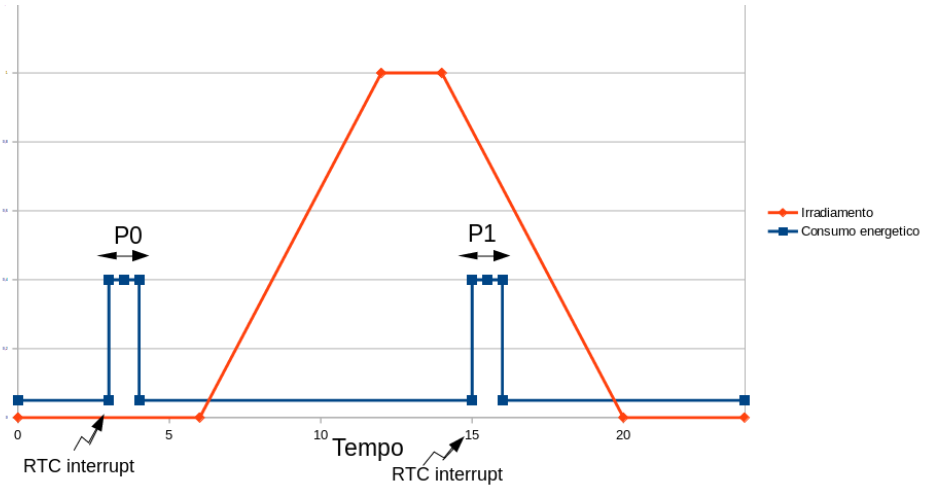

Figura 2.1: Esempio del comportamento energetico di un sensore nel sistema finale

Il sistema finale serve per monitorare le una struttura come ponti o dighe per verificare la bontà della struttura. Prendendo il caso di un ponte, sono montati sensori accelerometrici su delle travi esterne sia sulle campate, sia sulle pile del ponte da entrambe le parti. Su ogni trave vengono messi dei sensori ogni 6 metri. La lettura di tali sensori dovrà essere fatta poche volte al giorno e per raggiungere una certa precisione, il risultato sarà mediato per 10 secondo con una frequenza di 100 kHz. Tali sensori saranno dotati di moduli radio che serviranno per comunicare con una centralina. Questa centralina è formata da una Rasperry Pi, su cui è presente un database

che immagazzinerà i dati raccolti e da una board che fa da interfaccia tra i sensori e la Raspberry Pi. Ciascun sensore comunica con la centralina attraverso delle rotte generate utilizzando alcuni dei sensori che si trovano tra di esso e la centralina. I sensori utilizzeranno come alimentazione delle batterie ricaricabili a celle solari. Il suo comportamento energetico è indicato dalla Figura 2.1. Questa scelta comporta che venga risparmiata energia e che i sensori rimangano accesi solamente per il tempo necessario per la misura e per il restante tempo rimanga in uno stato a basso consumo energetico. In questo capitolo viene analizzata la piattaforma del singolo nodo e vengono aggiunte tutte le parti software mancanti affinchè il nodo abbia le funzionalità necessarie perché questo si possa comportare nel modo indicato sopra. Sono quindi necessari driver per Contiki OS per le modalità low-power e per l'RTC, in modo che questi possano entrare in modalità low-power e ritornare in modalità *run* nel momento prestabilito. Questi tempi sarà necessario che siano memorizzati in memoria di massa in modo che siano disponibili anche in caso di guasto del sensore o se la modalità low-power scelta cancella i dati in memoria. Saranno quindi necessari anche driver per la memoria di massa.

2.1 Sistema esistente

Per il prototopo, ST Microelectronics ha già un porting di Contiki OS su STM32 L152 con piattaforma Nucleo e l'expansion board X-Nucleo-IKDS04 su cui è montato un modulo Spirit1. In seguito sono spiegate le parti di cui è composto.

2.1.1 STM32L152

ST fornisce una vasta gamma di microcontrollori che si differenziano per potenza di calcolo, memoria e funzionalità. La scelta ricade sull'STM32 L152 [17]. È dotato di processore ARM Cortex-M3 con frequenza massima di clock di 32 MHz, 512 kByte di Flash, 80 kByte di RAM e 16 kByye di EEPROM. Presenta un oscillatore interno ad alta frequenza a 16 MHz (High-Speed Internal, HSI), un oscillatore interno a bassa frequenza a 37 kHz (Low-Speed Internal,LSI) e un oscillatore interno a media frequenza regolabile a 7 diverse frequenze(Medium-Speed External, MSI). È presente il supporto per oscillatori esterni al quarzo ad alta frequenza (High-Speed Ex-

Ips	Run/Active	Sleep	Low-power Run	Low-power Sleep	Stop	Standby
					Wakeup capability	Wakeup capability
Wakeup time to Run mode	0 µs	0.4 µs	3 µs	46 µs	< 8 µs	58 µs
Consumption V_{DD}=1.8 to 3.6 V (Typ)	Down to 195 µA/MHz (from Flash)	Down to 38 µA/MHz (from Flash)	Down to 11 µA	Down to 4.6 µA	0.53 µA (no RTC) V_{DD}=1.8V	0.285 µA (no RTC) V_{DD}=1.8V
					1.2 µA (with RTC) V_{DD}=1.8V	0.97 µA (with RTC) V_{DD}=1.8V
					0.56 µA (no RTC) V_{DD}=3.0V	0.29 µA (no RTC) V_{DD}=3.0V
					1.4 µA (with RTC) V_{DD}=3.0V	1.11 µA (with RTC) V_{DD}=3.0V

Figura 2.2: Modalità low-power con consumo e velocità di risveglio

ternal, HSE) e a bassa frequenza a 32,768 kHz (Low-Speed External, LSE), nel caso sia presente nella piattaforma che monta il microcontrollore. All'interno ha un modulo RTC che può contare di quattro registri che indicano la data (giorno, giorno della settimana, mese e anno) e cinque registri che indicano l'orario (A.M o P.M., ora, minuto, secondo e sub secondo). Inoltre ha due allarmi programmabili con interrupt, un timer per il wakeup, due timestamp e tre tamper-detection, tutti con interrupt attivabili e 80 byte di registri di backup. Questo circuito è alimentato da una diversa sorgente di tensione e perciò al reset del sistema non viene resettato. Il clock sorgente deve essere LSI o LSE. Una delle peculiarità di questo microcontrollore è che presenta 5 modalità low-power che sono:

sleep in cui la CPU è ferma mentre le altre periferiche rimangono attive e possono risvegliare la CPU attraverso degli interrupt o eventi ;

low-power run in cui la CPU è messa funziona alla frequenza minima di 131 kHz e viene settato il regolatore interno in modalità low-power, in modo da minimizzare la corrente. Funziona solo se la sorgente per il clock è MSI;

low-power sleep in cui la CPU è in sleep-mode con il regolatore di voltaggio in Low-power. Le periferiche utilizzabili sono limitate e come nella

modalità *sleep*, si esce da questo stato con un interrupt o un evento ;

stop con cui si raggiunge il minimo consumo mantenendo però il contenuto dei registri e della RAM. Tutti i clock principali vengono stoppati; rimangono accesi solo LSE o LSI, nel caso in cui fossero stati attivati, generando un maggior consumo. Il regolatore di voltaggio viene messo in modalità low-power. Il microcontrollore può essere risvegliato da una qualsiasi degli interrupt esterni o dagli eventi o interrupt generati dall'RTC.

standby che raggiunge il minor consumo energetico. Tutti gli oscillatori principali sono spenti, LSI o LSE rimangono attivi se attivati in precedenza, generando un maggior consumo. I dati in RAM e nei registri vengono persi eccetto quelli nella circuiteria di standby. Il microcontrollore può essere riattivato attraverso il reset esterno oppure un evento o interrupt dell'RTC, se attivo.

In Figura 2.1, sono presenti il consumo energetico per ogni modalità e il tempo di risveglio da queste. ST fornisce due tipi di librerie: le Standard Peripheral Libraries(SPL) e le Hardware Abstraction Layer. Entrambe forniscono delle astrazioni per l'accesso all'hardware, ma con la differenza che le prime sono relative ad un tipo di processore, mentre le seconde supportano diversi microcontrollori di una stessa famiglia e diverse piattaforme su cui essi sono montati.

2.1.2 Spirit1

Spirit1 [15] è un modulo radio proprietario di ST. Funziona a diverse frequenze (150-174 MHz, 300-348 MHz, 387-470 MHz, 779-956 MHz) e modulazioni (2-FSK, GFSK, MSK, GMSK, OOK, and ASK) con velocità di trasmissione da 1 a 500 kbps con antenna estensibile. Ha consumi bassi: 9 mA in ricezione 21 mA in trasmissione a 11 dBm, che è la sua potenza massima. Ha un buffer di ricezione e uno di trasmissione, entrambi di 96 byte.Ha un modulo per il carrier sense e permette di ottenere i valori di RSSI ed LQI dei pacchetti ricevuti attraverso i registri appositi.

2.1.3 STM32 Nucleo

STM32 Nucleo [16] è una evaluation platform disponibile con diversi microcontrollori. Presenta due bottoni, uno per il reset del sistema non programmabile e uno programmabile. Ha un cristallo al quarzo a bassa frequenza a 32,768 kHz e tre led di cui uno programmabile dall'utente. Ha il connettore ST-Morpho, che permettere di accedere agli I/O del microcontrollore e i connettori Arduino a cui possono essere collegate delle Extension Boards. Ha un debugger integrato STLink a cui si può accedere via USB. Per questa piattaforma sono disponibili le HAL, per varie tipologie di microprocessori.

2.1.4 X-Nucleo-IKDS04

X-Nucleo-IKDS04 è una Extension Board prodotta da ST che monta una radio Spirit1. Permette il funzionamento della radio ad 868 MHz, che è lo standard europeo, e monta un'antenna in ceramico. ST fornisce delle librerie che permettono di accedere alle funzionalità della scheda.

2.1.5 Contiki OS su STM32Nucleo con Spirit1

Il team STcLAB ha effettuato un porting di Contiki OS sulla piattaforma STM32Nucleo con L152 e l'expansion board X-Nucleo-IKDS04. Come librerie per la hanno utilizzato le HAL e le librerie della extension board per la radio. Sono state esposte le funzionalità che sono platform-dependent necessarie per il funzionamento base del sistema operativo e per il funzionamento della radio. La radio funziona con una velocità di 38,5 kbps con modulazione FSK e per l'accesso al mezzo utilizza l'unslotted CSMA/CA fornito da Contiki OS. Questo porting funziona utilizzando lo stack 6LoWPAN. Questo stack ha il vantaggio di utilizzare RPL come algoritmo di routing che permette il multihop. Il range con questa piattaforma è di circa 20/25 metri. L'utilizzo di un'antenna di tipo diverso porterebbe a notevoli migliorie da questo punto di vista.

2.2 Modifiche alla Piattaforma

Il sistema complessivo necessita l'utilizzo di modalità low-power per le piattaforme con radio. Infatti l'utilizzo in modalità Run del processore provoca un consumo eccessivo per una batteria anche di dimensioni di migliaia di mAh. Inoltre è necessario l'utilizzo di tutte le funzionalità dell'RTC i cui interrupt permettono il risvegli da alcune modalità low-power. Inoltre è necessario l'utilizzo della EEPROM se si vogliono memorizzare informazioni persistenti che utilizzando la modalità standby verrebbero perse. Inoltre è stato notato una perdita di pacchetti consistente nel caso in cui siano molto frequenti l'invio di pacchetti. Visto che Contiki OS ha ormai una storia decennale, si inputa questo problema al porting su questa piattaforma.

2.2.1 Driver

Contiki OS: Compilazione

Contiki OS è un sistema operativo che supporta numerose piattaforme. Il sistema di compilazione è basato su Makefile in modo che sia supportata la cross-compilation. La struttura del progetto presenta le seguenti cartelle principali:

app che contengono delle applicazioni che possono essere opzionalmente inserite all'interno di un progetto. Queste sono funzionalità che prescindono dal funzionamento di base del sistema operativo;

core contiene tutte le funzionalità e le interfacce del sistema operativo necessarie al suo corretto funzionamento che sono indipendenti dalla piattaforma. Sono qui contenuti i moduli per i timer, i processi, il clock di sistema, e gli stack di rete Rime (con primitive a livello MAC) e uIP, che possono essere compilati opzionalmente in mutua esclusione;

cpu contiene tutti i moduli che hanno bisogno dell'accesso a funzionalità Hardware. Sono presenti tutte le implementazioni delle CPU che supporta contiki. Sono qui per esempio l'implementazione del clock di sistema, dell'UART;

platform contiene tutte le informazioni che riguardano una particolare piattaforma. Qui sono presenti la funzione main, le funzioni che servono ad inizializzare ed utilizzare la piattaforma.

Come mostrato in Figura 2.3, la compilazione parte da un Makefile di progetto, in cui si settano la cartella principale dove è situato il sistema operativo e alcuni parametri specifici del progetto come le applicazioni che si vogliono utilizzare o il tipo di stack di rete. Dopodiché, attraverso la direttiva

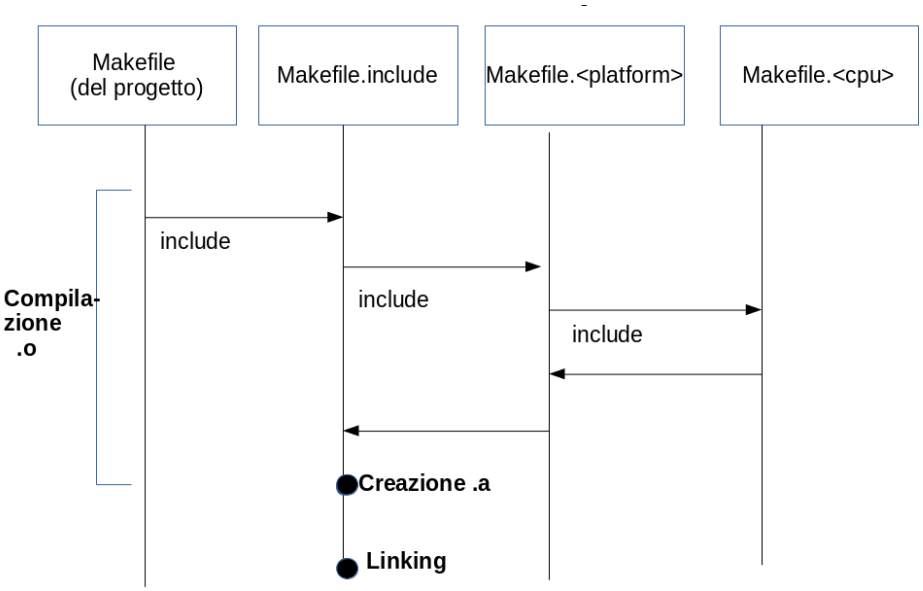

Figura 2.3: Schema di compilazione di Contiki OS

"include" viene chiamato il Makefile principale, che si occuperà di compilare le applicazioni selezionate e a sua volta di chiamare prima il Makefile che si occupa di compilare i file propri del microcontrollore da utilizzare e poi quello della piattaforma. Una volta fatto ciò compilerà i file del *core* del sistema operativo, indipendenti dalla piattaforma. I file oggetto (.o) ottenuti in questa fase vengono messi compressi in un file archivio (.ar). In seguito sono compilati i file del progetto e linkati insieme al modulo contenente la funzione *main* con l'archivio creato in precedenza. Ogni cartella ha il proprio Makefile che compila dei file dipendenti dai parametri di compilazione settati nel progetto. Contiki OS permette di definire delle variabili di compilazione a livello di progetto, piattaforma. I moduli che hanno importato il

modulo *contiki.h* permettono di definire alcune variabili a livello di progetto utilizzando le seguenti linee di codice:

```
#ifdef NAME_CONF_VAR
#define NAME_VAR
#else
#define NAME_VAR NAME_DEFAULT_VALUE
#endif
```

dove le variabili NAME_CONF_VAR sono definite in *project-conf.h* o in *platform-conf.h* entrambi importati in *contiki.h*, metre NAME_DEFAULT_VALUE è il valore di default di tale configurazione.

RTC

Per quanto riguarda l'RTC, sono stati introdotti i moduli *rtc.h* ed *rtc.c*: tutte le funzioni delle HAL necessitano una struttura particolare detta *RTC_Handle* che contiene le informazioni sull'RTC.

```
typedef struct
{
  RTC_TypeDef                 *Instance;
  /*!< Register base address  */
  RTC_InitTypeDef             Init;
  /*!< RTC required parameters*/
  HAL_LockTypeDef             Lock;
  /*!< RTC locking object     */
  __IO HAL_RTCStateTypeDef State;
  /*!< Time communication state*/
}RTC_HandleTypeDef;

HAL_StatusTypeDef HAL_RTC_SetTime
(RTC_HandleTypeDef *hrtc, RTC_TimeTypeDef *sTime,
uint32_t Format);
HAL_StatusTypeDef HAL_RTC_GetTime
(RTC_HandleTypeDef *hrtc, RTC_TimeTypeDef *sTime,
uint32_t Format);
HAL_StatusTypeDef HAL_RTC_SetDate
```

```
(RTC_HandleTypeDef *hrtc, RTC_DateTypeDef *sDate,
uint32_t Format);
HAL_StatusTypeDef HAL_RTC_GetDate
(RTC_HandleTypeDef *hrtc, RTC_DateTypeDef *sDate,
uint32_t Format);
```

Quindi questo nuovo modulo avrà una variabile di questo tipo. Nel metodo di inizializzazione viene utilizzato il metodo di inizializzazione delle HAL. Il formato scelto per l'ora è quello a 24 ore e vengono definiti i parametri RTC_-SYNCH_PREDIV e RTC_ASYNCH_PREDIV, che indicano il numero di tick necessari per un certo clock sorgente per avere un secondo, nell'header file,impostabile a seconda se si usa LSI o LSE. Per quanto riguarda le funzioni che leggono l'orario è necessario chiamare sia la funzione che legge la data sia quella che legge il tempo in questo ordine per ottenerne uno solo di questi; questo perché vengono letti dei registri ombra dell'RTC che vengono aggiornati solo se il registro contiene sia l'ora che la data precedente. Non ci sono ulteriori problemi per il settaggio della data, mentre per settare l'ora in modo preciso è necessario utilizzare, oltre alla funzione delle HAL che setta ora, minuti e secondi, utilizzare anche lo Shift Register per la coarse calibration. Per settare il numero di millisecondi è stata usata la formula:

$$SHIFT\_REG \leftarrow (1000 - milliseconds) \times (RTC\_SYNCH\_PREDIV)/1000$$

perché il registro viene decrementato e deve essere settato per un valore corrispondente al numero di tick corrispondenti a tali millisecondi. Allo stesso modo per ottenere i millisecondi si usa la formula inversa, perché anche il registro SUB_SECONDS viene decrementato. Il numero di tick per secondo è indicato dal valore RTC_SYNC_PREDIV, stabilito a compilazione e dipendente dal tipo di clock sorgente. Viene fatto un'accorgimento nel metodo che prende l'ora: è possibile ottenere un numero di millisecondi negativo nel caso ci sia appena stato l'utilizzo dello Shift Register, poiché leggendo il registro SUB_SECONDS si ottiene la somma tra questo e lo Shift Register: perciò in questo caso il dirver si occupa di modificare l'ora ottenendo il formato standard con tutti i valori positivi. Per ora il riporto di questo valore negativo fino alla data non è supportato. Per l'allarme è stata utilizzata la

funzione delle HAL che genera l'interrupt. L'allarme presenta una maschera binaria con cui si indica quale dei registri dell'orario ignorare per l'allarme. Il driver finale non permette la scelta di questa maschera, ma di default è settata quella che ignora la data e il giorno della settimana. L'allarme scatterà quando i registri dell'RTC di ora, minuti e secondi sarà uguale a quello dell'allarme.

Modalità Low-Power

Per quanto riguarda le modalità Low-Power sono stati creati i moduli *low-power.h* ed *low-power.c*: in questo caso è stato necessario vedere gli effetti che ciascuna modalità ha tenendo conto del resto della piattaforma, prima e dopo l'invocazione della funzione che agisca su questi. Nel caso della modalità "Low-Power Run" è stato necessario fornire dei metodi sia per l'attivazione e per la disattivazione. Nel caso delle modalità Sleep e Low-Power Sleep è stato necessario disattivare il Timer2 e del tick di sistema dell'STM32 altrimenti il microcontrollore sarebbe uscito praticamente subito per colpa del suo interrupt. Nel caso della modalità Stop all'uscita della modalità è stato ripristinato l'oscillatore di partenza, poiché alla sua uscita viene attivato l'oscillatore MSI a 4 kHz di default.

```
void MCU_Enter_SleepMode(void)
{
  /*Suspend Tick*/
  HAL_SuspendTick();
  HAL_TIM_Base_Stop_IT(&htim2);
  SLEEP_POWER_CONFIG();
  HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON,
  PWR_SLEEPENTRY_WFI);
  SLEEP_POWER_CONFIG_RESUME();
  /*when returning from sleep mode resume systick*/
  HAL_ResumeTick();
  st_lib_hal_tim_base_start_it(&htim2);
}
```

Nel caso della modalitá Standby viene vengono utilizzate delle istruzioni per avere un minor consumo e un risveglio più veloce. Tutte le modalità

tranne lo Standby, che alla sua uscita comporta il reboot del sistema, possono essere definite funzioni che permettono di ridurre ulteriormente il consumo disattivando altri PIN per poi riattivarle alla sua uscita. Affinché vengano attivate prima dell'entrata in modalità low-power devono essere definite nel file *project-conf.h* o in *platform-conf.h*. Prendendo come esempio il caso sopra, la funzione SLEEP_POWER_CONFIG() è definita come:

```
#ifdef SLEEP_CONF_POWER_CONFIG
#define SLEEP_POWER_CONFIG() SLEEP_CONF_POWER_CONFIG()
#else
#define SLEEP_POWER_CONFIG()
#endif
```

Cio vuol dire che se definisco una funzione che disattivi i pin chiamata "deactivate_pin" per esempio in *project-conf.h* dovrò definirla come:

```
#define SLEEP_CONF_POWER_CONFIG deactivate_pin
```

Così facendo per quel progetto verrà eseguita sempre prima di entrare in modalità low-power. Allo stesso modo si fa per le funzioni per gli altri stati low-power, anche all'uscita di tali stati.

EEPROM

Per quanto riguarda la EEPROM, Contiki OS fornisce un file system chiamato coffee. Presenta delle funzioni che permettono di leggere, scrivere ed eliminare file. Mentre l'interfaccia di tali funzioni rimane la stessa, le implementazioni sono diverse. Questa scelta è dovuta al fatto che si ha un livello di astrazione maggiore e al fatto che viene memorizzato il valore dei file scritti. È stata scelta l'implementazione detta *cfs-eeprom.c*, presente nella cartella *contiki/core/cfs*, che garantisce un accesso alla EEPROM con le interfacce del file system. Per fare ciò innanzitutto è stato necessario dichiarare nel linker script della piattaforma la sezione dedicata alla EEPROM in questo modo:

```
MEMORY
{
FLASH (rx)          : ORIGIN = 0x8000000, LENGTH = 512K
DATAEEPROM(rw)      : ORIGIN = 0x8080000, LENGTH = 16K
```

```
RAM (xrw)       : ORIGIN = 0x20000000, LENGTH = 80K
}
........
  .eeprom (NOLOAD):
   {        . = ALIGN(4);
            *(.eeprom)
            . = ALIGN(4);
   } >DATAEEPROM
```

In seguito è stato necessario creare le funzioni che poi verranno utilizzate dal file system *eeprom_read* ed *eeprom_write* nel file *eeprom.c* se che sono platform-dependent utilizzando le HAL. Queste permettono di leggere, scrivere e cancellare a partire da un offset compreso tra ed EEPROM_SIZE dall'inizio della EEPROM un numero arbitrario di byte e vengono utilizzate di base da tutte le implementazioni del file system. Nel caso di *cfs-eeprom.c* è stato riscontrato che lo status del file non viene salvato ed è stata predisposta una struttura posta in fondo alla EEPROM contenenti le informazioni sulla lunghezza del file, oltre ad un flag che indica la validità del numero letto. Il flag è valido se assume il valore esadecimale 0xDEADBEEF. Questo serve per evitare che vengano caricati valori preesistenti che non hanno quel senso.

```
struct file_eeprom {
        eeprom_addr_t filesize;
        uint32_t valid;
};
```

Questa opzione è attivata se definita la variabile di compilazione CFS_EEPROM_EOF_ENABLED. Tale struttura viene caricata alla prima apertura del file e modificata ad ogni chiamata di scrittura. L'unica convenzione che la piattaforma deve avere è che la sezione della EEPROM denominata nel linker script si chiami "eeprom", poiché non ci sono disponibili funzioni delle HAL che leggano il contenuto e quindi è stato utilizzato il seguente codice:

```
unsigned char eeprom[EEPROM_SIZE] __attribute__
((section(".eeprom")));
```

Ciò vuol dire che l'array eeprom fa riferimento all'inizio di tale sezione.

Aggiunta dei file alla procedura di compilazione

Per tutti i nuovi moduli è stata modificato il file *Makefile.stm32l152*. Questo è il Makefile che si trova nella sezione cpu, ed è stato scelto poiché tutte le funzionalità a ciò sono stati aggiunti i driver si trovano nella cpu. Affinché tutti i nuovi moduli venissero compilati, sono stati aggiunti i file *.c* alla variabile CONTIKI_CPU_ARCH ed il file *cfs-eeprom.c* alla variabile CONTIKI_TARGET_SOURCEFILES.

```
CONTIKI_CPU_ARCH+= watchdog.c \
                   rtimer-arch.c \
                   clock.c \
                   eeprom.c \
                   low-power.c \
                   rtc.c \

#eeprom file system
CONTIKI_TARGET_SOURCEFILES+= cfs-eeprom.c
```

Tali variabili verranno utilizzate in istruzioni del Makefile che genereranno dei file *.o*.

2.2.2 CSMA/CA

Uno dei problemi che si è presentato è il fatto che nel caso in cui un nodo invii molti messaggi in breve tempo si verifica una perdita di pacchetti consistente. La causa è stata attribuita al CSMA/CA. In questo caso il CSMA/CA è di tipo unslotted ed è quello fornito da Contiki OS. Il funzionamento corretto è illustrato in Figura 2.4. Confrontando l'algoritmo corretto con quello che si ha nella piattaforma, si è notato è che il CSMA/-CA non fa sensing del canale prima di trasmettere, ma si occupa solamente di tenere il conto delle ritrasmissioni fatte e dei pacchetti in coda, senza compiere il Clear Channel Assessment. Guardando le altre implementazioni del driver radio si è notato che questa procedura viene fatta dal metodo che si occupa di trasmettere il pacchetto prima di inviare la trama sul mezzo. Così nel driver di spirit che si occupa di trasmettere il pacchetto chiamato spirit_radio_transmit è stata aggiunta la seguente porzione di codice:

```
 #ifdef CSMA_SW
  BUSYWAIT_UNTIL(SpiritQiGetCs(),2*RTIMER_SECOND / 1000);
  if(SpiritQiGetCs()){
          PRINTF("Spirit1: detected collision\r\n");
          return RADIO_TX_COLLISION;
        }
#endif
```

SpiritQiGetCs() è una funzione fornita dalle librerie di Spirit che legge il segnale di Carrier Sense della radio e BUSYWAIT_UNTIL è una macro che aspetta o che il primo termine diventi vero o aspetta fino al tempo indicato dal secondo parametro. In sostanza quello che fa è rimanere in sensing per 2 millisecondi e se il canale e libero permette la trasmissione, altrimenti viene inviato un numero che corrisponde indica che è stato trovato il canale occupato al livello superiore, cioè al livello MAC. Nel caso di collisione il modulo di Contiki OS del CSMA/CA si occupa di tenere in memoria il numero di collisioni del pacchetto e di calcolarsi il tempo di backoff. Il tutto è racchiuso da una direttiva a precompilazione che verifica se è definito il parametro CSMA_SW. Questo è stato fatto perché Spirit1 ha un modulo per il CSMA/CA in Hardware e nel caso si voglia utilizzarlo non bisogna definire tale variabile. Questa può essere scelta a livello di progetto o di piattaforma o di progetto.

2.3 Conclusioni

In questo capitolo sono stati creati i driver necessari affinchè un nodo riesca ad entrare in modalità low-power e riaccendersi mediante un interrupt generato dall'RTC. Sono stati creati driver ad alto livello per entrare nelle modalità low-power, per settare i registri dell'RTC e dell'allarme che genererà un interrupt. Inoltre è stato integrato nella piattaforma il file sistem *coffee* nella sua versione per EEPROM, in modo da creare un'interfaccia più astratta verso di questa. Inoltre è stata aggiunto il Clear Channel Assesment, step del CSMA/CA mancante nella piattaforma precedente, che generava perdite di pacchetti.

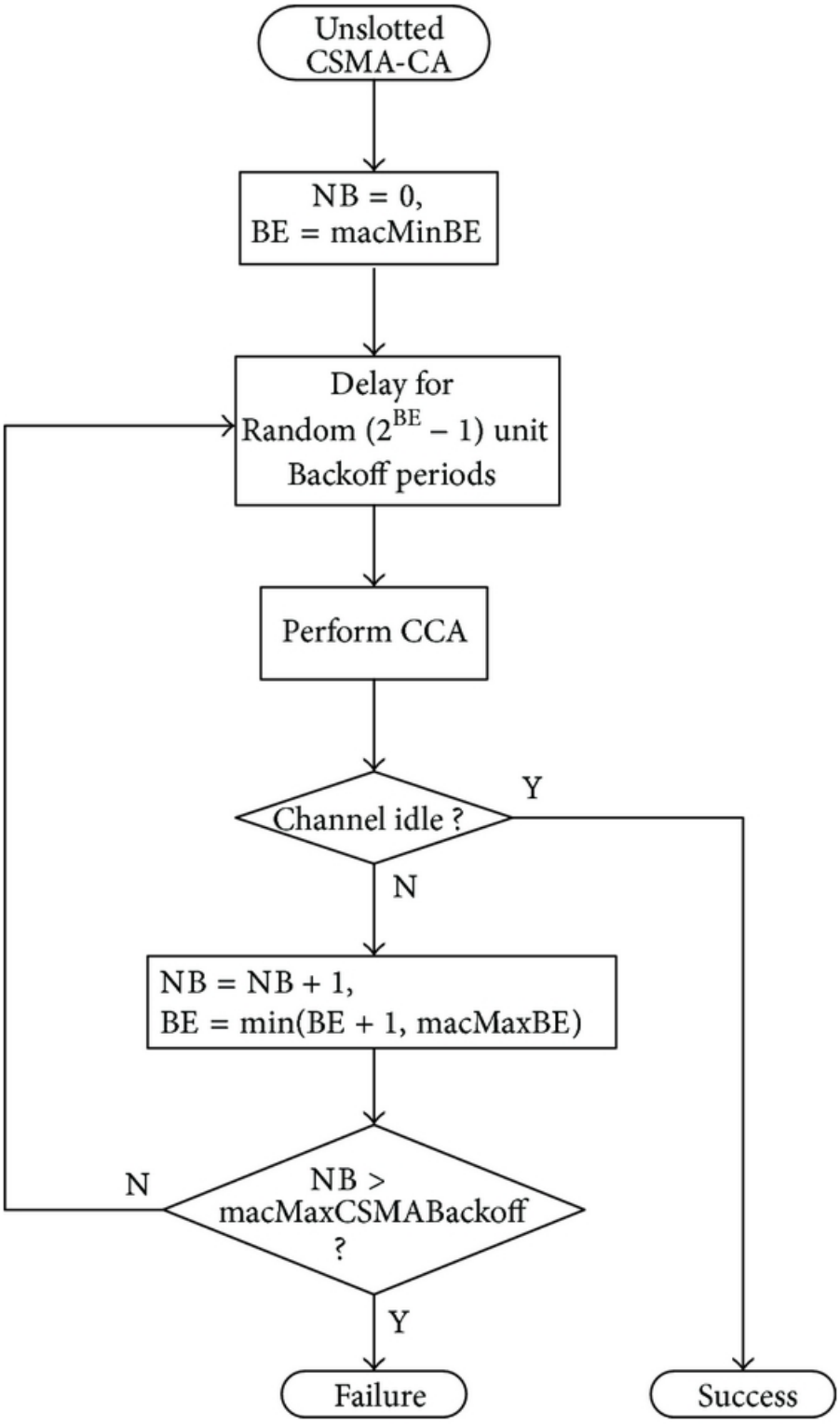

Figura 2.4: Funzionamento CSMA/CA unslotted [12]

Capitolo 3

Comunicazione e Sincronizzazione

In questo capitolo vengono affrontate e risolte delle problematiche che interessano l'intera rete di sensori: nella prima sezione viene presentata una nuova metrica, mentre nella seconda sezione viene presentato un metodo per la sincronizzazione dell'RTC, entrambi studiati per lo stack 6LoWPAN con il protocollo RPL, utilizzati dalla piattaforma.

3.1 Metrica di routing

Questa sezione si occupa di valutare come avviene il routing con la piattaforma STM32 L152 Nucleo fornita da ST. La parte software caricata su questa è la versione di Contiki OS 2.6, e nel caso ci siano dei problemi, migliorare il codice del sistema operativo in modo che questo avvenga in modo corretto. Lo stack per la rete è lo stack 6LoWPAN con RPL come algoritmo di routing.

3.1.1 Problematica

Contiki OS presenta il modulo *rpl_mrhof.c* che contiene la metrica e la funzione obiettivo. La funzione obiettivo si chiama Rango Minimo con Isteresi: come dice il nome, la funzione selezione il percorso con il rango minimo cambiando il percorso solo se questo varia di più di una certa soglia. Le metriche che possono essere selezionate sono una detta NULL METRIC e la

seconda ETX, che in sostanza sono equivalenti, la prima utilizza il campo "rango" del pacchetto per selezionare il percorso, mentre la seconda utilizza il campo dedicato alla metrica. Il numero di trasmissioni è calcolato dal livello MAC, che informa il modulo che riguarda la metrica, utilizzando un messaggio di Acknowledgement per ogni trama inviata. Nel caso l'Acknowledgement non venga ricevuto, il messaggio viene ritrasmesso finchè non viene raggiunto un numero massimo di ritrasmissioni impostato a compilazione. L'ETX calcolato non è bidirezionale, ma tiene conto solo delle trasmissioni che un nodo fa verso i propri vicini e non viceversa. L'algoritmo di routing che utilizza è RPL, un protocollo di routing di tipo Distance Vector che funzione come spiegato nel paragrafo 1.1.5. Sono state fatte numerose prove con la piattaforma Nucleo con STM32L152, descritta sopra, creando reti da 2 a 9 nodi: la rete creata è stata testata attraverso uno uno script che invia un ping a ciascun nodo della rete e in seguito, in caso di successo, sono stati richiesto i valori di RSSI ed LQI all'ultimo hop. Quello che è accaduto è che le reti hanno formato percorsi con pochi hop, circa a 2/3 hop, nonostante ci si aspettasse di averne di più, con risultati di successo per i ping molto scarsi.

Analizzando i link ad un hop, che sono stati quelli con probabilità di successo maggiori, si è notato che questa potesse dipendere dal valore dell'RSSI. Evidente dal grafico di una rete più piccola a con due nodi e il gateway posti come in Figura 3.2. L'algoritmo di routing, con questa metrica, crea i percorsi per questi due nodi: il nodo a9d2 comunica direttamente con il gateway, mentre il percorso del nodo 52b1 passa per il nodo a9d2, facendo 2 hop. Su questa rete è stato fatto lo stesso test fatto sulle reti precedenti, richiedendo solo il valore di RSSI: mentre il nodo ad 1 hop ha risposto con una probabilità vicina al 100%, il nodo a 2 hop ha risposto con una probabilità del 90%: il RSSI medio del primo nodo è di -75,4 dbm mentre quello del secondo è di -95,4 dbm . Disegnando il grafico temporale, mostrato in Figura 3.1 si nota che più l'RSSI si avvicina al valore di -100 dBm e più è improbabile la possibilità di successo dell'invio di un pacchetto, visto che nella prima parte sono stati pochi i fallimenti, mentre nella seconda parte questi sono stati maggiori. Michele Rondinone et al. [13] indicano che c'è una correlazione tra RSSI e Packet Error Rate. Inoltre, da vari capture dei pacchetti si è notato che i valori del rango da cui si è ricavato il valore del link è sempre di 256,

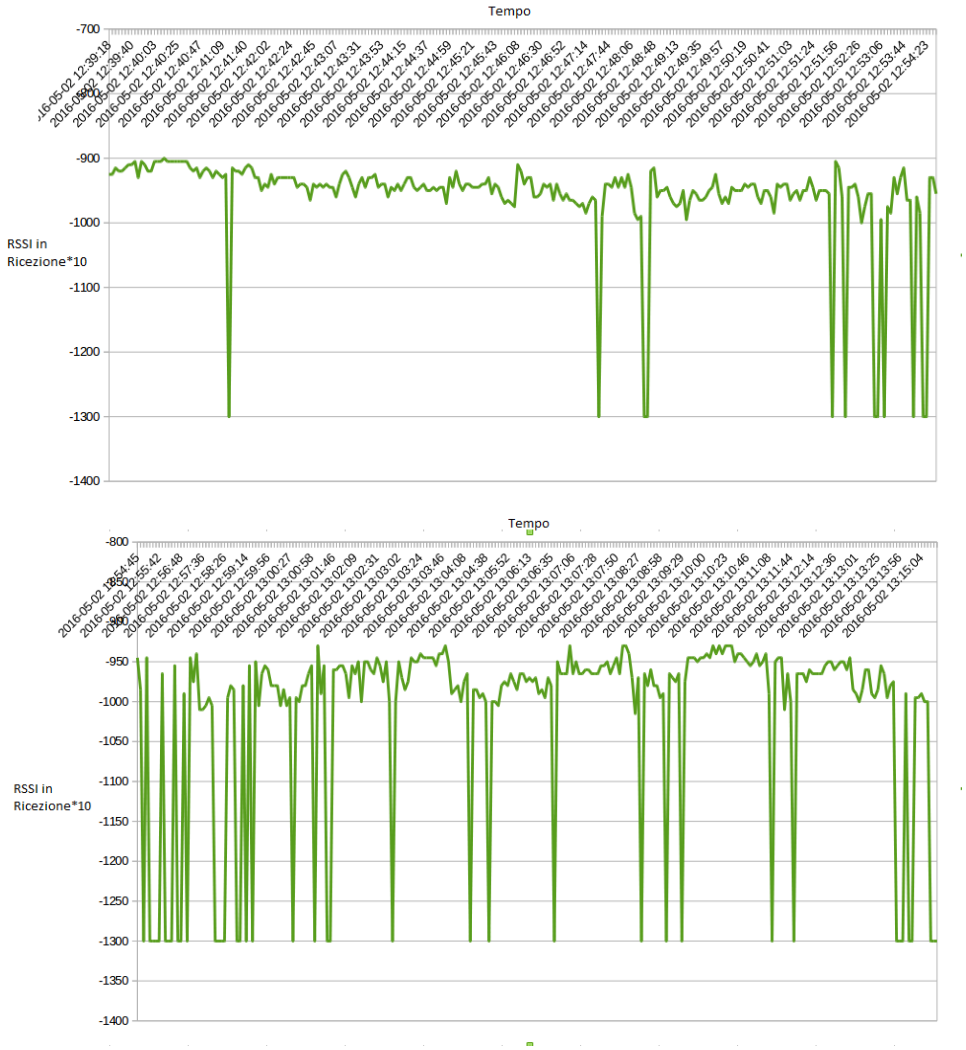

Figura 3.1: Analisi RSSI nodo a 2 hop. Il valore -1300 indica che non c'è stata risposta

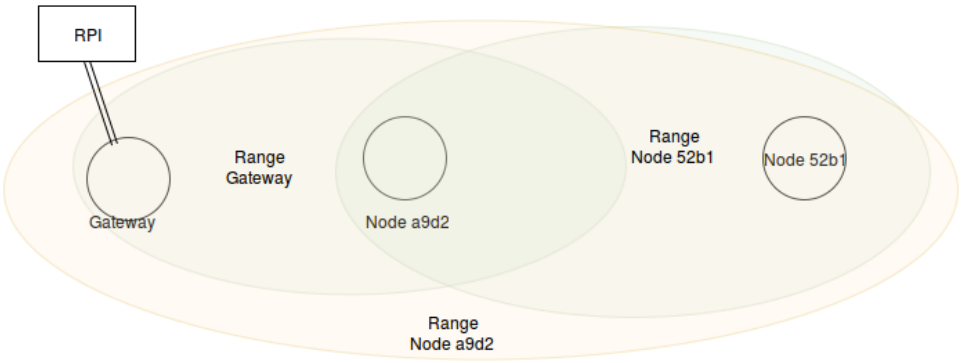

Figura 3.2: Topologia della rete su cui sono stati fatti esperimenti su PSR

che corrisponde ad un ETX di 1, anche per link con scarsa probabilità di successo. La spiegazione data per questo fenomeno è che la piattaforma non supporta ancora l'invio e la ricezione dell'Acknowledgement a livello MAC. Quello che accade è che il driver radio, non appena il canale è libero, invia il pacchetto e immediatamente informa i livelli superiori che la trasmissione ha avuto successo. Quindi, sia nel caso in cui il ricevente ha ricevuto, sia che

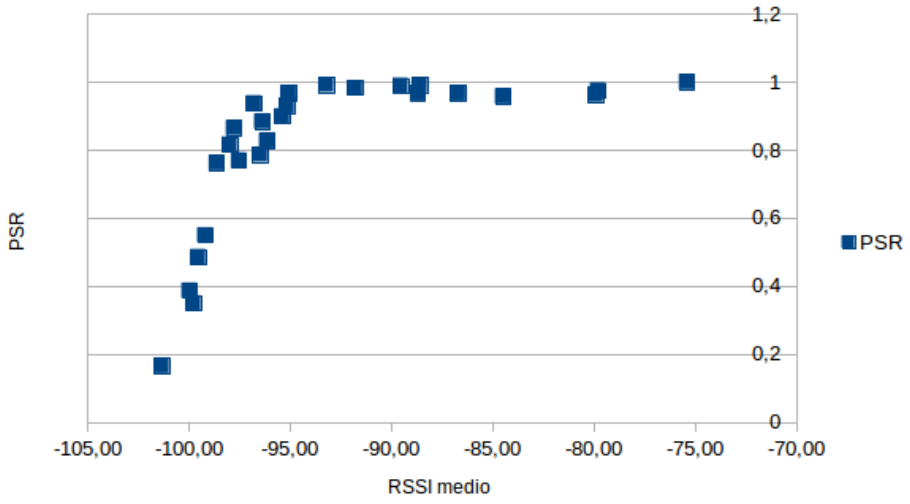

Figura 3.3: Packet Success Rate vs RSSI

non lo abbia ricevuto il pacchetto per motivi di qualità del link, il mittente ha l'informazione che il pacchetto sia stato inviato correttamente. Le reti che si formano sono equivalenti a quelle generate da una metrica del tipo hop count: non appena viene ricevuto un messaggio da parte di un vicino,

se questo ha un Rango inferiore rispetto a quello selezionato in precedenza, quest'ultimo viene scelto come Parent, non importa quale sia la qualità del link tra i due nodi. Perciò le reti che vengono formate possono essere poco affidabili poiché la scelta ricade su un percorso con meno hop, ma con link meno robusti, rispetto ad un percorso con più hop ma più affidabili. Una metrica,comunque, del genere però necessita un lungo tempo di setup della rete, poiché i pacchetti considerati sono solo quelli unicast. Sarebbe necessario che tutti i nodi inviino a tutti i propri vicini una serie pacchetti unicast per calcolarsi l'ETX. Visto che si è notato è che la probabilità di successo è legata al valore di RSSI, si vuole implementare una metrica che tenga conto di questo fattore. Quello che è stato fatto è disegnare un grafico per vedere che tipo di correlazione vi è tra queste due grandezze utilizzando i valori di RSSI ad un hop. Il riusltato, mostrato in Figura 3.3, è stato che a circa -95 dBm comincia un calo lineare fino a circa -100 dBm

3.1.2 Soluzione Proposta: Hop count with RSSI threshold

La soluzione proposta si basa su due principi: l'RSSI dopo un certo valore riduce drasticamente il PER di un certo link e che un link sia simmetrico. La prima affermazione è stata supportata dai dati raccolti in precedenza mentre la seconda deriva dal fatto che si assume che i dispositivi radio che vengono utilizzati sono dello stesso tipo. A questo proposito viene aggiunto alla struttura che memorizza lo stato dei vicini un array con 10 valori, che indicano gli ultimi 10 valori di RSSI misurati da un pacchetto ricevuto da tale vicino, e da un numero che indica il numero di pacchetti ricevuti. Per ricevere tali valori viene modificato in parte il modulo *sicslowpan.c* di Contiki OS. Alla struttura che contiene le informazioni del vicino vengono quindi aggiunte le linee in seguito:

```
#if RPL_OF==rpl_rssiof
  short int rssi_values[10];
  uint8_t rssi_nvalues;
#endif
```

Le strutture dei vicini nel caso di ETX e NULL METRIC vengono modificate nel momento in cui un pacchetto viene inviato; invece in questo caso deve

essere modificato nel caso venga ricevuto un pacchetto. Per questo è stata introdotta la seguente porzione di codice nel metodo input:

```
/*if objective function equals to rssiof, then link callback*/
if(RPL_OF.ocp==2){
   uip_ds6_link_neighbor_callback(MAC_TX_OK,
   sicslowpan_get_last_rssi());
}
```

che controlla se la funzione obiettio abbia l'objective code point (ocp) che corrisponde a quello della nuova metrica, in questo caso 2, chiamando la funzione *uip_ds6_link_neighbor_callback*, prendendo in input il valore di RSSI ricevuto e utilizzarà quello per modificare il contenuto per calcolare l'RSSI medio. Affinchè non avvengano errori è stata modificata anche la funzione di output, che avrà la condizione negata rispetto a quella riportata sopra. Tale funzione chiamerà la funzione di callback del link relativo alla metrica utilizzata. Per questa metrica prima di tutto viene memorizzato il nuovo valore di RSSI nell'array shiftando a destra i valori già presenti e perdendo eventualmente l'ultimo se l'array è pieno e in seguito viene calcolato il valore medio. Il valore del link viene settato ad infinito se il link ha una media inferiore ad una threshold, oppure il valore è al di sotto di una threshold più alta della precedente ma l'array è pieno metà, altrimenti assume il valore minimo per un link. La scelta dell a treshold più bassa deve essere quella in cui il packet error rate comincia a decrescere, mentre quella più alta è consigliato essere circa 2 volte la deviazione standard. Questa scelta è dovuta serve nel momento di setup della rete, perchè i valori msurati potrebbero appartenere ad una distribuzione aleatoria con media minore o uguale alla threshold più bassa. La funzione obiettivo, invece, è la stessa usata per l'ETX, ovvero il Rango Minimo con Isteresi. Questa soluzione funziona bene nel caso in cui non ci sia duty-cycling, come nella nostra applicazione finale, poiché utilizza i pacchetti inviati in broadcast, poco costosi in questo caso, per valutare la metrica dei propri vicini.

3.1.3 Valutazione Sperimentale

La simulazione è stata fatta con schede Nucleo con STM32L152 con l'expansion board X-Nucleo-IKDS04. L'ipotesi della simmetria del link vale,

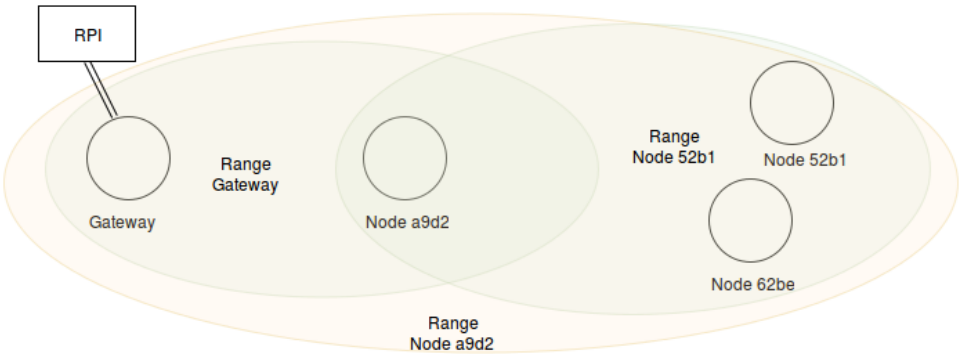

Figura 3.4: Topologia della rete su cui sono stati fatti gli esperimenti

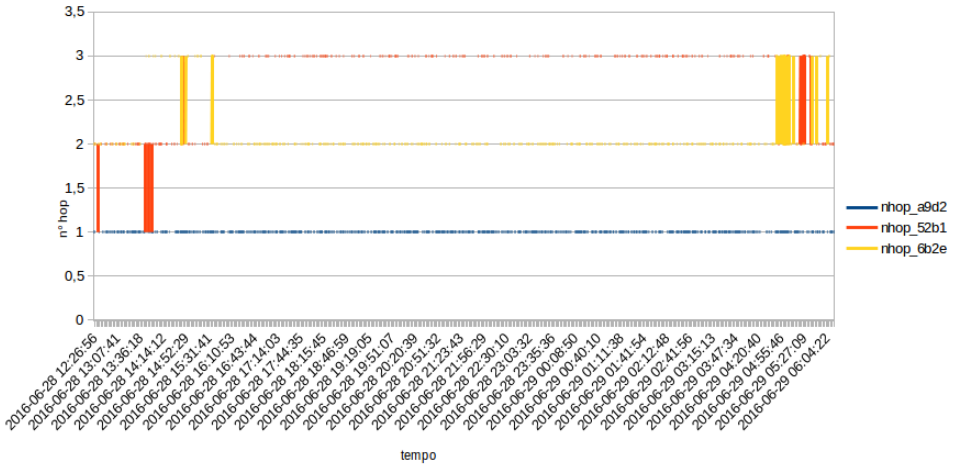

Figura 3.5: Variazione del numero di hop nel tempo con la metrica proposta

poiché i nodi montano tutti dispositivi uguali. Le due threshold settate per questa simulazione sono -95 e -90 dBm, la prima ricavata dal grafico sopra, la seconda ricavata dalle misurazioni di deviazione standard campionaria di circa 2 dBm. Innanzitutto si calcola il costo in termini di memoria. Le variabili aggiuntive sono 11 interi a 16 bit, quindi occupano 22 byte aggiuntivi. Tenendo conto che generalmente il numero dei vicini è dell'ordine di poche decine si ha un occupazione aggiuntiva nella RAM di alcune centinaia di byte. Questo è un costo accessibile per microcontrollori come l'STM32L152 che hanno complessivamente 80 kilobyte di RAM, ma potrebbe essere eccessivo per microcontrollori con pochi byte di RAM. Per quanto riguarda le performance riguardo le probabilità di successo, è stata fatta una simulazione con 4 nodi, posti come in Figura 3.4, di cui uno che funge da gateway, in modo

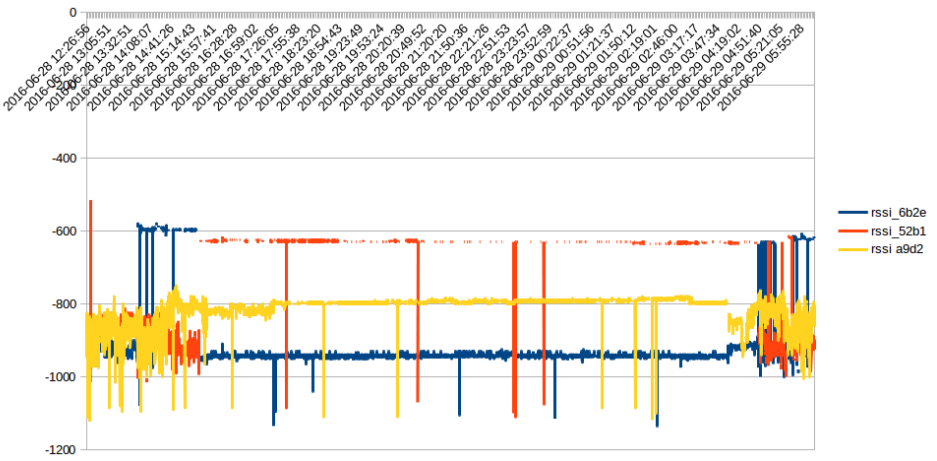

Figura 3.6: Variazione del valore dell'RSSI all'ultimo hop visto dai vari nodi nel tempo

che uno di questi facesse da ponte per altre due posti approssimativamente alla stessa altezza. Il nodo più vicino al gateway ha avuto successo il 95%, facendo sempre un hop, mentre gli altri due nodi hanno avuto rispettivamente l'83% e l'86%. Quello che si è avuto è un miglioramento della probabilità di successo per i ping per quanto riguarda la rete, a scapito della stabilità di questa. Guardando la Figura 3.6, che mostra la variazione di RSSI nel tempo, la curva relativa al nodo a9d2 ha avuto un andamento essenzialmente continuo, dovuto al fatto che il percorso che lo raggiunge non è mutato. I nodi 6b2e e 52b1 hanno avuto delle discontinutà, dovute al fatto che i percorsi che li raggiungono sono diventati da 2 a 3 hop e viceversa come è possibile vedere dalla Figura 3.5. Questo fatto è dovuto al fatto che vi è stato un peggioramento di uno dei due link, facendo sì che il percorso verso uno di questi due nodi passasse attraverso l'altro. Da notare che tra le 16:40 e le 4 del giorno dopo la rete sia rimasta stabile con il nodo 62be raggiunto in 2 hop e il nodo 52b1 raggiunto in 3 hop, che ha fatto in modo che questo potesse essere raggiunto. Ha avuto comunque una peggiore probabilità di successo poiché il link precedente a 2 hop è peggiorato a sua volta, mantenendo in quel periodo un RSSI medio vicino al limite dei -95 dBm, ma comunque si ha avuto un esito migliore, nello stesso caso ma con la metrica ETX: come mostrato in Figura 3.7, il nodo 52b1 ha risposto solo una volta nello stesso caso con un percorso a 2 hop: questo è dovuto al fatto che la

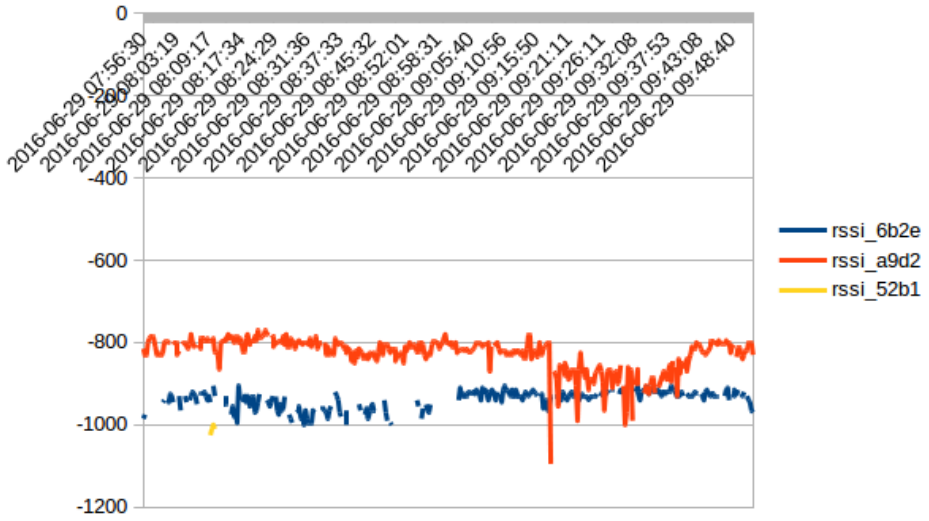

Figura 3.7: Variazione del valore dell'RSSI all'ultimo hop visto dai vari nodi nel tempo con metrica ETX/Hop Count

metrica, poiché funziona come un hop count, una volta che trova un percorso con meno hop predilige questo, senza tenere conto della qualità del link. Un percorso migliore si sarebbe ottenuto creando un percorso attraverso il nodo 62be, che ha avuto una probabilità di successo migliore e quindo un valore di RSSI in ricezione migliore. Il valore di RSSI del link tra questi due nodi è di circa -62 dbm e quindi ritenuti raggiungibili. Dall'esperimento, inoltre, emerge che la Threshold di -95 dBm è bassa per garantire un link di qualità. Questo comportamento avviene anche con una rete numerosa: ragionando per induzione, se questa metrica garantisce un link di qualità fino ad un certo nodo n con n hop, due nodi posti nel range del nodo n conb valori di RSSI in ricezione pari a quelli della simulazione, si comporterebbe sceglierebbero il percorso con n+1 hop, se questi hanno il valore di RSSI medio inferiore alla Threshold -95 dBm. Se invece solo uno di questi ha un valore medio superiore alla Threshold, questi farebbe n+2 hop mentre l'altro farebbe n+1 hop. Se entrambi hanno il valore di RSSI in ricezione maggiore della Threshold, entrambi sceglierebbero il percorso con n+1 hop, ma nessun'altro percorso passerebbe verso di loro, visto che il valore del link è infinito.

3.2 Sincronizzazione dell'RTC

In questa sezione si cerca una soluzione per il problema della sincronizzazione dell'RTC. Per l'applicazione finale è importante che questi siano sicronizzati perchè da questi dipende il consumo energetico finale. L'obiettivo è quello di creare ed implementare un algoritmo che consenta di mantenere sincronizzati gli RTC dei vari sensori della rete in modo efficace.

3.2.1 Perchè sincronizzare l'RTC in questa applicazione

La sincronizzazione dell'RTC è importante per varie ragioni. Come detto nel capitolo 1, alla misura che ciascun sensore fa, si può associare il valore di tempo locale per sapere qual è l'istante di tempo in cui è avvenuto. Tuttavia l'oscillatore che permette di incrementare l'orario ha una certa imprecisione che causa un offset tra l'orologio di riferimento e quello del sensore e tra i vari sensori. In questo caso la sincronizzazione è fondamentale perchè viene utilizzato per risvegliare i sensori e in caso si verifica che, a lungo andare, l'offset tra l'orologio di riferimento e quello dei sensori tende a diventare alto. Se vengono utilizzati gli allarmi, il valore registrato in tali allarmi non sarà quello reale e la misura verrà presa in orari differenti da quelli prestabiliti; inoltre l'offset, dovuto allo skew tra i vari oscillatori, tenderebbe ad aumentare. Quello che accadrebbe è che la rete rimarrebbe accesa per un tempo maggiore poiché ciascun sensore si risveglierebbe in un istante di tempo che può differire di molto rispetto a quello degli altri sensori. L'obiettivo di questo capitolo è di creare un algoritmo che permetta di sincronizzare gli RTC in modo efficiente per una rete multi-hop in una rete 6LoWPAN, prendendo spunto da protocolli presenti in letteratura che però sono a livello MAC, utilizzando lo stack 6LoWPAN: come nello stack IP, il livello applicativo si occupa di analizzare il contenuto ed elaborare le informazioni contenute nel pacchetto.

3.2.2 Algoritmo di sincronizzazione

L'algoritmo si basa, come idea per la propagazione del timestamp, sul Flooding Syncronization Time Protocol (FSTP). L'FSTP utilizza trame MAC per propagare il timestamp, mentre in questo caso viene utilizzata la trama e la porta dell'NTP. Il protocollo descritto necessita che il border router sia

sincronizzato esternamente con l'orologio di riferimento, perchè questo periodicamente invia una trama con il proprio timestamp in broadcast, che ha come indirizzo l'indirizzo broadcast RPL All Nodes ff02::1a. Il funzionamento dell'algoritmo è illustratto nella Figura 3.8. Ogni nodo ha memorizzato un ID e l'ultimo numero di versione ricevuto. La trama contiene questi valori, oltre al timestamp in cui il messaggio è stato inviato. L'ID del border router è inizializzato a 0 e periodicamente un processo invia la trama NTP utilizzando l'indirizzo RPL-ALL-Nodes con il proprio timestamp. Un qualsiasi nodo, invece alla ricezione di una qualsiasi trama NTP controlla il numero di sequenza e l'id: se il numero di sequenza è maggiore oppure l'ID del messaggio è minore rispetto al proprio, setta il timestamp ricevuto nell'RTC. Il nodo memorizza la nuova versione e setta come ID il numero successivo quello del messaggio. Se è il caso di una versione maggiore ricevuta inizializza un timer che genererà un evento dopo un tempo compreso casuale tra 0 e un numero massimo scelto a compilazione. Questo timer viene disattivato se riceve una trama con ID maggiore. Questa scelta di utilizzare il timer è stata fatta per evitare che piú nodi inviino contemporaneamente la propria trama che genererebbe delay. Quando il messaggio è stato inviato non vengono più valutate nuove trame la versione memorizzata. Quando il timer è scaduto il nodo può inviare la propria trama con l'ID e la versione memorizzati e il timestamp. Il delay di un nodo che ci si aspetta è pari al tempo di trasmissione sul link per l'ID del nodo.

3.2.3 Adattamento per NTP

L'NTP è un protocollo che non è pensato per reti di sensori wireless e non presenta i campi ID e versione. Per ovviare a questo problema si cerca una modalità per cui un campo non viene utilizzato. È il caso in cui il campo Stratum è a 0. Questo campo indica il tipo di clock di origine e il valore 0 indica il caso che sia indefinito e di conseguenza il campo Reference ID, che indica il nome del clock di origine, non è utilizzato. Questo algoritmo, quindi, è attivato quando il campo Stratum è 0 e il campo Reference ID è diverso da 0. Questo campo è a 32 bit ed è stato diviso a metà: i 16 bit più significativi indicano l'ID di chi invia il messaggio e i 16 bit meno significativi indicano la versione del messaggio. Questa modalità necessita anche che il valore Mode sia corrispondente a broadcast.

3.2.4 Implementazione in Contiki OS

In Contiki OS è stato aggiunto un nuovo modulo nella cartella app chiamato *SNTP*. Questa contiene un header file e un file *.c*. Per il messaggio è stata creata la struttura timesync_messagge che ricalca i campi del messaggio NTP. Per utilizzare il campo Reference ID è viene utilizzato un tipo usando la union in questo modo:

```
 struct broadcast_info{
        uint16_t id;
        uint16_t seq_number;
};

typedef  union broadcast_info_t{
        struct broadcast_info infos;
        uint32_t inpt;
} broadcast_info_t;
```

Il campo broadcast_info serve per mettere e ricavare le informazioni broadcast e il campo inpt serve per assegnarlo al campo Referenece ID. Il tipo di clock che deve essere sincronizzato non è il clock dell'orologio di sistema, ma l'RTC che è formato da una serie di registri. Per semplicità di implementazione, il timestamp è scomposto nei valori del registro. Viene utilizzato il solo transmit timestamp, diviso da un campo da 32 bit che contiene i secondi e un campo a 32 bit in cui è presente la frazione di secondi. In questo caso sono stati creati due tipi, adattando il timestamp in questo modo:

```
struct timestamp {
        uint8_t year;
        uint8_t month;
        uint8_t day;
        uint8_t hour;
        uint8_t minute;
        uint8_t second;
        uint16_t millisecond;
};

struct input{
```

```
        uint32_t input1;
        uint32_t input2;
};

typedef union timestamp_t{
          struct timestamp ts;
          struct input inpt;
 } timestamp_t;
```

dove *input* serve per dividere il timestamp nel tipo *timestamp_t*. L'header file contiene le funzioni sntp_init e sntp_root_init. Queste funzioni hanno il compito far cominciare rispettivamente i processi *sntp_process* e *sntp_process_root*. Il primo implementa il protocollo descritto sopra, mentre il secondo si occupa di inviare periodicamente il valore del timestamp. I timer utilizzati sono di tipo *etimer*, timer che creano un evento per il processo al loro scadere.

3.3 Conclusioni

In questo capitolo è sono stati affrontati problemi riguardo l'intera rete. Nella prima sezione è stato affrontato il problema del routing, dovuto al fatto che simulazioni hanno creato delle reti instabili con la piattaforma Nucleo con radio Spirit1. Il problema identificato è stato che la scelta dei percorsi avviene con una metrica, ETX, che in questa piattaforma, che non ha Acknowledgement MAC, funziona come una di tipo Hop-Count. Dall'analisi di misure di Packet Error Rate fatti su reti composte di questi nodi, è risultata una corrispondenza fra PER e RSSI su cui si basa l'implementazione di una nuova metrica, l'Hop-Count with RSSI threshold, che stima la qualità del link in base all'RSSI. Dagli esperimenti compiuti è risultato un miglioramento del routing rispetto alla metrica utilizzata con questa piattaforma. Nella seconda sezione è stato proposto un algoritmo per la sincronizzazione dell'RTC della rete wireless.

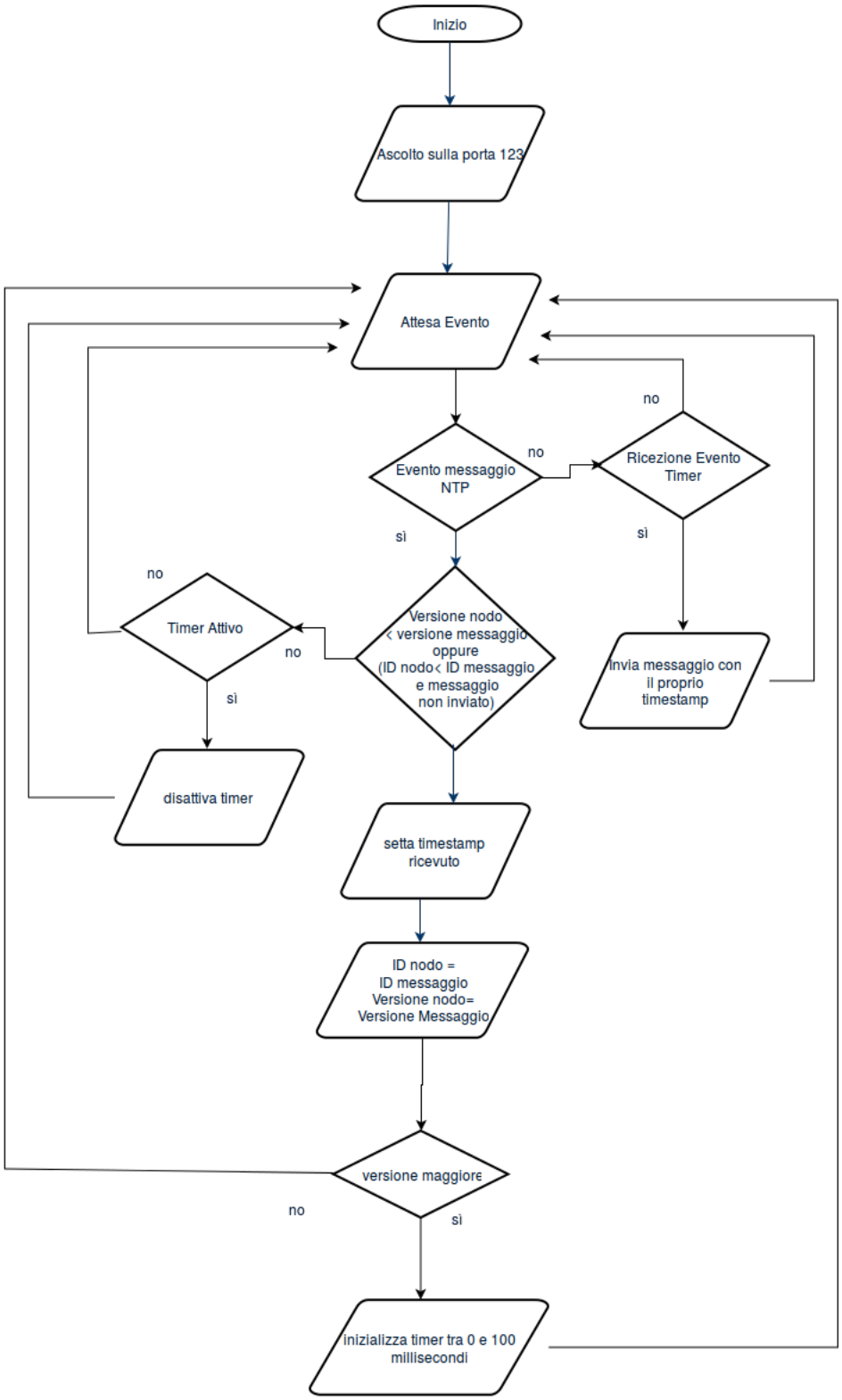

Figura 3.8: Algoritmo di sincronizzazione RTC

Capitolo 4

Realizzazione e Valutazione del Sistema

In questo capitolo viene presentato un algoritmo che serve per raccogliere i dati della rete di sensori wireless, che fa in modo che i sensori rimangono accesi solamente nel tempo in cui questi. In seguito viene fatta una prima valutazione del consumo energetico.

4.1 Requisiti e scelte di progetto

Esistono due tipi di componenti: una centralina e i sensori. La centralina si occupa di interrogare i sensori e immagazzinare i dati che ricevono dai sensori, i quali permettono accesso esterno alle proprie risorse, quali l'accelerometro e il loro tempo locale. L'algoritmo finale deve fare in modo che i sensori si accendano in un istante prestabilito e fornire una misura dell'accelerometro solo dopo aver campionato i valori, mediandoli per 10 secondi. Una volta raccolti i dati da tutti i sensori, questi si spengono. È stato scelto di ricreare la rete ogni volta per 3 motivi:

- la topologia ottimale potrebbe cambiare nel tempo in cui i vari sensori sono spenti per via del mezzo wireless;
- si ha la possibilità di sapere se effettivamente un nodo è connesso alla rete: in questo caso il nodo potrebbe essere spento e una richiesta verso di questo causerebbe il timeout e l'incremento del tempo di accensione dei sensori;

- si ha la possibilità di utilizzare la modalità standby del microcontrollore STM32 L152 che offre il minor consumo energetico.

4.2 Componenti

4.2.1 Sensore

Il sensore è una piattaforma contenente un microcontrollore STM32 L152, una radio Spirit1 e un accelerometro MEMS. Su questo è installato il sistema operativo Contiki. Questo è dotato di stack 6LoWPAN e server coap con cui è possibile accedere alle risorse fisiche. Per i test viene utilizzata una board STM32 Nucleo con l'expansion board X-Nucleo-IKDS04 che monta un modulo Spirit1.

4.2.2 Raspberry PI

La Raspberry Pi è il componente che permette di immagazzinare i dati e su cui è installato un client CoAP che permette di interrogare i sensori. In questo caso viene utilizzato mongodb come database e come client CoAP viene utilizzato il plugin python Txtxings.

4.2.3 Centralina

La centralina è composta da una Raspberry PI e da una piattaforma simile identica a quella del sensore, ma senza accelerometro. Questa fungerà da border router: attraverso un cavo seriale vengono trasmessi i pacchetti che provengono dalla rete verso la Raspberry PI e viceversa attraverso un'interfaccia IPv6 per la Raspberry PI con il protocollo SLIP (Serial Line Internet Protocol). Anche su questa è presente il sistema operativo Contiki.

4.3 Algoritmo di raccolta dati

Ciascun sensore ha una lista degli allarmi, che indicano l'istante di tempo in cui questi vengono risvegliati in una giornata, memorizzati nella propria EEPROM. Inoltre i sensori necessitano le seguenti risorse CoAP:

- una risorsa che gestisce gli allarmi (contenuta in res-alarm.c);

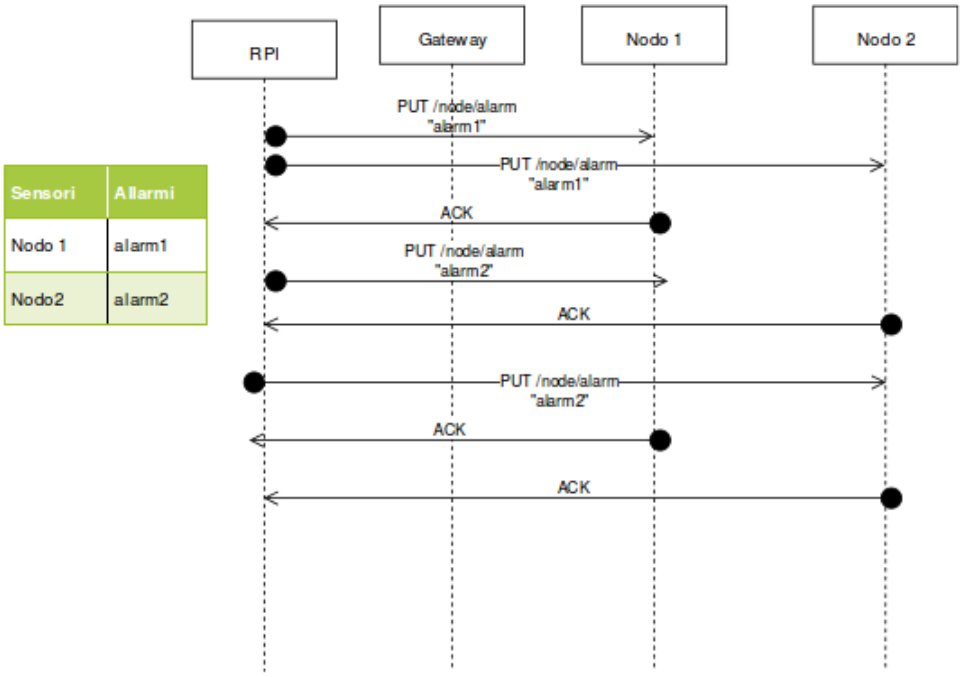

Figura 4.1: Fase di inizializzazione dei sensori

- una risorsa che acceda all'accelerometro (res-sensors.c).
- una risorsa che mette il sensore in modalità low-power dopo aver settato l'allarme successivo più prossimo all'orario del nodo (res-low-power.c).

La prima permette di aggiungere un allarme alla lista (con il metodo PUT), accede (GET) e cancellare la lista (REMOVE). Per aggiungere un allarme è necessario che questo sia cronologicamente maggiore di quello precedente. La seconda restituisce con il metodo GET una variabile elaborata da un processo che comincia al boot di sistema che accede all'accelerometro e media più volte la misura presa. Questa risorsa utilizza l'opzione "blocco 2" di CoAP: una variabile booleana indica se la misura e pronta oppure no. Se la misura non è ancora pronta, questa invia un messaggio di Acknoledgement vuota con codice 0. Quando questa è pronta, il processo che si occupa di fare il sampling invia un nuovo messaggio con all'interno la misura ottenuta. La terza, con il metodo POST, permette di mettere in modalità low-power il sensore, dopo che questo ha inviato lo stesso tipo di messaggio in broadcast radio senza inviare Acknowledgement. Questo è possibile in Contiki OS settando la variabile esterna *erbium_ status_ code* il valore della macro

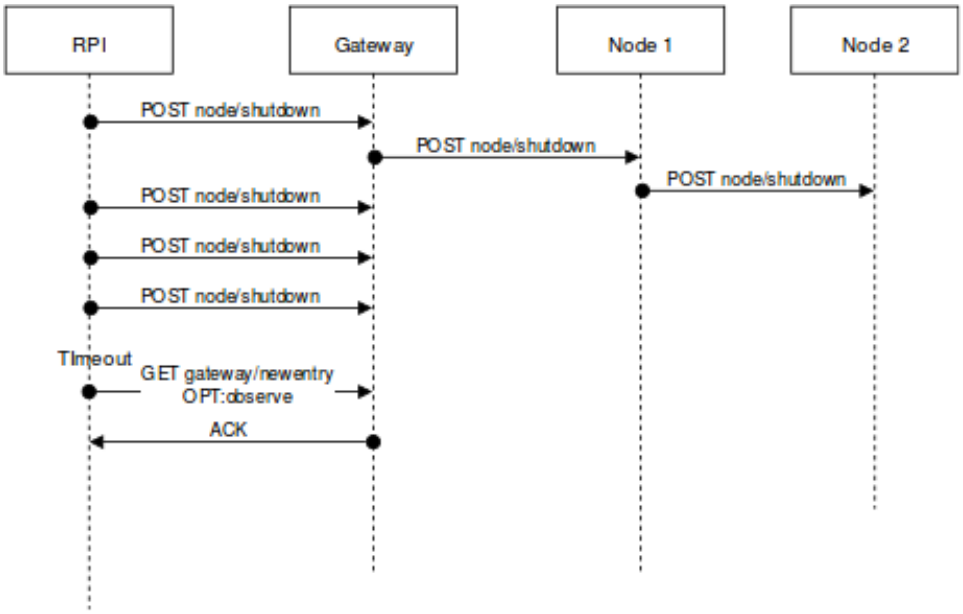

Figura 4.2: Fase di spegnimento dei sensori

MANUAL_RESPONSE, che non fa inviare l'Acknowledgement e inviando un segnale di POLL su un processo esterno, all'arrivo del quale esegue l'istruzione di spegnimento. Il border router, su cui è presente anche un server coap ha una risorsa osservabile che viene attivata non appena un nuovo nodo viene aggiunto alla rete. Contiki OS infatti permette di aggiungere una callback chiamata ogni volta che viene aggiunto o eliminato un nodo dalla propria tabella di routing. Quello che è stato fatto è stato di creare una risorsa a cui è legata una lista allocata dinamicamente. Quando un nodo viene aggiunto alla tabella di routing, viene chiamata una callback che aggiunge tale nodo alla lista e invia la notifica ai nodi che hanno sottoscritto a questa risorsa. Quando l'observer invia il messaggio di GET, viene fatto il pop dalla lista dell'ultimo nodo aggiunto alla tabella di routing. L'uso della lista è precauzionale, nel caso vengano aggiunti più nodi nel mentre ancora il client non ha inviato o non sia ancora arrivato il messaggio legato al metodo GET. Per funzionare nel caso preso in considerazione è necessario che il client che osserva sia unico e che la lista sia inizialmente vuota. Inoltre necessita la stessa risorsa per lo spegnimento del nodo, con la differenza che il border router si deve riavviare invece che andare in modalità low-power.

Il client CoAP presente sulla Raspberry PI ha memorizzato l'indirizzo

dei sensori e gli allarmi. Il client CoAP ha il compito di inizializzare i sensori con gli allarmi predefiniti come mostrato in Figura 4.1: viene inizializzato un dizionario che ha come chiave l'indirizzo IP dei vari sensori e gli elementi saranno delle liste che che contengono tutti gli allarmi in ordine di orario. Per ogni sensore, viene inviato il primo allarme utilizzando la risorsa che inserisce l'allarme. Alla ricezione dell'Acknoledgement, l'allarme inviato viene eliminato dalla lista del sensore che ha risposto, contenuta nel dizionario, e inviato il successivo. Una volta inviati tutti gli allarmi, l'entry corrispondente al sensore viene eliminato e quando questo dizionario è vuoto, è possibile spegnere la rete. La Figura 4.2 mostra come questo avviene. La risorsa che viene chiamata per spegnere la rete funziona in questo modo: ricevuto il messaggio, il metodo associato a questa richiesta invia in braodcast radio utilizzando l'indirizzo IP RPL-All-Nodes lo stesso tipo di messaggio senza inviare alcun Acknoledgement e successivamente si spegne. Perché funzioni per spegnere tutta la rete, il messaggio viene inizialmente inviato al border router che lo propagherà per tutta la rete. Il nodo, prima di spengnersi, controlla la lista dei propri allarmi e setta l'allarme successivo. A differenza degli altri nodi, il border router viene riavviato, invece che andare in standby e in seguito riceverà la richiesta di osservere la risorsa della tabella di routing dal client sulla Raspberry. In caso il messaggio non venga ricevuto per qualsiasi motivo, questo si spegnerà comunque dopo un certo tempo. Quando l'allarme fa risvegliare i nodi della rete, come si vede dalla Figura 4.3, la risorsa osservata dal client sul border router inizerà a notificare al client CoAP gli indirizzi dei nodi che vengono aggiunti e a tale nodo il client invia la richiesta alla risorsa legata all'accellerometro. Il nodo, come detto prima, invierà il valore misurato, se questo è pronto, altrimenti invia un messaggio di Acknoledgement vuoto e invierà il messaggio successivamente. Il client immagazzina i dati nel database e cancella dalla lista il sensore che ha risposto. Quando questa lista è vuota viene nuovamente inviato il messaggio di shutdown al border-router. Questo algoritmo presuppone che ci sia un meccanismo per la sincronizzazione dell'RTC che funzioni mentre ciò avviene.

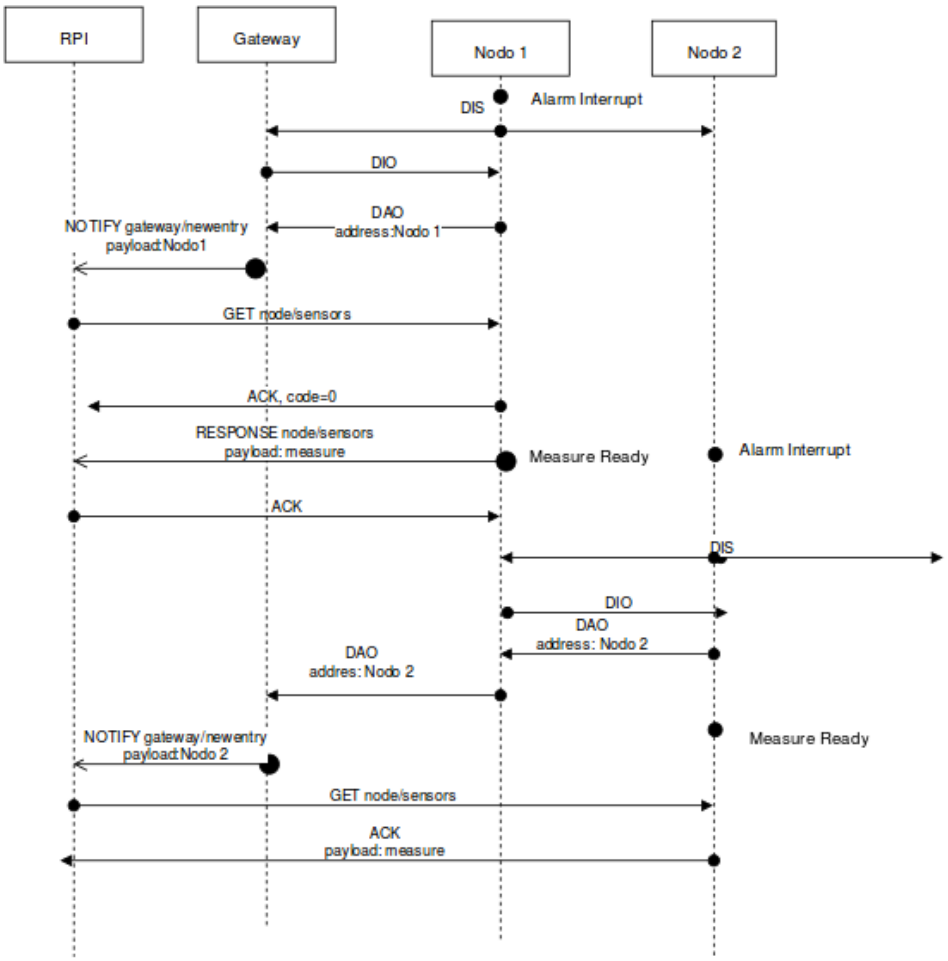

Figura 4.3: Fase di sampling

4.4 Implementazione in Contiki OS

Per l'ambiente contenente i file con i sorgenti del firmware sono contenuti nella cartella *workspace* creata nella directory principale di Contiki OS. Per la connessione "slip", è stato utilizzato il programma *tunslip6*, generato dal file sorgente *tunslip6.c* nella cartella *tools*, sempre nella directory principale.

4.4.1 Node Program

È il firmware che è caricato su tutti i nodi con i sensori. La base del progetto è l'esempio *er-example-server*, proveniente dalla cartella *examples* nella directory principale e modificata opportunamente. Contiene il Makefile preparato per aggiungere l'applicazione *coap*, con il riferimento alla cartella che contiene le risorse per compilarle, in questo caso *resources*, nella cartella del progetto, che contiene le risorse descritte sopra. Il processo principale, contenuto nel file *er-example-server.c*, inizializza il motore rest e le risorse rispettivamente con il metodo rest_init_engine e con il metodo *rest_activate_resource*, che prende il tipo *resource_t*, che contiene la risorsa definita in un altro file e dichiarata con la parola chiave "extern", e una stringa che è il nome della risorsa con cui viene identificata da client esterni.

4.4.2 Border Router

È stato utilizzata come base l'esempio *rpl-border-router* presente nella cartella *<contiki-folder>/example/ipv6*. È composta dai file *border-router.c*, che contiene il processo principale, un server HTTP con i file *httpd-simple.c* e *http-simple.h* e il file *slip-bridge.c* che contiene l'interfaccia per la connessione slip. Il server HTTP, opzionalmente attivabile attraverso il Makefile, invia le informazioni sui propri vicini e sui nodi raggiungibili attraverso questo. Le modifiche si siono concentrate sul primo file: è stato aggiunto il motore rest per attivare le risorse coap attraverso il metodo *rest_init_engine* e le varie risorse con il metodo *rest_activate_resource*, che prende il tipo *resource_t*, che contiene la risorsa definita in un altro file e dichiarata con la parola chiave "extern", e il nome della risorsa. È stato attivato il server sntp client, che serve per mettere in ascolto sulla porta 123 e nel metodo *set_prefix_64*, chiamato quando è attivata la connessione slip, è chiamata la funzione *sntp_root_init*, che si occupa di iniziare ad inviare i messaggi SNTP.

4.5 Valutazione sperimentale

4.5.1 Memoria

Si vuole stimare quanti percorsi ciascun nodo possa contenere. Nell'implementazione di IPv6 sono presenti due strutture principali per memorizzare le informazioni sulla composizione della rete: quella per i vicini (neighbors) e le routes. La prima contiene le informazioni sui nodi che un dispositivo riesce a rilevare nel proprio range attraverso e contengono l'indirizzo link-local. Nel secondo caso, contiene tutti i nodi indirizzabili globalmente, compresi i vicini con il loro indirizzo globale. In RPL una tabella di routing indirizza singlolarmente i percorsi all'ingiù, mentre i percorsi allinsù sono indirizzati attraverso la default route individuata dal proprio parent. Per stimare quante route è possibile contenere nella piattaforma si sfrutta il fatto che i moduli di utilizzano liste in spazi stabiliti a compilazione e così come le route. Quindi sono state fatte alcune compilazioni variando la grandezza di queste liste e misurando la grandezza del binario, prendendo in considerazione le sezioni .data e .bss. Il binario generato consta di 3188 byte nella sezione *.data* e 13076 nella sezione nel caso in cui il numero di routes siano pari a 20 e i vicini siano 8 e 3188 byte nella sezione .data e 15344 nella sezione .bss nel caso in cui il numero di routes siano pari a 40 e i vicini siano 20. Visto che i vicini non saranno molti di più, vengono presi in considerazione solo il numero di routes vengono aumentati il numero di routes. Facendo prove di compilazione con vari valori è risultato che con circa 1150 routes e 20 vicini il binario abbia sempre 3188 byte nella sezione *.data* e 75288 nella sezione *.bss* che sommata dà 78476 byte che sono inferiori agli 80 kilobyte della piattaforma.

4.5.2 Stima del consumo energetico

L'algoritmo per la raccolta dei dati presenta due stati energetici: lo stato di *ON*, in cui la radio è accesa e il microcontrollore è in modalità *Run* e lo stato di OFF, in cui la radio è spenta e il microcontrollore è in *Standby*. Il consumo di corrente del primo stato I_{on} è approssimativamente di circa $22mA$, $12mA$ del microcontrollore e $10mA$ della radio in ricezione, mentre nello stato di *OFF* il consumo I_{off} è di $1.3\mu A$ dovuto al solo microcontrollore. La tensione interna del microcontrollore è di 1.8 V. Una prima stima del

consumo energetico è data dalla seguente formula:

$$E_{tot} = P_{on} \times T_{on} + P_{off} \times T_{off}$$

dove considerando un ciclo di 24 ore è possibile ricavare il tempo di *OFF* dal tempo di *ON*. Più interessante, poiché bisogna stimare la grandezza della batteria, è la quantità di corrente consumata, che si può definire:

$$Q_{tot} = I_{on} \times T_{on} + I_{off} \times T_{off}$$

Il tempo necessario affichè il client CoAP riceva il dato di un singolo nodo, non considerando i 10 secondi necessari per fare la media dei dati è:

$$T_{sample} = T_{addroute} + T_{request}$$

dove $T_{addroute}$ è il tempo necessario affinchè il nodo venga aggiunto alla tabella di routing del gateway e $T_{request}$ è il tempo necessario perché la richiesta arrivi al sensore e che il client riceva la risposta. Il primo, in questo caso, è il tempo inecessario affinchè il messaggio RPL DAO arrivi al gateway ed ha un minimo dovuto al fatto che questo venga inviato periodicamente dopo alcuni secondi. In questo caso, visto che sono necessari 10 secondi per avere una misura, questo può essere considerato pari a 10, se il tempo reale è inferiore a 10. Entrambi i tempi saranno delle medie, poiché i messaggi saranno soggetti a ritrasmissioni. Il tempo di *ON* è una funzione del numero totale dei nodi, del numero di hop massimo e del numero medio dei vicini che ha ogni nodo, ma se si considera che tutto ciò avviene in parallelo il tempo di *ON* è circa il tempo impiegato dal sensore più lontano quindi:

$$\begin{aligned} T_{on}(Nnodes, MAXnhop, AvgNeigh) = T_{sample}(MAXnhop, AvgNeigh, Nnodes) + \\ + T_{shutdown}(MAXnhop, AvgNeigh, Nnodes) \end{aligned} \tag{4.1}$$

dove $T_{shutdown}$ è il tempo necessario affinchè arrivi il messaggio di spegnimento al singolo nodo ed è diversa per ogni nodo, dipendendo dal numero di hop a partire dal gateway.

Test		Totale	Media 1	Media 2	Media 3	Media 4	Media 5	Media 6	Media 7	Media 8
1 Nodo, 1 Hop	Medie (secondi)		2,619	3,673	4,057	4,501	4,923	5,754	7,006	8,353
	Campioni	203	4	34	81	38	35	4	6	1
	Frequenza	1	1,97%	16,75%	39,90%	18,72%	17,24%	1,97%	2,96%	0,49%

Test		Totale	Classi											
			3 sec	4 sec	6 sec	7 sec	8 sec	10 sec	12 sec	15/18 s	20 sec	30 sec	40/50 s	60 sec
2 Nodi, 2 Hop	Medie (secondi)			3,98	5,696	7,054	8,195	10,148	12,221	18,219				62,764
	Campioni	207		19	44	40	47	23	32	1				1
	Frequenza	1		9,18%	21,26%	19,32%	22,71%	11,11%	15,46%	0,48%				0,48%
3 Nodi, 3 Hop	Medie (secondi)				5,703		7,516	9,728	12,129	14,806	19,653	27,289		66,051
	Campioni	211			35		57	46	39	16	12	2		4
	Frequenza	1			16,59%		27,01%	21,80%	18,48%	7,58%	5,69%	0,95%		1,90%
4 Nodi, 4 Hop	Medie (secondi)						7,742	10,288	12,565	15,088	20,875	27,197	37,688	66,512
	Campioni	206					44	52	64	20	17	3	2	4
	Frequenza	1					21,36%	25,24%	31,07%	9,71%	8,25%	1,46%	0,97%	1,94%
2 Nodi, 1 Hop	Medie (secondi)		3,236	4,299	5,852		7,701	9,536	11,485		21,496	30,654		
	Campioni	207	41	61	51		14	13	21		2	4		
	Frequenza	1	19,81%	29,47%	24,64%		6,76%	6,28%	10,14%		0,97%	1,93%		
3 Nodi, 1 Hop	Medie (secondi)			3,971	5,831		7,918	10,97	13,084	18,929		31,77		69,214
	Campioni	207		38	51		26	40	23	13		15		1
	Frequenza	1		18,36%	24,64%		12,56%	19,32%	11,11%	6,28%		7,25%		0,48%
4 Nodi, 1 Hop	Medie (secondi)			4,507	6,573		8,154	11,608	13,697	18,612		31,693	47,261	
	Campioni	207		28	50		33	45	18	12		19	2	
	Frequenza	1		13,53%	24,15%		15,94%	21,74%	8,70%	5,80%		9,18%	0,97%	

Figura 4.4: Classificazione dei dati del tempo di Sample totale.

4.5.3 Esperimenti

Gli esperimenti compiuti hanno misurato il tempo di sample. Questo tempo non tiene conto dei 10 secondi necessari per avere una misura, poiché questo potrebbe nascondere delle ritrasmissioni. Partendo da una rete composta solo da un nodo, si è misurato il tempo di sample totale. A questa rete è stato aggiunto un nodo, in modo che i pacchetti indirizzati a questo passero attraverso il precedente, compiendo 2 hop, ed è stato rimisurato il tempo. Questa operazione è stata compiuta fino a 4 hop. Allo stesso modo, partendo da un nodo, sono stati aggiunti un nodo alla volta, ma in modo che questi possano essere visti direttamente tra di loro fino a 4 nodi. Questo si è fatto per stimare quanto il numero di hop totali e il numero di nodi medio potesse influire sul tempo di sample. La metrica utilizzata è la Hop Count with Threshold e come meccanismo dell'RTC è stato utilizzato il Flooding SNTP, illustrati nel capitolo 3. I parametri RPL che si possono settare sono il Timer DAO, che indica ogni quanto il messaggio di tipo DAO viene inviato e il DIO Interval, che indica ogni quanto il messaggio DIO di un nodo viene inviato. Dal primo dipende il tempo minimo affinché un nodo venga aggiunto alla rete, poiché contiene l'indirizzo del nodo raggiungibile. Il secondo invece

è un intero ed è l'esponente da dare a 2 e il valore ottenuto è il numero di millisecondi che ci sono tra un messaggio e l'altro. Nella simulazione al primo è dato il valore di 4 secondi, mentre al secondo il valore 8. Questo valore basso è dovuto al fatto che la metrica necessita alcuni campioni di RSSI. Il client CoAP effettua ritrasmissioni nel caso in cui l'Acknowledgement non venga ricevuto se scade un timeout: questo raddoppia ad ogni ritrasmissione fatta. In questo caso il timeout iniziale è settato a 2 secondi. Il tempo di trasmissione di un link che è stato misurato è di circa $40ms$. Nella Figura 4.4 vi sono i risultati. È stato utilizzato il metodo del K-Means, che suddivide i campioni in n parti secondo un criterio di fitness, di cui viene fatta la media. In questo caso i campioni sono divisi in 8 parti. Nel caso di un solo sono tutti risultati che sono intorno ai 4 secondi, che indicano che non ci sono state ritrasmissioni, al più alcuni delay dovute al backoff a livello MAC. L'introduzione di nodi in più, sia se tutti sono nello stesso range sia in multi hop, hanno fatto sì che avvengano delle ritrasmissioni o di mancate ricezioni del messaggio DAO. Negli altri casi questo non avviene, e la distanza in secondi delle classi diventa maggiore: ciò significa che sono avvenute delle ritrasmissioni. Le medie dei vari test sono state suddivise in 10 classi e 8 ritenute significative. La classe "3 secondi" è dovuta al fatto che il timer per l'invio del DAO ha un delay che non è precisamente 4 secondi, ma è un numero casuale con che varia tra 2 e 6 secondi. La classe "4 secondi", indica che il tutto ha avuto successo senza ritrasmissioni, la classe "6 secondi" indica che c'è stata una ritrasmissione di un messaggio CoAP, la "8 secondi", che la trasmissione di un messaggio DAO è stata persa. La "10 secondi" indica che ci sono state o 1 trasmissione perse del messaggio DAO e una ritrasmissione CoAP o 2 ritrasmissioni CoAP. La "12 secondi" indica che ci sono state 2 perdite del messaggio DAO. La classe "15/18 secondi", la "20 secondi" e la classe "30 secondi" indicano che ci sono state più di 2 perdite del messaggio DAO e 1 o più ritrasmissioni CoAP. Quello che è possibile vedere è che all'aumentare del numero di Hop è praticamente impossibile non avere ritrasmissioni già con 3 hop o 4 hop, e molto improbabile già con 2 hop. Questo potrebbe essere dovuto al fatto che il meccanismo del CSMA non riesce a rilevare la stazione nascosta. Inoltre rifare ogni volta la tabella di routing può avere un costo rilevante in termini di tempo che dipende dall'affidabilità del mezzo e dei percorsi creati. Sono necessari ulteriori test per verificare meglio

come il tempo aumentare dei nodi e dei salti, tenendo in considerazione che il numero di ritrasmissioni CoAP ha un limite oltre il quale la richiesta va in timeout. Questo primo risultato indica che sono necessari ulteriori meccanismi per l'affidabilità come l'utilizzo di Acknowledgement a livello MAC. Inoltre si può tentare di aumentare il DIO Interval e verificare se comunque la rete viene creata correttamente dalla metrica.

4.6 Conclusioni

È stato creato un algoritmo che permetta la raccolta dati: questo permette di far accendere e spegnere i nodi in momenti prestabiliti attraverso degli allarmi precaricati sui nodi. Questo algoritmo, per funzionare necessita che le tabelle di routing vengano rifatte ogni volta. Questo, come mostrato dalla parte sperimentale, causa un aumento del tempo di accensione, perché i messaggi che permettono la creazione di entry nelle tabelle di routing possono essere perse. Questo indica che è necessario l'utilizzo di meccanismi per l'affidabilità come l'Acknoledgement.

Conclusioni

L'obiettivo della tesi è stato di creare un prototipo di un sistema acquisizione dati da una rete di sensori wireless, in cui i sensori si accendono in momenti prestabiliti della giornata per raccogliere i dati acceleromentrici e che poi li faccia spegnere quando questi hanno terminato, permettendo così l'uso di batterie ricaricabili per poi valutare il consumo energetico. Il punto di partenza è stata la piattaforma STM32 L152 Nucleo con la radio Spirit1 su cui è caricato un porting di Contiki 2.6 con stack 6LoWPAN. A questo sono stati aggiunti i driver per l'RTC, le modalità low-power e della EEPROM, necessari affinchè un sensore potesse spegnersi e ritornare in modalità *Run* quando un interrupt dell'RTC si verifica, e per mantenere le informazioni rilevanti quando un sensore si spegne. È stato corretto il meccanismo del CSMA/CA implementato, che non comprendeva la fase di Clear Channel Assesment, necessaria per verificare la presenza di una trasmissione in corso prima di trasmettere il pacchetto. Dal punto di vista della rete, è stato implemetato una metrica che permette di creare una rete a partire dal valore di RSSI ricevuto dagli altri nodi: se questo valore è superiore ad una certa soglia, il link viene settato ad infinito, altrimenti questo ha il valore minimo. Questo ha permesso di creare una rete più affidabile rispetto alla metrica ETX: questa, però, non funziona correttamente, poichè la piattaforma non invia Acknoledgement in risposta a paccehtti MAC, necessari affinchè la metrica funzioni correttamente. Inoltre è stato necessario creare un meccanismo per la sincronizzazione dell'RTC, per evitare che si creino offset tra i vari sensori o tra i sensori e l'orologio reale. Il border router 6LoWPAN, sincronizzato esternamente, periodicamente invia pacchetti NTP, che vengono propagati dagli altri sensori attraverso la rete. È stato necessario adattare il pacchetto NTP per il funzionamento di questo algoritmo, che prevede i

campi “ID” e “Version”, non presenti. Questo permette la sincronizzazione automatica dell'RTC e il delay che ci si aspetta è il tempo di trasmissione del link per il numero di hop dal border router al nodo. Infine l'algoritmo di raccolta dati necessita che i sensori abbiano un server CoAP, con delle risorse che gestiscono gli alarmi per il risveglio, per interrogare l'accelerometro e per spegnerli. Un client CoAP, installato in una centralina, si occupa in una prima fase di inizializzare gli allarmi e succesivamente viene spenta la rete. Lo spegnimento avviene attraverso la propagazione in broadcast del messaggio di spegnimento. Il client osserva una risorsa del border router che notifica appena un nuovo nodo si aggiunge alla rete. Quando ciò avviene, il border router indica l'indirizzo IP del nuovo nodo, che verrà poi interrogato. Quando tutti questi nodi rispondono, viene inviato il messaggio di spegnimento. I test condotti utilizzano la metrica e l'algoritmo di sincronizzazione descritti sopra. Una prima indicazione sul consumo energetico è data dal tempo di *ON*, poichè da questo dipende il consumo. Da questo è emerso che già con una rete da 4 hop è praticamente impossibile non perdere un pacchetto, con il conseguente aumento di consumo energetico. Sono necessari ancora meccanismo come l'introduzione dell'Acknowledgement a livello MAC per aumentare l'affidabilità della rete, in vista dell'aumento del numero di sensori.

Bibliografia

[1] Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pages 1–314, Sept 2011.

[2] Industrial Internet Book. The model for 6lowpan architecture, 2013. [Online; accessed September, 2016.

[3] Carsten Bormann, Klaus Hartke, and Zach Shelby. The Constrained Application Protocol (CoAP). RFC 7252, October 2015.

[4] Anders Brandt, JP Vasseur, Jonathan Hui, Kris Pister, Pascal Thubert, P Levis, Rene Struik, Richard Kelsey, Thomas H. Clausen, and Tim Winter. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, October 2015.

[5] Phil Buonadonna, David Gay, Joseph M. Hellerstein, Wei Hong, and Samuel Madden. Task: Sensor network in a box. In *In Proceedings of European Workshop on Sensor Networks*, pages 133–144, 2005.

[6] L. H. Chang, T. H. Lee, S. J. Chen, and C. Y. Liao. Energy-efficient oriented routing algorithm in wireless sensor networks. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3813–3818, Oct 2013.

[7] Dr. Steve E. Deering and Robert M. Hinden. Internet Protocol Version 6 (IPv6) Addressing Architecture. RFC 3513, October 2015.

[8] S. Deering and R. Hinden. *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification.* Internet Engineering Task Force, December 1998.

[9] Adam Dunkels. The contikimac radio duty cycling protocol, 2011.

[10] Vinay Kumar and Sudarshan Tiwari. Routing in ipv6 over low-power wireless personal area networks (6lowpan): A survey. *Journal Comp. Netw. and Communic.*, 2012:316839:1–316839:10, 2012.

[11] T. H. Lee, X. S. Xie, and L. H. Chang. Rssi-based ipv6 routing metrics for rpl in low-power and lossy networks. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1714–1719, Oct 2014.

[12] Tsung-Han Lee, Hung-Chi Chu, Lin-Huang Chang, Hung-Shiou Chiang, and Yen-Wen Lin. Modeling and performance analysis of route-over and mesh-under routing schemes in 6lowpan under error-prone channel condition. *J. Applied Mathematics*, 2013:242483:1–242483:9, 2013.

[13] Michele Rondinone, Junaid Ansari, Janne Riihijärvi, and Petri Mähönen. Designing a reliable and stable link quality metric for wireless sensor networks. In *Proceedings of the Workshop on Real-world Wireless Sensor Networks*, REALWSN '08, pages 6–10, New York, NY, USA, 2008. ACM.

[14] Zach Shelby and Carsten Bormann. *6LoWPAN: The Wireless Embedded Internet*. Wiley Publishing, 2010.

[15] STMicroelectronics. Dm00047607. `www.st.com/resource/en/datasheet/spirit1.pdf`, 2015.

[16] STMicroelectronics. Dm00105823. `www.st.com/resource/en/user_manual/dm00105823.pdf`, 2015.

[17] STMicroelectronics. Cd00277537. `www.st.com/resource/en/datasheet/stm32l152r6.pdf`, 2016.