Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Progettazione, implementazione ed analisi di un gateway multi-radio per reti di sensori mesh 6LoWPAN

Relatore:
Chiar.mo Prof.
Marco Di Felice

Presentata da:
Giovanni Modica

Correlatore:
Chiar.ma Prof.ssa
Eleonora Franchi Scarselli

II
2016/2017

"Society is now created for technological,
rather than for human, requirements.
And that's where tragedy begins."

C. Virgil Gheorghiu, The Twenty-Fifth Hour, 1950

Introduzione

L'Internet of Things permette a qualsiasi oggetto utilizzato nella vita quotidiana di essere connesso alla rete internet, come lo sono dispositivi personali quali PC, notebook, smartphone e tablet. In questo modo vi è la possibilità di creare reti isolate formate da sensori, le cosiddette Wireless Sensor Networks (WSNs). Le WSNs sono una particolare tipologia di reti wireless ideate per dispositivi a basso costo, di limitate dimensioni e prestazioni. Ogni dispositivo è un nodo della rete.

I nodi possono comunicare tra di loro per coordinarsi, e collaborare per l'elaborazione e la trasmissione dei dati da un punto ad un altro della rete. Un nodo particolare chiamato gateway raccoglie le informazioni dei sensori e le rende disponibili attraverso Internet.

L'aumento del numero di queste reti ha portato alla progettazione, allo sviluppo di architetture di rete e protocolli per la comunicazione tra i nodi della rete interna ed esterna. Tra i vari standard è presente 6LoWPAN un protocollo che permette l'utilizzo di IPv6 nelle reti di sensori. Esso permette alla rete di integrarsi facilmente con Internet.

Il compito di stabilire la logica secondo la quale un nodo ne identifica un altro come destinatario del proprio messaggio, e quindi delle politiche di routing spetta a RPL. RPL è un protocollo di routing creato appositamente per le reti di sensori.

La comunicazione verso l'esterno avviene attraverso MQTT, un protocollo di livello applicativo (es. HTTP, COAP).

In questa tesi si descrive l'implementazione di un gateway che raccoglie ed

elabora le informazioni fornite dai nodi della 6LoWPAN e le rende disponibili su Internet attraverso MQTT.

Il nodo in questione è equipaggiato con due moduli radio dato che la comunicazione con i nodi interni ed esterni avviene su diverse frequenze.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

Al giorno d'oggi stiamo entrando sempre più nella terza era di Internet. La prima era cominciata con il World Wide Web che permetteva e permette tuttora a milioni di persone di condividere informazioni attraverso l'uso dei PC. La seconda era cominciata nel 2000 con l'introduzione dei moderni smartphones e quindi della possibilità di essere connessi in ogni luogo e ad ogni ora con il resto del mondo, pur non trovandosi in casa. Nella terza era con l'avvento dei dispositivi embedded, più comunemente chiamati *smart objects*, come ad esempio smartphones, dispositivi per il monitoraggio della salute, dispositivi per l'automazione casalinga e per il monitoraggio ambientale, entriamo nel mondo dell'*Internet of Things*.

1.1 Internet of Things

L'Internet of Things, identificato con l'acronimo IoT, permette di raccogliere informazioni da oggetti che utilizziamo nella vita di tutti i giorni e inviarle attraverso Internet per elaborarle e migliorare in questo modo, la qualità della vita. Proprio per questo le dimensioni dell'IoT sono difficili da stimare, in quanto non dipendono dal numero di utenti ma dalla quantità di dispositivi connessi alla rete Internet. Si stima però che le sue dimensioni possano superare quelle di Internet per come lo conosciamo, e continuare a

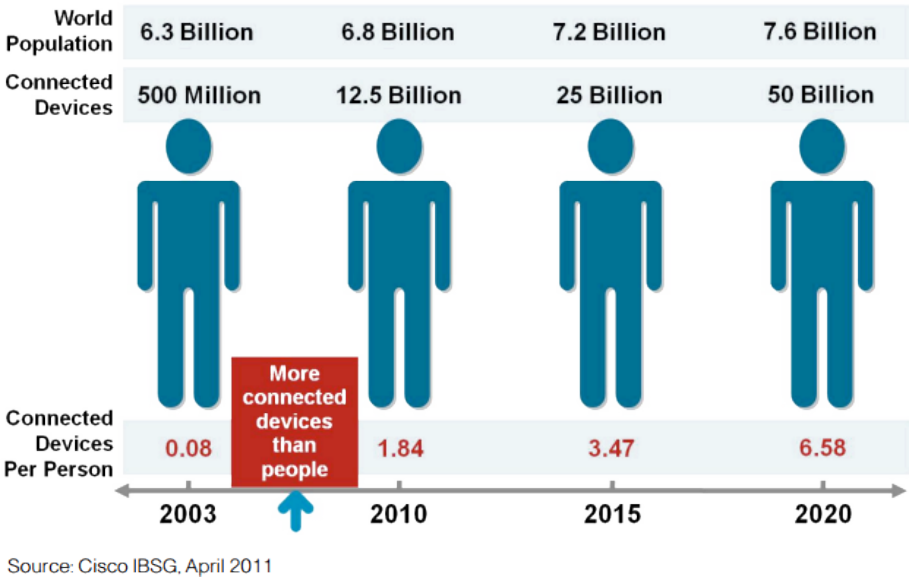

Figura 1.1: Crescita dell'IoT (fonte CISCO).

crescere rapidamente. L'IoT opera in diversi campi quali sanità, trasporti, gestione della casa, catene di approvvigionamento alimentare, gestione energetica delle strutture (industrie, abitazioni, ecc), ecc.

1.1.1 Esempio

Di solito le persone all'interno della propria abitazione interagiscono ogni giorno con l'ambiente che li circonda accendendo/spegnendo luci, regolando i riscaldamenti, ecc. Uno degli obiettivi dell'IoT è quello di automatizzare queste azioni creando sistemi che permettano di analizzare il comportamento degli inquilini e comportarsi di conseguenza. In genere questi sistemi sono formati da un insieme di sensori che collezionano diverse tipologie di dati riguardanti i comportamenti dei residenti e i consumi domestici. Questi dati verranno sfruttati dal sistema per migliorare gli standard di vita dei residenti e ridurre i consumi. Per fare ciò si deve creare un'infrastruttura composta da sensori.

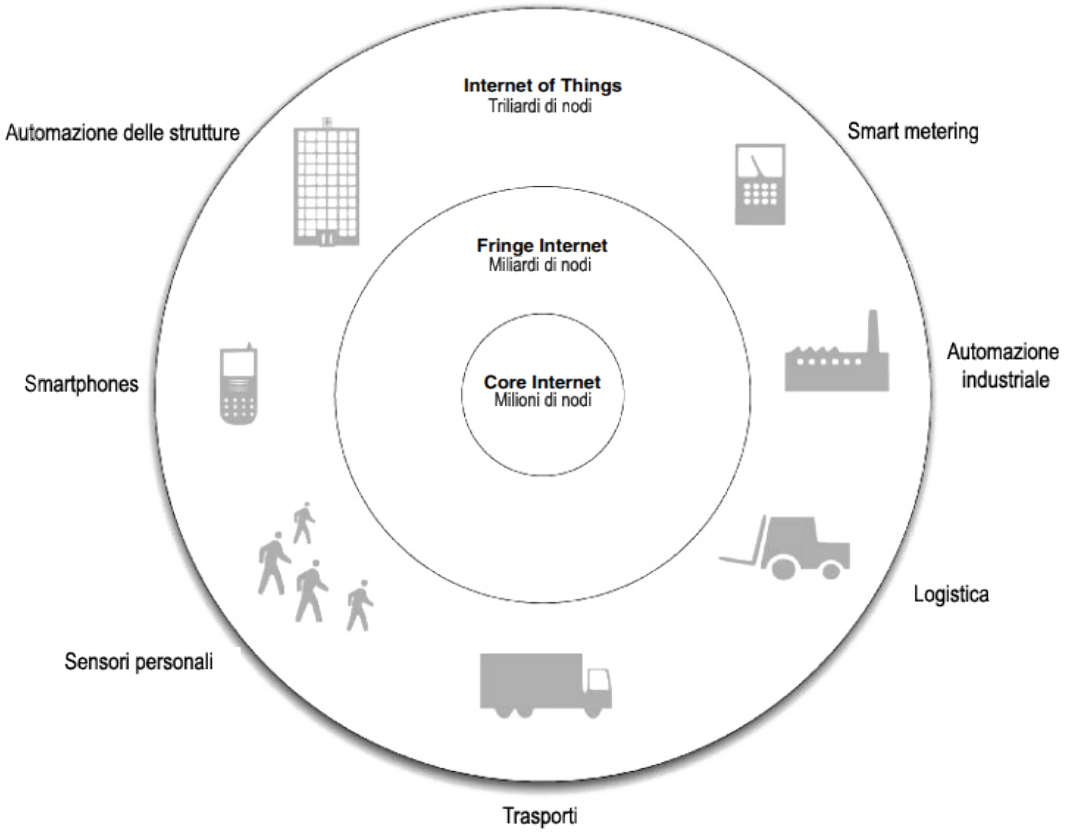

Figura 1.2: Internet of Things, dimensioni e casi d'uso.

1.2 Wireless Sensor Network

Le WSN sono reti formate da vari nodi connessi tra di loro attraverso la tecnologia wireless, che raccolgono varie tipologie di dati e li inviano ad un sistema centrale il quale li analizza ed elabora.

Questi nodi devono supportare la comunicazione senza fili, l'elaborazione e la raccolta dei dati. Per fare ciò la maggior parte di questi dispositivi deve avere un certo tipo di hardware: un microcontrollore per la computazione, una limitata quantità di RAM, uno o più banchi di memoria per l'installazione del codice che definisce il suo comportamento, una ricetrasmittente wireless, un'antenna, uno o più sensori e una fonte di alimentazione (batteria

o cavo). Il microprocessore presente in ogni nodo ha varie funzionalità che includono la gestione dei dati raccolti attraverso i sensori, la gestione del protocollo di comunicazione e di quella energetica. Una caratteristica fondamentale di ogni nodo è la minimizzazione del consumo energetico. In queste tipologie di nodi i moduli che utilizzano la quantità maggiore di energia sono quello radio e quelli dei vari sensori. Per questo l'hardware e il software devono essere progettati per garantire delle politiche di risparmio energetico.

I nodi che andranno a formare la rete devono essere disposti secondo una topologia di rete. Ne esistono diverse ed ognuna ha i suoi pro e i suoi contro.

1.2.1 Topologia di rete

Rete a stella

Le reti a stella sono reti formate da un nodo centrale, il coordinatore della rete, che può inviare e ricevere messaggi da un certo numero di nodi remoti. Questi nodi possono comunicare solamente con il coordinatore senza poter comunicare tra di loro. Questa topologia permette di limitare il consumo di energia dei nodi remoti, ma per contro se il coordinatore cessa di funzionare l'intera rete diventa inusabile. Ovviamente tutti i nodi remoti devono stare all'interno del range di comunicazione del coordinatore e questo permette solo la creazione di reti vicine spazialmente.

Rete mesh

Una rete mesh è composta da tanti nodi che possono comunicare tra di loro senza nessuna limitazione, purchè siano nel range di comunicazione radio. Questa topologia di rete permette la comunicazione multi-hop tra i nodi. Ciò vuol dire che se mittente e destinatario non si trovano nel range di comunicazione, il messaggio per raggiungere la destinazione finale passa per dei nodi intermedi che si occupano di inoltrarlo al nodo corretto. La rete in questione è molto più robusta di quella a stella, poichè se un nodo della rete smette di funzionare vi sono dei percorsi multipli che collegano gli altri

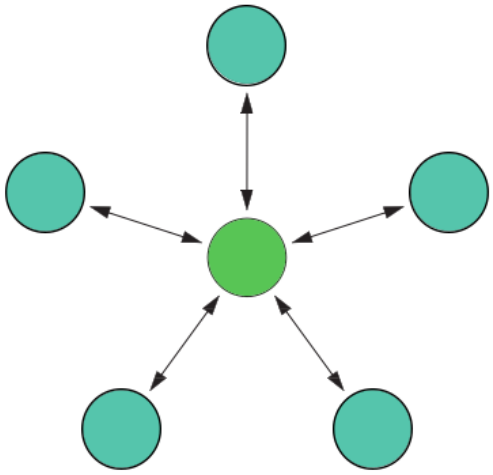

Figura 1.3: Topologia a stella. I nodi blu indicano gli host, i verdi i router.

nodi. Inoltre la dimensione della rete può essere elevata in quanto i nodi non devono trovarsi tutti nel range di comunicazione radio. Gli svantaggi derivano dall'elevato consumo di energia dei nodi che hanno il compito di inoltrare i messaggi verso la destinazione finale. In più, all'aumentare del numero di hop che separano la sorgente dalla destinazione aumenta anche il tempo per la consegna del messaggio.

Rete ibrida

Questa topologia di rete è un ibrido tra quella a stella e la rete mesh. In questa configurazione sono presenti due tipologie di nodi. I primi sono nodi ad elevata potenza trasmissiva generalmente collegati alla rete elettrica e hanno il compito di inoltrare messaggi verso altri nodi della rete (multihop). L'altra tipologia di nodi è rappresentata da dispositivi a bassa potenza che possono esclusivamente comunicare con nodi di tipo diverso (mai con quelli dello stesso tipo). Questa topologia fornisce una comunicazione robusta e versatile pur mantenendo basso il consumo di energia dei dispositivi.

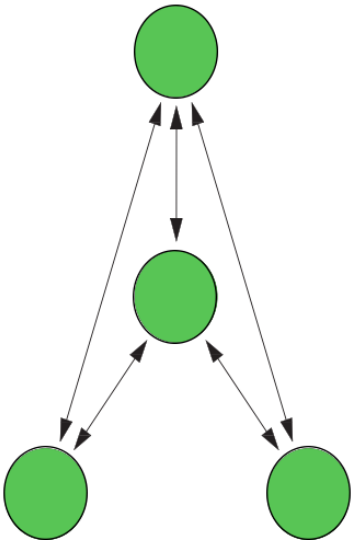

Figura 1.4: Topologia a mesh. Si hanno solo nodi router.

1.2.2 Tecnologia

Un'importante scelta durante la creazione di una rete di sensori è quella sulla tecnologia radio da utilizzare. Il livello radio definisce lo schema di modulazione da adottare, la frequenza radio sulla quale operare, ecc. Generalmente, conviene adottare un'interfaccia radio basata su uno dei tanti standard presenti.

IEEE 802.11x

Lo standard *IEEE 802.11* viene utilizzato maggiormente nelle *Local Area Network* (LAN) per le comunicazioni a banda larga tra computer e altri dispositivi. La velocità di trasferimento dati varia da un minimo di 1 Mbps fino ad un massimo di 50 Mbps. Tipicamente il range di trasmissione non è particolarmente elevato ma può essere migliorato tramite l'introduzione di un'antenna direzionale. Questa soluzione però non è particolarmente consigliata a causa degli elevati requisiti energetici della tecnologia.

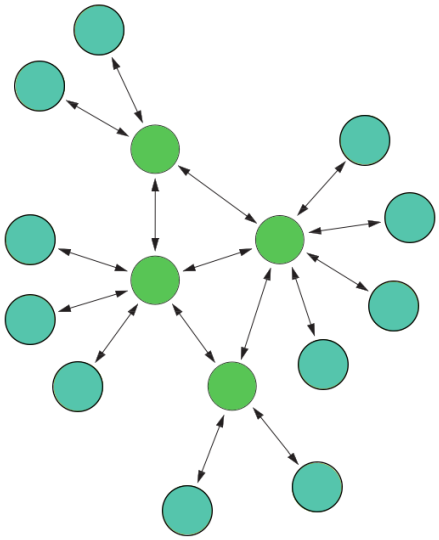

Figura 1.5: Topologia ibrida.

Bluetooth

Il *Bluetooth* è uno standard utilizzato nelle *Personal Area Network* (PAN) che ha dei requisiti energetici inferiori rispetto a quelli dell'802.11. Questo standard utilizza una topologia a stella che consente di connettere fino a 7 nodi remoti ad un nodo coordinatore. Bluetooth definisce una particolare modalità chiamata *sleep*, in cui i nodi della rete vengono spenti temporaneamente risparmiando così una notevole quantità di energia. Nonostante molte aziende abbiano utilizzato questa tecnologia durante la costruzione dei loro sensori, questo standard non ha mai preso piede nelle reti di sensori a causa di alcune limitazioni, come la possibilità di creare reti con un numero massimo di nodi molto basso, l'eccessivo consumo energetico per trasmissioni a breve distanza e l'eccessivo tempo dovuto alla sincronizzazione della rete dopo un periodo di sleep.

IEEE 802.15.4

Questo standard è stato specificatamente creato per le reti di sensori. L'*802.15.4* è molto flessibile in quanto permette di scegliere tra vari data rate e tra varie frequenze trasmissive. Inoltre, fornisce un meccanismo di sleep che permette ai nodi di ridurre al minimo il consumo energetico. La procedura di sincronizzazione che avviene dopo la fase di sleep è stata resa più efficiente e veloce. Grazie alle migliorie introdotte è possibile ottenere un elevato risparmio energetico grazie alla possibilità di spegnere periodicamente e temporaneamente il modulo radio del dispositivo. Lo standard trasmette dati su tre diverse frequenze 868 MHz, 902-928 MHz e 2.48-2.5 GHz. Il data rate va da un minimo di 20 Kbps fino ad un massimo di 250 Kbps. Supporta le topologie a stella e a mesh oltre a fornire un meccanismo per la stima della qualità del collegamento radio.

ZigBee

ZigBee è una tecnologia di reti wireless sviluppata dalla *ZigBee Alliance* per applicazioni che necessitano di una bassa velocità di trasmissione dati e comunicazioni a breve distanza. Lo stack del protocollo ZigBee è formato da 4 livelli: il livello fisico, il livello medium access control, il livello rete e il livello applicazione. I due livelli più bassi dello stack sono definiti dallo standard IEEE 802.15.4, mentre il resto dello stack è definito dalla ZigBee Alliance.

LoRa

LoRa è una tecnologia radio che modula i segnali nelle bande Sub-GHz ISM utilizzando una tecnica *spread spectrum* sviluppata e commercializzata dalla *Semtech Corporation*. Opera nelle bande ISM 433 MHz e 868 MHz in Europa, 430 MHz in Asia e 915 MHz negli Stati Uniti. Il data rate va da un minimo di 0.3 Kbps fino ad un massimo di 37.5 Kbps. Il range di comunicazione arrivo fino a 5 km. A livello MAC utilizza l'*unslotted ALOHA* per

l'accesso al canale. Questa tecnologia, contrariamente a ZigBee o 802.15.4, viene utilizzata in reti WAN (Wide Area Network) sviluppate per dispositivi low-power.

DASH7

Dash7 è una tecnologia che fornisce connettività a media distanza per sensori ed attuatori low-power. Opera nelle bande Sub-GHz 433 MHz, 868 MHz e 915 MHz utilizzando la modulazione Gaussian Frequency Shift Keying (GFSK). Il data rate va da un minimo di 9.6 fino ad un massimo di 166.7 kbps. A livello MAC utilizza il protocollo CSMA/CA per l'accesso al canale e prevede due topologie di rete: a stella e ad albero. Il range di comunicazione arriva fino a 5 km in scenari urbani.

1.2.3 Casi d'uso

Healthcare monitoring

L'idea di base, fornita da [17], è quella di formare una rete di sensori che monitora in tempo reale le condizioni di salute dei pazienti all'interno di una struttura ospedaliera.

La rete è formata da dei sensori, rappresentati da dispositivi indossabili assegnati ad ogni paziente, e da un gateway, un nodo che raccoglie le informazioni dai sensori, le elabora e le comunica attraverso Internet o Intranet. Tutti i nodi della rete utilizzano la tecnologia 6LoWPAN. Questo permette di semplificare la costruzione del gateway che si deve interfacciare con la rete Internet basata su IP.

Utilizzare dispositivi wireless per questi scopi semplifica la progettazione del sistema. I pazienti, infatti, sono liberi di spostarsi come vogliono rimanendo monitorati.

Home automation

L'idea è quella di creare una rete di sensori wireless, formata da un interruttore della luce, una valvola per termosifoni, un salvavita e un dispositivo di controllo remoto basato sulla tecnologia ZigBee.

Questo sistema, progettato da [18], è formato da 4 step. L'utente remoto può accedere al sistema attraverso Internet. I messaggi arrivano nella rete domestica per mezzo della rete Wi-Fi casalinga. I dati ricevuti vengono trasmessi al gateway. Sarà lui a comunicare i comandi agli attuatori o a rispondere all'utente remoto fornendo le informazioni richieste recuperate dai sensori.

Un difetto dei sistemi per reti di sensori che utilizzano protocolli proprietari come ZigBee deriva dal fatto che il gateway generalmente, deve implementare un meccanismo per la traduzione degli indirizzi IP in quelli utilizzati in ZigBee e viceversa.

Fence Monitoring

Le reti di sensori sono molto utili quando si parla di sicurezza. In questo caso [15], l'idea è quella di creare una rete di sensori e attaccarla ad una recinzione in modo da rilevare intrusioni o falle nella sicurezza. La recinzione è composta da vari elementi connessi tra loro, ognuno dei quali è equipaggiato con un sensore. Ogni sensore è progettato per funzionare anche in condizioni meteo critiche.

La rete di sensori riesce a identificare vari tipi di eventi:

- Kick: una persona da un calcio alla recinzione;
- Lean: una persona si appoggia;
- Shake (short o long): una persona agita il recinto per un certo periodo di tempo;
- Peek: una persona si arrampica sulla recinzione per vedere cosa c'è dietro;

- Climb: l'evento più critico, avviene quando una persona scavalca la recinzione.

La procedura utilizzata per identificare gli eventi è formata da 4 passi.

Inizialmente i dati grezzi vengono ripuliti dal rumore in sottofondo ed aggregati secondo varie proprietà.

Nel secondo step un algoritmo controlla se ci sono dei pattern conosciuti tra i dati aggregati. I pattern fanno riferimento agli eventi presentati precedentemente.

Successivamente i sensori decidono se un evento degno di nota è effettivamente avvenuto collaborando con i vicini attraverso lo scambio di informazioni. Infine, una volta rilevato l'evento, l'informazione viene inviata alla base station che agirà di conseguenza (allarmi).

Smart Energy

In questo campo la rete di sensori è formata da dispositivi collegati agli elettrodomestici presenti nell'abitazione e a qualsiasi altro oggetto che necessità di corrente elettrica per funzionare. L'obiettivo è quello di avere un consumo energetico più intelligente in grado di evitare gli sprechi [16]. La misura dei parametri elettrici degli elettrodomestici viene effettuata interfacciandosi con i loro moduli di rilevazione. I segnali in uscita dai sensori sono integrati e collegati al modulo *XBee* (radio) per la trasmissione dei dati. I moduli XBee sono interfacciati con diversi dispositivi di rilevamento e sono interconnessi sotto forma di rete mesh per garantire una ricezione affidabile delle informazioni.

Un coordinatore ZigBee collegato ad un PC raccoglie e memorizza i dati. Le informazioni vengono poi inviate ad un gateway per avere un controllo remoto dei parametri elettrici. Quindi gli apparecchi possono essere controllati automaticamente in locale o manualmente da remoto. Grazie a questo sistema è possibile ottenere un risparmio energetico configurando l'accensione degli elettrodomestici (es. lavatrice) in parti della giornata in cui c'è una tariffa energetica più conveniente.

Applicazioni militari

- **Self-healing land mines:** è una rete di sensori formata da mine anticarro in grado di posizionarsi e riposizionarsi autonomamente. Ogni mina è equipaggiata con sensori acustici ed accelerometri, monitora lo stato dei suoi vicini, rileva le minacce e risponde autonomamente muovendosi. Questo meccanismo viene utilizzato, ad esempio, per evitare che ci siano parti sguarnite nel campo minato.

- **Chemical, biological, and explosive vapor detection with micro cantilever array sensors:** la *Nevada Nanotech Systems* ha sviluppato una tecnologia in grado di rilevare tracce di esplosivi, agenti chimici tossici e agenti biologici. Il sistema attraverso una rete di sensori low-power a basso costo, è in grado di analizzare le proprietà elettriche e termiche delle molecole presenti nell'aria. Questo approccio permette il monitoraggio di aree a rischio come campi di battaglia, centri di ricerca che lavorano su agenti patogeni, ecc.

Capitolo 2

Tecnologie utilizzate

Nella fase di progettazione sono state effettuate delle scelte per quanto riguarda le tecnologie da utilizzare. Come protocollo di rete è stato utilizzato *6LoWPAN*, uno standard che permette di utilizzare il protocollo *IPv6* nelle reti di sensori. Per il routing la scelta è ricaduta sul protocollo *RPL*, definito dall'*IETF* come uno degli standard per le reti *6LoWPAN*. Infine, per la comunicazione tra la rete LoWPAN e il mondo esterno è stato adottato *MQTT*, un protocollo creato da *IBM*. In questo capitolo verranno, come prima cosa, presentati questi protocolli ed in seguito saranno elencate le componenti hardware utilizzate per creare la rete.

2.1 IPv6

IP version 6 (*IPv6*) è una nuova versione dell'*Internet Protocol* progettato per essere il successore del protocollo *IPv4*.

Il protocollo IPv4 è stato utilizzato in questi 20 anni data la sua robustezza, la facilità di implementazione e l'interoperabilità. Tuttavia, il progetto iniziale non ha tenuto conto di alcune condizioni:

- le dimensioni di Internet crescono esponenzialmente e questo porta all'esaurimento dello spazio di indirizzamento del protocollo IPv4;

- la necessità di avere dei meccanismi di autoconfigurazione degli indirizzi più semplice;
- requisiti di sicurezza a livello Rete (IP, IPSec);
- la necessità di un miglior supporto per la consegna dei dai in tempo reale (QoS).

Il formato degli indirizzi IPv4 a 32 bit permette di avere un massimo di 4.3 miliardi di indirizzi univoci. Sebbene questo valore possa sembrare elevato, non riesce a stare al passo con la crescita esponenziale che sta subendo Internet. Grazie ai 128 bit utilizzati negli indirizzi IPv6 è possibile avere un numero molto elevato di indirizzi univoci ($3.4 * 10^{38}$). Oltre a questo IPv6 ha altri vantaggi:

- *Header* semplificato per una gestione più efficiente dei pacchetti;
- Dimensioni del *payload* più grandi in modo da avere un maggior *throughput* e un meccanismo di trasporto più efficiente;
- Autoconfigurazione e supporto al *plug-and-play*;
- Viene eliminata la necessità di un *NAT* (*network address translation*);
- Supporto per protocolli di routing già ampiamente utilizzati;
- Maggior numero di indirizzi multicast.

2.1.1 Indirizzamento

IPv6 fornisce il supporto per tre modalità di indirizzamento: *unicast*, *multicast* e *anycast*.

Nella modalità unicast, un nodo è identificato univocamente dall'indirizzo. Il pacchetto contiene sia l'indirizzo del mittente che quello del destinatario. Quando un router riceve il pacchetto, lo inoltra scegliendo l'interfaccia IPv6 connessa all'indirizzo unicast presente nel messaggio.

Nella modalità multicast il pacchetto destinato a più nodi viene inviato attraverso un particolare indirizzo multicast. Tutti i nodi che vogliono ricevere informazioni attraverso multicast devono prima entrare a far parte del gruppo multicast. I nodi all'interno di esso riceveranno i pacchetti attraverso il multicast e li processeranno, quelli non interessati ignoreranno il messaggio.

La terza modalità è stata introdotta da IPv6. A più nodi viene assegnato lo stesso indirizzo anycast. Quando un nodo vuole comunicare con uno avente l'indirizzo anycast, invia un messaggio in unicast. Attraverso un meccanismo di routing, il pacchetto viene inviato al nodo più vicino al mittente in termini di costo del routing.

2.1.2 Indirizzi

Un indirizzo IPv6 è formato da 128 bit divisi in otto blocchi da 16 bit ciascuno. Ogni blocco viene poi convertito in blocchi formati da 4 cifre esadecimali, separati da un simbolo speciale (**:**). La prima parte dell'indirizzo IPv6 è composta dal prefisso IP. La seconda parte, gli ultimi 64 bit, è utilizzata per l'*Interface ID*. In genere questo campo è riempito utilizzando l'indirizzo MAC dei dispositivi. Ciò avviene poichè questi indirizzi sono considerati univocamente assegnati in tutto il mondo.

IPv6 definisce tre tipologie di indirizzi unicast:

- *Global Unicast Address*: questo indirizzo è equivalente all'indirizzo pubblico IPv4 ed è assegnato globalmente e univocamente.

- *Link-Local Address (FE80)*: è l'indirizzo utilizzato durante la procedura di autoconfigurazione. Questi indirizzi vengono utilizzati per le comunicazioni on-link tra nodi IPv6.

- *Unique-Local Address (FD00)*: questo tipo di indirizzo è globalmente univoco ma dovrebbe essere utilizzato nelle comunicazioni locali. Non è possibile effettuare il routing attraverso Internet ma solo localmente.

2.2 6LoWPAN

Lo standard 6LoWPAN è stato definito dall'*Internet Engineering Task Force* (IETF), un ente che ha il compito di creare e curare la manutenzione di tutti gli standard e le architetture riguardanti Internet.

Il 6LoWPAN è uno standard che permette la creazione di reti wireless formate da dispositivi che utilizzano il protocollo IP per la comunicazione. Tutto ciò avviene per mezzo di un livello posto tra quello MAC e Rete, l'Adaptation Layer. Oltre all'utilizzo di indirizzi IP per la gestione della comunicazione, è possibile utilizzare delle ottimizzazioni di protocolli già esistenti sviluppati per IPv6.

Questo standard nasce per applicazioni usate in reti di sensori in cui:

- I dispositivi embedded hanno la necessità di comunicare con servizi basati su Internet (IPv6 / IPv4);
- Si ha la necessità di collegare insieme più reti anche diverse tra loro;
- La rete dovrebbe essere scalabile e riusabile in modo da accogliere nuovi servizi.

Ovviamente la tecnologia 6LoWPAN può essere utilizzata in tutte quelle applicazioni costruite per reti di sensori, come l'automazione casalinga e delle strutture in generale, efficienza energetica, sicurezza delle infrastrutture, automazione veicolare, ecc.

Il mondo dell'IoT è formato da una moltitudine di "isole" composte da dispositivi embedded, connesse tra di loro. Le isole in questione vengono chiamate *stub networks*. Ogni isola è formata da una rete LoWPAN connessa ad Internet attraverso un nodo particolare presente in ogni rete, il gateway o edge router.

Le reti LoWPAN sono composte da un numero variabile di nodi, ognuno dei quali può assumere ruoli differenti. Esistono nodi host, router o gateway. Le interfacce di rete dei nodi in una 6LoWPAN condividono lo stesso prefisso IPv6 che viene distribuito dal gateway e dai router attraverso tutta la rete.

Per fare ciò si utilizza un meccanismo costruito per l'IPv6 e riadattato per il 6LoWPAN, la *Neighbor Discovery*. Attraverso esso i nodi registrano i loro indirizzi e li comunicano al gateway. I nodi possono far parte di una o più LoWPAN, spostarsi senza problemi attraverso tutta la rete e attraverso più edge router. Grazie al routing a livello IP (Route-over) o all'inoltro dei pacchetti attraverso il livello Datalink è possibile ottenere la comunicazione multi-hop tra i nodi.

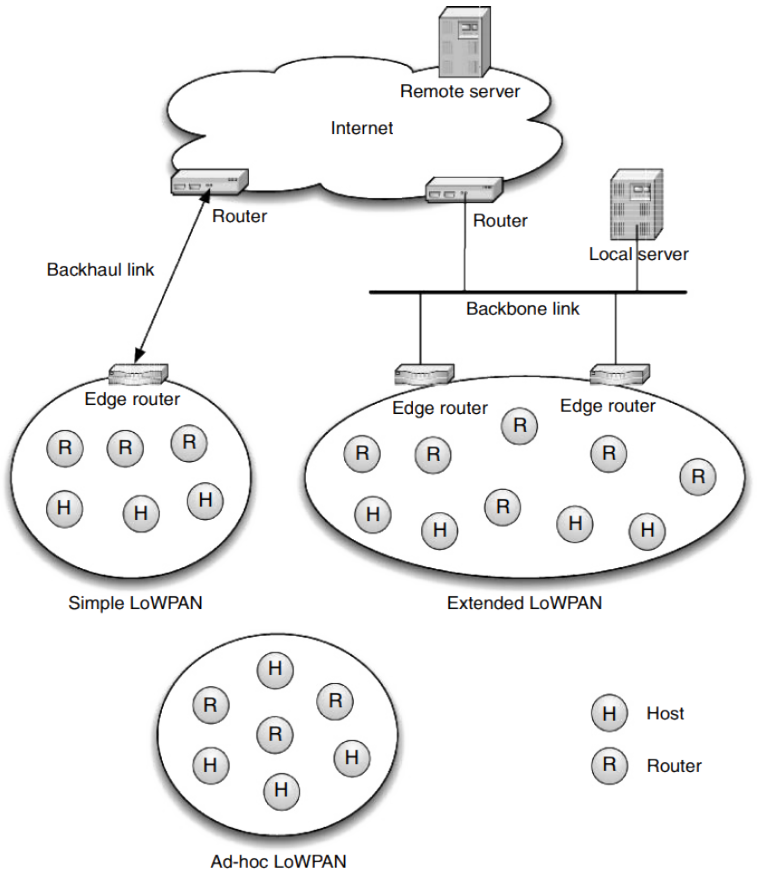

Figura 2.1: Architettura 6LoWPAN.

Lo standard definisce tre tipologie di reti LoWPAN:

- *Simple LoWPAN*: è connessa attraverso un edge router ad una rete IP esterna. Il collegamento è realizzato attraverso un *backhaul link*.

- *Extended LoWPAN*: questa rete collega più edge routers appartenenti a reti diverse attraverso un *backbone link*.
- *Ad-hoc LoWPAN*: è una rete che opera senza l'utilizzo di un'infrastruttura e che quindi non è connessa ad Internet. In questa topologia, un router viene configurato come gateway con funzionalità minimali. Esso si occupa della generazione di un indirizzo ULA (*Unicast Local Address*) e della gestione degli indirizzi registrati attraverso la *6LoWPAN-ND*.

2.2.1 Stack protocollare

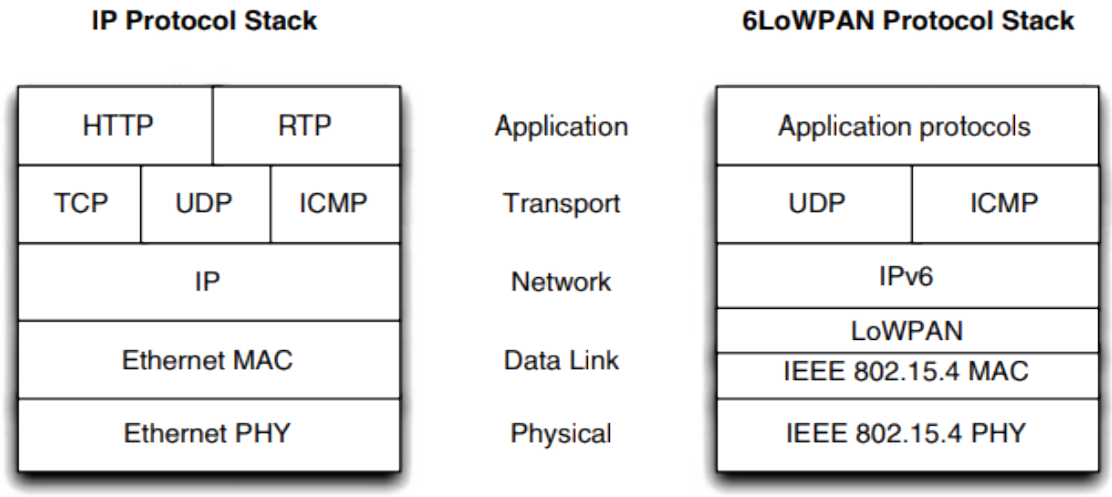

Figura 2.2: Stack protocollo IP e stack 6LoWPAN a confronto.

Lo stack dello standard 6LoWPAN è praticamente identico a quello del protocollo IP con alcune differenze.

Il livello fisico e quello Data Link seguono lo standard 802.15.4 con comunicazioni su varie frequenze:

- 2.4 GHz utilizzata in tutto il mondo, fornisce una velocità di trasmissione abbastanza elevata;
- 915 MHz utilizzabile solo nel Nord America;

- 868 MHz adottata nell'Unione Europea.

Lo standard permette di inviare dati tramite varie velocità trasmissive che vanno dai 20 ai 250 kbps. Per l'accesso al canale viene utilizzato il protocollo *CSMA*. L'indirizzamento avviene per mezzo di indirizzi a 64 bit o a 16 bit con la possibilità di utilizzare sia il broadcast che l'unicast. L'indirizzo IPv6 è formato da 128 bit ed è composto dal prefisso IPv6 a 64 bit e dall'indirizzo MAC anch'esso a 64 bit. In questo modo è possibile utilizzare la *Stateless Address Autoconfiguration* ed evitare la generazione di indirizzi ridondanti all'interno della 6LoWPAN.

Dall'immagine 2.2 si può notare la presenza di un livello posto tra quello Data Link e il livello Rete. L'*Adaptation Layer* ha il compito di ottimizzare il protocollo IPv6 per l'utilizzo con lo standard IEEE 802.15.4 o altri standard simili pensati per le reti di sensori.

Questo livello ha tre compiti principali:

- Compressione dell'header: i campi dell'header IPv6 vengono compressi assumendo l'utilizzo di informazioni che possono essere derivate da protocolli di livello più basso, o da altre che sono conosciute a priori da tutti i nodi;

- Frammentazione e riassemblaggio: i pacchetti IPv6 vengono frammentati poichè a causa della loro dimensione non possono essere utilizzati come frame a livello MAC. La frammentazione viene anche utilizzata per payload (dati da trasmettere, non include l'header) molto grandi;

- Link-layer forwarding: per supportare l'inoltro dei datagram IPv6, l'Adaptation Layer mantiene informazioni relative agli indirizzi MAC. Questi verranno utilizzati per il routing dei frame.

Compressione dell'header

Il concetto chiave dell'Adaptation Layer è quello della compressione, raggiunta tramite l'elisione di campi degli header di livello rete e trasporto, in

modo da avere header di pochi byte. Ciò è possibile in quanto molti campi degli header trasportano informazioni comuni e ridondanti come ad esempio indicazioni sul prefisso di rete, condiviso da tutta la LoWPAN. Inoltre è possibile eliminare informazioni superflue come gli indirizzi IPv6, in quanto possono essere dedotte dagli header dei livelli inferiori.

Frammentazione e riassemblaggio

IPv6 definisce un minimo *MTU* (*maximum transmission unit*) pari a 1280 byte. Questo è un valore troppo elevato per i frame dell'802.15.4 la cui dimensione non può superare i 127 byte. A causa di ciò non è possibile incapsulare un pacchetto IPv6 in un frame 802.15.4. L'Adaptation Layer definisce un meccanismo con il quale è possibile dividere un singolo pacchetto IPv6 in tanti piccoli frammenti le cui dimensioni andranno a soddisfare le specifiche dello standard 802.15.4. In questo modo sarà possibile inviare questi frammenti che saranno riassemblati una volta raggiunta la destinazione finale. L'header IPv6 sarà contenuto solo nel primo frammento della sequenza, gli altri conterranno delle informazioni utili al riassemblaggio.

Link-layer forwarding

Quando l'instradamento e l'inoltro dei pacchetti avviene a livello MAC vengono utilizzati indirizzi definiti da quel livello, ossia quello a 64 bit (EUI-64) o a 16 bit. I frame del livello MAC contengono informazioni relative all'indirizzo del mittente (*originator address*) e a quello del destinatario (*final destination address*). Nel caso della comunicazione multi-hop però, questi indirizzi vengono sovrascritti da ogni nodo intermedio. Ad ogni hop, l'indirizzo del destinatario verrà sovrascritto con quello del prossimo hop mentre l'indirizzo del mittente sarà quello del nodo che effettua l'inoltro. In questo caso si ha la necessità di archiviare l'indirizzo della destinazione finale in modo da inviare il pacchetto correttamente. Per fare ciò 6LoWPAN prevede l'utilizzo di un header particolare chiamato *Mesh Header*.

Quando si utilizza questo tipo di routing l'intera rete viene vista come un singolo hop IP. L'unico router IP è l'edge router. La frammentazione e il riassemblaggio avvengono alla sorgente ed alla destinazione finale rispettivamente. I frammenti vengono inviati in multicast ed ogni frammento può seguire un percorso diverso. Il nodo che riceve il messaggio controlla l'indirizzo di destinazione presente nel Mesh Header, se questo non corrisponde al suo indirizzo sovrascrive gli indirizzi presenti nei campi dell'header MAC e inoltra il messaggio. Dato che il messaggio viene riassemblato esclusivamente una volta che tutti i frammenti raggiungono la destinazione finale, se c'è una perdita di pacchetto e il numero di hop che separano la sorgente dalla destinazione è elevato, si ci accorge della perdita solo alla fine del percorso. In questo modo i meccanismi di ritrasmissione possono dilatare i tempi per gli invii dei pacchetti.

Livello rete

Oltre al routing a livello link layer è possibile utilizzare un altra tipologia di routing che permette l'utilizzo degli indirizzi IPv6. In questa configurazione ogni nodo della rete è visto come un singolo hop IP. Tutti i nodi sono router IP e possono inoltrare i messaggi verso la destinazione corretta utilizzando delle tabelle di routing. I messaggi vengono frammentati e riassemblati ad ogni hop dato che gli indirizzi di livello rete sono contenuti nei byte iniziali dell'header IPv6, che è presente solo nel primo frammento della sequenza. In questo modo il tempo per l'invio dei pacchetti da un punto ad un altro risulta più lento ma, la rilevazione delle perdite dei frammenti è più veloce.

Livello trasporto

Il protocollo di livello trasporto più comunemente utilizzato è l'**UDP** dato che è possibile comprimere il suo header grazie alla formattazione adottata dal protocollo 6LoWPAN. UDP (User Datagram Protocol) è un protocollo di tipo *connectionless*. Non gestisce il riordinamento dei pacchetti nè la

ritrasmissione di quelli persi ed è considerato poco affidabile. Il suo punto di forza è la rapidità e l'efficienza della comunicazione per applicazioni leggere. Ciò è dovuto al fatto che UDP non tiene conto dello stato della connessione e ha meno informazioni da memorizzare e questo permette di avere un header piccolo.

Il protocollo **TCP**, d'altro canto, a causa della sua complessità e dell'eccessivo overhead non è particolarmente utilizzato in quanto non soddisfa le specifiche energetiche per le reti di sensori. TCP è un protocollo orientato alla connessione, ciò vuol dire che prima di trasmettere deve stabilire una comunicazione tra mittente e destinatario. Esso è più affidabile dell'UDP in quanto garantisce la consegna dei messaggi alla destinazione attraverso il meccanismo degli ACK. Offre inoltre un meccanismo di ritrasmissione dei pacchetti persi. Il suo header è abbastanza grande in quanto mantiene informazioni relative alla connessione.

Inoltre è possibile adottare il protocollo **ICMPv6** per la distribuzione di messaggi di controllo attraverso la rete. Questo protocollo è l'implementazione dell'Internet Control Message Protocol per IPv6. Permette di utilizzare delle funzioni di diagnostica ed effettua la segnalazione degli errori.

Livello applicativo

A causa delle limitate capacità energetiche e di calcolo dei dispositivi utilizzati nelle reti di sensori e del basso payload richiesto dal protocollo 6LoWPAN, non è possibile utilizzare protocolli di livello applicativo come HTTP. Questo protocollo creato molti anni fa, è troppo complesso per le reti di sensori in quanto utilizza molte risorse di rete ed è difficilmente comprimibile. Questo protocollo inoltre utilizza TCP a livello trasporto che come è stato spiegato non è adatto per reti di sensori.

COAP è un protocollo che fornisce lo stesso insieme di servizi base offerti da HTTP utilizzando però un quantitativo limitato di risorse. Uno dei punti focali di COAP è l'utilizzo di UDP come protocollo di livello trasporto (HTTP utilizza TCP) che, come è stato spiegato precedente, può essere

compresso dall'Adaptation Layer. Questo protocollo comunica con HTTP, infatti, la traduzione da COAP ad HTTP avviene senza dover implementare funzioni di codifica specifiche.

MQTT è un protocollo *publish/subscribe* progettato da IBM per dispositivi con capacità limitate utilizzati in applicazioni di telemetria. L'architettura del protocollo richiede la presenza di un broker che ha il compito di distribuire i messaggi ai client interessati basandosi sull'argomento del messaggio. Il punto di forza del protocollo deriva dal fatto che la complessità del sistema si trova sul broker, mentre l'implementazione della parte client è molto semplice e leggera. MQTT supporta tre livelli di Quality of Service (QoS), che dipendono dall'affidabilità con la quale i messaggi vengono consegnati ai client.

1. QoS 0: è il più semplice, offre un servizio di consegna best effort. Ciò vuol dire che i messaggi possono o non possono essere consegnati ai client e non è prevista nessun procedura di ritrasmissione o conferma della ricezione.

2. QoS 1: i messaggi inviati vengono ritrasmessi fino a quando non si riceve un messaggio a conferma della ricezione. In questo modo si ha la sicurezza che i messaggi vengono ricevuti correttamente. Si possono avere, però, dei messaggi ridondanti a causa delle ritrasmissioni.

3. QoS 2: questo livello non solo assicura la corretta ricezione dei messaggi da parte dei client ma elimina i messaggi ridondanti. Quindi ai riceventi verrà consegnato soltanto un messaggio.

MQTT è un protocollo orientato alla connessione nel senso che necessita di un client per configurare la connessione col broker prima di poter scambiare messaggi di pubblicazione o sottoscrizione.

Anche se il protocollo MQTT è stato sviluppato per l'utilizzo in reti con limitata capacità di banda, l'adozione di TCP come protocollo di livello

trasporto lo rende inefficiente per l'utilizzo in reti 6LoWPAN. Per soddisfare i requisiti della limitata dimensione dei frame (802.15.4) e delle limitate capacità dei dispositivi utilizzati nelle reti di sensori, è stato creato MQTT-S.

Il protocollo **MQTT-S** prevede una procedura di scoperta del gateway non presente in MQTT, che può essere evitata se si conosce a priori l'indirizzo del gateway. I client si connettono al gateway attraverso un messaggio CONNECT ottenendo in risposta un ACK. La connessione termina attraverso l'invio di un messaggio DISCONNECT. I client possono connettersi a più di un gateway e quindi a più broker, e questo permette di utilizzare tecniche di bilanciamento del carico. MQTT-S include un meccanismo di registrazione al topic attraverso un messaggio REGISTER. Questo permette di risparmiare energia disaccoppiando la registrazione ad un topic dall'effettiva pubblicazione dei messaggi. I client possono pubblicare i loro dati utilizzando un messaggio di tipo PUBLISH in cui sono contenuti i dati, l'identificativo del topic e informazioni relative alla QoS. Per sottoscriversi ad un topic i client utilizzano un messaggio di tipo SUBSCRIBE contenente il nome del topic ricevendo in risposta un SUBACK che include l'identificativo del topic. Per rimuovere la sottoscrizione, infine, i client utilizzano un messaggio UNSUBSCRIBE.

2.3 RPL

RPL è un protocollo di routing progettato per le reti di sensori che utilizzano IPv6. I dispositivi di rete che implementano il protocollo sono connessi tra di loro in modo da non formare nessun ciclo. A questo scopo si costruisce un *Destination Oriented Acyclic Graph* (*DODAG*) che punta verso una destinazione specifica. Essa prende il nome di *DODAG root*.

Il grafo viene costruito attraverso l'utilizzo di una *Funzione Obiettivo* (*OF*) che definisce come i vincoli di routing ed altre informazioni vengono prese in considerazione al momento della costruzione della topologia. Il grafo così costruito è una topologia logica che soddisfa dei vincoli specifici creata

su una rete fisica già esistente. L'amministratore della rete può prevedere varie topologie che rispettano criteri diversi in modo da diversificare il comportamento dei vari dispositivi presenti. Ogni topologia viene chiamata *RPL Instance*.

Il protocollo prova ad evitare cicli stabilendo la posizione di ogni nodo rispetto alla "distanza" dalla radice del grafo. Questa posizione viene chiamata Rank e cresce con l'aumentare della distanza del nodo dalla radice, viceversa diminuisce. Il Rank può essere uguale al numero di hop che separano il gateway dal nodo corrente o può essere calcolato considerando altre tipologie di vincoli (come l'RSSI).

RPL prevede un insieme di messaggi di controllo ICMPv6 che permettono la distribuzione delle informazioni relative al grafo in tutta la rete. Questi messaggi sono: *DODAG Information Solicitation (DIS)*, *DODAG Information Object (DIO)* e *DODAG Destination Advertisement Object (DAO)*.

- DIS: questo messaggio è utilizzato dai nodi per sollecitare l'invio dei messaggi contenenti le informazioni sul grafo da parte dei vicini.

- DIO: è la fonte principale delle informazioni che sono necessarie durante la costruzione della topologia. Questo messaggio permette a un nodo di scoprire in quale RPL Instance si trova andandola ad archiviare in uno dei suo campi. Include anche la versione del grafo, il DODAG ID ovvero l'indirizzo della radice e il Rank del nodo mittente. Un'altra informazione presente nei messaggi DIO è quella relativa al tipo di routing verso il basso stabilito per il DODAG corrente.

- DAO: è utilizzato da RPL per propagare le informazioni relative al routing. Questi messaggi contengono, tra i vari campi presenti, l'ID dell'RPL Instance e l'ID del DODAG che corrisponde all'indirizzo IPv6 della radice.

- DAO-ACK: messaggio di risposta ad un DAO.

2.3.1 Costruzione del grafo

RPL prevede tre tipologie di nodi:

- *Gateway*: identificato con la radice del grafo, fornisce la connettività verso altre reti;
- *Router*: nodi che hanno il compito di distribuire le informazioni sulla topologia ai vicini;
- *Leaf*: possono unirsi ad un grafo esistente. Non possono inviare messaggi DIO.

La procedura di costruzione del grafo comincia dalla DODAG root la quale invia ai nodi vicini (range di comunicazione fisica) un messaggio DIO contenente il suo Rank. Ogni nodo potrà ricevere un numero elevato di messaggi DIO dai vicini ma sceglierà il padre al quale collegarsi attraverso un algoritmo. La scelta si basa sulla metrica utilizzata e sui vincoli stabiliti dalla funzione obiettivo. Dopodichè ognuno di essi calcola il suo Rank e, nel caso il nodo sia un router, aggiorna i valori contenuti nel messaggio DIO e lo inoltra a tutti i suoi vicini. Questa operazione viene ripetuta da tutti i nodi nelle vicinanze e termina quando si raggiunge un nodo foglia o quando non rimane nessun nodo nel range di comunicazione.

Per evitare la formazioni di cammini ciclici all'interno del grafo, RPL fornisce un meccanismo di rilevazione dei cicli che impedisce ai nodi del grafo di accettare messaggi DIO dai nodi del loro sottografo. Nel momento in cui un nodo perde il collegamento al padre a causa di interferenze o altri fattori, esso deve ricominciare da zero la procedura di registrazione al grafo.

2.3.2 Point-to-multipoint

RPL supporta le comunicazioni *point-to-multipoint* (*P2MP*) che permettono l'invio di messaggi dalla radice a tutti i nodi del suo sottografo. Questo permette agli amministratori della rete di controllare all'occorrenza lo stato dei nodi, in modo da rilevare eventuali malfunzionamenti. Il protocollo

definisce due modalità a supporto del P2MP: *storing mode* e *non-storing mode*.

Non-storing mode

In questa modalità ogni nodo della rete invia un messaggio DAO in unicast verso la radice. I nodi intermedi non archiviano il contenuto del messaggio, ma aggiungono semplicemente il proprio indirizzo al DAO ricevuto. Una volta fatto ciò, inoltrano il messaggio al proprio padre fino alla radice. Quando la radice deve richiedere qualche informazione ai nodi del sottografo crea un source route header che verrà utilizzato per effettuare il routing e inviare correttamente il messaggio a destinazione. Se un qualsiasi nodo vuole inviare un informazione ad un altro inoltra il messaggio alla radice la quale si occuperà della consegna.

In questa modalità i nodi della rete non mantengono tabelle di routing, diminuendo così la quantità di memoria richiesta ad ogni nodo. Solo il gateway ha una tabella di routing contenete tutti i cammini del suo sottografo. Ovviamente in questo caso la dimensione dei messaggi che partiranno dalla radice e raggiungeranno gli altri nodi avranno dimensioni maggiori, dato che l'header di ciascun messaggio conterrà gli indirizzi dei nodi che separano la sorgente dalla destinazione.

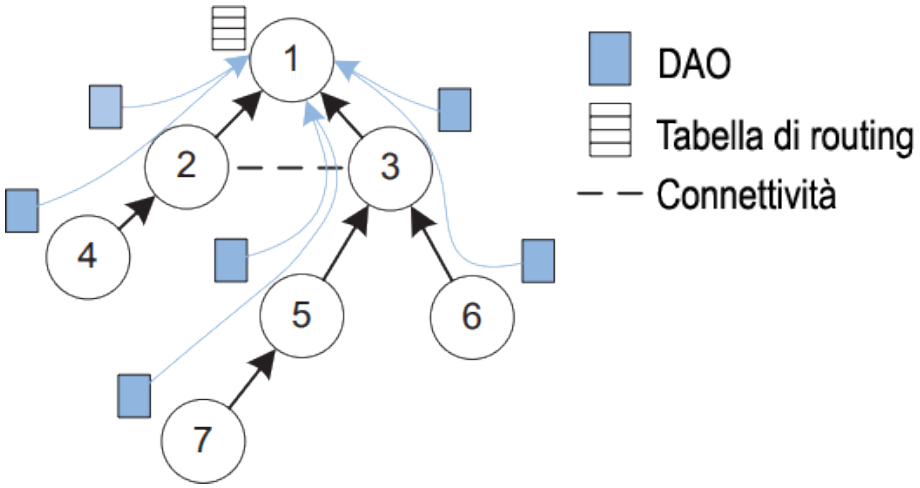

Figura 2.3: Modalità non-storing.

Storing mode

Nella modalità storing il messaggio DAO viene inviato dal figlio al padre, che archivia il contenuto del messaggio ricevuto. Dopodiché il padre crea un nuovo messaggio DAO che conterrà le informazioni relative al suo sottografo e lo invia a suo volta al padre. Ogni nodo memorizza le informazioni relative al figlio, non necessariamente diretto, e al primo hop per raggiungerlo.

In questa modalità quando la radice deve inviare dei messaggi ai nodi del sottografo, sceglie il figlio più vicino alla destinazione consultando le informazioni contenute nella sua tabella di routing. Il nodo che riceve questo messaggio a sua volta consulterà le informazioni presenti nella sua tabella di routing per scegliere il prossimo hop, e così via. Quando un nodo che non sia la radice deve inviare un messaggio ad un altro nodo, passa l'informazione al padre che controlla se il nodo destinazione si trova nel suo sottografo. Se la destinazione è presente il messaggio verrà inoltrato grazie alle informazioni contenute nella tabella di routing. Altrimenti, il messaggio verrà inoltrato nuovamente al padre che effettuerà la stessa procedura.

Nella modalità storing tutti i nodi della rete mantengono una tabella di routing contenente tutti i percorsi dei nodi del suo sottografo. Il carico di

lavoro in questo modo è distribuito in tutti i nodi del grafo.

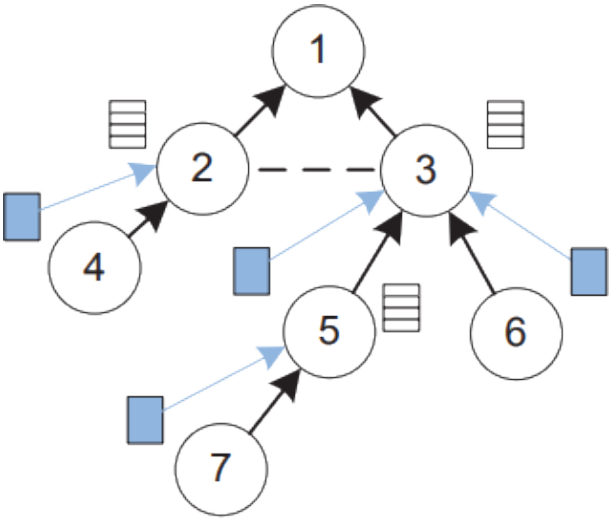

Figura 2.4: Modalità storing.

2.4 Contiki

Contiki è un sistema operativo open source scritto in linguaggio C, altamente portabile, leggero e multitasking. Questo sistema è progettato per l'utilizzo in sistemi embedded con capacità energetiche e di memoria limitate. Contiki supporta molti tra gli standard presenti in letterature come IPv4 e IPv6, oltre ai più recenti standard sviluppati per le comunicazioni wireless a bassa potenza come 6LoWPAN, RPL, CoAP e MQTT.

Consiste in un *kernel event-driven* in cui le applicazioni sono caricate dinamicamente a runtime. I processi di Contiki utilizzano dei *protothreads* che permettono una programmazione lineare simile a quella con i *thread*, sul kernel event-driven. Contiki fornisce due stack di comunicazione: *uIP* e *Rime*. uIP implementa lo stack TCP/IP e rende possibile la comunicazione delle applicazioni sviluppate con Contiki su Internet. Rime è uno stack di comunicazione creato per componenti radio a bassa potenza e fornisce un ampio insieme di primitive di comunicazione.

2.4.1 Architettura

Questo sistema operativo è basato su un'architettura modulare. A livello kernel segue un modello event driven, ma fornisce ai processi la possibilità di funzionare come i thread. Il kernel prevede un modulo per schedulare gli eventi che si occupa di inviarli ai processi in esecuzione. L'esecuzione dei processi è innescata dall'invio di un evento da parte del kernel o da un meccanismo di polling degli eventi. Ogni evento che viene schedulato, ad un certo punto, terminerà la sua esecuzione ma esiste anche un meccanismo di interrupt che permette l'alternanza dei processi.

Contiki supporta eventi sincroni e asincroni. Quelli sincroni sono inviati immediatamente al processo permettendo la sua esecuzione. Gli eventi asincroni vengono inseriti in una coda e inviati quando possibile al processo.

2.4.2 Servizi

In Contiki un servizio è un processo che implementa delle funzionalità che possono essere utilizzate da altri processi. Può essere visto come una specie di libreria condivisa. I servizi si possono dinamicamente rimpiazzare a runtime e devono essere linkati dinamicamente. Alcuni esempi di servizi sono: lo stack del protocollo di comunicazione, i driver dei sensori e così via.

2.4.3 Supporto alla comunicazione

La comunicazione in Contiki è implementata come un servizio in modo da essere rimpiazzata all'occorrenza a runtime. Questo permette anche di poter caricare contemporaneamente più protocolli di comunicazione. I servizi di comunicazione utilizzano il meccanismo fornito da Contiki per tutte le tipologie di servizi per collegarsi ad altri e gli eventi sincroni per comunicare con le applicazioni.

2.4.4 Multi-threading

Il multi-threading con prelazione è implementato da una libreria che viene eseguita sul kernel. Essa può essere linkata con l'applicazione che vuole operare con il modello multi-thread. Questa libreria è divisa in due parti: una parte indipendente dalla piattaforma che si interfaccia con gli eventi schedulati dal kernel, e una parte specifica per ogni piattaforma che implementa più stack, il passaggio tra di essi e varie primitive.

2.4.5 Processi

Contiki condivide l'hardware tra le varie applicazioni attraverso processi e protothread. Un processo è una funzione in C, che generalmente contiene un loop infinito e alcune chiamate bloccanti. Dato che il kernel di Contiki è non preemptive, ogni processo quando comincia la sua esecuzione continuerà ad essere eseguito fin quando non verrà bloccato da un evento. Il sistema definisce varie chiamate bloccanti che permettono di modificare il flusso di controllo. Il meccanismo utilizzato per gestire i processi utilizza la libreria dei protothread. I protothread sono dei thread che non possiedono uno stack dedicato, progettati per dispositivi che hanno dei vincoli di memoria.

Un processo in Contiki è formato da due parti: un *process control block* e un *process thread*. La prima parte contiene le informazioni su ogni processo come lo stato, un puntatore al process thread e una stringa contenente il nome del processo. La seconda parte contiene il codice del processo. Un protothread è dichiarato utilizzando la macro *PROCESS_THREAD(name, ev, data)*, in cui name indica il nome della struttura processo, il secondo campo identifica l'evento inviato al thread durante l'invocazione mentre il terzo parametro è un puntatore ai dati passati al thread. Questa macro dichiara semplicemente l'inizio del process thread.

Ogni processo deve cominciare con la macro *PROCESS_BEGIN()* e terminare con *PROCESS_END()*.

2.5 Componenti hardware

In questa sezione verranno descritte le componenti utilizzate per l'implementazione fisica della rete. I dispositivi in questione sono prodotti dall'azienda STMicroelectronics che ha fornito l'hardware necessario per la comunicazione e l'elaborazione dei dati.

Nucleo IDS01A4 e SPIRIT1

SPIRIT1 è un radiotrasmettitore a bassa potenza, progettato per applicazioni wireless che utilizzano la banda sub-GHz. Opera sia nelle bande ISM che in quelle SRD con frequenze che vanno dai 169 MHz ai 915 MHz. Può essere anche programmato per operare con altre frequenze nelle bande: 300 MHz - 348 MHz, 387 MHz - 470 MHz, e 779 MHz - 956 MHz. La velocità di trasmissione dati è programmabile e può andare da un minimo di 1 kbps fino ad un massimo di 500 kbps. SPIRIT1 integra un modem baseband configurabile, che supporta la modulazione, la demodulazione e la gestione dei dati.

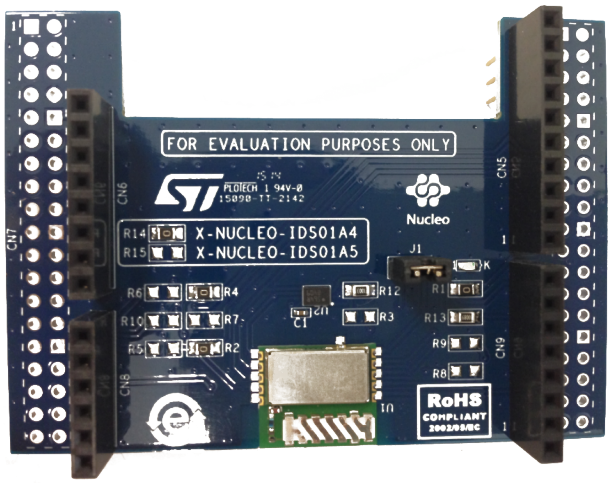

Figura 2.5: Modulo radio IDS01A4 (SPIRIT1).

Funzionalità:

- Schemi di modulazione: 2-FSK, GFSK, MSK, GMSK, OOK, ASK;
- Velocità di trasmissione dati da 1 a 500 kbps;
- Sensore di temperatura integrato;
- Indicatore della batteria e rilevatore di batteria scarica;
- Buffer di ricezione e trasmissione (96 byte ciascuno);
- Configurabile attraverso l'interfaccia SPI;
- Co-processore per la cifratura AES 128-bit;
- Potenza trasmissiva programmabile, da -31 a 11.6 dBm;
- Wake-up tramite timer interno o wake-up attraverso evento esterno;

La X-NUCLEO-IDS01A4 è un'Evaluation Board prodotta da ST basata sul modulo SPIRIT1 SPSGRF-868 che opera nella banda degli 868 MHz ISM. Dato che in commercio esistono due tipologie di IDS01Ax, l'identificazione della frequenza utilizzata dalle due board avviene per mezzo di due resistori (R14 e R15). I connettori della board sono compatibili con la configurazione Arduino R3 e MORPHO. Questa board si interfaccia con il microcontrollore STM32 attraverso la connessione SPI e i pin GPIO. L'utente può effettuare delle modifiche al GPIO montando o rimuovendo i resistori.

STM32F401RE e NUCLEOSTM32

STM32F401RE è dotato di un processore ARM Cortex-M4 con frequenza di clock massima pari a 84 MHz, 512 kByte di memoria Flash, 96 kByte di RAM. Presenta tre tipologie di oscillatore interno: uno a bassa frequenza (16 kHz), uno a media frequenza regolabile a 7 diverse frequenze e uno ad alta frequenza (37 MHz). All'interno ha un modulo RTC che utilizza quattro registri per indicare la data (giorno, giorno della settimana, mese e anno) e cinque registri che indicano l'orario (AM o PM, ora, minuto, secondo, millisecondo). Include vari registri di backup che rimangono disponibili durante la

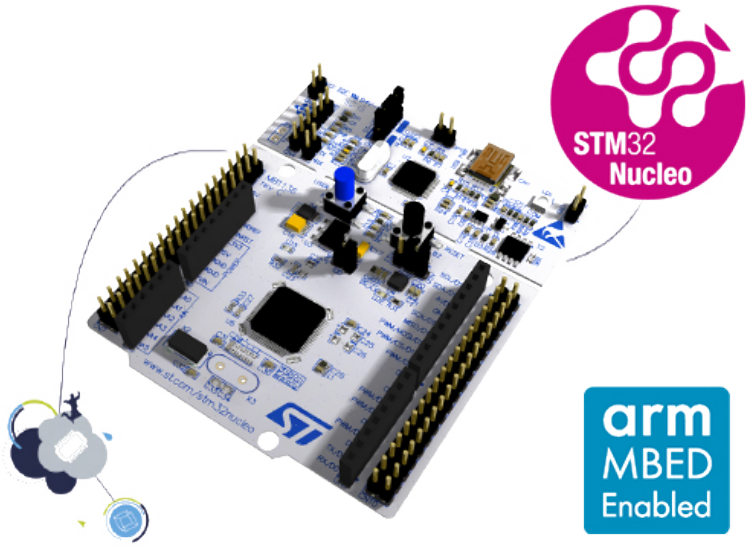

Figura 2.6: Nucleo board STM32F401RE.

fase di Standby. Fornisce un insieme di modalità per il risparmio energetico che permettono la progettazione di applicazioni low-power:

***Sleep*:** la CPU sospende la sua attività mentre la altre periferiche continuano la loro esecuzione. Sono loro a risvegliare la CPU attraverso interrupt o eventi;

***Stop*:** in questa modalità si raggiunge il minimo consumo energetico mantenendo il contenuto dei registri e della RAM. Tutti i clock principali vengono stoppati, a parte qualche eccezione. Viene utilizzato il regolatore del voltaggio impostato in modalità low-power. Il microcontrollore può essere risvegliato da un interrupt esterno, da un evento o da un interrupt generato dall'RTC.

***Standby*:** attraverso questa modalità è possibile ottenere il consumo energetico minimo. Stavolta i dati nei registri e nella RAM vengono persi. Gli unici registri che vengono mantenuti sono quelli relativi alla circuiteria di standby. Il microcontrollore può essere riattivato dagli stessi interrupt utilizzabili nella modalità stop.

X-NUCLEO-IDW01M1

La X-NUCLEO-IDW01M1 è un'evaluation board per il Wi-Fi sviluppata da ST, utilizzabile con le Nucleo boards STM32. Il modulo SPWF01SA ha un microcontrollore integrato, un modulo Wi-Fi b/g/n low-power con un gestore e amplificatore di potenza e un'antenna SMD. Possiede una memoria flash esterna di 1 MB. Il firmware fornisce uno stack IP completo in grado di aprire fino ad 8 socket TCP/UP. Il modulo può comportarsi contemporaneamente come socket server e client. Si interfaccia con il microcontrollore sulla nucleo board attraverso la porta seriale UART. X-NUCLEO-IDW01M1 è compatibile con la configurazione dei connettori Arduino UNO R3 e Morpho.

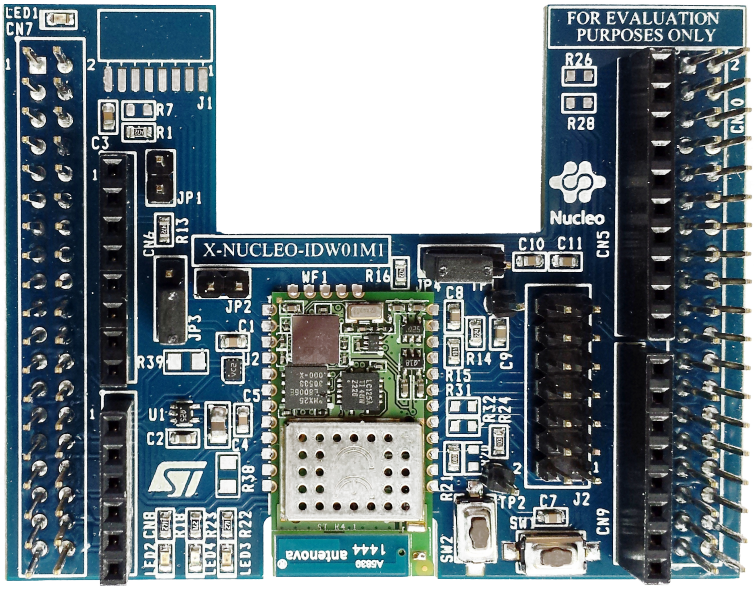

Figura 2.7: Modulo radio IDW01M1 (WiFi).

Capitolo 3

Implementazione

In questa sezione verranno descritte le scelte effettuate durante la fase di implementazione. Il dispositivo gateway che verrà implementato dovrà raccogliere i dati provenienti dalla rete 6LoWPAN utilizzando la tecnologia SPIRIT1 per la comunicazione. Successivamente il nodo comunicherà questi dati alla rete esterna attraverso il protocollo MQTT utilizzando la tecnologia radio 802.11. La rete sarà formata da vari dispositivi dotati di sensori e da un gateway che immagazzinerà le informazioni ricevute da ciascuno di essi e le invierà ad un broker MQTT rendendoli disponibili in remoto. Questo sistema verrà installato in un laboratorio di ricerca per monitorare la temperatura e l'umidità dei vari ambienti. Un nodo verrà posto nella sala server, dove i livelli di temperatura non devono superare una determinata soglia.

Questo progetto si è focalizzato sulla costruzione del nodo gateway, l'elemento principale della rete. Il gateway comunica con i nodi della sottorete attraverso messaggi. Il suo compito è quello di tenere aggiornati i valori ottenuti dai sensori grazie ad un meccanismo di richiesta delle informazioni. Per fare ciò utilizza una tipologia particolare di messaggio. Periodicamente inoltre, questo dispositivo crea un messaggio contenente le informazioni sull'ambiente e le invia ad un broker per la visualizzazione remota dei dati.

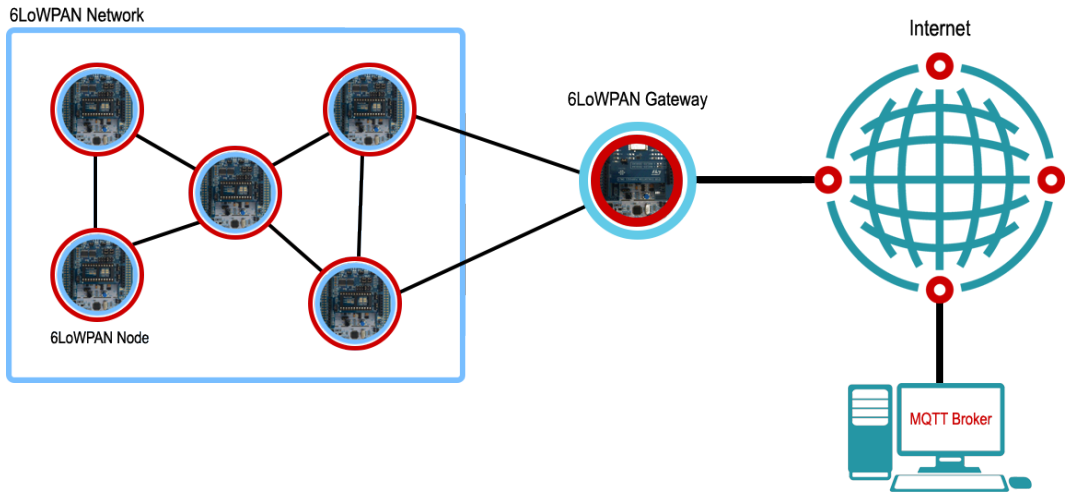

Figura 3.1: Rete progettata.

3.1 Installazione e configurazione di Contiki

Descrizione dell'IDE

Durante la fase di implementazione mi sono servito dell'ambiente di sviluppo System Workbench for STM32. Si tratta di un ambiente di sviluppo software multi piattaforma basato su Eclipse. Supporta un vasto insieme di board e microcontrollori di ST. Le componenti chiave di questo ambiente sono:

- Supporto per microcontrollori STM32, Nucleo boards STM32, evaluation boards, firmware STM32 (Standard Peripheral library o STM32Cube HAL).
- compilatore GCC C/C++;
- Debugger basato su GDB;
- Eclipse IDE;
- Compatibile con plugin di Eclipse;

- Supporto a ST-LINK;
- Nessun limite sulla dimensione del codice;
- Supporto di vari SO.

Ottenere Contiki

Il sistema operativo Contiki è ottenibile in tre modi:

- ***Instant Contiki*:** è un ambiente di sviluppo completo disponibile con un singolo download. Si basa su una macchina virtuale Ubuntu Linux eseguita su un player VMWare ed ha Contiki e tutti gli strumenti di sviluppo, compilatori, e simulatori utilizzati dal sistema operativo;
- ***Github*:** è possibile scaricare Contiki dal repository ufficiale su GitHub;
- **Esempi dal sito ST:** ST fornisce vari esempi di applicazioni realizzate con Contiki in cui sono già configurati i driver per varie board.

Il team *STcLAB* ha effettuato un porting di Contiki OS sulla piattaforma STM32-F401RE e l'expansion board X-NUCLEO-IDS01A4. Come librerie sono state utilizzate quelle per il modulo radio SPIRIT1 e le *HAL* (*Hardware Abstraction Layer*), le quali forniscono delle astrazioni per l'accesso all'hardware. Queste librerie supportano diversi microcontrollori di una stessa famiglia unitamente a varie piattaforme compatibili.

Come esempi di partenza sono state utilizzate le applicazioni *UDP-receiver* e *UDP-sender* sviluppate da ST per Contiki OS. Questi esempi forniscono la comunicazione di base tra due nodi all'interno di una rete 6LoWPAN utilizzando SPIRIT1.

Dopo aver scaricato l'esempio ed averlo importato all'interno dell'IDE, ho avuto modo di analizzare la struttura del progetto andando a studiare le varie cartelle da cui era composto per una migliore comprensione.

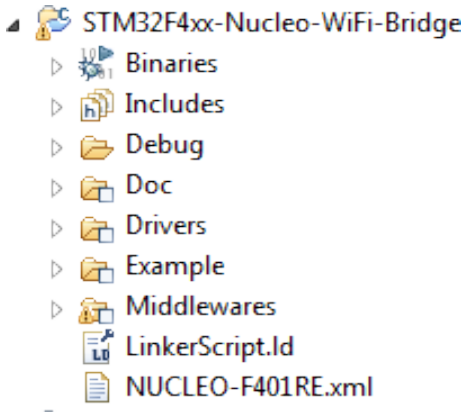

Figura 3.2: Struttura progetto Contiki.

Binaries: in questa sezione si trovano i file .elf creati dopo la compilazione del progetto, i quali costituiscono l'eseguibile che verrà installato sul dispositivo fisico;

Includes: qui si trovano tutte le librerie necessarie alla corretta esecuzione del sistema. Sono presenti librerie per la comunicazione, quelle per applicazioni di terze parti (MQTT-PAHO), ecc;

Drivers: questa sezione raccoglie tutti i driver dei moduli hardware utilizzati dall'applicazione. Sono presenti i driver per il modulo radio SPIRIT1, per il modulo 802.11 e quelli della NUCLEO board. Qui troviamo le librerie HAL che permettono di interfacciarsi con le componenti hardware della board NUCLEO.

Middlewares: contiene i file che definiscono ed implementano i servizi utilizzati dall'applicazione.

Example: in questa sezione si trova il file *main.c* e *receiver.c*, in cui è implementata l'applicazione.

Configurazione

L'applicazione esempio fornisce un programma di partenza già configurato per la creazione di un applicativo che utilizza la tecnologia 6LoWPAN, unitamente ad RPL, per la creazione di un gateway rappresentato da una NUCLEO board (STM32-F401RE) su cui è montato il modulo radio SPIRIT1 (board IDS01A4).

```
#define POWER_DBM                11.6
#define CHANNEL_SPACE            100e3
#define FREQ_DEVIATION           127e3
#define BANDWIDTH                540.0e3
#define MODULATION_SELECT        GFSK_BT1
#define DATARATE                 64000
#define LENGTH_WIDTH             8
#define CRC_MODE                 PKT_CRC_MODE_16BITS_2
#define EN_WHITENING             S_DISABLE
#define SYNC_WORD                0x88888888
```

Figura 3.3: Parametri di configurazione SPIRIT1.

Attraverso il file *spirit1.h* contenuto in *../Middlewares/ST/Contiki_STM32_Library/Inc* è possibile configurare i parametri del modulo radio scegliendo in questo modo la modulazione da utilizzare, il datarate e altri parametri. Solo alcuni di questi sono stati ritoccati o adattati alle esigenze dell'applicazione durante la progettazione del sistema.

Il campo POWER_DBM mostra la potenza trasmissiva del modulo radio. Questo parametro, in figura impostato con il valore massimo, è stato ritoccato più volte per testare la connettività dei vari dispositivi con potenze trasmissive diverse.

Il campo DATARATE permette di impostare la velocità trasmissiva del modulo radio. Questo campo è stato modificato più volte nel corso delle valutazioni effettuate sulla tecnologia SPIRIT1. MODULATION_SELECT permette di impostare il tipo di modulazione da utilizzare. Impostato a FSK di default, è stato modificato allo scopo di utilizzare datarate elevati.

Un altro passaggio nella configurazione dei dispositivi consiste nello scegliere quali driver dovranno utilizzare per realizzare la comunicazione, il Radio Duty Cycle, ecc. I driver utilizzati per l'RDC sono compatibili con qualsiasi dispositivo radio ma per contro non permettono lo sleep del dispositivo.

```
#define NETSTACK_CONF_FRAMER  framer_802154
#define NETSTACK_CONF_NETWORK sicslowpan_driver
#define NETSTACK_CONF_RDC nullrdc_driver
#define NETSTACK_CONF_MAC csma_driver
#define NETSTACK_CONF_RADIO spirit_radio_driver
```

Figura 3.4: Configurazione Contiki (Driver).

Infine all'interno dei file *rpl.h*, *rpl-private.h* sono presenti i parametri per la configurazione del comportamento di RPL.

```
#define RPL_OF_OCP RPL_OCP_MRHOF
#define RPL_LEAF_ONLY 0
#define RPL_WITH_DAO_ACK 0
#define RPL_CONF_MOP RPL_MOP_STORING_NO_MULTICAST
#define RPL_DIS_SEND 1
#define RPL_MRHOF_SQUARED_ETX 1
```

Figura 3.5: Parametri di configurazione RPL.

Ovviamente per il corretto funzionamento del protocollo sono necessari molti più parametri di configurazione, in figura, sono mostrati quelli più critici/importanti per l'applicazione.

RPL_OF_OCP: definisce la funzione obiettivo da utilizzare. Contiki fornisce due tipologie di OF: hop e ETX. In questo caso si utilizza ETX;

RPL_LEAF_ONLY: disabilita la modalità leaf. Quindi tutti i nodi della rete saranno router o gateway;

RPL_WITH_DAO_ACK: disabilita i messaggi DAO-ACK;

RPL_MOP_DEFAULT: definisce la modalità utilizzata per le downward route. Si hanno tre opzioni: modalità non-storing, modalità storing con multicast e modalità storing senza multicast. In questa applicazione viene utilizzata l'ultima;

RPL_DIS_SEND: abilita l'utilizzo dei messaggi DIS;

RPL_MRHOF_SQUARED_ETX: una versione della funzione obiettivo ETX che crea i cammini padre/figlio basandosi più sulla qualità del collegamento che sul numero di hop.

3.2 Implementazione main.c

In questa sezione verrà illustrato il file *main.c* in cui vengono eseguite le inizializzazioni per il corretto funzionamento della board e dei protocolli di comunicazione. Il corpo del main si occupa di eseguire le istruzioni relative all'inizializzazione del meccanismo di interrupt, alla configurazione del clock di sistema, all'inizializzazione dei bottoni e dei led presenti sulle board NUCLEO, l'inizializzazione del modulo RTC (clock) e dello stack 6LoWPAN.

Il metodo *Stack_6LoWPAN_Init()* si occupa dell'impostazione dei parametri per l'utilizzo del protocollo 6LoWPAN.

All'interno di questo metodo troviamo varie istruzioni:

***set_rime_addr()*:** imposta l'indirizzo MAC del dispositivo e lo assegna ad una locazione di memoria disponibile globalmente;

***netstack_init()*:** questo metodo inizializza i driver radio, MAC, quelli per il RDC, per la rete e per la tipologia di frame da utilizzare a livello datalink;

***spirit_radio_driver.on()*:** con questa istruzione viene attivato il modulo radio;

In questo frammento di codice troviamo l'inizializzazione di *queuebuf*, una coda utilizzata per archiviare temporaneamente pacchetti ricevuti ma

```
  set_rime_addr();
  netstack_init();
  spirit_radio_driver.on();

#if NETSTACK_CONF_WITH_IPV6
  memcpy(&uip_lladdr.addr, node_mac, sizeof(uip_lladdr.addr));

  queuebuf_init();
  process_start(&tcpip_process, NULL);

  uip_ipaddr_t ipaddr;
  uip_ip6addr(&ipaddr, 0xfe80, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
#endif /* NETSTACK_CONF_WITH_IPV6*/
```

Figura 3.6: Inizializzazione stack 6LoWPAN.

non ancora processati. L'istruzione successiva fa partire il processo TCP che inizializza il modulo *uip* il quale permette di utilizzare IPv6 su Contiki, e RPL solo se il progetto è configurato per il suo utilizzo.

Una volta fatto questo viene deciso il prefisso IPv6 che ogni nodo della rete dovrà avere; viene scelto il prefisso fe80 (link local). Il prefisso occuperà i primi 64 bit dell'indirizzo IPv6.

La restante parte ovvero gli altri 64 bit saranno occupati dall'indirizzo MAC del dispositivo. In questo modo si elimina la necessità di avere un meccanismo per il rilevamento di indirizzi duplicati.

Dopo l'inizializzazione dello stack 6LoWPAN, il main continua con un ciclo in cui è contenuta l'istruzione *process_run()*. Questo metodo effettua la procedura di polling dei processi e la generazione degli eventi.

3.3 Implementazione gateway

In questa sezione verrà mostrata l'implementazione del comportamento del gateway. Questa applicazione è formata da due processi: processo *receiver*

e processo *sender*.

Il file *message.h* definisce la struttura dei messaggi scambiati dai nodi della rete 6LoWPAN.

Struttura dei messaggi

```
typedef enum {ROOT_GET, ROOT_GET_ALL, SET_VALUE, UPDATE } tMessage_t;

typedef struct sensor_value{
    int32_t decPart;
    int32_t intPart;
}sensor_value_t;

typedef struct our_message {
    int message_id;
    int hop_count;
    tMessage_t message_type;
    uip_ipaddr_t final_ip_destination;
    sensor_value_t temperature;
    sensor_value_t humidity;
    uip_ipaddr_t original_ip_address;
    clock_time_t timestamp;
} our_message_t;
```

Figura 3.7: Definizione dei messaggi utilizzati all'interno della rete 6LoWPAN.

La struttura, come si può vedere nell'immagine, è composta da vari campi:

message_id: un indice incrementale che indica il numero di messaggi generati da un nodo;

hop_count: indica a quale distanza si trova il nodo mittente;

message_type: tipologia del messaggio, viene utilizzato per stabilire il comportamento dei nodi alla sua ricezione (ROOT_GET, ROOT_GET_ALL, SET_VALUE, UPDATE);

temperature: fornisce il valore della temperatura rilevato dal sensore, è formato da una parte decimale e da una intera;

humidity: fornisce il valore dell'umidità, ha una parte intera e una decimale;

original_ip_address: contiene l'indirizzo del nodo che ha generato il messaggio (es. nel caso di ROOT_GET_ALL questo indirizzo è rappresentato da quello del gateway);

final_ip_destination: contiene l'indirizzo del nodo finale (es. nel caso di ROOT_GET_ALL questo campo è vuoto dato che il messaggio che parte dalla radice raggiunge tutti i nodi del grafo).

Processo receiver

I punti chiave di questo processo sono due: la composizione di un indirizzo IPv6 globale che sarà utilizzato durante il routing e la composizione del grafo, e l'istruzione utilizzata dal gateway per far cominciare la procedura di creazione della rete LoWPAN attraverso il protocollo RPL.

La composizione dell'indirizzo IPv6 avviene tramite l'utilizzo delle stesse funzioni viste in precedenza nella parte riguardante il main. L'unica eccezione riguarda la tipologia di prefisso scelto che questa volta è rappresentato dal prefisso ULA (Unique Local Address). Esso permette di effettuare il routing all'interno della rete senza però fornire il supporto al routing verso la rete esterna (Internet).

```
uip_ip6addr(&ipaddr, UIP_DS6_DEFAULT_PREFIX, 0, 0, 0, 0, 0, 0, 0);
uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
```

Figura 3.8: Assegnamento dell'indirizzo ULA.

Con il metodo *create_rpl_dag()* il processo imposta il gateway come radice del grafo assegnando il suo indirizzo a quello della root e scegliendo l'RPL Instance. Infine, imposta il prefisso che ogni nodo della rete dovrà avere. Esso sarà contenuto nei messaggi DIO utilizzati nella composizione del grafo.

```
struct uip_ds6_addr *root_if;

root_if = uip_ds6_addr_lookup(ipaddr);
if(root_if != NULL) {
  rpl_dag_t *dag;
  uip_ipaddr_t prefix;

  rpl_set_root(RPL_DEFAULT_INSTANCE, ipaddr);
  dag = rpl_get_any_dag();
  uip_ip6addr(&prefix, UIP_DS6_DEFAULT_PREFIX, 0, 0, 0, 0, 0, 0, 0);
  rpl_set_prefix(dag, &prefix, 64);
```

Figura 3.9: Creazione del DODAG (RPL).

Processo sender

Questo processo ha il compito di inizializzare la connessione UDP e inviare messaggi ai nodi della sottorete per effettuare il poll dei dati dei sensori.

L'applicazione utilizza UDP come protocollo di livello trasporto. La connessione viene creata attraverso un metodo che consente di stabilire le porte e definire una funzione di callback. Questa funzione viene richiamata ogni volta che la connessione UDP rileva l'arrivo di un messaggio. All'interno di questo metodo è presente una struttura progettata per archiviare i messaggi ricevuti che verranno processati successivamente nel corpo del processo.

```
simple_udp_register(&unicast_connection, UDP_PORT_SEND, NULL, UDP_PORT_SEND, receiver);
```

Figura 3.10: Registrazione della connessione UDP.

Il corpo del processo è formato da un'istruzione while ripetuta all'infinito in cui vengono eseguite varie istruzioni. A causa della natura del kernel di Contiki è necessario utilizzare un meccanismo per far si che il processo in questione non monopolizzi tutte le risorse. Per fare ciò si utilizzano dei timer unitamente a dei metodi forniti opportunamente dalla libreria che implementa il multi-thread.

Ogni 30 secondi (questo valore può essere modificato in base alle necessità dell'applicazione) viene generato un messaggio di tipo ROOT_GET_ALL (tipo 1) e viene posto in una coda. Questo passaggio è necessario poiché ci possono essere altri messaggi da processare prima. In questo modo ci si assicura che tutti i messaggi verranno processati in ordine. Quando si estrae dalla coda un messaggio di tipo 1, il gateway grazie alle informazioni contenute nella sua tabella di routing, invia il messaggio a tutti i nodi del suo sottografo. Questo processo prevede un meccanismo di multi-hop. Il messaggio, infatti, partirà dalla radice ma ogni nodo che lo riceve avrà il compito di inoltrarlo ai nodi del suo sottografo.

```
for(r = uip_ds6_route_head(); r != NULL; r = uip_ds6_route_next(r)){
    neighbors_routes[count_neighbors] = r->ipaddr;
    count_neighbors+=1;
}

for(i = 0; i < count_neighbors; i++) {
    simple_udp_sendto(&unicast_connection,
                      &message,
                      sizeof(our_message_t),
                      &neighbors_routes[i]);
}
```

Figura 3.11: Invio dei messaggi ai nodi della rete interna.

Dall'immagine si può osservare come avviene l'invio. Tramite la struttura *uip_ds6_route_t* è possibile accedere alla tabella di routing del nodo. Nel caso specifico, la tabella della radice (gateway) conterrà le informazioni sui percorsi di tutti i nodi del suo sottografo corrispondente all'intero grafo generato da RPL.

L'invio effettivo dei messaggi avviene tramite l'istruzione *simple_udp_sendto()*. Essa prende come parametri la connessione UDP creata precedentemente, il messaggio, la sua dimensione e l'indirizzo IPv6 del nodo destinazione.

Elaborazione e visualizzazione dei dati ricevuti

Quando il gateway riceve un messaggio, come già detto, viene richiamata la funzione di callback definita dalla procedura di registrazione della connessione UDP. Al suo interno questo metodo implementa un meccanismo di archiviazione dei messaggi in una coda. Nel corpo del processo sender i messaggi vengono estratti dalla coda ed elaborati. Il gateway riceve messaggi di tipo SET_VALUE (tipo 2) contenenti le informazioni dei sensori raccolte dal nodo mittente oltre che dal suo indirizzo.

Le informazioni vengono processate e i dati archiviati in una struttura che periodicamente mostra gli ultimi valori di umidità e temperatura degli ambienti in cui sono posizionati i vari nodi della rete.

3.4 Implementazione della comunicazione MQTT

Il gateway fino a questo punto è un semplice nodo che implementa il protocollo 6LoWPAN, fungendo da sink per i nodi della rete interna. I dati raccolti da questo nodo devono però essere inviati ad un broker attraverso Internet. Tutto ciò avviene per mezzo del modulo WiFi che utilizza l'802.11 e del modulo MQTT-PAHO.

Per fare ciò si devono introdurre delle modifiche al progetto principale. Esse consistono nell'introduzione dei driver per la board IDW01M1, l'importazione delle librerie per il servizio MQTT e WiFi, l'implementazione delle procedure per l'inizializzazione di questi moduli e la connessione al broker.

I driver per la board, seguendo la struttura del progetto elencata precedentemente, vanno inseriti nella cartella Driver del progetto. Le librerie devono essere poste nella cartella Middlewares contenente tutti i servizi. L'implementazione del codice necessario al corretto funzionamento del sistema è posta nel file *main.c*.

Il funzionamento del gateway così implementato è il seguente: all'avvio del sistema il modulo WiFi permette di collegarsi alla rete Internet connettendosi ad un AP. Una volta fatto questo si potrà effettuare la connessione

al broker MQTT impostando il suo indirizzo e creando un socket per la comunicazione.

Il metodo *wifi_init(config)* si occupa della configurazione del modulo WiFi. La variabile *config* è una struttura che contiene i parametri di configurazione. C'è la possibilità di far andare in sleep il modulo. Essendo il gateway connesso alla rete elettrica si può scegliere di non utilizzare questa funzione andando ad impostare il parametro *config.power* con *wifi_active*.

L'SSID e la password necessari per la connessione all'AP sono contenuti in variabili il cui contenuto può essere definito a compile time o, eventualmente, a runtime attraverso la pressione prolungata del tasto USER (tasto blu sulla board NUCLEO). Una volta caricati, questi valori vengono salvati sulla memoria FLASH in modo da poter essere utilizzati nella successiva esecuzione. Tutto ciò avviene tramite il metodo *ConfigAPSettings()*.

Il metodo *MQTT_Init()* inizializza l'interfaccia di rete per MQTT andando a mappare le funzioni fornite dal WiFi in quelle di MQTT. Queste funzioni sono relative al socket ID e al meccanismo di scrittura/lettura sul socket. Attraverso il metodo *Config_MQTT_IBM()* si configurano i parametri per MQTT. Tra di essi troviamo l'indirizzo del broker, il nome del topic sul quale si vuole pubblicare, la QoS, ecc.

```
strcpy((char*)mqtt_ibm_setup->pub_topic, "SensorsValues");
strcpy((char*)mqtt_ibm_setup->sub_topic, "");
strcpy((char*)mqtt_ibm_setup->clientid,"d:quickstart:nucleo:");
strcat((char*)mqtt_ibm_setup->clientid,(char *)40);
mqtt_ibm_setup->qos = QOS0;
strcpy((char*)mqtt_ibm_setup->username,"");
strcpy((char*)mqtt_ibm_setup->password,"");
strcpy((char*)mqtt_ibm_setup->hostname,"192.168.43.30");
strcpy((char*)mqtt_ibm_setup->device_type,"");
strcpy((char*)mqtt_ibm_setup->org_id,"");
mqtt_ibm_setup->port = 1883;
mqtt_ibm_setup->protocol = 't';
```

Figura 3.12: Parametri connessione MQTT.

Il processo *wifi_process* effettua periodicamente un controllo su un timer. Quando il timer scade, viene richiamato il metodo *WiFi_Process()* passando come parametro la variabile *wifi_status* contenente lo stato della connessione. Il corpo del metodo è formato da un grande switch che scandisce le varie fasi della procedura di connessione attraverso la variabile status.

wifi_state_ready in questo blocco di codice avviene la connessione all'AP. Il sistema controlla che l'AP scelto sia effettivamente presente effettuando la scansione della rete. Una volta trovato chiama la funzione *wifi_connect()* passando come parametri l'SSID e la password scelti precedentemente;

```
wifi_connect(WIFITOKEN.NetworkSSID, WIFITOKEN.NetworkKey, mode)
```

Figura 3.13: Connessione all'AP.

mqtt_socket_create: qui avviene la creazione del socket per la comunicazione con il broker. Ciò avviene per mezzo del metodo *spwf_socket_create()* che necessita di vari parametri tra cui l'indirizzo del broker, la porta (1883 per MQTT) e il protocollo da utilizzare (TCP, secure TCP, ecc). Una volta fatto ciò vengono associati al socket appena creato i buffer di scrittura e lettura utilizzati da MQTT per l'invio e la ricezione dei messaggi;

```
spwf_socket_create (&n,
                    mqtt_ibm_setup.hostname,
                    mqtt_ibm_setup.port,
                    &mqtt_ibm_setup.protocol)
```

Figura 3.14: Creazine del socket per la comunicazione.

mqtt_connect: a questo punto attraverso il metodo *MQTT_Connect()* viene eseguita la connessione al broker. L'applicazione invia un pacchetto di connessione e aspetta l'ACK da parte del broker;

```
MQTTClient(&c,
           &n,
           MQTT_TIMEOUT,
           MQTT_write_buf,
           sizeof(MQTT_write_buf),
           MQTT_read_buf,
           sizeof(MQTT_read_buf));
```

Figura 3.15: Configurazione del client MQTT.

mqtt_pub: terminata la fase di connessione il gateway è ora in grado di pubblicare messaggi. Inizialmente viene creato il messaggio MQTT che contiene tra gli altri campi la QoS, il payload (rappresentato dai dati dei sensori) e la dimensione del payload. Utilizzando il metodo *MQTT_Publish()* è possibile inviare il messaggio di pubblicazione prendendo come parametri il nome del topic e il messaggio.

```
MQTT_msg.qos        = QOS0;
MQTT_msg.dup        = 0;
MQTT_msg.retained   = 1;
MQTT_msg.payload    = (char *)buffer;
MQTT_msg.payloadlen = strlen((char *)buffer);
```

Figura 3.16: Creazione del messaggio MQTT.

```
MQTTPublish(&c,(char*)mqtt_ibm_setup.pub_topic,&MQTT_msg)
```

Figura 3.17: Pubblicazione dei dati sul broker.

Durante la fase di test della comunicazione MQTT, come broker, è stato utilizzato un PC connesso alla rete locale. Il dispositivo dopo essersi collegato alla rete comunica con il PC inviando le informazioni prese precedentemente dai sensori. Attraverso l'applicazione *MQTTLens* di Chrome è possibile configurare un broker, creare topic e inviare messaggi di sottoscrizione e pubblicazione.

TeraTerm

La visualizzazione del comportamento del gateway e il debug è stato effettuata attraverso l'utilizzo del terminale seriale TeraTerm, impostato secondo questi parametri:

- baud rate: 115200;
- data: 8 bit;
- parity: none;
- stop: 1 bit;
- flow control: none.

Capitolo 4

Valutazione sperimentale

In questo capitolo verranno mostrati i risultati dei test effettuati sulla tecnologia SPIRIT1. I parametri che verranno analizzati sono il *goodput* e il *delay*.

4.1 Metriche

Goodput

Il throughput indica l'ammontare dei messaggi consegnati con successo in un certo intervallo di tempo. In questa analisi però, nel calcolo di questo indice non viene considerato l'*overhead* ma solo il payload, cioè l'effettiva dimensione del messaggio definito dall'applicazione. Per overhead si intende l'effettivo costo per la trasmissione di un messaggio che comprende, oltre al payload, il peso dell'header e di altri campi che compongono un messaggio. Quindi in questo caso si parla di goodput. L'indice è calcolato dividendo la quantità di dati inviati sul canale per il tempo necessario alla loro ricezione e viene rappresentato in KB/s (Kilobyte al secondo). Da qui la formula:

```
goodput = (pkt_size * numPkt) / delay
```

In cui *pkt_size* è la dimensione del singolo pacchetto, *numPkt* è il numero di pacchetti inviati al secondo e *delay* è il tempo necessario alla ricezione di tutti i messaggi spediti.

Delay

Il delay, o latenza, indica il tempo trascorso tra l'invio del messaggio e l'effettiva ricezione. Questo valore comprende il tempo impiegato per il processamento del messaggio. Esso è composto da eventuali ritardi dovuti a meccanismi di accodamento o altre operazioni che avvengono nel mittente e nel destinatario, più il tempo necessario per la trasmissione del segnale. Questo valore è calcolato mediante la differenza tra il tempo ottenuto subito dopo l'ultima ricezione e quello ottenuto prima del primo invio. Da qui:

```
delay = final_time - start_time
```

4.2 Risultati

4.2.1 Valutazione tecnologia SPIRIT1

Nella valutazione di SPIRIT1 sono stati utilizzati due dispositivi posti ad una distanza fissa pari a 3 metri. Dato che i clock dei dispositivi non sono sincronizzati, il calcolo dei tempi necessario per la computazione delle metriche avviene su un solo dispositivo. Per fare ciò il mittente invia periodicamente un certo numero di messaggi con payload di 70 byte (*bufsize*). La dimensione del payload è stata scelta bassa per evitare la frammentazione. Un messaggio di controllo inviato come ultimo della sequenza, scatena la risposta del destinatario che risponderà a sua volta con un messaggio della stessa tipologia. Quindi il destinatario riceve un certo numero di messaggi (5, 10, 15...*sLoad*) ma risponde solo al messaggio di controllo.

Il mittente una volta ricevuto questo messaggio, calcola il goodput attraverso la formula:

```
goodput = (sLoad * bufsize) / delay
```

Il delay si ottiene tramite la formula:

```
delay = final_time - start_time
```

in cui *start_time* è il tempo registrato subito prima dell'invio del primo messaggio della sequenza. *Final_time* è il tempo registrato alla ricezione del messaggio di controllo da parte del mittente.

Il grafico del goodput è stato costruito ponendo nelle ascisse il valore sLoad (*System Load* - numero di pacchetti generati al secondo) e nelle ordinate il valore del goodput. I test sono stati effettuati utilizzando diversi datarate. Questi risultati sono visibili nelle varie rette del grafico del goodput.

Il grafico del delay è stato costruito allo stesso modo con la sola differenza nei valori dell'asse Y.

Come si può notare con una rapida occhiata, i grafici sono speculari. Quando il delay è costante il goodput cresce. Quando il delay cresce il goodput si stabilizza.

Questo deriva dal fatto che, un delay stabile indica che la trasmissione dei pacchetti non subisce ritardi aggiunti dovuti, ad esempio, alla congestione della rete o all'accodamento di messaggi in attesa dell'invio. Il goodput cresce poiché pur aumentando il numero di pacchetti inviati in un secondo il delay si mantiene costante. Per semplicità nella visualizzazione dei risultati, nell'asse delle X, accanto al System Load è presente un valore che indica la quantità (in byte) di dati inviati in un secondo.

Confrontando questo valore con quello del goodput ci si accorge del divario che c'è tra i due. Il motivo di questo divario può essere attribuito a due fattori: la dimensione limitata del buffer di invio/ricezione e i limiti della tecnologia. Aumentare la dimensione del buffer non è di alcuna utilità in quanto, quello utilizzato da Contiki, processa un messaggio alla volta.

Goodput

Questo grafico mostra la variazione del goodput al variare del system load e del datarate. Come si può vedere il goodput aumenta all'aumentare

di questi due valori. Si nota che per ogni retta c'è un diverso punto di stacco in cui il goodput si stabilizza. Ciò deriva dal fatto che ad un certo punto la connessione si congestiona e gli invii subiscono dei ritardi. Si può notare come il goodput sia molto inferiore rispetto al datarate.

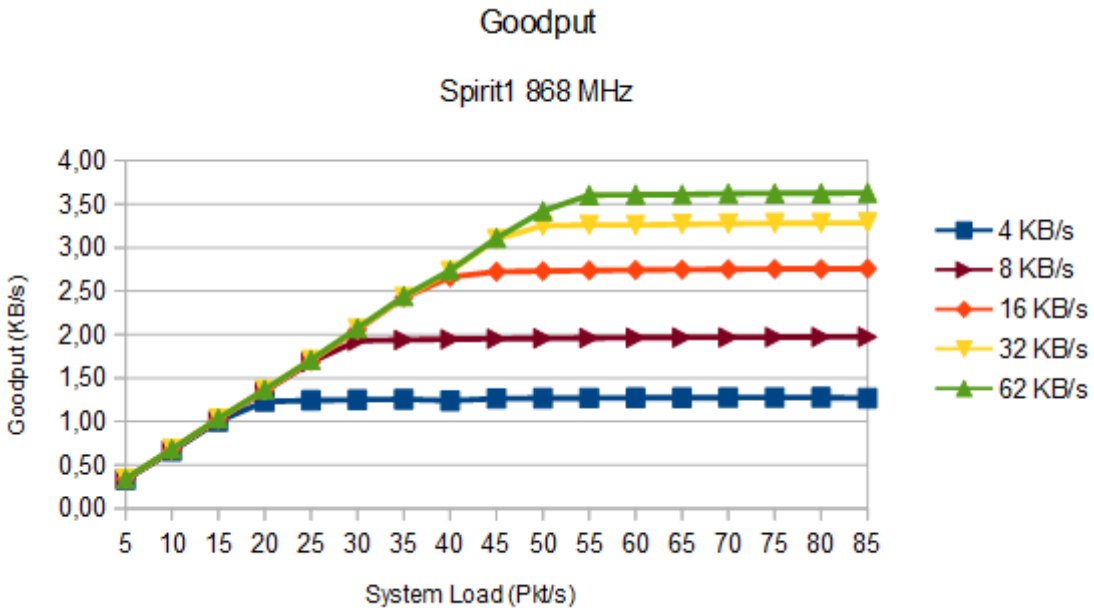

Figura 4.1: Goodput (SPIRIT1).

Delay

Il grafico mostra la variazione del delay al variare del system load e del datarate. I due grafici visti finora sono speculari in quanto, i punti di stacco, sono gli stessi in entrambi. Stavolta arrivate ad un certo punto, le rette si impennano e il delay cresce.

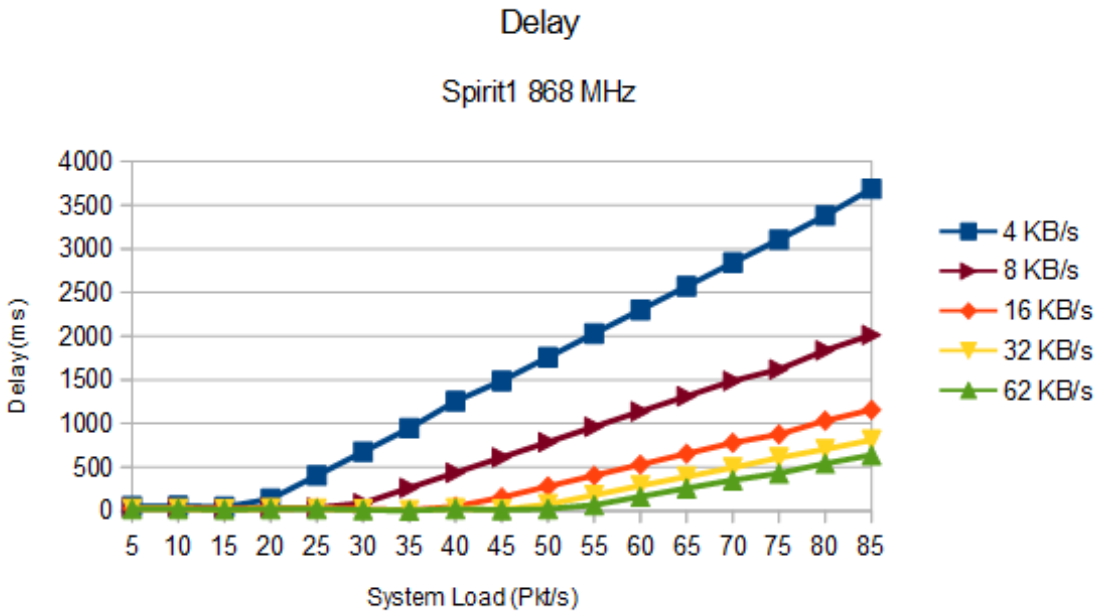

Figura 4.2: Delay (SPIRIT1).

Conclusioni

In questa tesi è stato implementato un gateway che attraverso il protocollo 6LoWPAN permetteva la raccolta di informazioni da una rete di sensori di cui lui era il coordinatore. Ogni sensore raccoglieva informazioni relative alla temperatura ed umidità di certi ambienti e li inviava al gateway. Dato che i dati così raccolti erano disponibili solo localmente si è implementata, tramite MQTT, una comunicazione verso un nodo broker presente su Internet. Periodicamente il gateway pubblicava le informazioni raccolte sul broker. Attraverso questo meccanismo qualunque utente interessato alle informazioni fornite dalla rete poteva interrogare il broker per ottenere i dati richiesti.

Tutto ciò è stato svolto secondo i programmi ma durante la fase di sviluppo e analisi sono sorte delle idee per il miglioramento dell'efficienza della rete.

In primo luogo, dato che i nodi della rete sono dispositivi equipaggiati con batterie si dovrà prevedere un meccanismo per il risparmio di energia. Ad esempio, ogni dispositivo potrebbe entrare in modalità sleep dopo l'invio di un messaggio. In questo modo il dispositivo spegne temporaneamente i moduli che consumano più energia, come quello adibito alla comunicazione, all'elaborazione o alla raccolta dei dati. Nell'implementazione mostrata in questa tesi la durata della batteria del sensore durerebbe qualche ora.

Dai grafici mostrati nella fase di valutazione si vede che ad un certo punto il delay comincia a crescere. Una delle cause è da ricercare nel modo in cui Contiki gestisce il multi-thread. Infatti i processi di invio e ricezione si alternano senza sovrapporsi. Ciò vuol dire che durante una procedura di invio, i

messaggi in entrata non vengono ricevuti poiché il processo di ricezione non è in esecuzione. Per evitare questo tipo di perdite di pacchetti si sono dovuti introdurre ritardi nella computazione dei messaggi che degradano le prestazioni di rete. Contiki fornisce la possibilità di avere un vero parallelismo sfruttando il multi-threading ma l'implementazione di questo meccanismo è specifica per ogni piattaforma.

Si potrebbe migliorare la rete introducendo più gateway per realizzare meccanismi di load balancing e fault tolerance. In questo caso se il gateway smettesse di funzionare per qualsiasi motivo l'intera rete non sarebbe più in grado di adempiere alla sua funzione.

Bibliografia

[1] SUNDMAEKER, Harald, et al. Vision and challenges for realising the Internet of Things. Cluster of European Research Projects on the Internet of Things, European Commision, 2010, 3.3: 34-36.

[2] GUBBI, Jayavardhana, et al. Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 2013, 29.7: 1645-1660.

[3] SHELBY, Zach; BORMANN, Carsten. 6LoWPAN: The wireless embedded Internet. John Wiley & Sons, 2011.

[4] CHRIS, T.; STEVEN, A. Wireless Sensor Networks: Principles and Applications. 2011.

[5] STANKOVIC, John A. Wireless sensor networks. computer, 2008, 41.10.

[6] CULLER, David; CHAKRABARTI, Samita. 6LoWPAN: Incorporating IEEE 802.15. 4 into the IP architecture. IPSO Alliance White Paper, 2009.

[7] BORMANN, Carsten; CASTELLANI, Angelo P.; SHELBY, Zach. Coap: An application protocol for billions of tiny internet nodes. IEEE Internet Computing, 2012, 16.2: 62-67.

[8] HUNKELER, Urs; TRUONG, Hong Linh; STANFORD-CLARK, Andy. MQTT-S?A publish/subscribe protocol for Wireless Sensor Networks. In: Communication systems software and middleware and workshops,

2008. comsware 2008. 3rd international conference on. IEEE, 2008. p. 791-798.

[9] VASSEUR, J., et al. RPL: The IP routing protocol designed for low power and lossy networks. Internet Protocol for Smart Objects (IPSO) Alliance, 2011, 36.

[10] TSVETKOV, Tsvetko; KLEIN, Alexander. RPL: IPv6 routing protocol for low power and lossy networks. Network, 2011, 59.

[11] GADDOUR, Olfa; KOUBÂA, Anis. RPL in a nutshell: A survey. Computer Networks, 2012, 56.14: 3163-3178.

[12] DUNKELS, Adam; GRONVALL, Bjorn; VOIGT, Thiemo. Contiki-a lightweight and flexible operating system for tiny networked sensors. In: Local Computer Networks, 2004. 29th Annual IEEE International Conference on. IEEE, 2004. p. 455-462.

[13] http://anrg.usc.edu/contiki/index.php/Main_Page

[14] Contiki and TinyOS, Information Technology for European Advancement, 2012

[15] WITTENBURG, Georg, et al. Fence monitoring?experimental evaluation of a use case for wireless sensor networks. Wireless Sensor Networks, 2007, 163-178.

[16] SURYADEVARA, Nagender Kumar, et al. WSN-based smart sensors and actuator for power management in intelligent buildings. IEEE/ASME transactions on mechatronics, 2015, 20.2: 564-571.

[17] SINGH, Dhananjay, et al. Global healthcare monitoring system using 6lowpan networks. In: Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on. IEEE, 2009. p. 113-117.

[18] GILL, Khusvinder, et al. A zigbee-based home automation system. IEEE Transactions on consumer Electronics, 2009, 55.2.

[19] HUSSAIN, Md Asdaque, et al. WSN research activities for military application. In: Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on. IEEE, 2009. p. 271-274.

[20] ?URI?I?, Milica Pejanovi?, et al. A survey of military applications of wireless sensor networks. In: Embedded Computing (MECO), 2012 Mediterranean Conference on. IEEE, 2012. p. 196-199.

[21] RAZA, Usman; KULKARNI, Parag; SOORIYABANDARA, Mahesh. Low power wide area networks: An overview. IEEE Communications Surveys & Tutorials, 2017, 19.2: 855-873.

Ringraziamenti

Qui possiamo ringraziare il mondo intero!!!!!!!!!!
Ovviamente solo se uno vuole, non è obbligatorio.