POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Delle Telecomunicazioni
Dipartimento di Elettronica e Informazione

Analisi e prestazioni di metriche energy aware nel protocollo di routing RPL per reti di sensori wireless IPv6

Relatore: Prof. Antonio Capone
Correlatore: Antimo Barbato

Tesi di Laurea di:
Emanuele Lomartire, matricola 725405

Anno Accademico 2011-2012

Alla mia famiglia

Sommario

Le reti di sensori wireless sono una particolare tipologia di rete formata da dispositivi di piccola dimensione, limitata capacità di calcolo e energia in grado di prelevare dati dall'ambiente circostante e di comunicare tra loro.
Negli ultimi anni la progettazione di protocolli di routing per reti di sensori wireless ha richiamato l'attenzione da parte degli addetti al settore, in quanto la comunicazione offre un contributo non indifferente al consumo di energia dell'intera rete.
La comunità IETF attraverso il gruppo di lavoro ROLL (Routing on Low power and Lossy networks) ha mostrato interesse su questo aspetto proponendo un nuovo protocollo di routing denominato Ipv6 Routing Protocol for Low power and lossy networks (RPL).
Il seguente lavoro di tesi prevede l'analisi e l'implementazione di metriche di routing energy aware in grado di ridurre il consumo energetico. Tali metriche, energia consumata e indice di vulnerabilità, vengono utilizzate dal protocollo RPL per la formazione del grafo di instradamento con l'obiettivo di preservare la limitata capacità energetica della rete.

Abstract

Wireless sensor networks (WSN) are a particular kind of network consisting of small devices with limited computational power and limited memory able to communicate among them and to monitor physical or environmental conditions. The energy consumption is the main issue in routing design. The IETF working group ROLL (Routing on Low power and Lossy networks) has proposed a new routing protocol called Ipv6 Routing Protocol for Low power and lossy networks (RPL).
In this thesis two energy aware routing metrics are analyzed and implemented. These metrics, called consumed energy and vulnerability index, are used in the RPL protocol for the routing tree formation in order to preserve energy lifetime of the entire network.

Ringraziamenti

Ringrazio ...

Indice

Capitolo 1

Introduzione

Negli ultimi anni nel settore delle telecomunicazioni si è assistito allo sviluppo e alla diffusione della tecnologia wireless come alternativa valida alle tradizionali reti cablate, consentendo di ottenere dei sistemi di connessioni dotati di maggiore flessibilità e versatilità.
Nei primi anni di sviluppo la rete cablata rappresentava ancora l'unica soluzione possibile, in quanto la tecnologia wireless si dimostrava poco conveniente in termini di costi e prestazioni. Tuttavia, gli enormi sforzi compiuti dalla comunità scientifica hanno portato a miglioramenti significativi nella qualità e affidabilità delle trasmissioni wireless. Di conseguenza, l'interesse da parte dell'industria per questo tipo di tecnologia è progressivamente cresciuto. Questo interesse si è tradotto in maggiori investimenti di risorse umane ed economiche che hanno permesso alle trasmissioni wireless di trovare un largo impiego in numerosi campi applicativi. Si può quindi facilmente immaginare come una tale flessibilità di utilizzo e di possibili applicazioni stimoli ulteriolmente una ricerca alquanto attiva e sempre crescente.
Nell'ambito delle tecnologie wireless, trovano uno spazio importante le reti di sensori (Wireless Sensor Networks, WSN). Una WSN è composta da un largo numero di dispositivi chiamati nodi sensore che comunicano tra loro e si occupano del monitoraggio dell'ambiente in cui operano. I nodi sensore sono generalmente dotati di una unità di calcolo, una unità di comunicazione wireless a basso raggio e componenti che permettono di rilevare grandezze fisiche (temperatura, umidità, ecc.). La comunicazione, realizzata tramite tecnologia wireless a corto raggio, è solitamente di tipo asimmetrico in quanto i nodi comuni inviano le informazioni raccolte ad uno o più nodi speciali

della rete, detti sink, che hanno il compito di raccogliere i dati e inoltrarli ad un server o ad un calcolatore.
Le reti di sensori wireless si distinguono da quelle tradizionali per via di alcuni aspetti critici che la caratterizzano: il limitato raggio di copertura del segnale wireless, la scarsa capacità di calcolo, di memorizzazione e di energia dei nodi sensore. Per questo motivo la progettazione e lo sviluppo di soluzioni specifiche per questo tipo di rete deve tenere conto inevitabilmente di queste criticità al fine di preservare il corretto funzionamento della rete stessa.
In questo lavoro di tesi sono state analizzate e implementate due metriche di routing energy aware, appartenenti alla tipologia di metriche che hanno come obiettivo principale il risparmio energetico. La comunicazione in una rete di sensori wireless concorre in maniera significativa al consumo di risorse energetiche. La comunità IETF si è interessata a questo aspetto proponendo un nuovo protocollo di instradamento per reti di sensori denominato Ipv6 Routing Protocol for Low power and lossy networks (RPL). RPL effettua la selezione dei cammini di routing in base a una funzione obiettivo che è espressione di una metrica o di una combinazione di metriche.
Le due metriche implementate sono state sviluppate all'interno di ContikiRPL, implementazione del protocollo nel sistema operativo Contiki per dispositivi embedded. La prima metrica energy aware permette al protocollo di selezionare come nodo di destinazione del pacchetto quello con minore energia consumata, mentre la seconda indica quanto un nodo sia vulnerabile energeticamente. La vulnerabilità misura in un certo modo l'attitudine di un nodo a consumare la propria energia. Solitamente i nodi più vicini al nodo sink dimostrano una vulnerabilità maggiore rispetto ai nodi foglia poichè inoltrano un numero maggiore di pacchetti.
L'implementazione ha portato alla definizione di un nuovo modulo all'interno di ContikiRPL che si occupa del calcolo, dell'aggiornamento e della gestione generale delle metriche. Le due metriche successivamente vengono testate su ambiente di simulazione Cooja del sistema operativo Contiki. I risultati ottenuti vengono poi confrontati con l'implementazione originaria di ContikiRPL, che prevede come metrica di routing il numero di trasmissioni attese (expected transmission count, ETX), ponendo l'attenzione sul ritardo medio end-to-end, sul packet loss rate, sul throughput e sull'energia consumata. Il lavoro di tesi è articolato nei seguenti capitoli:

- Capitolo 2 - Reti di sensori wireless: introduce gli aspetti fondamentali delle reti di sensori wireless;
- Capitolo 3 - Routing nelle reti di sensori wireless: illustra gli aspetti critici del routing, ovvero quei vincoli e limiti che regolano la progettazione a livello di comunicazione nelle reti di sensori wireless e fornisce una classificazione dei protocolli di routing implementati;
- Capitolo 4 - Protocollo di routing RPL: descrive in dettaglio il funzionamento del protocollo di routing IPv6 RPL;
- Capitolo 5 - Sistema operativo Contiki: presenta la piattaforma software in cui l'implementazione delle metriche è stata sviluppata;
- Capitolo 6 - Analisi e implementazione delle metriche di routing: descrive le metriche energy aware e la loro integrazione nel protocollo RPL;
- Capitolo 7 - Risultati: valuta le prestazioni dell'implementazione di RPL con le metriche energy aware in termini di ritardo medio end-to-end, throughput, energia. Inoltre le prestazioni sono state confrontate con la metrica di default expected transmission count di RPL;
- Capitolo 8 - Conclusioni: conclude il lavoro con la valutazione delle prestazioni della nuova implementazione di RPL con le metriche energy aware e riporta i commenti sui risultati sperimentali ottenuti. Inoltre propone possibili sviluppi futuri in grado di migliorare l'analisi delle metriche e il risparmio energetico.

Capitolo 2

Reti di sensori wireless

In questo capitolo viene data una visione d'insieme sulle reti di sensori wireless (Wireless Sensor Networks, WSN). L'attenzione maggiore è dedicata in particolare all'ambito del networking e alle applicazioni che fanno uso di tale tipologia di rete.

2.1 Introduzione

Una rete di sensori wireless è composta da un largo numero di dispositivi chiamati nodi sensore che comunicano tra loro nel monitoraggio di un determinato ambiente.
I sensori sono generalmente dotati di una unità di calcolo, una unità di comunicazione wireless a basso raggio e componenti che permettono di rilevare grandezze fisiche. Tipicamente le informazioni raccolte dai nodi vengono inviate verso una stazione base (sink) che è collegata a una unità di aggregazione ed elaborazione dati. Tali componenti possono essere utilizzati in ambienti molto differenti tra loro: dall'interno di un edificio ai dintorni di un qualsiasi ambiente naturale (mare, foresta, ecc.), oltre a poter essere equipaggiati con una qualsiasi tipologia di rilevatore di grandezze fisiche (umidità, calore, movimenti, ecc).
La comunicazione, realizzata tramite tecnologia wireless a corto raggio, è solitamente di tipo asimmetrico in quanto i nodi comuni inviano le informazioni raccolte ad uno o più nodi speciali della rete, detti nodi sink, i quali hanno lo scopo di raccogliere i dati e trasmetterli tipicamente ad un server o ad un calcolatore (figura 1). Una comunicazione può avvenire autonoma-

mente da parte del nodo quando si verifica un dato evento, o può venire indotta dal nodo sink tramite l'invio di una query verso i nodi interessati.

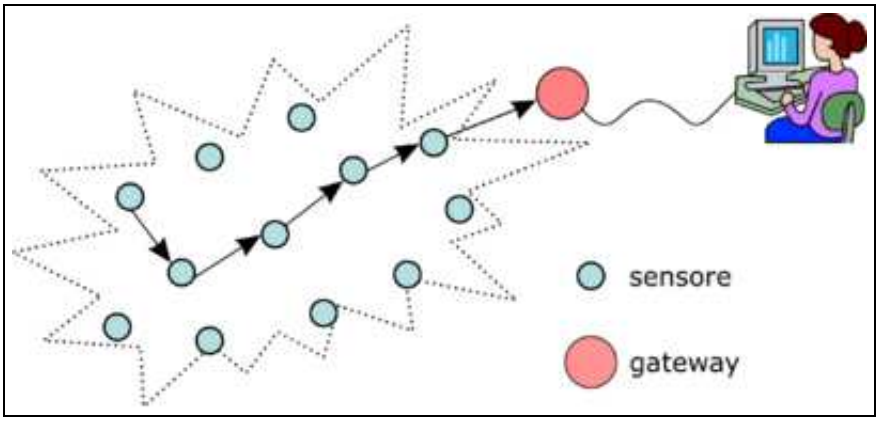


Figura 2.1: Rappresentazione di una Wireless Sensor Network

2.2 Campi di applicazione delle reti di sensori wireless

Le reti di sensori fanno parte di un'area di studio molto attiva dal punto di vista dell'innovazione tecnologica. Per l'analisi delle varie applicazioni, una prima suddivisione può essere fatta in base alle finalità della rete. Di seguito si riportano quattro macro-settori in cui è possibile dividere il mondo delle WSN.

- Monitoraggio. Questo tipo di rete viene utilizzata per tracciare in maniera continuativa una certa grandezza. La sorgente da monitorare può essere a capo di un singolo sensore o una sottorete da cui proviene un aggregato di dati. Una rete di questo genere richiede un campionamento solitamente isocrono e fisso con un consumo energetico considerevole.

- Riconoscimento di eventi. Un'altro importante scenario applicativo risulta essere l'*event detection*: la rete in questo caso deve accorgersi di situazioni di allarme, ovvero quando una determinata grandezza fisica non rientra nei livelli stabiliti. Questo monitoraggio è potenzialmente

meno esigente del precedente a livello energetico, in quanto il nodo entra in uno stato attivo solamente in casi eccezionali.

- Classificazione di oggetti. L'obiettivo di queste reti è il riconoscimento di alcune grandezze tra un insieme di prototipi noti. Questo implica un carico computazionale superiore agli altri tipi di rete.

- Tracciamento di oggetti. In questo caso la rete funge da sistema di sorveglianza, riconoscendo e stimando a posizione di alcuni oggetti in una determinata area geografica.

Oltre a questa prima suddivisione si possono descrivere i vari scenari in cui le WSN sono presenti. Inizialmente erano adoperate solo in contesti particolari come la rilevazione di radiazioni, il tracciamento di obiettivi e sorveglianza militare, la rilevazione di dati biomedici, il monitoraggio di un'area e la rilevazione di attività sismiche. Più recentemente, l'attenzione è stata rivolta verso reti di sensori biologici e chimici per applicazioni di interesse nazionale, nonchè verso lo sviluppo di soluzioni commerciali alla portata di molti potenziali utenti. Un piccolo elenco di possibili applicazioni è il seguente:

- Settore militare:
  - monitoraggio di forze nemiche;
  - monitoraggio di forze alleate ed equipaggiamenti;
  - sorveglianza di campi di battaglia;
  - stima dei danni;
  - rilevazione di attacchi nucleari o biochimici.

Come precedentemente detto, le WSN sono nate, assieme a molti altri sistemi e tecnologie oggi ampiamente presenti sul mercato, da sviluppi e sperimentazioni in campo bellico e militare.

- Settore ambientale:
  - rilevamento di incendi boschivi;
  - rilevamento di inondazioni, smottamenti o frane;
  - monitoraggio di microclimi;

  - agricoltura di precisione.

Uno dei campi di maggior utilizzo delle WSN è il monitoraggio ambientale. Con l'avanzare dell'industria e degli agenti inquinanti diventa essenziale controllare aree geografiche molto estese, soprattutto a ridosso dei centri abitati, per valutare l'impatto ambientle di alcune scelte umane. Un altro ambito di interesse è lo studio di zone pericolose, come vulcani o aree a alto rischio sismico. Gli studi scientifici su flora e fauna si possono avvalere di una rete di monitoraggio ambientale: questo porta anche nel settore agricolo un'ulteriore evoluzione scientifica, permettendo di controllare un certo tipo di coltura e di intervenire in tempo reale con trattamenti opportuni per debellare eventuali parassiti o limitare determinate situazioni critiche.

- Settore biomedico:
  - monitoraggio a distanza di parametri fisiologici;
  - tracking di personale medico o pazienti all'interno di un ospedale;
  - gestione di medicinali o attrezzatura medica di varia natura.

L'utilizzo di WSN all'interno di strutture ospedaliere può rappresentare un valido supporto per seguire cosstamente acuni parametri fisiologici dei pazienti e rendere più flessibile e portatile l'utilizzo di alcuni macchinari. Ma questo scenario è solo il primo passo verso una rete di sensori molto piccoli e distribuiti posti all'interno dei tessuti umani che possa misurare parametri quali la temperatura, la pressione sanguigna e altre grandezze di rilievo.

- Settore domestico:
  - home automation;
  - smart metering;

Tutti gli elettrodomestici più moderni hanno un'elettronica molto evoluta e il futuro consiste nella loro reciproca interazione. L'unità di controllo di questi dispositivi dispone di funzioni sempre più avanzate, dando spazio a uno scenario in cui i vari elettrodomestici possono collaborare autonomamente per gestire in maniera efficiente le faccende domestiche. La chiave quindi è la possibilità di comunicare, di conseguenza la tecnologia di comunicazione dovrà essere di tipo wireless. Un'altra applicazione importante è la gestione in maniera intelligente ed autonoma del riscaldamento, della ventilazione e dell'illuminazione.

- Settore commerciale (automotive):
  - controllo ambientale in costruzioni industriali ed edifici civili;
  - controllo di inventari;
  - tracking di veicoli;
  - monitoraggio della supply chain.

In questa categoria rientrano molti scenari diversi. Sicuramente alcuni fra i più interessanti sono: ambiente automotive, sistemi di allarme antifurto intelligenti e versatili, servizi all'interno di musei o scuole (localizzazione di persone e oggetti). Nell'ambito automobilistico trovano spazio sia il monitoraggio di traffico veicolare che l'utilizzo di strumenti di navigazione autonoma, anche se per questi ultimi il livello di affidabilità richiesto è molto elevato, superiore a quello ottenibile attualmente.

2.3 Vincoli

Le reti di sensori wireless si distinguono da quelle tradizionali per via di alcuni vincoli che la caratterizzano. Innanzitutto la principale fonte di alimentazione dei nodi sensore è costituita da batterie, le quali dispongono di una quantità di energia limitata nel tempo. Il fattore energetico risulta l'aspetto più critico per tale tipologia di rete, infatti nell'ambito della ricerca sperimentale i maggiori sforzi sono concentrati nell'ottenere la massima efficienza possibile con lo scopo di ridurre il consumo energetico complessivo. La mancanza di energia in un nodo sensore comporta inevitabilmente il suo spegnimento, impattando sulle prestazioni complessive o perfino sulla capacità della rete di svolgere il compito per cui è stata progettata. Inoltre, se si considera l'estrema difficoltà in taluni casi nella sostituzione dei nodi sensore spenti derivante dall'impossibilità di accedere in determinati luoghi naturali, si può bene immaginare quanto sia importante concentrare gli sforzi progettuali sugli aspetti energetici.
Esiste la possibilità naturalmente di equipaggiare i nodi sensore con ulteriori dispositivi in grado di acquisire energia dall'ambiente esterno. la tecnologia più utilizzata in questo senso è costituita dai pannelli solari, tuttavia vi sono anche studi riguardanti l'adozione di celle combustibile e lo sfruttamento delle vibrazioni e dei movimenti.
Al problema energetico si aggiungono il limitato raggio di copertura del

segnale wireless, la scarsa capacità di calcolo e di memorizzazione dei nodi sensore. Tuttavia è comune trovare differenti tipologie di nodi all'interno della rete che si differenziano in base al ruolo che devono ricoprire. I sensori generici, il cui scopo principale è quello di raccogliere misure dall'ambiente esterno, sono equipaggiati con dispositivi in grado di rilevare svariate grandezze fisiche come umidità, temperatura, calore, luce, pressione, accelerazione, posizione, segnli acustici, ecc. I sensori gateway invece hanno una capacità di calcolo, memorizzazione e di autonomia energetica superiori tale da permettere di svolgere operazioni di data aggregation e di inoltro verso la stazione base.
Oltre ai limiti tecnologici sopra elencati, la progettazione delle WSN dipende fortemente dalla tipologia di applicazione e di ambiente in cui essa si prefigge di operare. Nel corso degli anni sono stati sviluppati numerosi standard per WSN con lo scopo comune di soddisfare i requisiti di basso consumo energetico. Tra di essi si trovano:

- IEEE 802.15.4: è lo standard proposto per regolare i livelli MAC e fisico nelle reti che sfruttano comunicazioni wireless con basse capacità di trasferimento, basso raggio di copertura e scarsa autonomia energetica[1].

- ZigBee: comprende una serie di protocolli di comunicazione wireless di livello superiore operanti sopra IEEE 802.15.4. Essi sono progettati con l'obiettivo di risspettare i vincoli di semplicità ed energetici, sono integrati in molti dispositivi con il fine di creare reti di sensore adibite al monitoraggio ambientale[2].

- 6LowPAN: definisce un meccanismo di compressione e adattamento dell'header di un pacchetto IPv6 in modo da poterlo utilizzare con il protocollo IEEE 802.15.4. In questo modo risulta possibile stabilire comunicazioni tra i sensori e i dispositivi che utilizzano protocolli basati su IP[3, 4].

2.4 Parametri di valutazione delle reti

Le metriche rilevanti che permettono di valutare una WSN sono sostanzialmente il tempo di vita della rete, copertura, costo, tempo di risposta, accuratezza temporale, sicurezza. L'aspetto da tenere in considerazione è

legato alla possibile correlazione che intercorre tra queste metriche. Spesso potrebbe essere necessario degradare le prestazioni di una metrica allo scopo di migliorare le performance di un'altra. Tali metriche sono necessarie per descrivere le capacità prestazionali di una WSN.
L' obiettivo essenziale da tenere in considerazione è che i nodi possano operare per anni, il maggior tempo possibile, per questo motivo i nodi sensore devono operare a bassa potenza. Se si utilizzano i nodi sensore in scenari differenti, una architettura di rete WSN deve disporre di una adeguata capacità a rendersi flessibile per soddisfare i requisiti legati alle numerose applicazioni. Allo scopo di rendere la vita della rete il più duratura possibile, ogni nodo deve essere progettato per avere una determinata robustezza e in generale la rete deve essere capace di tollerare e adattarsi di fronte al guasto individuale di un sensore. La rete deve essere quindi in grado di adattarsi, monitorando il proprio stato di salute, aggiornando i propri parametri, decidendo tra nuovi compromessi (ad esempio diminuendo la qualità del servizio quando l'energia sta per terminare).
La definizione di lifetime non è univoca, nel senso che dipende dall'applicazione che si vuole misurare: a volte infatti si indica con lifetime il tempo entro il quale il primo nodo della rete finisce la propria energia (o comunque non funziona più); altre volte invece corrisponde al momento in cui metà dei nodi viene persa, oppure indica la prima volta in cui una regione sotto controllo non è più monitorata da alcun nodo. In questo contesto risulta quindi estremamente importante valutare le metriche per poter adottare una efficace strategia di prolungamento del tempo di vita della rete WSN.

2.5 Strategie progettuali

Per risolvere tutte queste sfide progettuali nel corso degli anni sono stati sviluppati diversi meccanismi per la comunicazione, architetture di sistema e protocolli di varie tipologie. Con tali sviluppi si riescono ad ottenere le seguenti importanti funzionalità:

- Connessioni wireless multihop. La comunicazione diretta tra due nodi non è sempre possibile, poichè potrebbero esserci ostacoli oppure perchè i nodi sono molto distanti tra loro e l'utilizzo di una potenza trasmissiva elevata comporterebbe un rapido esaurimento della batte-

ria. Quindi la soluzione è quella di adoperare dei nodi che fungano da relay verso altri nodi.

- Operazioni energeticamente efficienti. Risulta importante che tutte le operazioni compiute tengano in considerazione il risparmio energetico e bisogna, possibilmente, evitare la formazione di hotspot, ovvero regioni o gruppi di nodi che esauriscono la propria energia molto più rapidamente degli altri.

- Autoconfigurazione. La rete deve essere in grado di configurare automaticamente tutti i suoi parametri vitali. Per esempio deve gestire autonomamente l'ingresso di un nuovo nodo oppure l'aggiornamento delle tabelle di routing dopo la perdita di alcuni nodi.

- Collaborazione e in-network processing. In alcune applicazioni, un singolo nodo non è in grado di capire se si è verificato un evento. Per questo motivo è necessario che i nodi collaborino tra di loro e compiano in-network processing, ovvero eseguano alcuni calcoli sui dati, tipicamente data-aggregation (ad esempio il calcolo della temperatura media di una zona) in modo da ridurre la quantità di dati trasmessa attraverso la rete; oppure sfruttando la correlazione tra misurazioni di più sensori.

- Data-centric. In una rete di comunicazione tradizionale, lo scambio di dati avviene tra entità aventi ognuna un indirizzo di rete specifico, quindi si tratta di un'architettura address-centric. In una WSN, non importa tanto chi fornisce il dato, ma da quale regione proviene. D'altronde un nodo può essere ridondato da più nodi, e quindi si perde l'individualità dei vari componenti. Ciò che interessa è richiedere una certa informazione ad una certa area monitorata, e non richiedere una certa informazione ad un certo nodo. Concetto simile alla query di un database Visualizza tutte le aree in cui la temperatura è maggiore di X oppure Richiedi i dati di umidità della regione X.

- Località. Per risparmiare risorse hardware, il nodo deve interessarsi e memorizzare informazioni di routing solo verso i nodi vicini a lui. Così facendo, nel momento in cui la rete dovesse crescere esponenzialmente, le risorse hardware occupate rimarrebbero inalterate. Chiaramente,

conciliare località e protocolli di routing efficienti è una delle sfide da affrontare.

- Bilanciamento dei tradeoff. Sia durante la fase di progettazione della Wireless Sensor Network sia durante il suo runtime, bisogna ponderare diversi trade-off, anche contraddittori tra loro come lifetime e qualità del servizio, lifetime della rete e lifetime del singolo nodo, densità della rete ed efficienza del routing, solo per citarne alcuni.

Capitolo 3

Routing nelle reti di sensori wireless

In questo capitolo vengono illustrati gli aspetti critici del routing, ovvero quei vincoli e limiti che regolano la progettazione a livello di comunicazione nelle reti di sensori wireless. Viene fornita inoltre una esaustiva classificazione delle principali tecniche di routing sviluppate e implementate.

3.1 Introduzione

Come si è ampiamente discusso nel capitolo precedente, le reti di sensori permettono di raccogliere dati da un ambiente e trasferire tali informazioni verso una stazione base. Per raggiungere tale scopo, un ruolo fondamentale è ricoperto dal livello di networking dello stack di protocolli di comunicazione. Tale livello si occupa principalmente dell'instradamento delle informazioni di routing. Esso è di fondamentale importanza in una qualsiasi tipologia di rete, dai calcolatori ai sensori. Un protocollo di routing ha la funzione di determinare un cammino tra due nodi della rete che non hanno la possibilità di comunicare direttamente tra loro. Il routing nelle reti di sensori [5] è un problema di grande interesse sia per l'importanza del suo ruolo nel corretto funzionamento delle applicazioni, sia perchè lo sviluppo di questi protocolli è impegnativo a causa delle caratteristiche proprie delle reti di sensori che le distinguono dalle comuni reti wireless. Innanzitutto non è sempre efficiente costituire un sistema di indirizzamento globale stile IP a causa dell'elevato numero di nodi sensori. Inoltre, le esigenze di efficienza

energetica, computazione e capacità di memorizzazione limitata richiedono particolare attenzione anche nella progettazione del protocollo di routing. Infine, la grande varietà di possibili scenari in cui si trova ad operare una rete di sensori rende difficile individuare un protocollo di utilizzo generale.

3.2 Aspetti critici del routing

Le principali criticità legate alla progettazione di un protocollo di routing per WSN sono riportate di seguito:

- **Mezzo trasmissivo.** Le comunicazioni all'interno di una rete di sensori avvengono tramite componenti wireless con tutti i tradizionali problemi connessi a questo tipo di mezzo trasmissivo, quali fading, interferenze e alto tasso di errore oltre alla limitata potenza trasmissiva dei nodi.

- **Considerazioni energetiche.** La limitata disponibilità energetica dei nodi influenza notevolmente la durata del funzionamento della rete di sensori e il protocollo di routing deve contribuire a minimizzare il consumo di energia del nodo sensore. Inoltre in uno scenario multi-hop, ciascun nodo può assumere sia il compito di semplice monitoraggio dell'ambiente, sia il ruolo di router verso la stazione base e quindi una mancanza di energia di un nodo, con il suo conseguente spegnimento, può condurre alla necessità di dover calcolare nuovi cammini.

- **Limiti computazionali e di memorizzazione.** La limitata capacità di computazione e di memorizzazione del nodo impone al protocollo di routing una complessità non eccessiva accompagnata da una limitata necessità di informazioni da mantenere in memoria. Questi fattori devono inoltre essere scalabili rispetto alla dimensione della rete e quindi non crescere significativamente in funzione del numero di nodi.

- **Fault Tolerance.** I nodi possono incorrere in comportamenti anomali, oppure smettere di funzionare correttamente per diverse cause come la mancanza di energia, il danneggiamento fisico, oppure la presenza di un ostacoli. Il protocollo di routing deve essere in grado di gestire queste situazioni calcolando strade alternative verso la stazione base.

- **Tecniche di richiesta e consegna dellle informazioni.** I dati accumulati dai nodi possono essere inviati alla stazione base secondo tre fondamentali tecniche: time-driven, event-driven, query-driven. La prima tecnica prevede che i nodi raccolgano e trasmettano le loro informazioni secondo precisi intervalli di tempo mentre con la seconda tecnica i nodi reagiscono a particolari eventi tali da comportare significativi cambiamenti nei dati misurati dai sensori. Infine la terza strategia assume che sia la stazione base, o qualsiasi altro nodo, a chiedere informazioni inviando una richiesta direttamente al nodo che interessa la comunicazione.

- **Scalabilità.** Il numero di nodi che compongono una rete di sensori può variare dalle decine alle migliaia di unità. Tale paramentro della rete non dovrebbe influire sul funzionamento del protocollo di routing, che deve inoltre essere in grado di far fronte ad un improvviso aumento di traffico dovuto all'occorrenza di un particolare evento che stimola la trasmissione di dati da parte di un gran numero di nodi in precedenza silenti.

- **Distribuzione dei nodi.** Nelle reti di sensori strutturate, essendo i nodi distribuiti in posizioni stabilite a priori, risulta relativamente semplice specificare dei percorsi. Nelle reti non strutturate i nodi devono essere in grado di organizzarsi in maniera autonoma e determinare le rotte verso la stazione base. Inoltre in tale tipologia di rete non risulta raro trovare una distribuzione dei nodi non uniforme che quindi suggerisce l'idea di formare cluster di nodi ciascuno con un coordinatore in modo da garantire la connettività di tutte le regioni della rete attraverso una struttura multi-hop. Compito del protocollo di routing sarà anche quello di formare nella maniera più efficiente possibile tale gerarchia di cluster.

- **Dinamicità della rete.** Le reti di sensori in genere sono costituite principalmente da nodi la cui posizione non cambia nel tempo, ma esistono applicazioni che necessitano di vari livelli di mobilità. Gestire questa situazione da parte del protocollo di routing garantendo sempre una strada verso la stazione base è un problema di complessità non indifferente.

3.3 Tassonomia dei protocolli di routing

I protocolli di routing per WSN possono essere classificati secondo tre categorie principali in relazione alla struttura della rete: data-centric, gerarchico e geografico (o location-based). Queste categorie sono generali e costituiscono un paradigma di funzionamento del protocollo. A partire dalle prime versioni dei protocolli che specificavano tali paradigmi, nel corso degli anni sono stati sviluppati molte altre versioni modificate e migliorate per far fronte ai diversi scenari di applicazione. Un'ulteriore possibile distinzione è tra protocollo proattivi, reattivi oppure ibridi, a seconda di come sono determinate le rotte nella rete. Un protocollo di routing proattivo calcola i cammini prima che questi siano effettivamente necessari.Diversamente un protocollo reattivo determina la strada ogni volta che questa viene richiesta. I protocolli ibridi utilizzano una combinazione delle due precedenti tecniche. Le descrizioni dei protocolli che seguono fanno riferimento a studi piuttosto datati, ma hanno l'unico scopo di inquadrare le diverse strategie secondo le loro caratteristiche principali. Nel corso degli anni sono stati pubblicati una grande varietà di articoli che rielaborano queste idee sviluppando protocolli più performanti oppure specifici per un determinato scenario di applicazione.

3.3.1 Data centric routing

Questa tipologia di routing si adatta alle reti di sensori non strutturate costituite da un grande numero di nodi distribuiti nell'area da monitorare. A causa della grande quantità di nodi, non è efficiente creare un sistema di indirizzamento globale e inoltre la ditribuzione casuale e la densià della rete conducono spesso a una notevole ridondanza di informazione tra nodi appartenenti ad una stessa regione. Questi limiti hanno portato allo sviluppo di protocolli di routing dove si seleziona un insieme di nodi sensori e di utilizzare l'aggregazione di dati durante l'inoltro dei dati. Sono stati sviluppati vari protocolli secondo il paradigma data-centric, nel seguito vengono descritti i più significativi.

- **SPIN.** [6] L'idea chiave dietro al *Sensor Protocols for Information via Negotiation* è quella di costruire uno schema di metadati di alto livello che identifichi i dati semplici. Prima che avvenga la trasmissione del dato vero e proprio, il nodo invia un pacchetto *advertisement* che pubblicizza ai vicini il metadato corrispondente all'informazione

di cui è in possesso. Una volta ricevuto il messaggio, i nodi interessati a tale informazione inviano di ritorno una richiesta affinchè il nodo invii tramite il dato che aveva pubblicizzato in precedenza. A questo punto i nodi in possesso della nuova informazione possono a loro volta inviare un messaggio *advertisement* in modo da diffondere il dato nella rete. Tuttavia la consegna di un dato ad un nodo non è garantita dal protocollo. Un semplice esempio di tale scenario si ha quando un nodo interessato ad un'informazione è distante, nella rete, da quello che effettivamente la possiede e i nodi intermedi non sono interessati all'informazione stessa. Nella sua relativa semplicità, SPIN è stato progettato con l'intento di migliorare le prestazioni energetiche limitando una notevole ridondanza nella diffusione dei dati.

- **Directed Diffusion.** [7] In questo protocollo ogni nodo identifica i dati che genera con uno o più attributi. Gli altri nodi esprimono interessi basandosi sugli attributi. Gli interessi stabiliscono gradienti su cui vengono diretti i dati diffusi (tipicamente reverse paths). Questi possono essere rinforzati basandosi sulla qualità dei dati ricevuti. Directed Diffusion è un protocollo reattivo query-driven e quindi non si adatta ad applicazioni che richiedono un continuo invio di dati verso il nodo sink. Una versione leggermente diversa di Directed Diffusion è stata proposta in [8] sotto il nome di Gradient-based routing. L'idea è'quella di considerare il numero di hop come fattore di gradiente e quindi come costo di instradamento, in modo che ciascun nodo possa scoprire la sua distanza dal nodo sink.

3.3.2 Routing gerarchico

Per far fronte a problemi di scalabilità della rete e possibile sovraccarico dei gateway nei casi di aumento della densità dei nodi o di presenza di un unico nodo gateway viene adottato un approccio di tipo *clustering*. Nella rete vengono formati dei cluster di nodi che fanno riferimento a un nodo in particolare che assume il ruolo di *cluster-head* (CH). Tale nodo, che può disporre di risorse energetiche e di calcolo superiori, ha il compito di raccogliere le informazioni, svolgere operazioni di aggregazione e instradamento verso la stazione base. Si viene quindi a formare una struttura gerarchica organizzata in livelli, come raffigurato in figura. Il vantaggio di tale approccio è quello di ridurre il numero di pacchetti inviati al sink, garantendo una diminuzione

del consumo energetico all'interno dei cluster e quindi aumentando il tempo di vita generale della rete. A differenza del paradigma data-centric, il routing gerachico non è di tipo query-driven, ma le informazioni sono inviate dai nodi in maniera *time-driven* o *event-driven*.

- **LEACH.** [9] *Low-Energy Adaptive Clustering Hierarchy* è uno dei protocolli di router gerarchico più diffusi e popolari. I cluster-head sono scelti in maniera casuale e si occupano di comprimere i dati ricevuti dai nodi appartenenti al cluster, aggregarli e inviarli alla stazione base. La raccolta dei dati avviene in maniera centralizzata e periodica, approccio adeguato per una applicazione che necessita di monitorare costantemente parametri e grandezze fisiche. Le operazioni svolte dal protocollo si possono dividere in due distinte fasi: la formazione della struttura cluster e la consegna dei dati aggregati alla stazione base. Per quanto riguarda la selezione dei cluster-head, ciascun nodo sceglie un numero casuale r tra 0 e 1, e lo valuta in relazione alla seguente soglia *T(n)*

  $$T(n) = \left(\frac{p}{1-p(r \bmod 1/p)}\right) \quad n \in G$$

  dove p è la percentuale desiderata di cluster-head e G l'insieme dei nodi coinvolti nell'elezione. Se si verifica la condizione *r ¡ T(n)*, il nodo diventa cluster-head e comunica in broadcast la sua avvenuta elezione. I nodi non eletti selezionano a quale cluster agganciarsi sulla base della potenza del segnale del messaggio ricevuto comunicando al prescelto cluster-head la loro appartenenza. Dopo aver ricevuto le notifiche da parte di tutti i nodi, il CH assegna a ciascuno di essi un intervallo di tempo di trasmissione per ridurre al minimo le collisioni dovute a comunicazioni simultanee. Dopo un intervallo di tempo stabilito a priori viene ripetuto il processo di formazione dei cluster con la scelta di nuovi CH, in modo da portare ad un uniforme consumo di energia tra i nodi della rete. Sebbene questo protocollo sia in grado di aumentare il tempo di vita della rete, esistono delle limitazioni dovute ad alcune assunzioni sulla rete di sensori alla quale esso si applica. Innanzitutto si assume che i nodi abbiano una potenza di trasmissione tale da poter sempre raggiungere il sink, ipotesi che esclude l'utilizzo in reti distribuite in aree molto grandi. Inoltre non è garantita

una scelta omogenea nella rete dei cluster-head, quindi alcuni nodi potrebbero non avere alcun CH a cui agganciarsi.

- **PEGASIS.** [10] *Power Efficient GAthering in Sensor Information Systems* può essere considerato un miglioramento di LEACH. Il protocollo costituisce delle catene di nodi in modo tale che ciascuno di essi trasmetta e riceva pacchetti dal vicino più prossimo in termini di potenza di segnale. Una volta formata la catena, viene scelto in maniera casuale un nodo, diverso ad ogni round di comunicazione, che assuma il ruolo di leader nelle trasmissioni verso la stazione base in modo da distribuire tra tutti i nodi il maggiore carico di lavoro richiesto dalle operazioni di leader.

3.3.3 Routing geografico

Questa tipologia di routing permette di identificare i nodi per mezzo della loro posizione. La distanza tra i nodi può essere stimata sulla base della potenza del segnale entrante e, scambiando questa informazione, è possibile ottenere delle coordinate in un sistema di riferimento relativo al nodo stesso. In alternativa la posizione pu‘ essere determinata in maniera più precisa tramite strumenti GPS montati sui nodi. Tra i principali protocolli geografici si ricorda:

- **GEAR.** [11] *Geographic Energy Aware Routing* utilizza una selezione euristica dei nodi basata sulle informazioni geografiche e di consumo energetico con il fine di instradare un pacchetto verso una regione di destinazione. L'idea è quella di ridurre il numero di richieste di informazioni limitandole a determinate regioni invece di inondare tutta la rete, come avviene per esempio in Directed Diffusion. Ogni nodo che implementa GEAR memorizza due parametri di costo per raggiungere una destinazione: stimato e appreso. Il costo stimato è calcolato sulla base di una combinazione tra l'energia residua del nodo e la distanza dalla destinazione stessa. Il costo appreso invece è un raffinamento del costo stimato che tiene conto di eventuali buchi presenti nella rete. Si ha un buco quando un nodo non ha vicini maggiormente prossimi all'area di destinazione. Se la rete non presenta buchi, il costo stimato coincide con quello appreso e quest'ultimo è propagato a ritroso ogni volta che un pacchetto raggiunge l'area di destinazione. Per quanto

riguarda l'instradamento del pacchetto, esistono due scenari: il forward verso la regione target, oppure il forward all'interno della regione. Nel primo caso un nodo che riceve un pacchetto controlla se esiste un suo vicino più prossimo alla destinazione e, in caso positivo, il pacchetto è consegnato a tale vicino. Una risposta negativa al controllo potrebbe essere indice della presenza di un buco nella rete e quindi uno dei vicini è selezionato per il forward in accordo con il costo appreso, il quale sarà poi aggiornato seguendo il cammino del pacchetto. Nel secondo scenario invece, l'informazione può essere diffusa seguendo in maniera ricorsiva il procedimento appena descritto, suddividendo la regione in quadranti e inviando il pacchetto ad un nodo in ciascun quadrante, il quale opererà un'ulteriore partizione fino alla formazione di aree con un solo nodo. Un'altra tecnica, utile quando la rete non è molto densamente popolata, è un flooding dell'informazione all'interno della regione.

3.4 ROLL: Routing Over Low-power and Lossy Networks

La grande diffusione delle reti di sensori, chiamate in questa sede con il termine generale di *Low-power and Lossy Networks* (LLN), ha spinto la comunità *Internet Engeneering Task Force* (IETF) ad interessarsi ad esse con progetti come 6LoWPAN ma anche all'aspetto del routing. Come si è visto nella sezione precedente, la problematica dell'instradamento nelle reti di sensori richiede molta attenzione nella progettazione, conducendo allo sviluppo di svariati protocolli senza però avere la prospettiva futura di diventare standard. Scopo di IETF è invece quello di sviluppare un protocollo di routing che possa proporsi come uno standard per LLN. Il primo passo verso questa direzione è valutare la possibilità di utilizzare un protocollo esistente specificato da IETF nel contesto delle reti di sensori. Vengono presi in considerazione solo tale famiglia di protocolli perchè essi hanno una corrispondente descrizione in un RFC. I risultati di questa indagine sono esposti nel documento [12], in cui si conclude che nessun protocollo esistente specificato da IETF può essere utilizzato in una rete di sensori in quanto non rispetta uno o più degli aspetti critici descritti nella sezione 3.2. Per tale motivo all'interno di IETF si è costituito il *Routing Over Low-power and Lossy*

networks working group (ROLL) [13] con l'obiettivo di definire un nuovo protocollo di routing specifico per LLN. Il risultato è *Ipv6 Routing Protocol for Low power and lossy network* (RPL) [14], sarà ampiamente descritto e analizzato nel capitolo 4. Risulta interessante percorrere le considerazioni e le argomentazioni che hanno portato i ricercatori di IETF a bocciare l'utilizzo dei protocolli esistenti in LLN perchè mettono in luce i motivi di alcune scelte progettuali di RPL. Il documento [12] prende in considerazione cinque diversi criteri che un protocollo di routing deve soddisfare per poter essere utilizzato nelle reti di sensori. Tali criteri costituiscono dei vincoli di alto livello che devono essere necessariamente rispettati ma non rappresentano un elenco esauriente e completo e alcune applicazioni potrebbero richiedere ulteriori vincoli più stringenti. Essi, riprendendo quanto detto nella sezione 3.2, risultano:

- **Routing state.** Questo criterio si riferisce alla capacità del protocollo di essere scalabile rispetto alla quantità di memoria richiesta all'aumentare sia del numero di nodi, sia del numero di destinazioni a cui i dati raccolti devono essere consegnati. In particolare la relazione tra questi fattori non deve essere linerare ma sub-lineare.

- **Loss Response.** Poichè il mezzo trasmissivo wireless sfruttato nelle reti di sensori non è completamente affidabile, il protocollo di routing non deve richiedere l'utilizzo di molti link tra i nodi per propagarsi nella rete. Risulta infatti preferibile che il routing richieda la trasmissione di informazioni solo quando queste vanno effettivamente ad incidere sui cammini verso destinazioni attive oppure servono per la scoperta di nodi vicini.

- **Control cost.** Un protocollo di routing operante in una LLN non deve richiedere un elevato traffico di controllo sia per la scoperta che per il mantenimento delle rotte. Per dare un'idea numerica, una comunicazione ogni ora è di gran lunga più desiderabile di una ogni secondo, e in ogni caso il traffico di controllo deve essere limitato superiormente dal traffico dati. Questo implica che il protocollo deve essere in grado di diminuire il traffico generato quando necessario.

- **Link cost.** Il protocollo deve tenere in considerazione la qualità del link radio e includerla come metrica nella funzione che determi-

na la scelta dei cammini, con l'obiettivo di minimizzare la latenza e massimizzare l'affidabilità della comunicazione.

- **Node cost.** Analogamente al criterio precedente, il protocollo deve tenere in considerazione lo stato di un nodo nel calcolo dei cammini. In particolare valori come la potenza, la memoria e il livello di energia residuo devono influire sulla rotta che il pacchetto segue nella rete.

Nelle sezioni seguenti vengono brevemente descritti i protocolli IETF e i corrispondenti limiti del loro utilizzo in una rete di sensori. La suddivisione fatta tra protocolli basati sullo stato del collegamento (link-state) e sui vettori delle distanze (Distance Vector). All'interno di queste due categorie sono considerati protocolli di routing progettati per mobile ad hoc network (MANET), che comprendono reti mobili in cui i dispositivi hanno possibilità di muoversi liberamente in ogni direzione e le comunicazioni avvengono per mezzo del canale wireless.

3.4.1 Protocolli link-state

Il paradigma link-state prevede che ciasun router comunichi a tutti i nodi della rete lo stato dei suoi collegamenti adiacenti. Questo tipo di informazioni ricevute da tutti i partecipanti al protocollo permette di creare un database che rappresenta la topologia completa della rete. A questo punto, sfruttando algoritmi di tipo *shortest path*, risulta possibile calcolare un cammino verso la destinazione.

- **OLSR e OLSRv2.** [15, 16] *Optimized Link State Routing* è un protocollo link-state proattivo sviluppato per reti MANET. Secondo il paradigma link-state, ogni nodo ottiene una mappa della topologia della rete tramite uno scambio di messaggi contenenti informazioni sullo stato dei link. Per limitare il traffico di controllo che si generebbe a causa delle continue variazioni dei collegamenti, OLSR impone un intervallo di tempo minimo tra due successive trasmissioni che comunque possono essere opzionali. Ciascun nodo mantiene inoltre un insieme di vicini distanti un hop chiamati *Multipoint Relays* (MPR), i quali garantiscono connetività rispetto a vicini distanti due hop. Un nodo MPR contiene la lista dei nodi, detta *MPR selectors*, che lo hanno scelto come MPR. Essi inoltre sono gli unici che diffondono le informazioni link-state riguardati i nodi appartenenti alla *MPR selectors*.

In questo modo il protocollo riesce a limitare la diffusione di pacchetti di controllo e ottenere un rapido adattamento ai cambiamenti topologici. Tuttavia la dimensione della tabella di routing aumenta in relazione al numero di nodi, non rispettando il vincolo Routing State. Per quanto riguarda il traffico di controllo, esistono tecniche come la *fisheye routing* che permette di diminuire il numero di aggiornamenti inviati da un nodo riducendo il carico del traffico di controllo a livelli accettabili per una LLN, tuttavia non esiste una descrizione di come questo possa essere integrato nel protocollo OLSR. Un cambiamento nella topologia necessita l'invio di informazioni link-state attraverso la rete anche se tali link non sono utilzzati per il routing e OLSR indica che l'invio di tali informazioni è opzionale ma non descrive come gestire problemi di consistenza della topologia dovuti al mancato invio di queste informazioni. Quindi i criteri loss response e control cost necessitano di ulteriori specifiche per essere rispettati. Si può invece concludere che OLSR soddisfa i criteri link e node cost.

- **OSPF** [17, 18] *Open Shortest Path First* è un protocollo link-state che permette la suddivione della rete in diverse aree stabilendo un struttura gerarchica tra di esse. Lo stato dei collegamenti adiacenti al router è ricavato scambiando messaggi di tipo hello, e il cammino è determinato considerando metriche di link come la larghezza di banda e il ritardo di propagazione. OSPF non rispetta il vincolo routing state in quanto esso necessita della conoscenza di tutti i link della rete con conseguente dimensione linerare nel numero di router della tabella di routing. Il protocollo fallisce il criterio control cost in quanto necessita di un flooding delle informazioni nella rete sia nella fase di inizializzazione sia in caso di un cambiamento del link, anche se non utilizzato. Queste caratteristiche non permettono inoltre di rispettare il criterio loss response. Per quanto riguarda il vincolo link cost e node cost si può concludere che, viste le precedenti considerazioni sul funzionamento di OSPF, il primo è rispettato, mentre non si può dire altrettanto del secondo.

3.4.2 Protocolli distance vector

Un protocollo di tipo Distance Vector scambia informazioni sui cammini piuttosto che informazioni topologiche sulla rete. Le informazioni sono scam-

biate tra nodi vicini e riguardano i collegamenti diretti. In questo modo i nodi non hanno una conoscenza della topologia dell'intera rete, ma semplicemente sui collegamenti attivi con i vicini. Per ogni destinazione appresa viene creata una voce in tabella che ne indica la distanza basata su una qualche metrica e il primo passo per raggiungerla. I nodi inoltre comunicano informazioni sulle rotte effettivamente utilizzate con la possibilità di stabilirle on-demand. Se da una parte i protocolli Distance Vector richiedono uno scambio di dati solo con i vicini più prossimi, dall'altra essi hanno in generale tempi di convergenza più alti e possono portare alla formazione di cicli nei cammini della rete.

- **AODV.** [19] *Ad-hoc On-demand Distance Vector* è un protocollo progettato per reti MANET che costruisce i cammini solo quando questi sono effettivamente richiesti. A tale scopo, AODV inonda la rete con un pacchetto *Route Request* (RREQ) alla cui ricezione un nodo può memorizzare un percorso che permetta di raggiungere il nodo origine del pacchetto RREQ. Il destinatario della richiesta, risponde con un pacchetto *Route Replay* (RREP) che, grazie alla informazioni acquisite dai nodi durante la propagazione della richiesta, giunge alla sorgente creando e memorizzando una rotta di comunicazione tra i due nodi. AODV utilizza tecniche che permettono di controllare l'esplosione dei messaggi di richiesta nella rete in modo da indirizzarli verso il nodo destinazione insieme ad un meccanismo di numero di sequenza che previene la formazione di loop. La metrica di scelta del cammino è basata sul numero di hop. Quando un collegamento non risulta essere più valido, avviene una procedura di riparazione che permette di modificare i percorsi in modo da evitare il link rotto. Questo avviene tramite la propagazione di messaggi *Route Error* (RERR) in tutta la rete. Il numero di righe della routing table è proporzionale al numero di nodi necessari a raggiungere i capi della comunicazione e non alla densità totale della rete, caratteristica che permette di soddisfare il criterio routing state. Analogamente anche il criterio control cost è soddisfatto in quanto il protocollo genera traffico di controllo solo quando è necessario un cammino per instradare i dati. AODV invece fallisce il criterio loss response poichè la rottura di un link provoca la propagazione di messaggi necessari alla modifica dei cammini che utilizzano tale link anche se questo in realtà non è attivo per la comu-

nicazioni di dati. I criteri link cost e node cost non vengono soddisfatti poichè le proprietà del link e i nodi non vengono presi in considerazione per la scelta di percorsi.

- **RIP.** [20] *Routing Information Protocol* è un protocollo di tipo distance vector utilizzato nelle reti di calcolatori e come tale può incorrere nei problemi appena citati di loop e lento tempo di convergenza. Come metrica utilizza il numero di hop. La grandezza della tabella di routing, che tipicamente è lineare rispetto al numero di nodi della rete, può essere ridotta nel caso in cui si abbia una conoscenza a priori delle destinazioni da raggiungere senza quindi aumentare il numero delle righe in relazione al crescere della rete permettendo a RIP di soddisfare il criterio routing state. La versione del protocollo denominata Triggered RIP innesca l'invio di aggiornamenti solo quando la topologia della rete effettivamente cambia, quindi anche il criterio control cost può dirsi soddisfatto. Il criterio loss response invece non può essere considerato rispettato poichè la propagazione di aggiornamenti avviene anche se il cammino non è effettivamente utilizzato. Per quanto riguarda il link cost, l'RFC specifica la possibilità di utilizzare metriche più complesse del semplice hop-count, anche se questo potrebbe portare ad instabilità. Infine le proprietà del nodo non sono prese in considerazione dal protocollo e quindi il criterio node cost non risulta rispettato.

Capitolo 4

Protocollo di routing RPL

In questo capitolo è descritto il protocollo di routing IPv6 Routing Protocol for Low power and Lossy Networks (RPL), mettendo in evidenza i meccanismi e i principi legati alla formazione della topologia e alla segnalazione necessaria al suo funzionamento.

4.1 Introduzione

La progettazione del protocollo di routing RPL [14] è portata avanti dal gruppo di lavoro IETF ROLL (Routing Over Low power and Lossy networks) con lo scopo di fornire un protocollo standard per reti di sensori wireless. RPL è di un protocollo proattivo distance vector inquadrabile nel contesto del routing gerarchico, anche se non in senso stretto in quanto non prevede l'elezione e la formazione di cluster, ma forma una particolare struttura di grafo in cui è presente una radice rappresentata dal nodo sink o dalla stazione base. RPL è compatibile con un indirizzamento IPv6 dei sensori, caratteristica che consente di relazionare la rete di sensori wireless su cui opera con reti di calcolatori basate sullo stack TCP/IP.

4.2 Topologia e parametri

4.2.1 Grafo

Un Directed Acyclic Graph (DAG) è un grafo orientato in cui non esistono cicli e gli archi formano dei cammini verso i nodi radice, in cui tale cammino termina. I nodi radice hanno la caratteristica di non avere archi uscenti

ma solo entranti. Un DAG con un unico nodo radice che costituisce la destinazione di tutti i percorsi del grafo è detto Destination Oriented DAG (DODAG). Il protocollo RPL crea tra i nodi che lo eseguono una struttura di tipo DODAG, che può essere vista come un albero la cui radice è detta DODAG root (figura 4.1). Per tale motivo ciascun nodo può ricoprire il ruolo di genitore, figlio e foglia.

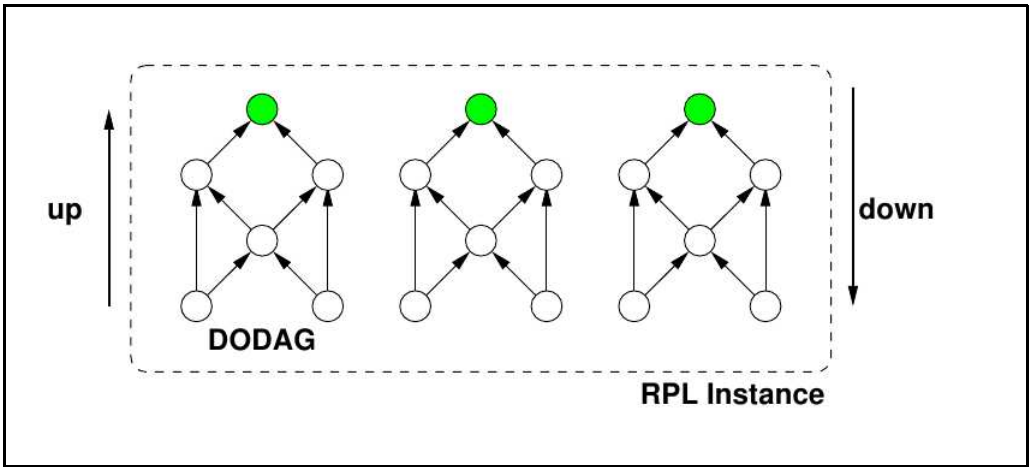


Figura 4.1: RPL DODAG e istanza

Una rete in cui opera RPL può comprendere differenti DODAG. Per mantenere coerenza tra di essi, il protocollo fornisce quattro parametri:

- *RPLInstanceID.* Questo valore identifica un'istanza di RPL. Una rete può avere una o più istanze, con differenti RPLInstanceID, ciascuna delle quali comprende un certo numero di DODAG indipendenti e non correlati basati su metriche, funzioni, vincoli differenti. Un nodo della rete può appartenere a un unico DODAG all'interno della stessa istanza.

- *DODAGID.* Scalare unico che identifica un DODAG e la sua corrispondente radice (tipicamente il nodo sink o il nodo che funge da gateway) all'interno di un'istanza. La coppia (RPLInstanceID, DODAGID) identifica in maniera univoca un DODAG nella rete.

- *DODAGVersionNumber.* Il nodo radice ha la possibilità di decidere la ricostruzione di un DODAG sulla base di certi eventi (per esempio la

creazione di un ciclo fra nodi) incrementando lo scalare DODAGVersionNumber. La tripla (RPLInstanceID, DODAGID, DODAGVersionNumber) identifica una versione del DODAG.

- *Rank.* Il rank stabilisce un ordinamento all'interno di una versione del DODAG e identifica la posizione di ciascun nodo rispetto alla radice. Esso cresce o decresce in maniera monotona a seconda che ci si allontani o si avvicini al nodo radice. Il nodo radice tipicamente presenta un rank nullo e i nodi foglia dispongono dei valori più alti, inoltre il rank di ogni nodo genitore risulta strettamente minore di quello dei figli. Ogni valore è determinato da una funzione (*objective function*) dipendente da un particolare metrica o da una composizione di metriche e vincoli.

RPL supporta tre diversi tipi di flusso di traffico: da molti a uno, multipoint-to-point (MP2P); da uno a molti, point-to-multipoint (P2MP); da uno a uno, point-to-point (P2P). Il caso MP2P è quello più comunemente richiesto da applicazioni operanti su LLN e comprende l'invio di dati da molti nodi verso uno specifico. Tale obiettivo può essere facilmente raggiunto sfruttando il traffico diretto verso la radice del DODAG.

4.2.2 Metriche

RPL non specifica esplicitamente le metriche da utilizzare per calcolare i cammini all'interno del grafo, poichè queste sono legate al tipo di applicazioni. Il documento [15] presenta in maniera generale quali tipologie di misurazioni e di vincoli dovrebbero essere seguiti da RPL nella scelta dei percorsi tra i nodi.

4.3 Messaggi RPL

I messaggi di controllo di RPL sono definiti all'interno di un pacchetto ICMPv6 (figura 4.2) e sono costituiti da un'intestazione ICMPv6 e da un corpo che comprende il messaggio stesso e una serie di possibili opzioni. Il campo Type assume il valore 155, mentre il campo Code assume uno dei seguenti codici a seconda del tipo di messaggio di controllo contenuto:

- *0x00*: DODAG Information Sollecitation (DIS);

- *0x01*: DODAG Information Object (DIO);
- *0x02*: Destination Advertisement Object (DAO);
- *0x03*: Destination Advertisement Object Acknowledgment (DAO-ACK);
- *0x80*: Secure DODAG Information Sollecitation;
- *0x81*: Secure DODAG Information Object (DIO);
- *0x82*: Secure Destination Advertisement Object (DAO);
- *0x83*: Secure Destination Advertisement Object Acknowledgment (DAO-ACK);
- *0x8A*: Consistency Check.

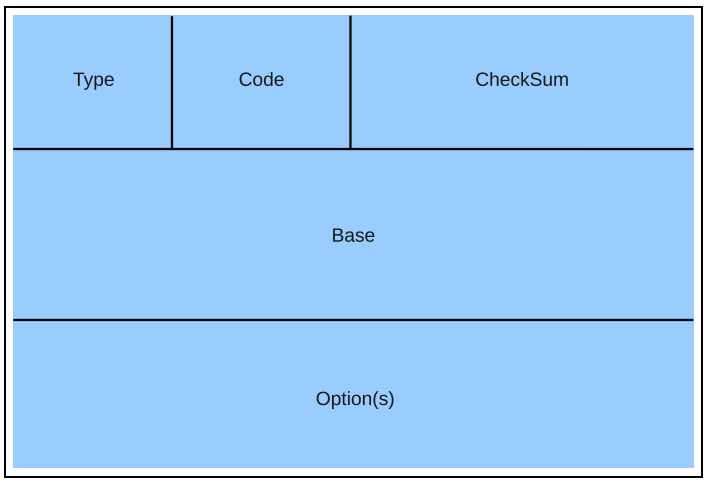


Figura 4.2: Messaggio ICMPv6

4.3.1 DODAG Information Object (DIO)

Il messaggio DIO è utilizzato per la formazione e il mantenimento dei cammini verso il nodo radice (rotte verso l'alto). Le informazioni contenute al suo interno permettono ad un nodo di scoprire l'esistenza di una istanza di RPL e apprenderne i corrispondenti parametri di configurazione. Inoltre esso permette l'identificazione di un insieme di nodi all'interno del DODAG tra i quali il nodo può scegliere il genitore diventando così parte del DODAG. I

parametri di configurazione sono generalmente inizializzati dal nodo che crea l'istanza RPL, tipicamente il sink, assumendo il ruolo di radice (root) nel DODAG. Di seguito vengono elencti i campi che costituiscono il messaggio DIO (figura 4.3):

- *RPLInstanceID*: campo di 8 bit impostato dal DODAG root che definisce l'identificativo dell'istanza di RPL di cui fa parte il DODAG.
- *Version*: campo intero di 8 bit senza segno impostato dal DODAG root che identifica la versione del DODAG.
- *Rank*: campo intero di 16 bit che identifica il rank di un nodo.
- *Grounded G*G: questo bit impostato dal root definisce se il DODAG è di tipo grounded oppure no. Un DODAG si definisce grounded se offre connettività ai nodi necessari all'applicazione per realizzare applicazioni. I nodi non grounded sono detti floating e spesso il loro unico scopo è quello di fornire connetività tra nodi che non svolgono funzioni nell'ambito dell'applicazione in esecuzione nella rete.
- *Mode Of Operation (MOP)*: campo di tre bit definito dal nodo radice che stabilisce la tecnica utilizzata per costruire i cammini dal DODAG root verso gli altri nodi. Ciascun elemento del DODAG deve essere in grado di adempiere al ruolo di router come specificato dal campo MOP, altrimenti può agganciarsi al DODAG solo come foglia. I valori possibili del campo sono i seguenti:
  - *000*: nessun cammino dal nodo radice è mantenuto da RPL;
  - *001*: modalità di non memorizzazione, non storing mode;
  - *010*: modalità di memorizzazione senza supporto per il multicast, storing mode without multicast;
  - *011*: modalità di memorizzazione con supporto per il multicast, storing mode with multicast.

  Gli altri valori sono considerati riservati. La modalità non storing sfrutta la tecnica del source routing per costruire i cammini, mentre la modalità storing utilizza le tabelle di routing.
- *DODAGPreference (Prf)*: campo intero di 3 bit senza segno impostato dal root che definisce il livello di preferenza (in ordine crescente da 0x00

a 0x07) del DODAG rispetto ad un altro all'interno della stessa istanza RPL.

- *DSTN*: il campo intero di 8 bit senza segno Destination advertisement Triggered Sequence Number (DTSN) è utilizzato per il mantenimento delle rotte dal nodo radice.

- *DODAGID*: campo intero di 128 bit senza segno impostato dal root che identifica in maniera univoca il DODAG all'interno dell'istanza di RPL.

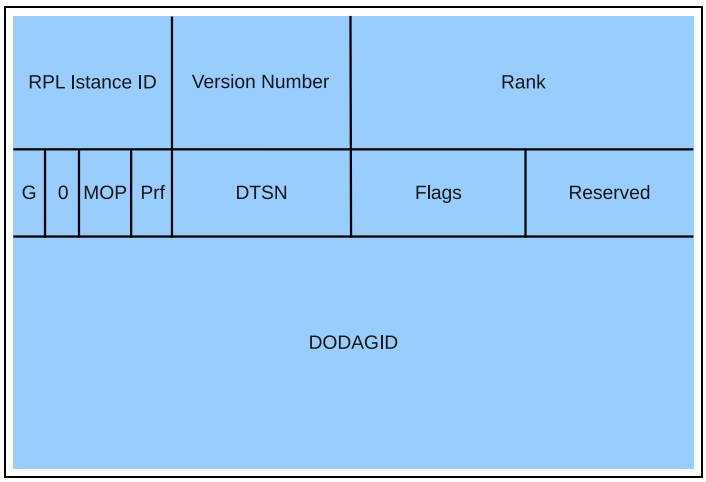


Figura 4.3: Messaggio DIO

4.3.2 DODAG Information Solicitation (DIS)

Il messaggio DIS viene utilizzato con lo scopo di sollecitare l'invio di un pacchetto DIO da un altro nodo al fine di scoprire o verificare la presenza di nodi vicini. Nella figura 4.4 sono illustrati i campi del messaggio.

4.3.3 Destination Advertisement Object (DAO)

Il messaggio DAO è utilizzato per la costruzione dei cammini dal nodo radice verso gli altri nodi del DODAG ed è mandato in modalità unicast da un figlio ad uno (o più) genitore. Il DAO puè essere opzionalmente confermato

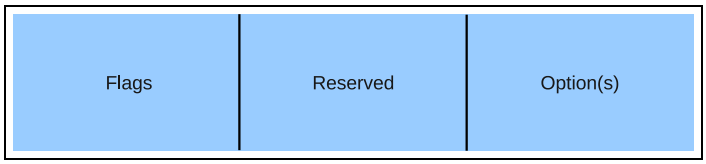


Figura 4.4: Messaggio DIS

dal genitore inviando in risposta un pacchetto DAO-ACK al corrispondente figlio. Di seguito vengono elencati i campi del messaggio DIS (figura 4.5):

- *RPLInstanceID*: campo intero di 8 bit che identifica un'istanza di RPL. Questo valore viene ricavato dal messaggio DIO.
- *K*: bit che indica se la sorgente si aspetta un pacchetto DAO-ACK in risposta al DAO inviato.
- *D*: bit indicante la presenza o meno del valore DODAGID nel pacchetto. Esso deve essere presente nel caso in cui vi siano più DODAG appartenenti alla stessa istanza di RPL.
- *DAOSequence*: valore incrementato a ciascun invio del DAO, e ricevuto uguale come risposta in un DAO-ACK.
- *DODAGID*: campo opzionale intero di 16 bit senza segno che identifica univocamente il DODAG all'interno di un'istanza RPL. Il campo è presente solo se il bit D è attivo.

4.3.4 Destination Advertisement Object Acknowledgement (DAO ACK)

Il messaggio DAO ACK è mandato dal nodo genitore al nodo figlio in risposta ad un DAO. Il formato del pacchetto è il seguente (figura 4.6):

- *RPLInstanceID*: campo intero di 8 bit che identifica un'istanza di RPL. Questo valore è appreso dal messaggio DIO.
- *D*: bit indicante la presenza o meno del valore DODAGID nel pacchetto. Esso deve essere presente nel caso in cui vi siano più DODAG appartenenti alla stessa istanza di RPL.

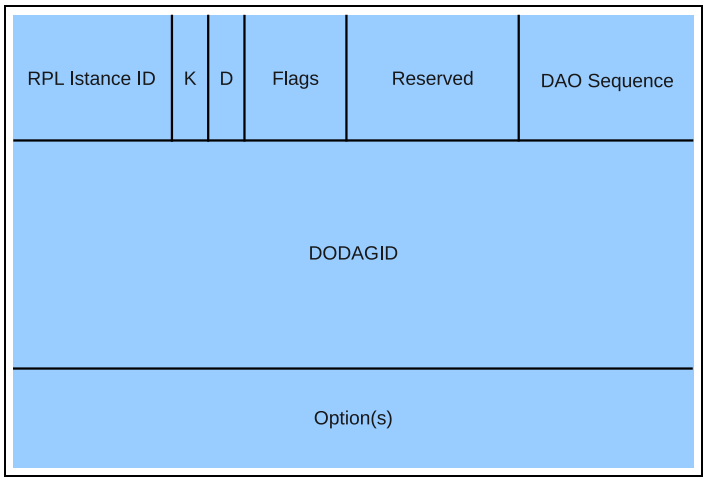


Figura 4.5: Messaggio DAO

- *DAOSequence*: valore corrispondente allo stesso campo del DAO ricevuto.
- *Status*: un valore sopra il 128 indica che il nodo dovrebbe selezionare un genitore alternativo.
- *DODAGID*: campo opzionale intero di 16 bit senza segno che identifica univocamente il DODAG all'interno di un'istanza RPL. Il campo è presente solo se il bit D è attivo.

4.3.5 Opzioni

Le opzioni che possono essere incluse nei pacchetti di controllo di RPL sono nove:

- *0x00 Pad1*: permette di inserire un ottetto di padding nel messaggio con il fine di ottenere un allineamento delle opzioni seguendo la convenzione adottata per il protocollo IPv6.
- *0x01 PadN*: permette di inserire N ottetti di padding al messaggio sempre con fini di allineamento.
- *0x02 Metric Container*: permette di diffondere e riportare valori di metrica lungo il DODAG.

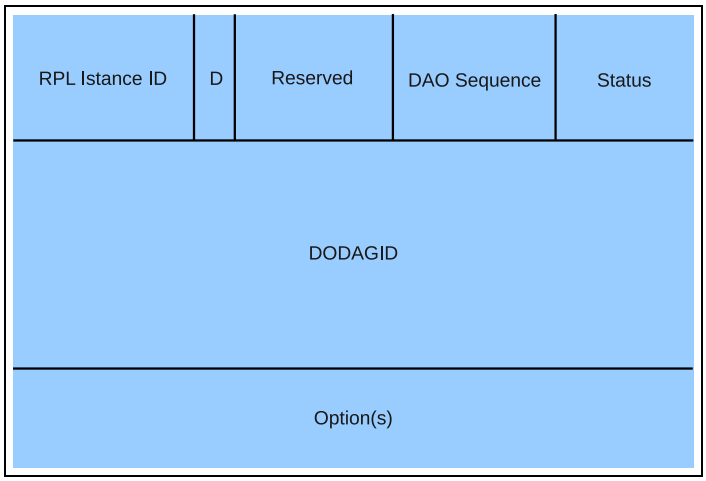


Figura 4.6: Messaggio DAO ACK

- *0x03 Route Information*: è usata per indicare se è presente connettività dal nodo root verso una destinazione specificata tramite un prefisso.

- *0x04 DODAG Configuration*: è utilizzata per diffondere parametri di configurazione necessari per il funzionamento del protocollo RPL. Queste variabili devono essere specificate dal nodo radice e solo lui deve avere il permesso di modificarle.

- *0x05 RPL Target*: utilizzata per indicare l'indirizzo IPv6 di una particolare destinazione. Può identificare una particolare risorsa che il nodo radice sta cercando di raggiungere.

- *0x06 Transit Information*: utilizzata da un nodo per indicare gli attributi di un cammino per una o più destinazioni specificate dalle opzioni RPL Target che precedono le opzioni Transit Information. Sono utilizzate principalmente quando il protocollo lavora in modalità non storing.

- *0x07 Solicited Information*: utilizzata da un nodo per richiedere l'invio di un messaggio DIO da uno o più nodi vicini.

- *0x08 Prefix Information*: permette di distribuire prefissi di indirizzamento utilizzati nel DODAG e hanno un'intestazione comune rappresentata in figura 4.7 i cui campi identificano:

– *Option Type*: campo intero di 8 bit che identifica il tipo di opzione.

– *Option Length*: campo intero di 8 bit senza segno che rappresenta la lunghezza in ottetti dell'opzione escludendo i campi Option Type e Option Length.

– *Option Data*: campo di lunghezza Option Length che contiene le informazioni specificate dall'opzione.

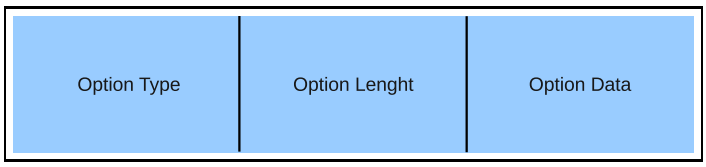


Figura 4.7: Formato comune delle opzioni

4.4 Formazione dei cammini verso il nodo radice

I cammini verso il nodo radice sono costruiti e mantenuti da RPL tramite l'invio di un pacchetto DIO tra i nodi della rete. I campi del pacchetto DIO G, MOP, Prf, Version, RPLInstanceID e DODAGID sono impostati dal nodo radice e i nodi che ricevono il messaggio devono adottare una configurazione coerente con tali paramentri e inoltrarli inalterati. Ciascun nodo invece può aggiornare i campi Rank e DSTN.

4.4.1 Scoperta dei nodi vicini, selezione dei nodi genitori e del DODAG

Un nodo scopre l'esistenza di una versione di un DODAG tramite la ricezione di un messaggio DIO da parte di altri nodi. L'origine di tale messaggio è il DODAG root e il DIO è propagato attraverso tutti i nodi della rete secondo le regole definite in seguito. Un nodo quindi riceve un messaggio DIO relativo a un DODAG da un numero arbitrario di nodi. Esso seleziona tra di essi un insieme di nodi vicini, neighbor, e un insieme di nodi genitori, parent. L'esatta specifica con cui vengono definiti tali insiemi dipende dall'implementazione, tuttavia vi sono delle regole da seguire. In particolare i neighbor sono nodi raggiungibili direttamente tramite il mezzo trasmissivo wireless. I nodi parent costituiscono un sottoinsieme dei nodi neighbour

il cui rank è strettamente maggiore di quello del nodo appena affiliato al DODAG, il cui valore è determinato, incrementato e poi propagato a partire dal rank letto nel pacchetto DIO ricevuto. Tra i nodi parent viene selezionato un unico genitore preferito, preferred parent, che rappresenta il nodo di collegamento nel cammino verso la radice il quale avrà un insieme di nodi genitore vuoto. Dalle regole appena descritte si deduce che il rank descresce di passo in passo nel cammino verso il nodo radice. Esso propaga un rank pari a ROOTRANK che costituisce il valore minimo di rank all'interno del DODAG. Nel caso in cui siano presenti più DODAG all'interno dell'istanza di RPL, un nodo sceglie quello a cui agganciarsi sulla base delle funzioni specificate nell'implementazione che tuttavia deve tenere conto della variabile Prf.

4.4.2 Gestione delle versioni di un DODAG

Una versione di DODAG è definita dalla tripla (RPLInstanceID, DODAGID, DODAGVersionNumber) stabilita dal nodo radice. Ogni elemento dell'insieme dei genitori di un nodo deve appartenere alla stessa versione del nodo che non può a sua volta comunicare all'interno del DIO un valore di versione diverso da quello di appartenenza. Quando il root incrementa il valore DODAGVersionNumber una nuova versione del DODAG viene diffusa nella rete. Non appena un nodo riceve un DIO con DODAGVersionNumber maggiore del suo attuale, deve migrare verso la nuova versione e comunicarlo nei DIO trasmessi.

4.4.3 Processamento e trasmissione del pacchetto DIO

Quando un nodo riceve un DIO, il cui contenuto non è corrotto, da parte di un nodo vicino, questo viene processato con il fine di stabilire se tale nodo puè essere posto nell'insieme dei genitori e, in caso, essere eletto preferred parent. Da notare che il rank dei nodi all'interno dell'insieme dei parent può essere minore o uguale a quello del genitore preferito. Uno dei criteri fondamentali di progettazione del protocollo è la possibilità di diminuire il traffico di controllo a seconda delle necessità. Per raggiungere tale scopo, RPL sfrutta un algoritmo detto *Trickle Timer* [22]. Esso necessita di tre parametri comunicati nel pacchetto DIO tramite le DODAG Configuration Option:

- *Imin*: rappresenta un tempo e costituisce l'intervallo di partenza dell'algoritmo.

- *Imax*: il suo valore rappresenta il massimo numero di raddoppi del valore Imin ed è un numero naturale.

- K: numero naturale maggiore di zero che rappresenta la costante di ridondanza.

Ad esse si aggiungono altre 3 variabili necessarie al funzionamento dell'algoritmo.

- I: rappresenta il corrente intervallo di tempo.

- t: valore di tempo all'interno del corrente intervallo.

- c: variabile contatore.

L'inizializzazione dell'algoritmo avviene assegnando I = Imin e c = 0. All'inizio di un intervallo, che termina dopo I unità di tempo, t viene scelto in maniera casuale all'interno del range [I/2, I]. Ogni volta che viene rilevata una trasmissione consistente, la variabile c è incrementata di una unità e quando termina l'intervallo definito da t l'algoritmo permette la trasmissione del DIO solo se c è minore di K. Infine, al termine dell'intervallo definito da I, il suo valore è raddoppiato fino a raggiungere il valore Imax, nel qual caso rimane I = Imax e viene impostato c = 0. Quando viene rilevata una trasmissione inconsistente, l'algoritmo effettua un reset delle variabili impostando nuovamente I = Imin e c = 0. Nell'ambito di RPL possiamo definire consistente la ricezione di DIO che non comporta cambiamenti del genitore preferito o del rank. Si ha invece una trasmissione inconsistente alla ricezione di un DIS multicast senza l'opzione Solicited Information oppure, nel caso in cui sia presente tale opzione, il nodo verifica i parametri definiti al suo interno prima di giudicarla inconsistente. Anche l'agganciamento ad una nuova versione di un DODAG deve essere considerata inconsistente. Il nodo non dovrebbe fare il reset del Trickle Timer alla ricezione di un pacchetto DIS diretto esplicitamente a lui, ma inviare semplicemente un pacchetto DIO con le opzioni DODAG Information Option in unicast al nodo origine e, nel caso in cui sia presente l'opzione Solicited Information, l'invio è vincolato alla coincidenza dei parametri presenti nell'opzione stessa.

4.5 Formazione dei cammini dal nodo radice

La gestione e la creazione dei percorsi dal nodo radice verso gli altri nodi (rotte verso il basso), che permette di gestire il traffico P2MP e P2P, avviene tramite i pacchetti DAO inviati verso il nodo radice. Questo è possibile tramite la selezione di uno o più next-hop, detti DAO parent, all'interno dell'insieme dei DODAG parent che può coincidere o meno con il preferred parent. I campi RPLInstanceID e DODAGID del pacchetto DAO devono avere gli stessi valori presenti nel DIO, mentre il campo DAOSequence è incrementato al massimo di una unità per ogni nuovo DAO inviato. Infine l'abilitazione del bit K e la conseguente richiesta di un DAO-ACK è opzionale, ma un nodo può comunque inviare un DAO-ACK in risposta a un DAO senza K impostato con il fine di segnalare degli errori. Come già precedentemente accennato, è possibile disabilitare la creazione delle rotte verso il basso, oppure decidere la modalità con cui queste vengono mantenute, impostando il campo MOP del pacchetto DIO. Nel caso in cui tale valore sia 0, nessun nodo agganciato al DODAG deve inviare messaggi DAO e ignorarli in ricezione. Per la formazione dei cammini dal nodo radice sono previste due modalità, *storing* e *non storing*, che verranno descritte di seguito.

4.5.1 Modalità non storing

Nella modalità non storing, il cammino in direzione downlink (dal nodo radice verso un nodo) viene formato ricorrendo alla tecnica di IP source routing, non ricorrendo alla formazione e utilizzo di tabelle di routing. In tale modalità sono previste le seguenti regole:

- il campo Parent Adress di Transit Information Option deve contenere uno o più indirizzi. Tutti questi indirizzi devono essere DAO parent del nodo sender;
- i DAO vengono inviati direttamente al nodo radice mediante un cammino di default installato nella fase di selezione del parent;
- quando un nodo rimuove un altro nodo dal suo insieme di DAO parent, esso deve generare un nuovo messaggio DAO con il valore di Transit Information Option aggiornato.

Nella modalità non storing, ogni nodo utilizza i messaggi DAO per riportare i suoi parent al nodo radice. Il nodo radice può ricostruire il cammino verso un nodo utilizzando gli insiemi dei parent ricevuti da ogni nodo. Lo scopo di questa modalità consiste nel minimizzare il traffico di pacchetti di instradamento nel caso di frequenti cambiamenti nell'insieme dei DAO parent.

4.5.2 Modalità storing

Nella modalità storing, la creazione dei cammini in direzione downlink viene effettuata ricorrendo agli indirizzi di destinazione IPv6. In questo caso il messaggio DAO non contiene informazioni sul percorso, ma questo è derivato implicitamente dal pacchetto DAO stesso tramite l'indirizzo sorgente e l'indirizzo destinazione verso cui il pacchetto è stato instradato. Anche in tale modalità i pacchetti sono indirizzati verso il nodo radice, ma ciascun nodo che riceve il DAO memorizza in una tabella l'informazione su chi ha generato tale DAO e il nodo vicino che permette di raggiungerlo. Quindi ciascun DAO parent avrà una tabella relativa ai nodi del proprio sub-DODAG.

Capitolo 5

Sistema operativo Contiki

In questo capitolo è stata effettuata una descrizione di Contiki, sistema operativo che opera su dispositivi caratterizzati da scarsa disponibilità di memoria e energia. Si pone l'attenzione sull'architettura del kernel, sulle funzionalità fornite dai servizi e dalle librerie, sulle modalità di gestione e di supporto alla comunicazione.

5.1 Introduzione

Nell'ambito delle reti di sensori wireless, le risorse di cui dispongono i sensori sono solitamente molto limitate, l'alimentazione fornita dalle batterie o celle solari è ridotta e limitata nel tempo e la necessità di limitare fortemente i costi energetici legati al dispositivo complica ulteriolmente la progettazione dell'architettura. Contiki, il cui nome deriva dalla zattera Kon-Tiki con la quale Thor Heyerdahl fece una spedizione per dimostrare che i polinesiani erano in grado di ritornare dal Sud America al loro arcipelago [23], è un sistema operativo implementato in linguaggio C, event-driven, con preemptive multithreading opzionale che può essere applicato a singoli processi. È stato sviluppato per essere utilizzato su un vasto numero di sistemi che vanno dai computer a 8-bit fino ai sistemi embedded su microcontroller. Contiki è disponibile per molteplici piattaforme (Sky/TelosB, Atmel AVR, etc.) che hanno in comune l'architettura della CPU basato su un modello di memoria senza segmentazione e senza meccanismi di protezione. Il codice è installato nella ROM riprogrammabile (EEPROM) ed in RAM. Le uniche astrazioni fornite dal codice di base sono il multiplexing della CPU ed il supporto per il caricamento dinamico di applicazioni e servizi. Per il resto le altre as-

trazioni sono implementate tramite librerie, come ad esempio il preemptive multi-threading.

5.2 Caricamento del codice in run-time

Nelle reti di sensori wireless, composte da un elevato numero di dispositivi, la possibilità di caricare codice tramite il network risulta rilevante, specialmente nei casi in cui è difficile recuperare fisicamente i sensori, per aggiornare le applicazioni e per risolvere velocemente i bug del codice. Sono disponibili alcuni metodi per il caricamento dinamico del codice, ma comune a tutti vi è la necessità di limitare il numero di bytes che transitano in rete, per ridurre la quantità di energia necessaria. Molti sistemi operativi richiedono il caricamento completo dell'immagine binaria, che include il sistema operativo, le librerie di sistema e le applicazioni che funzionano sopra al sistema operativo. Contiki permette il caricamento e la sostituzione dinamica di programmi e servizi [6], in modo flessibile, mantenendo il sistema base leggero e compatto. Molto spesso le singole applicazioni sono di dimensione trascurabile rispetto all'intero sistema e perciò richiedono meno energia nella trasmissione nel network ed il tempo di trasferimento risulta sensibilmente inferiore. I programmi caricabili sono implementati tramite una funzione di rilocazione run-time ed il codice contiene le informazioni necessarie per la rilocazione: quando un programma è caricato nel sistema, il loader tenta di rendere disponibili le risorse secondo le informazioni nel codice e se le risorse non sono disponibili l'esecuzione viene arrestata. Dopo che il codice viene caricato, viene chiamata una funzione di inizializzazione.

5.3 Modello event-based o multi-threaded

Nei dispositivi con limitate risorse di memoria, un modello di programmazione multi-threaded può richiedere un elevato ed indefinito consumo di memoria: ogni thread deve disporre di uno stack proprio, inoltre è difficile conoscere a priori con esattezza la quantità di memoria richiesta e quindi si tende a sovradimensionarlo e a caricarlo al momento dell'esecuzione. La memoria contenuta nello stack non può essere condivisa con gli altri thread. Quando si ha a che fare con dispositivi dalle capacità di memoria e energia limitate, i sistemi operativi vengono progettati seguendo un approccio di

tipo event-based, evitando in questo modo un utilizzo cospicuo di memoria e cercando nello stesso tempo di fornire supporto ai processi concorrenti. In sistema di tipo event-based, i processi hanno il compito di gestire gli eventi e di raggiungerne il completamento. E' possibile modellare il comportamento del processo come flusso di eventi (e quindi come una serie di transizioni di stato) all'interno di una macchina a stati finiti.Il gestore dell'evento non può essere bloccato, ma per dare spazio ad un processo successivo deve essere prima portato a compimento (semantica di tipo run-to-completion). In questo caso lo stack di memoria è condiviso da tutti i processi e la memoria utilizzata risulta essere molto esigua rispetto al caso di multithreading. Tuttavia tale astrazione comporta anche alcuni aspetti negativi. Un sistema orientato agli eventi è composto da una sezione di rilevazione (operazione che avviene mediante polling o interruzioni hardware) e da un event-dispatcher che provvede ad invocare le routine di gestione degli eventi (event-handlers) definite. Un'applicazione definisce tali procedure. Eventuali identificatori locali definiti nel handler non saranno più accessibili nel momento in cui il controllo viene ritornato al dispatcher in quanto il record di attivazione della procedura che li contiene è disallocato. Essendo il flusso del dispatcher unico, ogni event-handler non può definire statement esplicitamente bloccanti (come ad esempio il wait) perchè ciò provocherebbe l'interruzione del flusso. L'esecuzione di un handler termina (con il ritorno del controllo al dispatcher) solo nel momento in cui tutti gli statement contenuti in esso sono stati eseguiti o quando non è necessario attendere un evento (gli handlers non sono preemptive). Questo significa che l'utilizzo dei sistemi orientati agli eventi è giustificabile solo nel caso di processi che fanno largo utilizzo delle interfacce I/O in quanto permettono l'avvicendamento dei processi e quindi la massimizzazione del rendimento di CPU. Al contrario programmi con prevalenza d'elaborazione, come routine di calcolo matematico, rischiano di monopolizzare l'utilizzo del processore visto che l'esecuzione degli event-handlers non è preemptive. C'è da aggiungere inoltre che la maggior parte delle applicazioni di alto livello, come ad esempio le applicazioni legate allo stack TCP/IP, richiedono nei rispettivi programmi un numero alquanto elevato di stati rendendone lo sviluppo, il debug e la manutenzione compiti alquanto difficoltosi. I protothread, che sono alla base del funzionamento di contiki, cercano di ovviare a tali problemi, venendo incontro alle esigenze di utilizzo di memoria limitato e di semplicità e flessibilità nello stile

programmativo. Per il funzionamento dei protothread si veda l'appendice X.

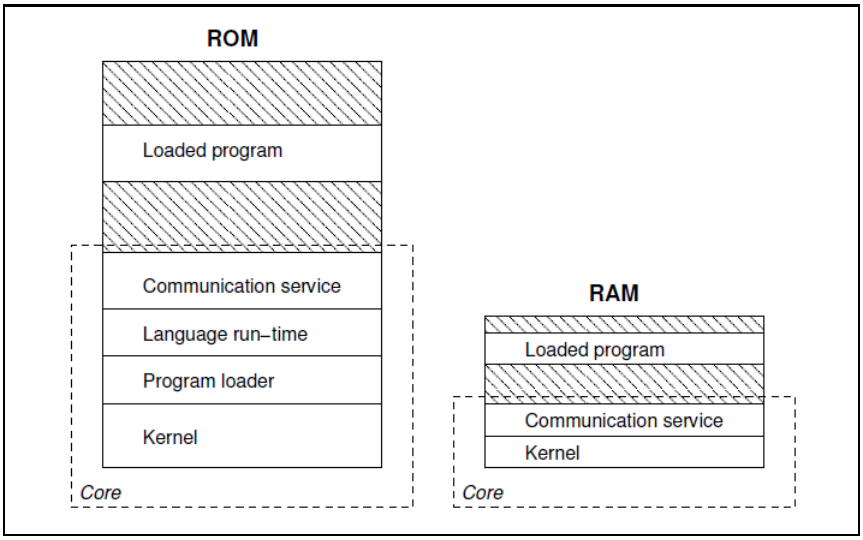


Figura 5.1: Contiki

5.4 Panoramica del sistema

Il sistema operativo Contiki è composto dalle seguenti componenti: il kernel, il program-loader e un insieme di processi (figura 5.1). Un processo può essere un'applicazione o un servizio. Un servizio implementa funzionalità che possono essere caricati in maniera dinamica in run-time. La comunicazione tra i processi avviene tramite il kernel, il quale non fornisce un livello di astrazione hardware ma ne permette la comunicazione diretta con i drivers e le applicazioni. Un processo è definito da un gestore degli eventi (event-handler) e da una funzione opzionale poll handler. Lo stato del processo è mantenuto in una memoria privata, mentre il kernel mantiene solamente un puntatore allo stato del processo. Tutti i processi condividono lo stesso spazio di indirizzamento. La comunicazione tra i processi avviene tramite il posting degli eventi. Contiki è partizionato in due parti: il core ed i loaded programs. Il partizionamento è realizzato in fase di compilazione e tipicamente il core è composto dal kernel, il program loader, le più comuni parti del language run-time, le librerie di supporto e lo stack di comunicazione con i driver per l'hardware di comunicazione. Il core è compilato in una

singola immagine binaria che non viene generalmente modificata dopo il deployment, anche se con un boot loader speciale è possibile sovrascrivere il core oppure applicare patch. Il program loader può caricare i programmi nel sistema sia tramite lo stack di comunicazione oppure utilizzando l'EEPROM locale: tipicamente un programma viene prima posizionato nella EEPROM e poi caricato nella code memory.

5.5 Architettura del kernel

Il kernel di Contiki consiste di uno scheduler di eventi che assegna gli eventi ai processi e periodicamente interpella i polling handlers dei processi. L'esecuzione dei processi è scatenata dagli eventi distribuiti dal kernel oppure tramite il meccanismo del polling. Il kernel non sospende un event handler scheludato ma deve attenderne il completamento,tuttavia sono previsti meccanismi interni che ne permettono la sospensione (preemption). Gli eventi supportati dal kernel sono di due tipi: asincroni e sincroni. Gli eventi asincroni vengono messi in coda dal kernel e distribuiti con leggero ritardo, mentre gli eventi sincroni causano l'immediata schedulazione del processo incaricato della gestione. In aggiunta alla gestione degli eventi il kernel supporta il meccanismo di polling che può essere visto come un evento ad elevata priorità che viene gestito in mezzo a due eventi asincroni. Il polling è utilizzato dai processi che necessitano di verificare lo stato dell'hardware e quando un poll è schedulato, tutti i processi che implementano il poll handler vengono eseguiti in base alla loro priorità. Contiki utilizza un singolo stack condiviso tra tutti i processi e l'utilizzo di eventi asincroni riduce lo spazio utilizzato, dato che lo spazio è riutilizzato ad ogni gestione di nuovo evento. La schedulazione degli eventi è effettuata in un unico livello e nessun evento può portare alla sospensione di altri eventi. L'unica possibilità di sospensione di un evento è data da interrupt che normalmente sono implementati mediante hardware interrupt oppure mediante esecuzioni real-time. L'ultima tecnica è stata precedentemente utilizzata nell'ambito del kernel di Linux [24]. Contiki non permette che i gestori di interrupt possano generare eventi, per evitare race-conditions, in alternativa è possibile richiedere un poll event tramite un flag per effettuare una richiesta immediata di polling. I programmi caricabili vengono implementati utilizzando una funzione di riallocazione run-time e un formato binario che contiene le informazioni di

riallocazione. Quando un programma viene caricato nel sistema, il program loader cerca di allocare una certa quantità di memoria basandosi sulle informazioni di memoria fornite dal binario. Se l'allocazione di memoria fallisce, il caricamento del programma si arresta. Una volta che viene effettuato con successo il caricamento in memoria del programma, il program loader chiama la funzione di inizializzazione del programma. Tale funzione può inizializzare o rimpiazzare uno o più programmi. Nelle reti di sensori solitamente è richiesta la possibilità di spegnere i nodi sensore quando la rete è inattiva al fine di ridurre fortemente il consumo energetico e preservarne la durata di funzionamento. I meccanismi di conservazione dell'energia dipendono principalmente sia dalle applicazioni [25] che dai protocolli di routing [26]. In merito a questo aspetto, il kernel di Contiki non contiene esplicitamente funzioni di risparmio energetico ma ne demanda l'implementazione allo strato al livello applicativo. Tuttavia, lo scheduler fornisce la misura della coda degli eventi al fine di aiutare l'applicazione a capire quando decidere di porre il nodo in stato di sleep, solitamente quando non ci sono eventi in coda nello scheduler. In caso di interrupt, il processore si sveglia per riprendere le sue normali funzioni e un poll handler si occuperà di gestire l'evento esterno.

5.6 Servizi

Un servizio è un processo che implementa una funzionalità che può essere utilizzata da altri processi, come una libreria condivisa. Un service può essere dinamicamente sostituito durante il run-time e perciò deve essere linkato dinamicamente. I servizi sono gestiti da un service layer che tiene traccia dei servizi in esecuzione. Tipici esempi di servizi sono costituiti da stack di protocolli di comunicazione, da driver dei dispositivi e da funzionalità di alto livello come algoritmi di gestione dai dati catturati dai sensori. I servizi sono gestiti da un service layer concettualmente situato a ridosso del kernel. Il service layer tiene traccia dei servizi in esecuzione e fornisce la possibilità di trovare i servizi installati. Un servizio è identificato da una stringa che lo descrive ed è costituito da una interfaccia e da un processo che la implementa . L'interfaccia consiste di un numero di versione ed di una tabella contenente i puntatori alle funzioni che implementano l'interfaccia (figura 5.2). Le applicazioni utilizzano una service stub library per comunicare con

il servizio, la quale utilizza il service layer per trovare il servizio ed ottenerne il process ID per le successive richieste. In particolare, la prima volta che il servizio è richiamato, la service interface stub lo ricerca nella service layer e se il servizio esiste viene ritornato un puntatore all'interfaccia, che contiene un numero di versione e i puntatori a tutte le implementazioni delle funzioni contenute nel servizio. Se i numeri di versione coincidono, l'interface stub richiama l'implementazione delle funzioni richieste. Come gli altri processi, anche i servizi possono essere caricati dinamicamente e rimpiazzati. E' cruciale che il process ID venga mantenuto, dato che è utilizzato per l'identificazione del servizio. Quando un servizio deve essere rimpiazzato, il kernel fornisce la versione in esecuzione inviando un evento particolare al service process. Il servizio prima di rimuoversi autonomamente dal sistema deve trasferire l'internal state al nuovo servizio tramite passaggio di puntatore e la memoria deve essere prelevata da una zona condivisa, dato che la memoria del vecchio processo viene completamente liberata. La descrizione dello stato del servizio è etichettata col numero di versione del servizio, in modo che versioni incompatibili dello stesso servizio non tentino di caricare la descrizione del servizio.

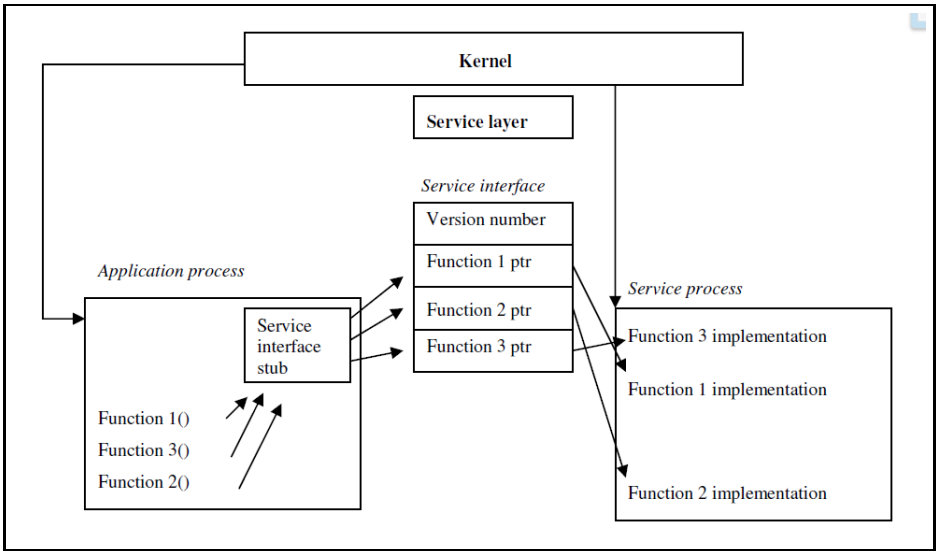


Figura 5.2: Invocazione di un servizio di Contiki

5.7 Librerie

Il kernel di Contiki fornisce le sole funzionalità di base di multiplexing della CPU e la gestione degli eventi. Il resto del sistema è implementato con librerie di sistema che vengono opzionalmente linkate nei programmi in tre modalità. Nel primo modo i programmi possono essere staticamente linkati alle librerie che fanno parte del core, nel secondo modo i programmi possono essere linkati con librerie che fanno parte dei programmi caricabili e nel terzo modo i programmi possono richiamare i servizi che implementano una libreria specifica. Le librerie implementate come servizi possono essere rimpiazzate dinamicamente durante il run-time. Le librerie che fanno parte del core sono sempre presenti nel sistema e non devono essere incluse nei binari dei programmi caricabili. Tipicamente, le librerie di cui si prevede un largo uso dovrebbero far parte del core di Contiki, invece le librerie di cui si prevede un uso non frequente o che sono specifiche di una particolare applicazione dovrebbero essere linkate con i programmi caricabili. Come esempio chiarificatore, si consideri un programma che utilizza le funzioni memcpy() (copia in memoria) e atoi() (conversione di stringhe in interi). La prima funzione è usata frequentemente come funzione di libreria del linguaggio C mentre la seconda risulta essere meno frequente. Quindi, in questo caso la prima funzione dovrebbe essere inclusa nel core del sistema mentre non la seconda. Quando il programma viene linkato per la creazione del codice binario, la prima funzione viene linkata tramite il suo indirizzo statico al core, mentre la seconda funzione dovrà essere assolutamente inclusa nel codice binario del programma.

5.8 Supporto alla comunicazione

La comunicazione è un concetto fondamentale nelle reti wireless di sensori. In Contiki la comunicazione è implememntata come servizio al fine di consentirne il rimpiazzo in run-time e di prevedere la possibilità di caricare simultaneamente più stack di comunicazione. Nell'ambito della ricerca sperimentale, questo aspetto consente di valutare e confrontare i differenti protocolli di comunicazione. In aggiunta a questo, lo stack di comunicazione può essere suddiviso in più servizi coe mostrato nella figura 5.3. Ciò permette la sostituzione in run-time di singoli elementi dello stack. I servizi di comunicazione utilizzano il meccanismo dei servizi per richiamarsi tra loro

e gli eventi sincroni per comunicare con gli applicativi. PoichÃ© gli eventi sincroni devono tassativamente arrivare a completamento è possibile utilizzare un unico buffer per i processi di comunicazione, senza la necessità di copiare dati. Un driver di dispositivo legge i pacchetti in ingresso nel buffer di comunicazione e poi richiama i servizi dei livelli superiori con il meccanismo dei servizi. Lo stack di comunicazione elabora l'header del pacchetto e genera un evento sincrono all'applicazione destinataria del pacchetto. L'applicazione lavora sui contenuti del pacchetto ed eventualmente posiziona una risposta nel buffer, prima di restituire il controllo allo stack di comunicazione, che aggiunge i propri header al pacchetto in uscita e restituisce il controllo al driver del dispositivo, in modo da consentirne la trasmissione.

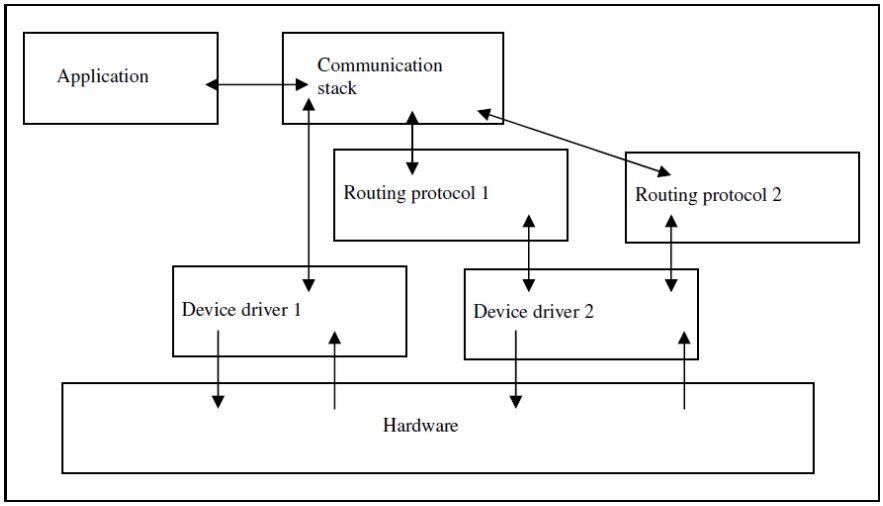


Figura 5.3: Stack di comunicazione di Contiki

5.9 Preemptive multi-threading

Il preemptive multi-threading è implementato in Contiki come libreria sopra al kernel che può essere linkata opzionalmente alle applicazioni che lo prevedono esplicitamente. La libreria è divisa in due parti: una parte indipendente dalla piattaforma, che si interfaccia con il kernel ed una parte specifica per piattaforma che implementa lo switching dello stack e le primitive di preemption (25 linee di codice C per MSP430). La preemption è implementata utilizzando un timer interrupt che salva i valori dei registri

del processore nello stack e ritorna allo stack del kernel. Ogni thread, a differenza dei normali processi, richiedono uno stack separato ed ogni thread esegue col proprio stack finchè non viene rilasciato. La API della libreria multi-threading consiste di quattro funzioni che possono essere chiamate dal thread in esecuzione (mt yield(), mt post(), mt wait(), and mt exit()) e di due funzioni che vengono invocate per il setup e l'esecuzione del thread (mt start() and mt exec()). La funzione mt exec() esegue l'effettiva schedulazione ed è invocata da un gestore di eventi.

5.10 Dimensione del codice

Un sistema operativo per sistemi dalle risorse limitate deve risultare compatto sia in termini di dimensione del codice che in termini di occupazione della memoria RAM, al fine di consentire sufficiente spazio per le applicazioni. La seguente tabella 5.4 evidenzia i dati ricavati dall'analisi della dimensione del codice e l'occupazione della RAM da parte di Contiki in versione 2.1 compilato per due piattaforme: Texas Instruments MSP430 e Atmel AVR. I valori riportati riguardano sia la dimensione dei componenti del core che quella di una applicazione di esempio: un servizio di replica dei dati catturati dai sensori. Tale applicazione consiste di una interfaccia di servizio e dell'implementazione del servizio stesso. La dimensione del codice di Contiki risulta maggiore rispetto a quella di TinyOS [27] ma minore rispetto a quella del sistema operativo Mantis [28]. La dimensione del kernel di Contiki risulta molto più grande rispetto a quella del kernel di TinyOS data l'elevata disponibilità di servizi forniti. Mentre il kernel di TinyOS fornisce solamente uno scheduler degli eventi di tipo FIFO, il kernel di Contiki supporta sia eventi FIFO che gestori di polling con priorità. La quantità di RAM richiesta dipende dal massimo numero di processi configurati per il sistema (p), dalla massima dimensione della coda asincrona degli eventi (e) ed in caso di multi-threading, dalla dimensione dello stack del thread (s).

Module	Code Size (AVR)	Code Size (MSP430)	RAM Usage
Kernel	1044	810	10+4e+2p
Service Layer	128	110	0
Program Loader	-	658	8
Multi-threading	678	582	8+s
Timer Library	90	60	0
Replicator Stub	182	98	4
Replicator	1752	1558	200
Total	**3874**	**3876**	**230+4e+2p+s**

Figura 5.4: Dimensione del codice e di RAM

Capitolo 6

Analisi e implementazione delle metriche di routing

In questo capitolo vengono valutate le metriche energy-aware che sono oggetto della tesi. Viene inoltre fornita una descrizione dell' implementazione delle due metriche proposte e del loro adattamento all'interno dei moduli di ContikiRPL, versione del protocollo presente nel sistema operativo Contiki.

6.1 Descrizione delle metriche di routing

Il protocollo RPL specifica la modalità di costruzione e mantenimento del grafo di routing (Destination Oriented Directed Acyclic Graph (DODAG)) utilizzando una funzione obiettivo (objective function). La funzione obiettivo è rappresentata da una metrica, un vincolo o una combinazione dei due. Le implementazioni del protocollo RPL presenti attualmente sono due: TinyRPL (nel sistema operativo TinyOS) e ContikiRPL (in Contiki). Al fine di implementare e valutare le metriche proposte, ho scelto di ricorrere a ContikiRPL perchè al momento della scelta era l'implementazione più completa del protocollo, o meglio quella che copriva la maggior parte delle funzionalità e caratteristiche. Entrambe le implementazioni al momento presentano solo due funzioni obiettivo Of0 e Of-ETX: la prima prevede la selezione del nodo parent con il valore minimo di hop count, mentre la seconda porta a scegliere il nodo parent con valore minimo di ETX (expected transmission count). In questo lavoro ho scelto di non prendere in considerazione la funzione obiettivo di minimizzazione del hop count in quanto essa porta all'ottimizzazione

del cammino di routing solo in condizioni ideali di trasmissione e ricezione e non persegue il criterio di risparmio energetico. La scelta della metrica da confrontare con le due metriche energy-aware è ricaduta su ETX, in quanto ho considerato che rispetto a hop count la sua minimizzazione è mirata alla ricerca di cammini caratterizzati da una bassa probabilità di ritrasmissione dei pacchetti che ovviamente portano non solo ad avere ritardi end-to-end contenuti ma anche a un consumo minore di energia rispetto ai cammini che presentano alta probabilità di ritrasmissione. Per la descrizione dell'implementazione di ETX all'interno di ContikiRPL si veda successivamente nella sezione che descrive l'implementazione delle metriche.
Di seguito invece si passa a descrivere la metrica di confronto ETX e inoltre vengono presentate le due metriche energy-aware (con principale obiettivo il risparmio energetico) che ho implementato in ContikiRPL: energia consumata nel tempo e indice di vulnerabilità del dispositivo wireless.

6.1.1 Expected transmission count

Expected transmission count (ETX) rappresenta il numero totale atteso di trasmissioni (comprese le ritrasmissioni) richieste per consegnare con successo un pacchetto alla sua destinazione. Il ETX di un cammino corrisponde alla somma dei ETX dei lati che lo compongono. Per esempio, il ETX di un cammino costituito da tre hops con links operanti in perfette condizioni ideali risulta pari a, il ETX di un link con percentuale di consegna con successo del pacchetto pari a 50 vale 2.
ETX di un link è calcolato ricorrendo a due parametri: D_f e D_r. D_f rappresenta la probabilità che un pacchetto dati venga consegnato consegnato con successo al nodo di destinazione, mentre D_r rappresenta la probabilità che un pacchetto ACK venga ricevuto con successo dal nodo sorgente. La probabilità attesa che un pacchetto venga ricevuto e risocontrato con successo risulta pari a $\cdot D_f D_r$. Poichè ogni singolo tentativo di trasmissione del pacchetto può essere considerato come una prova di Bernoulli, ETX vale

$$ETX = \frac{1}{D_f \cdot D_r} \tag{6.1}$$

ETX come metrica basata sulle probabilità di consegna influisce in maniera diretta sul throughput e riflette l'asimmetria del link incorporando in sè le

informazioni riguardanti le percentuali di perdita di pacchetti in entrambe le direzioni (uplink e downlink). In aggiunta, ETX porta alla riduzione dell'energia consumata per pacchetto, poichè ogni trasmissione o ritrasmissione causa un ulteriore consumo di energia da parte del nodo.
Il calcolo delle probabilità necessarie alla definizione di ETX avviene mediante meccanismo di probing, ossia mediante l'invio di pacchetti broadcast di misurazione.

6.1.2 Energia consumata

L'energia consumata indica l'ammontare di carica energetica esaurita del dispositivo all'istante di tempo t. Il suo valore risulta essere determinato da numerosi fattori, tuttavia gli stati di trasmissione e ricezione sono tra quelli che incidono maggiormente. L'energia consumata di un nodo è già stata utilizzata come metrica (nel gradient routing), tuttavia non è stata implementatata nel protocollo RPL.

6.1.3 Indice di vulnerabilità

L'indice di vulnerabilità (o Battery Index) [29] indica intuitivamente se un nodo è destinato a consumare più o meno energia nel tempo. Esso permette quindi di individuare con buona probabilità quei nodi che hanno un tempo di vita potenziale relativamente basso rispetto agli altri. Il Battery Index è pari a

$$BI = K \cdot \frac{w_{tx} \cdot T_{tx}(t) + w_{rx} \cdot T_{rx}(t) + w_{idle} \cdot T_{idle}(t)}{T_{sleep}(t) \cdot D_c} \tag{6.2}$$

dove $T_{tx}(t)$, $T_{rx}(t)$, $T_{idle}(t)$, $T_{sleep}(t)$ sono rispettivamente i tempi accumulati negli stati di trasmissione, ricezione, idle e sleep all'istante t; $D_c(t)$ è il duty cycle (il rapporto tra tempo di attività del nodo e la somma tra tempo di attività e tempo in stato sleep); w_{tx}, w_{rx} e w_{idle} rappresentano il consumo di energia per unità di tempo nei relativi stati mentre K è un fattore di normalizzazione.
Nell'implementazione di questa metrica si sono fatte alcune considerazioni che portassero alla semplificazione della formula col fine di ridurre l'impatto di calcolo sulla CPU. Innanzitutto la modalità idle può essere trascurata nel

calcolo dato che la comunicazione radio è uno dei fattori che incide fortemente sul consumo di energia. Inoltre, dato l'utilizzo (mediante emulazione) in questo lavoro di dispositivi Tmote Sky con chip radio CC2420 in cui il consumo di potenza negli stati di trasmissione e ricezione è pressochè identico w_{tx} e w_{rx} possono essere considerati uguali. L'espressione diventa quindi

$$BI = K^I \cdot \frac{T_{tx}(t) + T_{rx}(t)}{T_{sleep}(t) \cdot D_c} \tag{6.3}$$

dove K^I rappresenta la nuova costante di normalizzazione che ingloba i valori di w_{tx} e w_{rx}. Intuitivamente, la teoria alla base di questa metrica considera che i nodi maggiormente vulnerabili all'interno della rete siano quelli che consumano mediamente più energia nel tempo (o nel tempo di attività, ossia il tempo speso per la trasmissione e la ricezione e nello stato idle).
In una tipica rete di sensori wireless i nodi sono distribuiti geograficamente in modo tale da riportare periodicamente delle misure verso un nodo sink, situazione che viene risaltata maggiormente nel caso del protocollo di routing RPL in cui la tipologia di traffico predominante è di tipo MP2P o P2MP (multi-point to point e viceversa). Per questo motivo un sottoinsieme di nodi, in particolare quelli vicini al nodo sink, tendono a consumare più energia degli altri e quindi a essere vulnerabili. L'intento è quindi quello di evitare che questi nodi esauriscano la loro energia troppo velocemente consentendo di conseguenza un allungamento del tempo di vita dell'intera rete.

6.2 Implementazione delle metriche in ContikiRPL

Nella sezione seguente viene descritto il processo di implementazione delle metriche energy-aware. Tuttavia, prima di analizzare nel dettaglio l'implementazione delle metriche energy-aware, risulta necessario fornire una descrizione dei moduli che formano la struttura del protocollo RPL nel sistema operativo Contiki. Averne una chiara comprensione permette di capire le modalità di inserimento delle metriche all'interno del protocollo e le relative modifiche da apportare al codice.

6.2.1 Panoramica di ContikiRPL

ContikiRPL implementa il protocollo RPL seguendo le direttive della relativa specifica con numero di versione 18 [14] e due objective functions Of0 e Minimum Rank Objective Function with Hysteresis (MRHOF) come accennato nella prima sezione. L'implementazione di RPL separa la logica di protocollo, la costruzione e gestione dei pacchetti, gestione e controllo della objective function in moduli differenti.
Il modulo inerente la logica di protocollo (rpl-dag.c) si occupa della gestione del grafo di routing (DAG), mantiene l'insieme dei nodi potenziali parent con le loro relative informazioni, comunica con i moduli preposti alla gestione della objective function, convalida i messaggi RPL a livello logico seguendo le direttive della specifica RPL.
Il modulo destinato alla costruzione e all'analisi dei messaggi (rpl-icmp6.c) prevede l'adattamento dei messaggi ICMPv6 (Internet Control Message Protocol IPv6) nelle strutture dati proprie di Contiki.
Infine, i moduli destinati alla gestione della objective function (rpl-of0.c e rpl-of-etx.c) forniscono le API per l'implementazione e l'aggiornamento della stessa. ContikiRPL non prende direttamente decisioni sull'instradamento del singolo pacchetto, ma si limita a fornire e formare delle tabelle di instradamento e demanda il vero lavoro di instradamento al livello IPv6. I pacchetti IPv6 uscenti passano dal livello IPv6 al livello 6LowPAN per il processo di compressione e frammentazione. Il livello 6LowPAN invia poi i pacchetti al livello MAC sottostante. Il livello MAC di default di Contiki prevede un meccanismo CSMA/CA per l'accesso multiplo sistemando i pacchetti in attesa code. Dalle code i pacchetti sono trasmessi in ordine attraverso il livello RDC (radio duty cycling). Il livello MAC prevede la ritrasmissione del pacchetto fino a quando non riceve un ACK dal ricevitore. Se avviene una collisione, il livello MAC ricorre a una finestra di back-off e ritrasmette il pacchetto colliso.
L'implementazione attuale di ContikiRPL prevede inoltre la presenza di un modulo esterno che prevede la raccolta di alcuni parametri che caratterizzano i nodi vicini, nel particolare effettua e aggiorna una stima del costo per ogni collegamento ricorrendo a una funzione callback. Nel caso si adotti come metrica di routing ETX, ContikiRPL quindi aggiorna il costo del cammino da o verso il sink basandosi sulle stime effettuate dal modulo. Il costo per ogni collegamento è rappresentato dal suo ETX, stimato ricorrendo a

una funzione EWMA (exponentially-weighted moving average function) con parametro di configurazione a = 2. Il valore di ETX è utilizzato poi per la definizione del rank di MRHOF. Questa objective function prevede nel dettaglio la definizione e l'utilizzo di una soglia di confronto tra i potenziali parent: un nodo viene scelto come parent preferito solo se il suo rank risulta minore dell'altro di almeno il valore stabilito con la soglia, tutto questo per evitare frequenti aggiornamenti e cambiamenti nella topologia che provocherebbe instabilità alla rete. La scelta del parent nell'insieme dei potenziali parent si riflette sul nodo il cui collegamento è caratterizzato dal valore minimo di ETX.
ContikiRPL prevede la costituzione di una singola istanza con un singolo DODAG (Destination Oriented Directed Acyclic Graph). La modalità di formazione dei cammini dal nodo radice adottata è storing mentre il traffico supportato al momento è quello di tipo multipoint-to-point e viceversa.

6.2.2 Calcolo delle metriche energy-aware in ContikiRPL

Per l'effettivo calcolo dell'energia residua e dell'indice di vulnerabilità ho sfruttato le funzionalità del modulo *energest.c* di Contiki (in figura 6.1 la definizione della struttura e dei relativi stati) includendolo all'interno di ContikiRPL. Tale modulo indica le procedure necessarie al calcolo dei tempi cumulativi degli stati o attività in cui si trova a operare un dispositivo (accensione e spegnimento dei leds, trasmissione radio attiva, scrittura e lettura su memoria flash,ecc.).
Ai fini del risparmio energetico e per semplicità di trattazione, sono stati considerati solo i tempi associati ai seguenti stati: trasmissione, ricezione, cpu, low power mode. Gli altri stati o attività sono stati trascurati in quanto non necessari all'analisi dell'implementazione e in quanto il loro contributo in termini di energia dissipata risulta trascurabile.
Nel calcolare i contributi di energia associati ai singoli stati, ho tenuto conto dei differenti valori di corrente associati. Tali valori sono stati ricavati dal datasheet del mote Tmote Sky (principale modello di dispositivo adottato in Contiki) e risultano:

- $I_{tx} = 19,5$ mA
- $I_{rx} = 21,8$ mA
- $I_{cpu} = 1,8$ mA

- $I_{lpm} = 0,0545$ mA

Per quanto riguarda il voltaggio di esercizio di un Tmote Sky ho assunto come valore 3 V. L'energia consumata (espressa in milliJoule) in un dato istante t è determinata quindi dalla somma dei consumi energetici relativi ai quattro stati sopra elencati. Tenendo conto della definizione generale di energia, la relativa formula risulta:

$$E_c = (I_{cpu} \cdot t_{cpu} + I_{lpm} \cdot t_{lpm} + I_{tx} \cdot t_{tx} + I_{rx} \cdot t_{rx}) \cdot V \tag{6.4}$$

Nella figura 6.1 viene riportata la dichiarazione della variabile di energia consumata e il relativo calcolo all'interno del modulo di ContikiRPL. Il valore di RTIMERSECOND corrisponde al numero di ticks per secondo e dipende dalla CPU del dispositivo adottato, la relativa divisione serve semplicemente a esprimere i tempi in secondi.

```
Definizione della metrica
typedef unsigned long rpl_metric_energy_t;
rpl_metric_energy_t energy;

Calcolo dei tempi associati agli stati considerati
cpu = energest_type_time(ENERGEST_TYPE_CPU);
lpm = energest_type_time(ENERGEST_TYPE_LPM);
Transmit = energest_type_time(ENERGEST_TYPE_TRANSMIT);
listen = energest_type_time(ENERGEST_TYPE_LISTEN);

Calcolo della metrica
energy  = ((1.8 * cpu + 0.0545 * lpm + 20.0 * listen + 17.7 *
transmit) * 3) / RTIMER_SECOND);
```

Figura 6.1: Formula dell'energia consumata

Per il calcolo dell'indice di vulnerabilità ho seguito una procedura analoga a quella utilizzata per l'energia consumata. Anche in questo caso i tempi definiti in precedenza contribuiscono alla definizione della metrica e il loro calcolo in Contiki rimane identico a quanto visto nel caso della prima metrica. Il duty cycle è dato dal rapporto tra tempo di attività della CPU

e tempo totale di vita del dispositivo. Per quanto riguarda la costante K definita durante la descrizione della metrica, ho scelto di stabilire come valore 10, identico a quello utilizzato dai progettatori nella fase di studio e simulazione come stabilito nel relativo documento [29].

```
Calcolo dei tempi associati agli stati considerati
cpu = energest_type_time(ENERGEST_TYPE_CPU);
lpm = energest_type_time(ENERGEST_TYPE_LPM);
Transmit = energest_type_time(ENERGEST_TYPE_TRANSMIT);
listen = energest_type_time(ENERGEST_TYPE_LISTEN);

Calcolo della metrica
radio_time = transmit + listen;
duty_cycle = cpu / (cpu + lpm);

Battery index = (10 * radio time) / (lpm * duty cycle);
```

Figura 6.2: Formula dell'indice di vulnerabilità

6.2.3 Dichiarazione delle nuove variabili in ContikiRPL

Per l'inserimento delle metriche energy-aware all'interno di ContikiRPL ho previsto una nuova dichiarazione delle variabili di codice necessarie al corretto funzionamento del protocollo. Nella figura 6.3 sono riportate le relative dichiarazioni contenute nei file header di cui dispone il programma. Innanzitutto, ho definito due nuovi DAG Metric Container Object Types relativi alle due metriche. Attraverso delle direttive di precompilazione opportunamente definite, il programma ogni volta seleziona il tipo di DAG Metric Container opportuno a seconda della objective function che viene adottata. Nella nuova rappresentazione logica del DAG Metric Container ho ritenuto comodo includere entrambe le metriche energy-aware.

6.2.4 Creazione del nuovo modulo di gestione delle metriche

Come accennato in precedenza, ContikiRPL nella forma originale con ETX demanda il calcolo della metrica a un modulo esterno (neighbour-info.c) che contiene le informazioni necessarie sui nodi vicini. Neighbour-info.c calcola il valore di ETX del collegamento tra due nomi semplicemente ricorrendo a

Definizione dei DAG Metric Container Object

```
#define RPL_DAG_MC_ENERGY          0
#define RPL_DAG_MC_BATTERY_INDEX   1
```

Definizione delle metriche e del rank

```
typedef unsigned long rpl_metric_energy_t;
typedef unsigned long rpl_metric_BI_t;
typedef unsigned long rpl_rank_t;
```

Nuova rappresentazione logica di un DAG Metric Container

```
struct rpl_metric_container {
  uint8_t type;
  uint8_t flags;
  uint8_t aggr;
  uint8_t prec;
  uint8_t length;
  union metric_type {
    rpl_metric_energy_t Ec;
    rpl_metric_BI_t BI;
  } metric;
};
typedef struct rpl_metric_container rpl_metric_container_t;
```

Direttive per la selezione della metrica

```
#ifdef RPL_CONF_DAG_MC
#define RPL_DAG_MC RPL_CONF_DAG_MC
#else
#define RPL_DAG_MC RPL_DAG_MC_ENERGY /7oppure BATTERY_INDEX
#endif
```

Figura 6.3: Dichiarazione delle variabili nel header di ContikiRPL

un meccanismo di probing. Il modulo di gestione della metrica ne aggiorna il valore semplicemente estrapolandolo dal modulo esterno e procede al calcolo del rank e del costo dell'intero cammino di instradamento. Con l'introduzione delle due metriche energy-aware, ho ritenuto necessario definire un nuovo modulo, chiamato genericamente rpl-of-metric.c, includendo anche il calcolo della semplice metrica rendendolo di fatto interno a ContikiRPL. In esso sono presenti entrambe le metriche energy-aware, il modulo sceglie semplicemente la metrica corretta in base all'informazione contenuta nel campo type del DAG Metric Container.
Le principali funzioni implementate nel nuovo modulo sono quelle relative al calcolo della metrica del nodo, alla detrminazione del rank del nodo all'interno del DAG, al confronto tra due nodi potenziali parent.
La funzione di calcolo della metrica del nodo (figura 6.4) risulta abbastanza semplice: verifica il tipo di metrica adottata dal protocollo attraverso il campo type del DAG Metric Container, impone il valore di metrica pari a zero se il nodo interessato è il sink, altrimenti procede al calcolo vero e proprio della metrica. Il valore nullo della metrica riflette direttamente il ruolo di sink del nodo all'interno della rete. Infatti, supponendo che al ruolo di sink sia destinato un nodo con maggiori capacità energetiche non avendo alimentazione a batteria, sarebbe come affermare che il sink non ha problemi nel consumare energia o nell'essere un nodo vulnerabile. La funzione di calcolo del rank del nodo è presentata in figura 6.5. Per semplicità di esposizione ho omesso la parte relativa all'indice di vulnerabilità in quanto in entrambi i casi il procedimento risulta analogo e ho evitato inoltre di mettere tutte le dovute condizioni di verifica. La funzione riceve in ingresso la struttura del nodo parent e ritorna il valore del rank associato al nodo. Il rank di un nodo è ottenuto come somma dei valori di metrica di tutti i nodi intermedi che compongono il cammino verso il sink. La funzione quindi in maniera semplice ottiene il rank calcolando la metrica sul nodo e sommandola al rank del nodo parent. Il rank rispecchia le direttive della objective function che in entrambi i casi stabilisce di scegliere i cammini che presentano un costo energetico minore. Allontanandosi dal sink (con rank nullo) il rank tende ad aumentare. La funzione di confronto tra due potenziali nodi parent (figura 6.6) stabilisce semplicemente se è opportuno o meno da parte del nodo cambiare parent preferito in base all'informazione del rank. In generale ogni nodo della rete prevede una lista di potenziali nodi parent il cui numero è

```
static void update_metric(rpl_instance_t *instance)
{

rpl_dag_t *dag;
dag = instance->current_dag;


#if RPL_DAG_MC == RPL_DAG_MC_ENERGY

instance->metric_container.type = RPL_DAG_MC_ENERGY;

      If(dag->rank == ROOT_RANK) {

            instance->metric_container.metric.energy = 0; }

      else {

            instance->metric_container.metric.energy = valore dell'energia consumata
       }


#elif RPL_DAG_MC == RPL_DAG_MC_BATTERY_INDEX

...stesso procedimento di sopra


#endif
```

Figura 6.4: Funzione di calcolo della metrica di nodo

```
static rpl_rank_t calculate_rank(rpl_parent_t *p, rpl_rank_t base_rank)
{

rpl_rank_t new_rank;
rpl_rank_t rank_increase;


rank_increase = calcolo della metrica del nodo;

base_rank = p->rank;

new_rank = base_rank + rank_increase;


return new_rank;
}
```

Figura 6.5: Funzione di calcolo del rank

configurabile all'interno del file header di ContikiRPL. La funzione prende in ingresso le strutture di due nodi potenziali parent, verifica che entrambi appartengano allo stesso DAG e restituisce il parent preferito se il suo valore di rank risulta minore del rank dell'altro parent di almeno un certo valore di soglia. Tale valore cambia a seconda della metrica adottata nella objective function. Nel mio caso, in base all'osservazione delle simulazioni effettuate e cercando di evitare che la funzioni provochi continui cambiamenti al grafo, ho ritenuto ragionevole impostare la soglia a 20 nel caso di energia consumata, a 2 nel caso dell'indice di vulnerabilità. In questo modo viene preservata la stabilità della rete.

```
static rpl_parent_t * best_parent(rpl_parent_t *p1, rpl_parent_t *p2)
{
  rpl_dag_t *dag;
  rpl_rank_t min_diff;
  rpl_rank_t p1_metric;
  rpl_rank_t p2_metric;

  dag = p1->dag;

  min_diff = valore da definire a seconda della metrica adottata

  p1_metric = p1->rank;
  p2_metric = p2->rank;


  if(p1 == dag->preferred_parent || p2 == dag->preferred_parent) {
      if(p1_metric < p2_metric + min_diff && p1_metric > p2_metric - min_diff) {

          return dag->preferred_parent;

        }
    }

  return p1_metric < p2_metric ? p1 : p2;
}
```

Figura 6.6: Funzione di confronto dei potenziali parent

Capitolo 7

Risultati

Nel seguente capitolo vengono presentati i risultati relativi alle prestazioni del protocollo RPL con le metriche energy-aware. Inoltre viene mostrato il confronto con l'implementazione di RPL con metrica ETX.

7.1 Scenario della simulazione

Nel seguente paragrafo viene presentato lo scenario di simulazione per la valutazione del protocollo di routing RPL nelle sue tre differenti implementazioni: ContikiRPL-ETX, ContikiRPL-EnergiaResidua, ContikiRPL-BatteryIndex. Per le simulazioni ho optato per la scelta di Cooja, simulatore offerto all'interno del sistema operativo Contiki. Cooja si differenzia dai numeri ambienti di simulazione per reti di sensori wireless in quanto emula in maniera accurata il funzionamento dei dispositivi reali. I dispositivi adottati in Cooja per la verifica di questo lavoro fanno parte della famiglia Tmote Sky e sono dotati di microcontroller MSP430 e chip radio CC2420.
I nodi impiegati nella formazione della topologia sono 31 (di cui un sink) e collocati in maniera casuale su un'area quadrata di lato pari a 200 metri circa (figura 7.1). Il sink è collocato ai margini della topologia. In realtà la collocazione del sink risulta importante nella determinazione del rank dei nodi della rete, tuttavia non rientra negli obiettivi prefissati di questo lavoro. La scelta di una topologia casuale è motivata dall'avvicinarsi il più possibile ad un contesto reale.
Il modello di canale è Unit Disk Graph Medium (UDGM) - Distance Loss ove il range di trasmissione è modellato come un cerchio centrato nel nodo e dipende dal rapporto tra potenza corrente in uscita del dispositivo em-

ulato e potenza massima in uscita. Tutti i nodi all'interno del raggio di trasmissione possono ricevere i messaggi del nodo mentre quelli al di fuori non lo ricevono. Viene presa in considerazione anche l'interferenza anche se in maniera semplificata rispetto alla realtà. Inoltre si ha la possibilità di impostare i valori di probabilità di trasmissione e ricezione del pacchetto.
Al fine di valutare le prestazioni del protocollo RPL, tutti i nodi sono dotati di un applicativo UDP in cui i 30 nodi client inviano periodicamente pacchetti dati UDP al sink. Quest'ultimo ha quindi la duplice funzione di RPL Router e UDP Server.

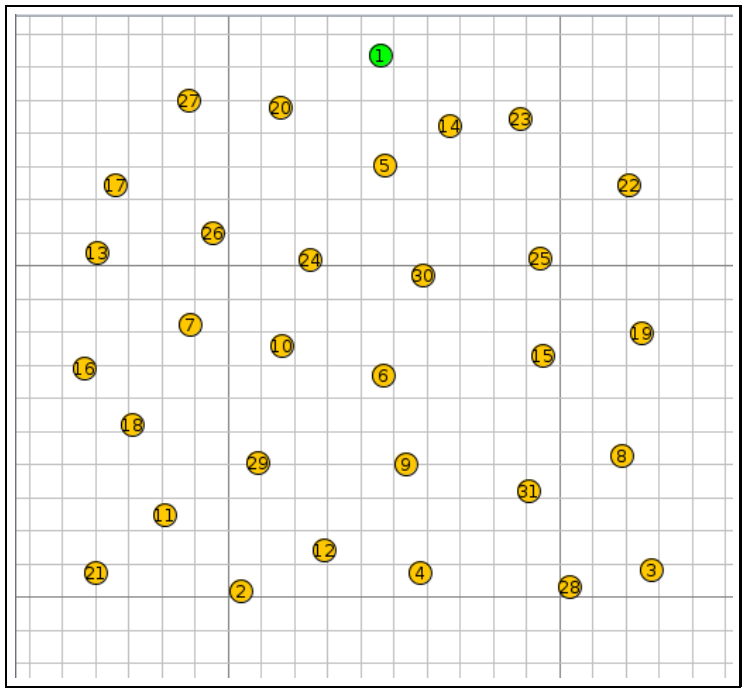


Figura 7.1: Topologia di rete

7.2 Parametri di prestazione di RPL e risultati

Di seguito vengono riportati i parametri necessari a valutare le prestazioni di RPL nelle sue differenti implementazioni. Essi sono necessari alla valutazione del protocollo in termini di affidabilità, scalabilità e con una parti-

colare attenzione alle prestazioni di risparmio energetico, aspetto di primo piano nel routing di una WSN. I parametri di rete analizzati sono:

- Ritardo medio end-to-end. Esso è ottenuto dai singoli ritardi end-to-end calcolati per ogni datagramma UDP inviato dal client verso il sink.

- Packet Loss Rate. Esso indica la percentuale di pacchetti dati UDP inviati dai nodi client e che non raggiungono la destinazione (sink). La perdita dei pacchetti può essere causata da diversi fattori tra cui la congestione della coda dei nodi intermedi del cammino, le collisioni radio provocate principalmente da interferenza o errori casuali del canale radio. Il packet loss rate risulta essere pari a

  $$PLR = 1 - \frac{N_{rx}}{N_{tx}}$$

  dove N_{rx} è il numero di pacchetti UDP ricevuti con successo dal sink mentre N_{tx} è il numero di pacchetti inviati dai nodi client. L'analisi delle prestazioni in termini di percentuale di perdita dei pacchetti dati fornisce inoltre una chiara indicazione sul throughput dati in ricezione: al diminuire dei valori di packet loss rate si ha un aumento dei pacchetti ricevuti in un dato intervallo di tempo e viceversa.

- Energia consumata media. Indica l'ammontare di carica di batteria consumata dalla rete in un dato intervallo di tempo.

- Distribuzione energetica della rete. Indica la distribuzione del livello di carica energetica all'interno della rete. L'intento è verificare che non ci sia un consumo troppo sproporzionato tra i vari dispositivi della rete, garantendo in tal modo un certo equilibrio in termini di energia e di copertura.

Il rate di invio pacchetti all'interno della rete viene variato in questo modo: l'applicazione UDPclient stabilisce un intervallo di tempo periodico T, ogni nodo invia un pacchetto in un istante casuale t all'interno dell'intervallo T. Nell'effettuare le simulazioni si è fatto variare l'intervallo T da 10 secondi a 1 secondo con variazione di 1 secondo. Di seguito sono riportati i risultati prestazionali delle tre implementazioni del protocollo RPL con

le tre metriche. Per semplicità le tre implementazioni di ContikiRPL sono state denominate ETX, RE (Residual Energy), BI (Battery Index).

7.2.1 Ritardo medio end-to-end

I risultati relativi al ritardo medio end-to-end della rete a livello UDP sono visibili nella figura 6.3. I valori catturati sono in funzione del rate di trasmissione e sono relativamente alti in parte considerando che tutti i dispositivi della rete fungono da sorgente attiva di traffico e i valori crescono sensibilmente con l'aumentare del rate di trasmissione dati. Inoltre bisogna tenere conto di aver adottato un protocollo a livello MAC che limita l'utilizzo del canale radio per fini di risparmio energetico per cui le code dei singoli nodi sono caratterizzate da un'alta utilizzazione che porta a probabili congestioni e quindi di conseguenza a ritardi più lunghi. In generale il valore del ritardo medio end-to-end ha una dipendenza non trascurabile dal numero di trasmissioni e ritrasmissioni necessarie a consegnare un pacchetto con successo.
Nel caso specifico ETX mostra una prestazione migliore in termini di ritardo in quanto come metrica tende a evitare links ove sono previste numerose ritrasmissioni, contribuendo alla creazione di cammini col minimo numero di trasmissioni e ritrasmissioni. Tale aspetto non viene ovviamente considerato in RE e BI, ove i valori di ritardo risultano relativamente più alti.

7.2.2 Packet loss rate

Nella figura 6.4 sono riportati i risultati relativi alle prestazioni delle tre metriche in relazione al packet loss rate. In questo caso ETX fornisce una migliore performance rispetto alle altre due metriche in quanto tende a privilegiare cammini non altamente congestionati. Le percentuali di perdita di pacchetti relative alle altre due metriche risultano relativamente più alte rispetto a ETX, tuttavia all'aumentare del rate di trasmissione la differenza si assottiglia risultando quindi equiparabili. Risulta facilmente intuibile come in termini di prestazioni di throughput ETX dimostri prestazioni migliori rispetto alle altre due implementazioni dato che dimostra una percentuale di perdita dati minore.

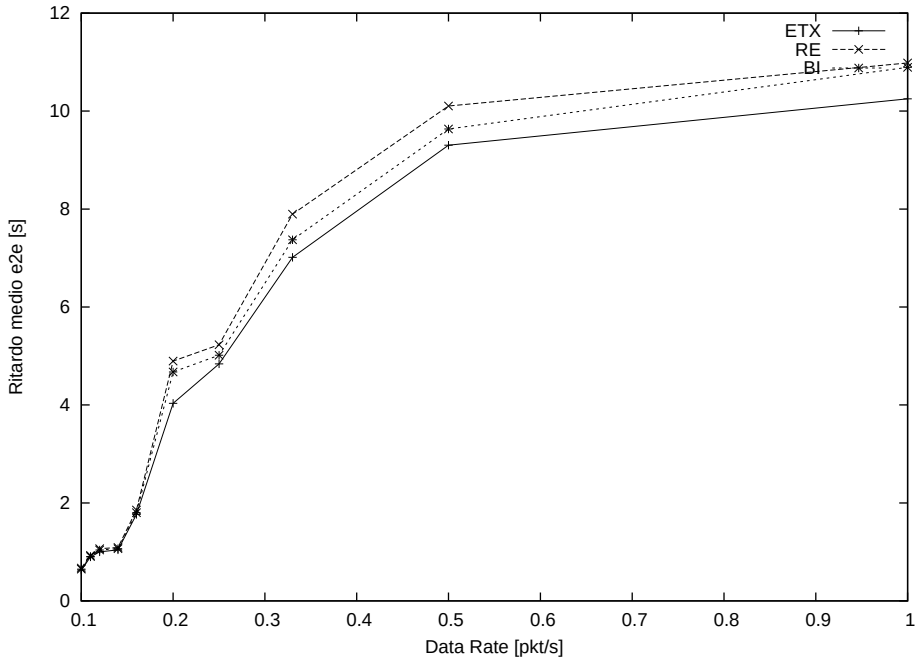


Figura 7.2: Ritardo medio end-to-end della rete.

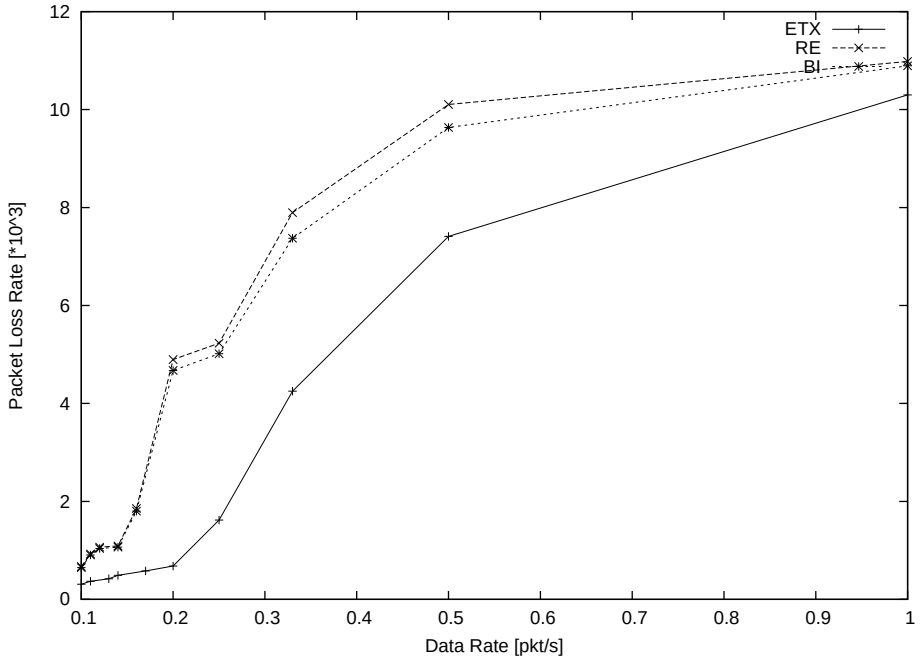


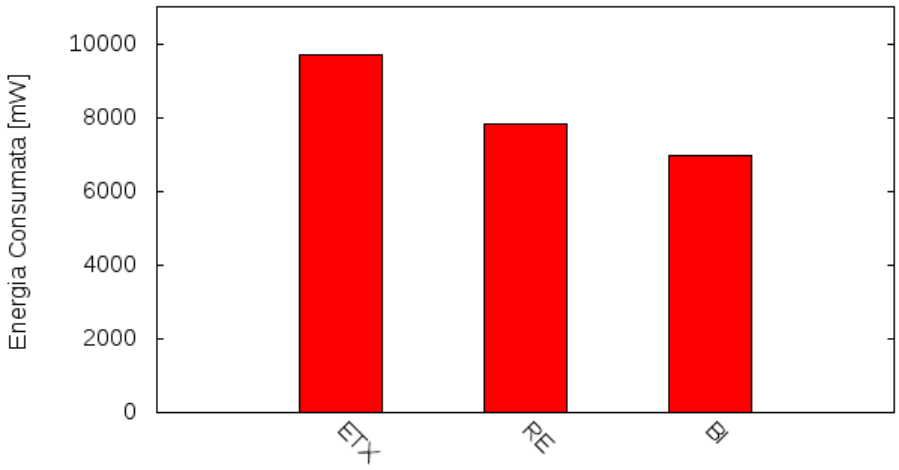
Figura 7.3: Packet Loss Rate.

7.2.3 Energia consumata media

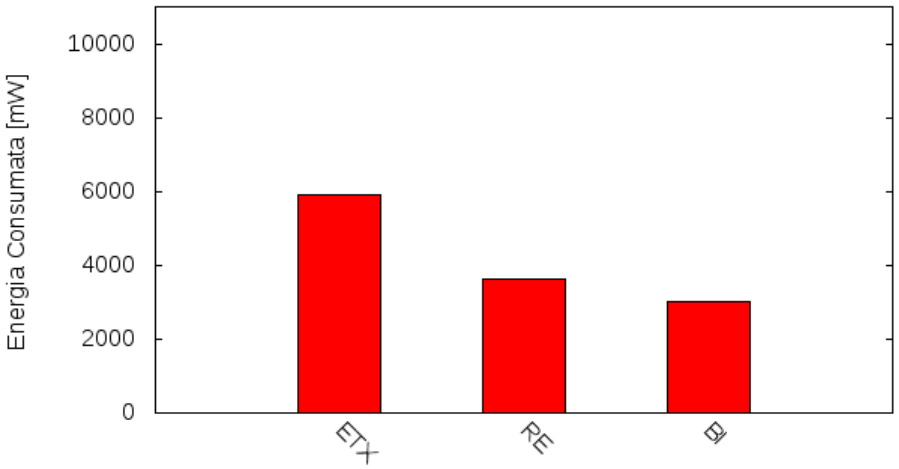
Nella sezione seguente vengono riportati i dati relativi al consumo medio della rete. La media viene effettuata sui valori di energia consumata dei nodi che compongono la WSN considerando come intervallo periodico di invio pacchetto T = 1, 5, 10s e con un tempo di simulazione pari a 500s, andando a verificare le prestazioni nelle differenti situazioni relative al traffico offerto della rete.
Dalle figura 6.5 si nota come in generale RE e BI forniscano migliori prestazioni energetiche rispetto a ETX. Il risparmio energetico di RE e BI tende a essere maggiore rispetto a ETX all'aumentare del traffico offerto dalla rete, tuttavia le differenze di ETX in termini di consumo medio rispetto a RE e BI non risultano drastiche. Questo è duvuto al fatto che ETX ponga comunque in parte l'attenzione al risparmio energetico in quanto porta alla selezione di cammini di instradamento col numero minimo di trasmissioni e ritrasmissioni, limitando quindi il tempo in cui ogni dispositivo si trova in stato di trasmissione e ritrasmissione.

7.2.4 Distribuzione energetica della rete

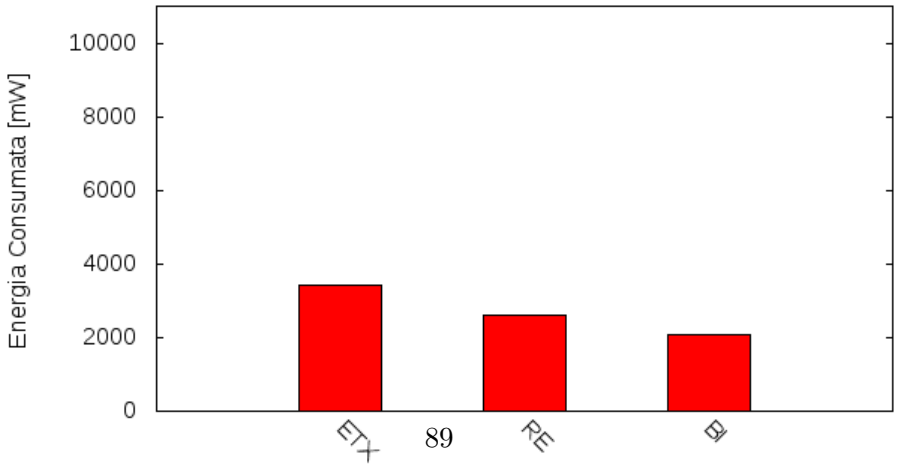
In questa sezione verranno mostrati i risultati relativi alla distribuzione energetica della rete. Nella figura 6.6 vengono riportati i valori relativi all'energia conzumata da ogni nodo della rete con T = 1s e tempo di simulazione pari a 500s. Nel grafico di ETX si nota un consumo energetico sbilanciato con alcuni nodi che presentano alcuni picchi energetici, mentre nel caso delle altre due metriche si nota un consumo più bilanciato e distribuito con più uniformità tra i nodi della rete. Questo è dovuto al fatto che ETX tende a utilizzare spesso gli stessi cammini di instradamento, in particolare quei link che presentano una bassa percentuale di perdita dei pacchetti (meno trasmissioni e ritrasmissioni richieste), di conseguenza alcuni nodi tendono a essere impiegati assiduamente. Nel caso di RE e BI si tende implicitamente ad evitare questa situazione, in particolare BI offre migliori prestazioni poichè individua proprio quei nodi che sono vulnerabili ossia quelli che presentano una più assidua attività nel ricevere e trasmettere pacchetti portando quindi per quanto sia possibile a un coinvolgimento più uniforme da parte dei nodi della rete.



(a) T=1s

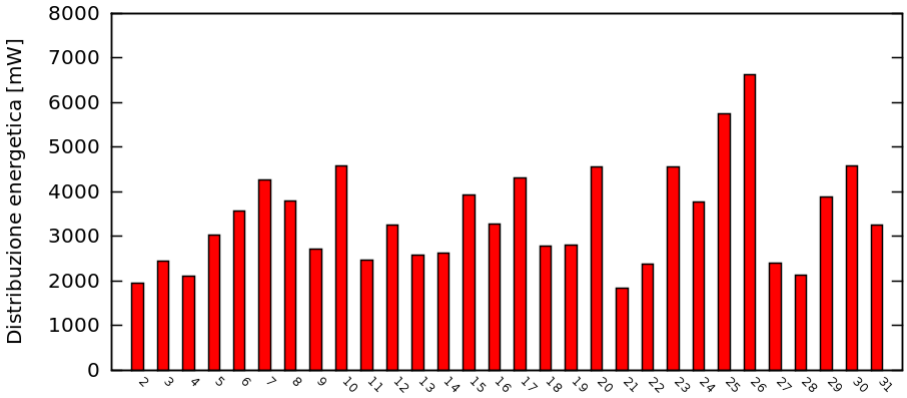


(b) T=5s

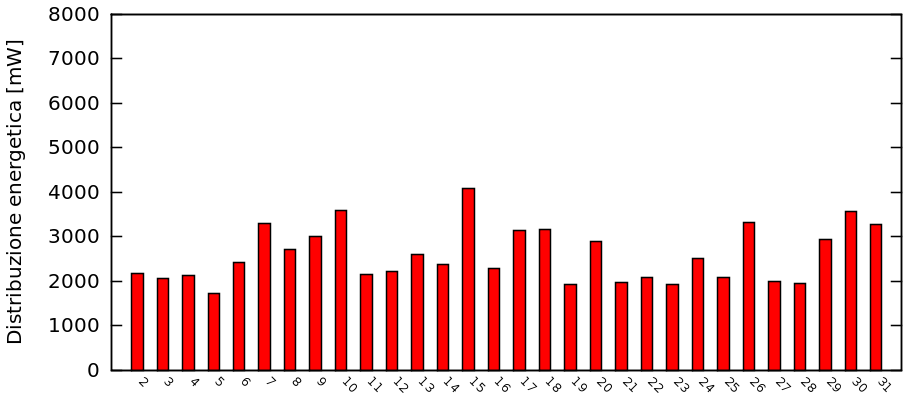


(c) T=10s

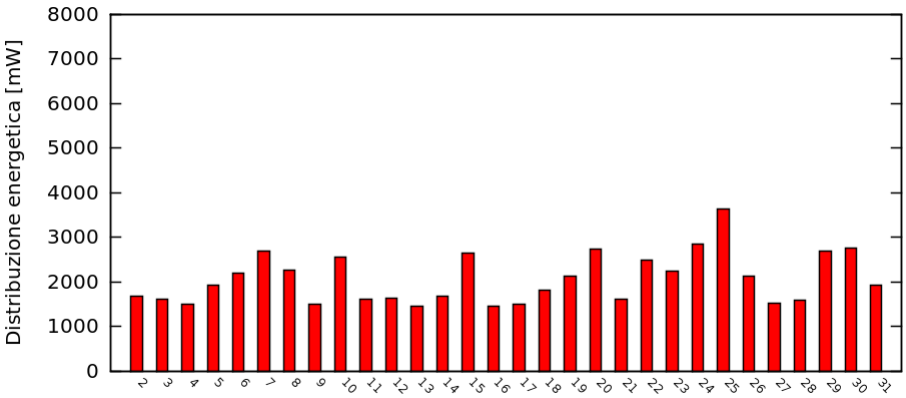
Figura 7.4: Energia consumata media



(a) ETX



(b) RE



(c) BI

Figura 7.5: Distribuzione energetica dei nodi della rete.

Capitolo 8

Conclusioni

Il seguente capitolo conclude il lavoro di tesi riportando alcune considerazioni sulle metriche energy aware analizzate e implementate ed eventuali sviluppi futuri.
Una rete di sensori wireless è composta da un largo numero di dispositivi chiamati nodi sensore che comunicano tra loro nel monitoraggio di un determinato ambiente. I dispositivi sono caratterizzati dall'avere limitato raggio di copertura del segnale wireless, scarsa capacità di calcolo e di memorizzazione, limitate risorse energetiche.
Il routing nelle reti di sensori è un problema di grande interesse sia per l'importanza del suo ruolo nel corretto funzionamento delle applicazioni, sia perchè lo sviluppo di questi protocolli è impegnativo a causa delle caratteristiche proprie delle reti di sensori che le distinguono dalle comuni reti wireless essendo costituite da dispositivi con limitato raggio di copertura del segnale wireless, scarsa capacità di calcolo e di memorizzazione, limitate risorse energetiche.
Il protocollo IPv6 Routing Protocol for Low power and Lossy Networks (RPL) rappresenta un buon punto di partenza per lo sviluppo di un protocollo di routing finalizzato al risparmio energetico, dato che offre la possibilità di utilizzare liberamente differenti metriche di routing che regolano la formazione del cammino di instradamento.
La necessità di integrare il protocollo di routing con metriche energy aware è nasce dalla volontà di rendere RPL efficiente dal punto di vista del risparmio energetico soprattutto in situazioni critiche in cui è previsto un numero elevato di nodi sensore alimentati a batteria e difficilmente rimpiazzabili. Tale necessità ha portato allo svolgimento di questo lavoro di tesi basato sul-

l'analisi e implementazione di due metriche energy aware da integrare con RPL, energia consumata e indice di vulnerabilità.
Le metriche energy aware hanno il duplice obiettivo di minimizzare il costo energetico dell'intera rete e di bilanciare adeguatamente il consumo di energie tra i vari dispositivi. In questo modo si evitano situazioni indesiderabili in cui alcuni nodi della rete, esaurendo la propria carica energetica preventivamente, portano a situazioni di instabilità e a un non corretto funzionamento della rete stessa.
Per poter implementare le due metriche a risparmio energetico in RPL è stato necessario uno studio accurato delle piattaforma software esistente (sistema operativo Contiki) per la gestione delle WSN descritto nel capitolo 5. Tale studio mi ha portato a tenere in debita considerazione la limitata capacità di memorizzazione e di elaborazione dei dispositivi che di fatto ha rappresentato il vincolo principale per la progettazione delle metriche. La formulazione delle metriche deve risultare quindi necessariamente leggera in termini di memoria ed evitare un calcolo troppo elaborato da parte della CPU.
Il calcolo delle metriche si basa sulla misura dei tempi degli stati in cui si trova un nodo sensore (stati di trasmissione, ricezione, CPU attiva e low power mode). Oltre all'implementazione delle formule di calcolo delle metriche è stato implementato un nuovo modulo in RPL con le funzioni necessarie all'intera gestione della objective function. Al fine di misurare le prestazioni delle due implementazioni è stato necessario estrapolare i relativi parametri di prestazione tipici del livello di routing (ritardo medio, throughput, packet loss rate, energia) in quanto il tool di simulazione utilizzato per i test (Cooja) permette solo la visualizzazione di parametri di prestazione dei livelli inferiori.
I test effettuati sulle due implementazioni energy aware e il confronto con RPL con metrica ETX hanno dimostrato come le due metriche energy aware non solo consentono un risparmio medio dell'energia maggiore rispetto a ETX, ma portano anche a un consumo di energia bilanciato sulla maggior parte dei dispositivi della rete, allungando il tempo di vita. La metrica ETX invece fornisce prestazioni migliori in termini di ritardo e throughput anche se all'aumentare del rate di trasmissione dei pacchetti le prestazioni tendono a essere analoghe.
La nuova implementazione del protocollo facilita in generale l'inserimento

e la gestione della metrica, offrendo la possibilità di verificare il comportamento di ulteriori metriche. Tuttavia, il lavoro svolto non può che essere considerato un punto di partenza per la valutazione di metriche finalizzate al risparmio energetico per diversi motivi che possono essere considerati validi punti da sviluppare in futuro.

Innanzitutto sarebbe necessario fornire una implementazione del protocollo che preveda il trasporto di più DAG, offrendo quindi la possibilità di valutare combinazioni di vincoli e metriche.

Pur avendo utilizzato un tool di simulazione che permette un'adeguata emulazione dei dispositivi della rete, risulta necessario verificare le prestazioni e il confronto delle metriche in uno scenario reale che tipicamente comprende aspetti che non possono o non sono ancora ben modellati in un ambiente simulativo. Per esempio, la mobilità dei dispotivi sarebbe un aspetto da non trascurare nella progettazione.

Un possibile intervento può riguardare la possibilità di fornire l'implementazione di RPL del supporto al traffico P2P e valutare le metriche in questo scenario.

Bibliografia

[1] I. Howitt, J.A. Gutierrez, IEEE802.15.4 low rate-wireless personal area network coexistence issues, Wireless Communications and Networking 3 (2003) 1481-1486.

[2] ZibBee Alliance http://www.zigbee.org/Home.aspx

[3] G. Mulligan, L.W. Group,The 6LoWPAN architecture, Proceedings of the EmNets, Cork, Ireland, 2007.

[4] 6LowPAN Working Grouphttp://datatracker.ietf.org .

[5] J.N: Al-Karaki, A.E. Kamal, Routing Techniques in wireless sensor networks: a survey, IEEE Wireless Comunication vol.11, no.6, pp.6-28, 2004

[6] W. Heinzelman, J. Kulik, H. Balakrishnan, Adaptive protocols for information dissemination in wireless sensor networks, Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Net-working (MobiCom99), Seattle, WA, August 1999

[7] C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom00), Boston, MA, August 2000.

[8] C. Schurgers, M.B. Srivastava, Energy efficient routing in wireless sensor networks, on Communications for Network-Centric Operations: Creating the Information Force, McLean, VA, 2001.

[9] W. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless sensor networks, Proceeding of the Hawaii International Conference System Sciences, Hawaii, January 2000.

[10] S. Lindsey, C.S. Raghavendra, PEGASIS: power efficient gathering in sensor information systems, Proceedings of the IEEE Aerospace Conference, Big Sky, Montana, March 2002.

[11] Y. Yu, D. Estrin, R. Govindan, Geographical and energy-aware routing: a recursive data dissemination protocol for wireless sensor networks, UCLA Computer Science Department Technical Report, UCLA-CSD TR-01-0023, May 2001.

[12] P.Levis, A.Tavakoli, S. Dawson-Haggerty Overview of existing routing protocols for low power and lossy networks, IETF, Internet-Draft draft-ietf-roll-protocols-survey-07, 2009.

[13] IETF Routing Over Low-power and Lossy networks https://datatracker.ietf.org/wg/roll/charter/.

[14] T. Winter, P. Thuber, RPL: Ipv6 Routing Protocol for Low power and lossy network, IETF, Internet-Draft, draft-ietf-roll-rpl-18, 2011.

[15] T. Clausen, P. Jacquet, Optimized Link State Routing Protocol (OLSR), RFC 3626, October 2003.

[16] T. Clausen, C. Dearlove, P. Jacquet, The Optimized Link State Routing Protocol version 2, draft-ietf-manet-olsrv2-07, July 2008.

[17] J. Moy, OSPF Version 2, STD 54, RFC 2328, April 1998.

[18] R. Coltun, D. Ferguson, J. Moy, OSPF for IPv6, RFC 2740, December 1999.

[19] Perkins, C., Belding-Royer, E., and S. Das, Ad hoc On-Demand Distance Vector (AODV) Routing, RFC 3561, July 2003.

[20] G. Malkin, RIP Version 2, STD 56, RFC 2453, November 1998.

[21] J. Vasseur, M. Kim, K. Pister, N. Dejean, D. Barthel, Routing Metrics used for Path Calculation in Low Power and Lossy Networks, draft-ietf-roll-routing-metrics-17 (work in progress), January 2011.

[22] P. Levis, T. Clausen, J. Hui, O. Gnawali, J. Ko, The Trickle Algorithm, draft-ietf-roll-trickle-08 (work in progress), January 2011.

[23] Contiki web site: http://www.sics.se/contiki

[24] M. Barabanov. A Linux-based RealTime Operating System. Master's thesis, New Mexico Institute of Mining and Technology, 1997.

[25] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler. The Emergence of Networking Abstractions and Techniques in TinyOS. In Proc. NSDI, 2004.

[26] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy aware wireless microsensor networks. IEEE Signal Processing Magazine, 19(2):40-50, 2002.

[27] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In Proc. ASPLOS-IX, November 2000.

[28] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. MANTIS: system support for MultimodAl NeTworks of In-Situ sensors. In Proc. WSNA03, 2003.

[29] Joan Cortes, Qi Wang, John Dunlop Novel Metric for Identifying Energy-Vulnerable Nodes and Corresponding Proactive Schemes in Wireless Sensor Networks. Mobile Communications Group, Department of Electronic and Electrical Engineering University of Strathclyde, Glasgow UK, 2009.

Appendice A

Principio di funzionamento dei protothreads in Contiki

Nella seguente appendice si riporta il principio di funzionamento dei protothreads che stanno alla base del funzionamento del sistema operativo Contiki. Per la descrizione dettagliata riguardo alle operazioni che si possono fare con i protothreads si rimanda alla letteratura. Nell'implementazione in linguaggio C, tutte le operazioni riguardanti i protothreads sono nascoste dietro delle macro. La ragione per la costruzione della libreria dei protothreads su macro invece che su funzioni risiede nel fatto che i protothreads alterano il flusso di controllo. Questa operazione è tipicamente difficile da realizzare ricorrendo a delle funzioni C poichè una tale implementazione richiederebbe codice assembly di basso livello per poter funzionare. Implementando invece i protothreads mediante macro, è possibile alterare il flusso di controllo solo ricorrendo ai costrutti C standard.
Per capire bene come i protothreads operano, viene dato un breve sguardo alle modalità con cui il preprocessore C espande le macro associate e a come il codice C risultante viene eseguito.
Per lo scopo si introduce un semplice programma scritto con i protothreads. Poichè il programma è piccolo in termini di linee di codice, si mostra l'intero programma inclusa la funzione main(). Il programma, mostrato in figura A.1, attende che un certo contatore raggiunga una determinata soglia, stampa a video un messagio e azzera il contatore. Tutto ciò è eseguito in un loop infinito di tipo while(). Il contatore viene incrementato nella funzione main().

```
#include "pt.h"

static int counter;
static struct pt example_pt;

static
PT_THREAD(example(struct pt *pt))
{
  PT_BEGIN(pt);

  while(1) {
    PT_WAIT_UNTIL(pt, counter == 1000);
    printf("Threshold reached\n");
    counter = 0;
  }

  PT_END(pt);
}

int main(void)
{
  counter = 0;
  PT_INIT(&example_pt);

  while(1) {
    example(&example_pt);
    counter++;
  }
  return 0;
}
```

Figura A.1: Programma sviluppato con i protothreads

Prima di mostrare come il preprocessore espande il codice, conviene dare uno sguardo alla definizione dei protothreads in macro. Nella figura A.2 viene fornita una definizione semplificata ai fini della comprensione.

```
struct pt { unsigned short lc; };
#define PT_THREAD(name_args)  char name_args
#define PT_BEGIN(pt)          switch(pt->lc) { case
0:
#define PT_WAIT_UNTIL(pt, c)  pt->lc = __LINE__;
case __LINE__: \
                              if(!(c)) return 0
#define PT_END(pt)            } pt->lc = 0; return 2
#define PT_INIT(pt)           pt->lc = 0
```

Figura A.2: Definizione di un protothread

Si nota che la struttura pt consiste di un singolo unsigned short chiamato lc, che sta per local continuation. Questa variabile unsigned short rappresenta la sorgente dei 2 byte di overhead che vengono frequentemente menzionati quando si parla di protothreads. In aggiunta, si nota che la macro PT-THREAD semplicemente inserisce un char prima del suo argomento. Le macro PT-BEGIN e PT-END aprono e chiudono rispettivamente uno switch statement. La macro PT-WAIT-UNTIL risulta più complessa delle altre. Essa contiene un assegnamento, un case statement, un if statement e anche un return statement. Inoltre, essa utilizza la macro built-in LINE due volte. La macro built-in LINE è una speciale macro che il preprocessore C espande sul numero di linea a cui la macro fa riferimento. La macro PT-INIT semplicemente inizializza la variabile lc a zero. molti degli statements utilizzate nelle macro dei protothread non sono utilizzati troppo spesso nelle macro di carattere generale. Il return utilizzato in PT-WAIT-UNTIL ferma il flusso di controllo nella funzione in cui la macro è utilizzata. Per questo motivo, la maggior parte dei programmatori preferisce non utilizzare il return nelle macro. Inoltre si nota come la macro PT-BEGIN apra uno switch, ma non lo chiude. Questo tipo di operazioni protrebbero sembrare non corrette se non si considera il punto di vista dei protothreads. Esse in realtà

consentono modifiche al flusso di controllo e questo costituisce il principale obiettivo dei protothreads. Di seguito si osserva invece come il preprocessore C espande il codice riportato nel programma di esempio formulato in precedenza (figura A.3).

```
static
PT_THREAD(example(struct pt *pt))
{
  PT_BEGIN(pt);

  while(1) {
    PT_WAIT_UNTIL(pt,
      counter == 1000);
    printf("Threshold reached\n");
    counter = 0;
  }

  PT_END(pt);
}
```

```
static
char example(struct pt *pt)
{
  switch(pt->lc) { case 0:

  while(1) {
    pt->lc = 12; case 12:
    if(!(counter == 1000)) return
0;
    printf("Threshold reached\n");
    counter = 0;
  }

  } pt->lc = 0; return 2;
}
```

Figura A.3: Definizione di un protothread

Nella prima riga di codice si nota come la macro PT-THREAD viene espansa in modo tale che l'esempio con i protothread risulti eseguibile in C come una normale funzione che restituisce un char. Il valore di ritorno della funzione protothread può essere utilizzato per vedere se il protothread risulta bloccato in attesa di un evento e perchè lo stesso protothread è terminato. La macro PT-BEGIN è stata espansa in uno switch con parentesi aperta. Si osserva che la corrispondente parentesi di chiusura dello switch si trova in corrispondenza della macro PT-END. Dopo la parentesi di apertura dello switch, si nota come l'espansione PT-BEGIN riporti un case 0 statement. Questo assicura che la macro PT-BEGIN sia la prima ad essere eseguita quando è attivato un protothread. Muovendosi lungo l'espansione di PT-WAIT-UNTIL e in particolare dopo la dichiarazione del ciclo infinito while(1), si nota come la variabile lc sia impostata a 12 e venga definito uno statement case pari a 12. Dopo questo, il programma verifica se il contatore ha raggiunto il valore 1000 o meno. Se no, il programma effettua un return esplicito. Tutto ciò potrebbe sembrare strano e incomprensibile, ma per capire veramente cosa il programma scritto in protothreads effettua, bisogna rivolgere l'attenzione alla chiamata successiva della funzione.

La volta successiva in cui la funzione main() richiama la funzione di esempio, la variabile lc non varrà zero ma 12, come stabilito nell'espansione della macro PT-WAIT-UNTIL. Questo porta lo switch(lc) a saltare al case 12 statement che è posto immediatamente prima del if. In questo modo si può ancora verificare se il contatore ha raggiunto la soglia o meno. Se no si effettua un return e si attende la chiamata successiva della funzione. Questa procedura continua fino a quando il contatore non ha raggiunto il valore 1000 di soglia. Una volta raggiunto, il programma stampa a video e azzera il contatore. Il numero 12 indica sostanzialmente il numero di riga in cui è presente PT-WAIT-UNTIL. L' aspetto interessante che caratterizza le righe di codice è che sono monotonicamente crescenti. Ovvero, se inseriamo nel nostro programma un altro PT-WAIT-UNTIL, il corrispondente numero di linea sarà differente da quello contenuto nel primo PT-WAIT-UNTIL. Quindi lo switch sa perfettamente dove effettuare il salto senza incorrere in ambiguità.