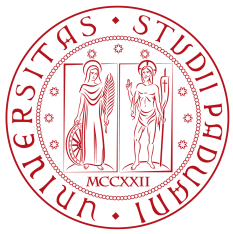

Università degli Studi di Padova
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Sistema Embedded per la gestione distribuita dell'energia in una Smart Microgrid

Relatore: Ch.mo Prof. Matteo Bertocco

Laureando: Marco Testolina
Matricola: 607244 - IF

24 Aprile 2012
Anno Accademico 2011/2012

A Laura e alla mia famiglia

Indice

Sommario

Nel presente lavoro si è affrontato il problema di una gestione efficiente dell'energia in una Smart Microgrid. E' stato quindi sviluppato un algoritmo distribuito in grado di bilanciare in modo efficace i picchi di potenza in linea così da migliorare la power quality e l'hosting capacity della rete di distribuzione.

Nel capitolo 2 saranno spiegate in dettaglio tutte le problematiche inerenti le Smart Microgrid e le entità che vengono coinvolte. Nel capitolo 3 verranno introdotti nei dettagli i moduli che sono stati individuati come principali attori nella gestione e nello scambio dell'energia. Nei capitolo 4 e 5, sarà quindi spiegata la struttura di tali moduli e le scelte progettuali, come questi sono stati definiti e come sono stati simulati sull'hardware in dotazione. Nel capitolo 6 infine saranno mostrati i risultati di alcune simulazioni effettuate in laboratorio.

Capitolo 1

Introduzione

La rete di trasmissione e distribuzione dell'energia è di solito basata sulla disponibilità di grandi centrali di produzione connesse a linee ad alta tensione, le quali poi distribuiscono l'energia tramite sottoreti a media e bassa tensione. Tale struttura centralizzata di produzione, è stata progressivamente affiancata da generatori distribuiti basati su fonti rinnovabili. La rete di distribuzione risulta quindi caratterizzata da flussi di energia bidirezionali dovuti ai *prosumer*, ossia utenti contemporaneamente produttori (*producer*) e utilizzatori (*user*). Presso ambiti scientifici si sta affermando l'idea che qualora la quota parte di energia dovuta ai *prosumer*, diventi una parte importante della produzione complessiva sia necessario modificare i modi di produzione e iniezione dell'energia ricorrendo a nuovi metodi basati anche sullo scambio di informazioni di stato e controllo a livello di utenti. Ne risulterebbe la necessità di una capacità di gestione e coordinazione robusta ed efficiente, utilizzando tecniche di comunicazione in uno scenario simile a quello di una rete internet dove ogni utente connesso, attraverso i propri impianti di generazione, assume una parte attiva e si coordina con le altre entità alla fine di gestire efficacemente la rete sotto sia il profilo energetico sia economico. Questa nuova concezione di rete elettrica, più efficiente e flessibile, viene riferita con il termine *Smart Grid*, ossia:

> ***Una rete in grado di offrire analisi, monitoraggio, controllo e comunicazione al sistema di distribuzione al fine di ottimizzare l'efficacia dello stesso e contribuire al risparmio energetico***[1]

Dunque, con il concetto di *Smart Grid* si aggiunge capacità di analisi, controllo e comunicazione al sistema di distribuzione e trasmissione elettrico, affinché possa ottimizzare l'efficienza del sistema e contribuire al risparmio energetico. In accordo a tale visione, il miglioramento dell'attuale sistema elettrico sarà ottenuto anche dal monitoraggio e dalla elaborazione distribuita per ottenere una conoscenza più accurata dello stato della rete e conseguentemente migliorare il controllo degli impianti.

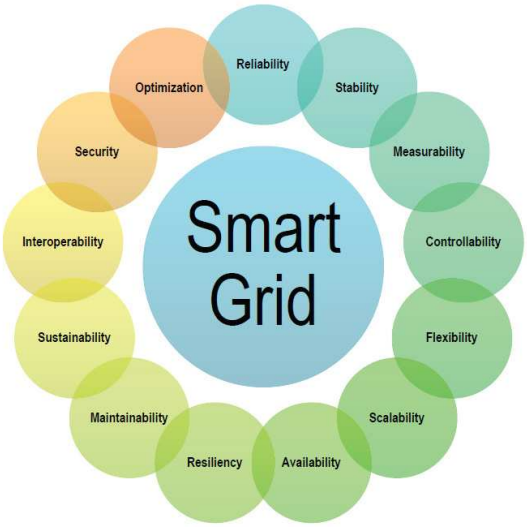

Figura 1.1: Caratteristiche e requisiti di una Smart Grid

Gli obiettivi perseguiti con tali strategie sono:

- rendere il più omogeneo possibile il flusso di energia tra i punti di produzione e quelli di assorbimento per limitare gli stress alla rete dovuti ai picchi di consumo;
- gestire in maniera più efficiente la generazione distribuita dell'energia anche in luoghi remoti, in particolare di quella originata da fonti rinnovabili (e.g., turbine eoliche, pannelli fotovoltaici) che ora causano per loro natura problemi di instabilità;
- aumentare l'affidabilità delle strutture e la qualità dell'energia fornita attraverso il monitoraggio dello stato della rete;
- fornire servizi aggiuntivi ai consumatori affinché abbiano la possibilità di scegliere il fornitore dell'energia e siano agevolati all'assumere comportamenti conformi al risparmio e all'evitare eccessivi carichi della rete.

Il concetto di *Smart Grid*, tuttavia, porta con se alcune sfide da risolvere. Le reti intelligenti, infatti, necessitano di sistemi di misurazione non ancora disponibili. La produzione di energia da fonti rinnovabili non è costante: nasce così la necessità di installare dispositivi di storage per ottimizzarne la gestione. Si devono quindi cercare delle soluzioni per coordinare in modo efficiente le entità presenti in una Smart Grid, consentendo un uso consapevole dell'energia prodotta localmente. A questo si aggiunga il concetto di *Smart MicroGrid* particolarmente interessante a livello di comunità o di quartiere. Infatti, al fine di perseguire gli obiettivi già citati, è possibile sfruttare le tecnologie e i protocolli delle reti di telecomunicazioni, non più su vasta scala, ma a livello locale.

Il lavoro di tesi qui presentato si pone l'obiettivo di gettare le basi per l'implementazione di un sistema di gestione dell'energia che possa interagire sia con futuri sistemi di domotica sia con la rete esterna nazionale, per comunicare e compiere operazioni sui dati scambiati.

Capitolo 2

Smart Grid

Una rete intelligente fornisce l'elettricità ai consumatori utilizzando tecnologie ICT per massimizzare il rendimento, ridurre i costi, accrescere l'affidabilità e la trasparenza dei sistemi elettrici. L'introduzione di procedure informatiche consente alla rete di effettuare calcoli e misure, collegando tra loro consumatori, enti distributori e produttori. Questa innovativa forma di comunicazione promette di rivoluzionare l'attuale contesto di produzione e distribuzione dell'energia elettrica. L'utente, conoscendo in tempo reale i dati relativi ai propri consumi, sarà in grado di adottare il comportamento più opportuno e più consono alle sue esigenze. L'energia in rete non proverrà esclusivamente dall'ente di distribuzione centrale, ma potrà essere messa a disposizione anche da fonti locali. Infine, tale concezione della rete elettrica aprirà la strada ad una vasta gamma di nuove tecnologie, nuovi servizi e nuovi mercati.

2.1 La generazione distribuita

Allo stato attuale, la struttura della rete elettrica si dimostra sempre meno idonea a supportare le necessità imposte dal mercato dell'energia. In particolare, i nuovi traguardi imposti dall'Unione Europea in merito a riduzione dei consumi, miglioramento dell'efficienza e diminuzione dell'inquinamento, impongono una ridefinizione dell'intera infrastruttura di produzione e distribuzione. Per far fronte a queste nuove sfide, una possibile soluzione è rappresentata dalla cosiddetta generazione distribuita. Un simile approccio non attua alcuna forma di controllo centralizzato né in fase di produzione, né in fase di distribuzione. L'adozione di un protocollo distribuito consente di aumentare la capacità di trasporto della rete elettrica, attraverso il ridimensionamento dei componenti di potenza delle linee (linee, trasformatori,...). Allo stesso modo, si rende possibile l'uso congiunto di diversi vettori energetici locali, con immediati riscontri in termini di efficienza energetica e costi di gestione. A tal proposito, la generazione distribuita consente, da un lato, di sfruttare risorse energetiche altrimenti inutilizzate, quali l'irraggiamento solare; dall'altro, di realizzare reti meno estese e

più economiche. D'altro canto, una simile innovazione porta con sé una serie di problematiche, quali:

- aumento dei punti di prelievo e di immissione della potenza;
- alterazione dei flussi di potenza;
- aumento del sovraccarico nelle singole linee;
- alterazione delle tensioni lungo la distribuzione;
- difficoltà di monitoraggio delle condizioni di funzionamento con utenze attive e passive difficilmente prevedibili;
- mancato riconoscimento di funzionamenti anomali della rete;

Nasce così l'esigenza di progettare dei metodi di gestione, in grado di fornire un monitoraggio diffuso della rete attraverso molteplici punti di misura. Allo scopo, si sono approntati algoritmi di validazione e elaborazione dei dati, che sfruttano le utenze attive e passive in gioco per gestire al meglio il funzionamento della rete.

2.2 Microrete residenziale

Una *Smart Microgrid* è assimilabile in quanto a dimensioni a un quartiere, o a un piccolo paese dove tutte le utenze si coordinano per raggiungere a livello locale gli stessi obiettivi di risparmio energetico e di gestione efficiente dell'energia. In questo contesto, una *Smart Microgrid*, o microrete residenziale, rappresenta la realtà più ristretta e vicina agli utenti finali. Tuttavia, può essere considerata come il primo passo verso l'estensione e lo sviluppo delle reti intelligenti a livelli più estesi. In termini operativi, tale microrete è caratterizzata dalla presenza di un discreto numero di nodi che contendono e condividono un numero limitato di risorse. Si possono quindi trasportare i concetti chiave delle reti Internet nel contesto in esame: in particolare, la condivisione della potenza prodotta localmente può essere intesa come uno scambio di informazione tra i nodi.

Una rete di questo genere è composta da edifici, come abitazioni civili, uffici pubblici o realtà industriali, che possono comportasi sia come utilizzatori sia come produttori di energia. Parimenti, non si esclude la presenza anche di dispositivi di storage che riescano ad immagazzinare l'energia prodotta in eccesso e a restituirla alla rete in caso di necessità.

A tal proposito, si introduce il concetto di *power quality*, intesa come la capacità della rete di sopperire ad eventuali variazioni o interruzioni nell'erogazione dell'energia. L'adozione di un protocollo distribuito promette appunto una maggiore continuità del servizio e un rapido adeguamento della fornitura elettrica alla domanda da parte degli utenti (*energy management*).

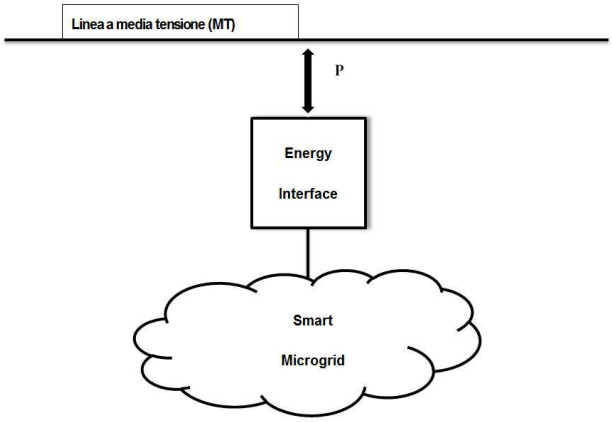

Figura 2.1: Ruolo di *Energy Interface*

La disponibilità di energia localizzata porta quindi a definire nuovi scenari di funzionamento, primo su tutti, il *funzionamento ad isola* che prevede la possibilità di disconnettersi dalla rete di distribuzione nazionale, disponendo di energia sufficiente a far fronte al consumo interno della *microgrid*. In quest'ottica, sorge la necessità di introdurre degli appositi dispositivi che gestiscano gli scambi di potenza. In particolare, l'**Energy Gateway** sarà deputato alla gestione della potenza (attiva e reattiva) utilizzata, mentre l'**Energy Interface** fornirà un'interfaccia di controllo tra la rete di distribuzione e la rete residenziale tale da replicare il funzionamento dell'*Energy Gateway* ad un livello gerarchico superiore[2].

2.3 Nodo Energy Interface

Come mostrato in figura 2.1, la rete a media tensione associa una comunità organizzata di *prosumer* (la smart microgrid) ad un unico utente per mezzo di una entità chiamata *Energy Interface*. Esso, infatti, somma tutte le richieste energetiche della sottorete gestita e dall'interno risulta essere omogeneo con gli altri *energy gateway*. Tutto ciò permette di introdurre una struttura scalabile di Smart Grid, infatti si può andare verso la definizione di uno scenario come quello mostrato in figura 2.2, dove i punti di accesso alle varie sottoreti sono appunto gli *energy interface*. Si vede quindi come un *energy interface* sia un'interfaccia tra la rete di distribuzione e la microrete, offrendo inoltre delle funzioni di misura sui flussi di energia tra le linee a media tensione e la microgrid implementando il supporto per la gestione e l'ottimizzazione di tali scambi energetici.

Il compito dell'*energy interface* è inoltre quello di coordinare il funzionamento a isola, in base a policy che sono decise sui dati raccolti sia dal lato microgrid sia dal lato sistema di distribuzione. Se la sottorete dispone di una autonomia energetica si può decidere che lo scambio di energia tra i due soggetti debba

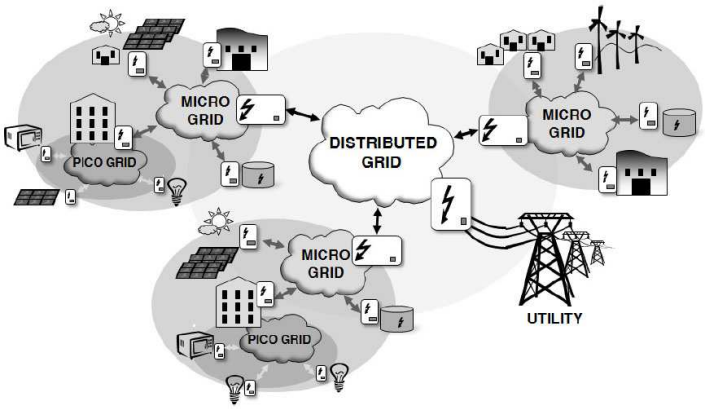

Figura 2.2: Achitettura scalabile di una MicroGrid

essere nullo, questa procedura prevede il coinvolgimento degli *energy gateway* al fine di attuare le operazioni necessarie: spegnimento carichi o attivazione moduli di storage.

2.4 Nodo Energy Gateway

Il nodo ***Energy Gateway*** (EG) fa idealmente da intermediario tra i *prosumer* e la rete elettrica a bassa tensione in modo da poter attuare delle azioni di controllo sui dispositivi interni alle utenze. Tale device, quindi, oltre alla capacità di comunicare nella rete, possiede delle funzioni di metering dei consumi. Assieme

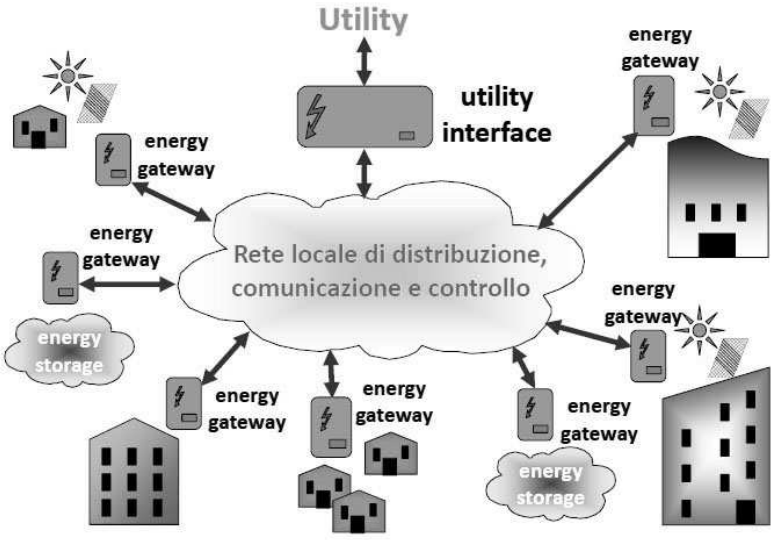

Figura 2.3: Struttura di una micro rete residenziale

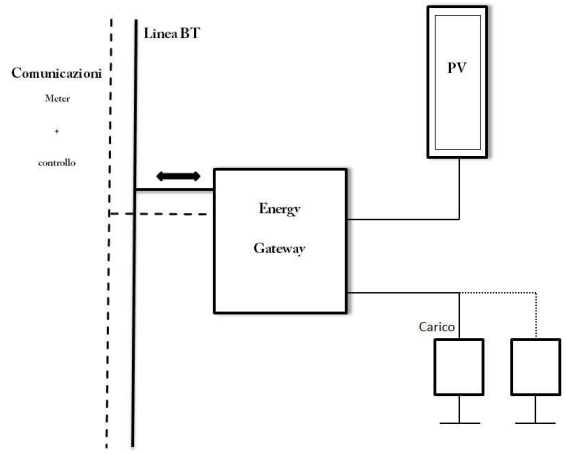

Figura 2.4: Nodo *prosumer*: visione Smart Microgrid

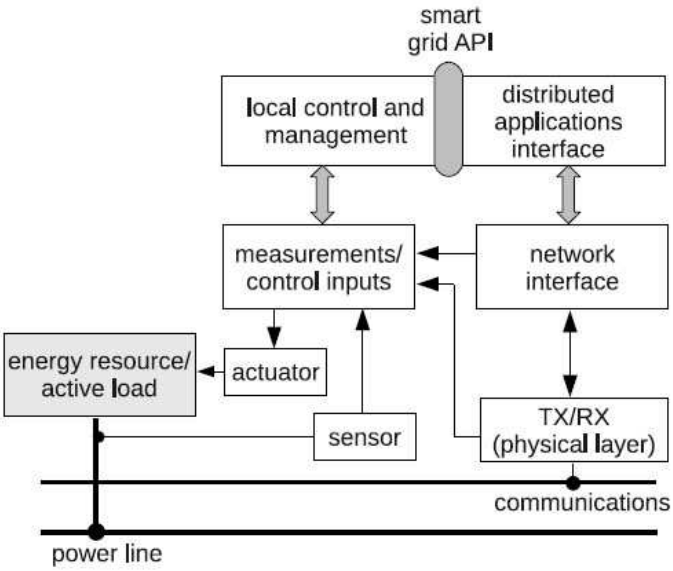

Figura 2.5: Elementi di un Energy Gateway

a dei metodi di *energy management* riesce a controllare l'energia gestendone in modo efficace la domanda interna.

In una *smart microgrid* quindi, *l'energy gateway* ha il compito di aggregare le richieste delle unità locali di una utenza e rappresenta un punto di accesso obbligato alla rete di distribuzione. Tramite tale accesso si riescono infatti a gestire i flussi di energia a due vie tipici degli ambienti presi in esame.

A questo scopo un *energy gateway* deve quindi poter supportare le funzionalità di:

- integrare funzioni di *meter* bidirezionale;
- controllare le sorgenti di potenza in modo da ottimizzarne il funzionamento, diminuire le perdite in linea e stabilizzare le tensioni di nodo (gestendo potenza attiva e reattiva[1]);
- instaurare comunicazioni con altri *energy gateway* al fine di scambiare informazioni e dati per il controllo distribuito;
- sfruttare meccanismi di sincronizzazione tra nodi e di monitorig della power quality[2];
- controllo locale assumendo il compito di gateway domotico.

Lo schema a blocchi di un *energy gateway* si può osservare in figura 2.5; tra tutti il blocco di *sensing* è importante sotto il punto di vista del riferimento temporale, infatti, la sincronizzazione degli orologi dei diversi nodi è fondamentale per la raccolta dei dati e per una loro corretta gestione da parte dell'algoritmo distribuito. Tra gli obbiettivi di progetto, che saranno definiti in seguito, c'è quello di utilizzare un qualche metodo che mantenga sincronizzati i clock dei diversi *energy gateway*. Un secondo blocco preso in considerazione in questo lavoro è il modulo di controllo e gestione (*local control and management*) che dialoga constatemene con il sensing e gli attuatori. Tale comunicazione permette di rendere effettive le modifiche dettate dall'algoritmo di gestione, in questo modo si può considerare la gestione dei carichi nei nodi un parametro modificabile dalla rete in caso di emergenza.

Particolare enfasi nel controllo locale va data ai consumi di elettricità, infatti in momenti di sovraccarico della rete, gli *energy gateway* potrebbero richiedere ai dispositivi elettrici meno prioritari di spegnersi, evitando blackout nella rete o nei singoli nodi, ad esempio spegnendo il solo condizionatore o ritardando il ciclo di accensione di alcuni elettrodomestici. Gli obbiettivi, che una tale logica di funzionamento del sistema di controllo, deve seguire è una questione ancora

[1] L'immissione in rete di potenza reattiva permette di stabilizzare la tensione ai nodi della rete [3]

[2] Stabilizzazione della tensione a livello locale tipicamente effettuata tramite iniezione in rete di potenza attiva o reattiva

aperta, tuttavia è possibile individuare alcuni punti principali che il controllo dovrà fornire:

- una più efficace integrazione e migliore utilizzazione dei generatori basati su fonti di energia rinnovabile;
- incremento dell'efficienza della rete;
- riduzione dei costi di esercizio e delle perdite in linea;
- dimensionamento più economico dei componenti di rete e riduzione dei costi di installazione;
- benefici ambientali;
- peak-shaving e di conseguenza aumento della hosting capacity[3];
- miglioramento della *power quality*, intesa come affidabilità e della continuità del servizio.

Nel lavoro sviluppato si è deciso di simulare alcuni blocchi dell'energy gateway creando diversi moduli software che ne riproducono il comportamento, tale astrazione sarà spiegata nei prossimi capitoli.

2.5 La comunicazione nelle SmartGrid

La comunicazione tra le diverse entità di una *SmartGrid* è strutturata a livelli differenti. In questo lavoro di tesi si gestiranno i nodi attraverso una rete *Ethernet* semplice da utilizzare visto che l'hardware implementava tale standard ma non saranno trattati i differenti standard di comunicazione con differenti bande.

Tuttavia le comunicazioni di una *SmartGrid* saranno suddivise in diversi livelli, ogni livello avrà bisogno di caratteristiche diverse sotto l'aspetto della comunicazione e dei valori di banda disponibile. A questo si somma il problema della *privacy* e della *security* ancora non ben definite né dagli organi prepositi alla tutela dei consumatori né dalle strutture istituzionali.

In particolare nella visione proposta in questo lavoro gli *energy gateway* faranno oltre che da meter anche da ***EMU*** (Energy Management Unit) cioè un ponte di comunicazione tra la ***HAN***[4] mostrata in figura 2.6 e la ***NAN***[5], cioè tra la domotica interna e la microgrid esterna[3]. A livello di microgrid gli edifici si troveranno connessi in una ***NAN*** che potrebbe avere come access point per le comunicazione l'*energy interface*.

[3] Capacità della rete, a parità di infrastrutture, di servire più utenze

[4] Home Area Network: rete costituita dai diversi dispositivi che saranno gestibili in una casa con funzionalità domotiche

[5] Neighborhood Area Network: rete costituita dalle adiacenze in una smartmicrogrid

Standard Wired	Ambito di utilizzo
DLS	Neighborhood Area Network
Fiber communication	Neighborhood/Wide Area Network
Power line communication	Home/Neighborhood Area Network

Tabella 2.1: Standard wired attualmente valutati per la comunicazione in NAN e WAN per smartgrid

Standard Wireless	Ambito di utilizzo
ZigBee[6] - IEEE 802.15.4	Home Area Network
WiMax	Neighborhood Area Network
LTE/UMTS	Wide Area Network

Tabella 2.2: Standard Wireless attualmente valutati per la comunicazione a diversi livelli di una SmartGrid

Dal canto suo l'*energy interface* quindi farebbe da ponte tra le comunicazioni tra la microgrid ed una ***WAN*** creata dall'unione di differenti smart microgrid o dalla rete nazionale di distribuzione. Si prospetta quindi l'utilizzo di diversi protocolli di comunicazione che dovranno essere efficacemente organizzati per supportare tali livelli; nella tabelle 2.2 e 2.1 sono elencati i diversi standard di comunicazione che, in ambito scientifico, si stanno valutando come possibili candidati all'utilizzo nelle smartgrid[3].

[6]Lo standard ZigBee è stato progettato per l'uso in applicazioni embedded che richiedano un basso transfer rate e bassi consumi. L'obiettivo attuale di ZigBee è di definire una *Wireless mesh network*, economica e autogestita che possa essere utilizzata per scopi quali il controllo industriale o nel caso delle HAN, nell'organizzazione interna della domotica

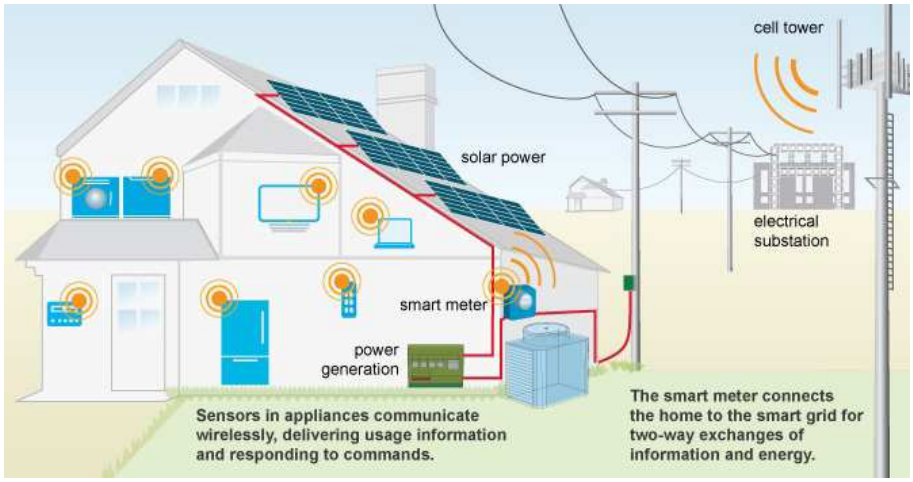

Figura 2.6: Illustrazione di una home area network

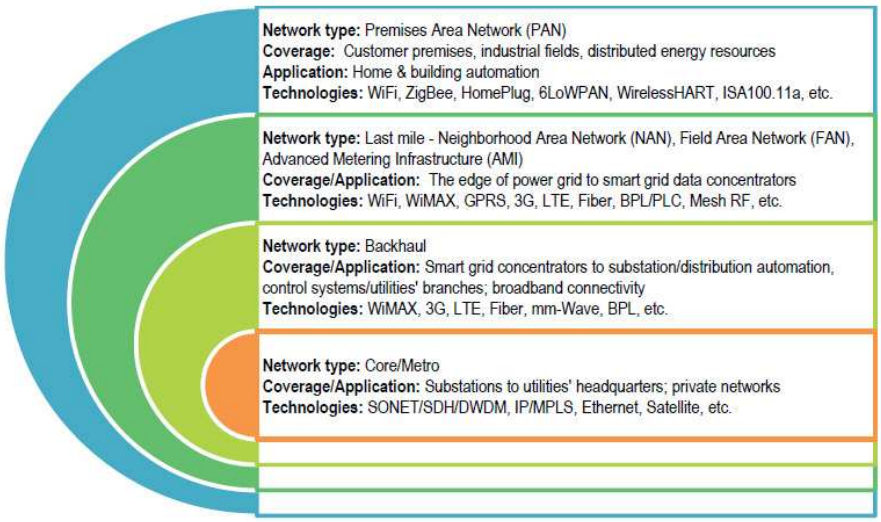

Figura 2.7: Livelli e rispettivi protocolli in una Smart Grid

Capitolo 3

Obiettivi del progetto

Il seguente capitolo descrive in dettaglio gli obiettivi del lavoro di tesi. In particolare il lavoro di sviluppo ha avuto un approccio *bottom-up* con la creazione, dapprima delle funzionalità legate all'hardware, e in un secondo momento delle funzioni di controllo. L'ordine con il quale vengono presentati i requisiti segue tuttavia una diversa logica; vengono presentati prima gli obbiettivi di gestione e controllo e successivamente ci si addentra nei dettagli legati hardware. Si è ritenuto, infatti, che tale ordine definisse meglio gli obiettivi prefissati. Nel lavoro sono state introdotte delle semplificazioni che hanno permesso un approccio più semplice al problema, tuttavia sono state mantenute tutte le funzionalità importati per la corretta implementazione della gestione. Le semplificazioni apportate sono:

- sono state considerate solo le potenze attive;
- il controllo è sullo spegnimento o l' accensione dei carichi o sulla disponibilità dello storage locale;
- lo storage è modellato come un carico che assorbe o cede sempre la stessa potenza senza considerare variazioni dovute alla quantità di carica residua;
- i nodi hanno lo stesso numero di carichi e produzione;

La gestione quindi è solo sulla potenza attiva, in questo modo si crea un algoritmo che cerca di aumentare la ***hosting capacity*** diminuendo i picchi di consumo, ma **tralascia i parametri di *power quality***. Il lavoro inoltre ha come obiettivo secondario quello di **testare un sistema embedded** in questo campo di applicazione, come già accennato in precedenza.

3.1 Energy Gateway e Energy Interface

Il lavoro di tesi prevede che siano sviluppati degli applicativi software che simulino il comportamento degli *energy interface* e degli *energy gateway*; la struttura

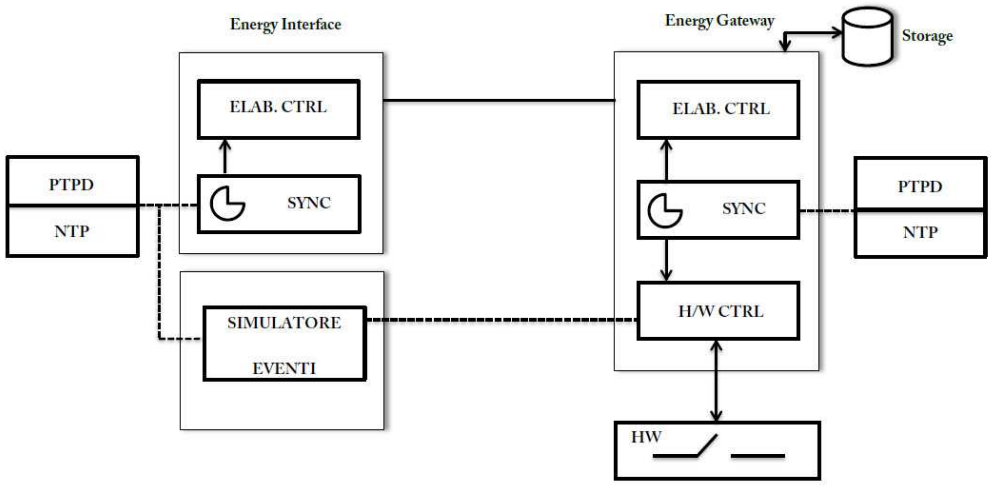

Figura 3.1: Organizzazione e struttura dei moduli

dei moduli è mostrata in figura 3.1. Nello schema a blocchi si possono vedere le diverse sezioni progettate, alcune sono identiche da entrambi i lati o a livello di codice subiscono qualche modifica altre invece sono specifiche del tipo di nodo. Tuttavia come più volte riportato la divisione hardware tra *energy interface* ed *energy gateway* è indicativa in quanto i moduli possono coesistere all'interno di uno stesso nodo.

3.1.1 Energy Interface

Nello sviluppo di questo lavoro tale entità è stata lasciata indipendente e le sue funzioni principali sono, oltre a quelle di contabilizzazione previste dalle Smart Microgrid, di elaborazione, di sincronizzazione e gestione del ciclo di lavoro. Il timing interno è sincronizzato con gli altri nodi della federazione e la procedura di inizializzazione prevede i seguenti passi:

- determinazione dei parametri di funzionamento t_0 e $\mathbf{T}$;
- verifica degli *energy gateway* attivi;
- una volta trovati, per ogni nodo, invio dei parametri del ciclo di lavoro, delle adiacenze[1] e dei parametri della rete, che nella versione sviluppata si limitano alla potenza massima disponibile al nodo;
- attesa di ulteriori *energy gateway* ripetendo i passi appena descritti.

[1]Per adiacenze si intende i nodi della rete fisicamente vicini, è importante che gli energy gateway conoscano i propri vicini per massimizzare il rendimento e diminuire le perdite in linea degli scambi di potenza

I servizi disponibili, invece, devono poter gestire al meglio una politica *plug&play*, quindi una volta che il nodo è attivo ed esegue una prima analisi dello stato della rete deve attendere le richieste di accesso di nuovi *energy gateway*.A queste risponde comunicando i parametri di funzionamento e aggiornando i parametri del ciclo di lavoro nei nodi coinvolti nelle adiacenze. Inoltre verifica periodicamente la funzionalità dei nodi connessi e li espelle dalle comunicazioni nel caso non avesse risposta. L'*energy interface* inoltre raccoglie i dati di consumo dei diversi nodi e su questi ad ogni ciclo di lavoro calcola l'obiettivo della federazione da lui controllata, comunicandolo nella fase di comunicazione a tutti gli *energy gateway*. Questi messaggi servono se agli *energy gateway*, in caso di malfunzionamenti del ciclo di gestione, per determinare le azioni da intraprendere anche in presenza di errori nelle comunicazioni o nelle adiacenze.

3.1.2 Energy Gateway

L'entità dell'***Energy Gateway*** è ripetuta in diversi nodi nella rete, il loro funzionamento è sincrono e comunicano tra nodi della stessa adiacenza. La sequenza di avvio prevede i seguenti passi:

- inizializzazione del controllo hardware;
- attesa del collegamento con un energy interface;
- ricezione dei parametri t_0 e $\mathbf{T}$;
- ricezione degli indirizzi IP delle adiacenze;
- ricezione dei parametri di funzionamento.

In particolare *l'energy gateway* ha un importante modulo, che nella figura 3.1 è definito con **HW** il quale gestisce la simulazione dei carichi e degli impianti di generazione. Tale funzionalità permette ad un utente di interagire in modo diretto con l'accensione e lo spegnimento dei carichi in modo tale da variare la potenza attiva prodotta e consumata dal nodo. Ciò implica, che ogni nodo della rete abbia un consumo simulato assegnato per ogni carico. Al primo avvio il modulo hardware carica da un file di configurazione locale il consumo e la priorità associata al nodo e al carico, in modo da definire diverse gerarchie di consumo e rendere lo spegnimento dei carichi più intelligente.

Un esempio di come potrebbero essere gestite le priorità in un'abitazione è il seguente:

- consumi più bassi e associati a illuminazione, **priorità alta**;
- consumi medi associati a intrattenimento o piccoli lavori domestici , **priorità media**;

- consumi medio alti e associati a lavori domestici pesanti (lavatrice, lavastoviglie, carica auto), **priorità bassa**.

Sempre in riferimento alla figura 3.1, il blocco definito con il nome di H/W CTRL ha il compito di gestire lo stato dell'hardware e, in base alle priorità e ai profili di consumo, di impartire comandi al modulo hardware. Tali comandi sono decisi dal modulo ELAB CTRL sulla base dei dati ricevuti dal corrispettivo ELAB CTRL dell'*energy interface*, oppure delle decisioni interne prese analizzando i dati ricevuti dalle adiacenze. D'altro canto, il modulo di memorizzazione consente ad ogni nodo di tener traccia dei consumi, producendo registri delle seguenti tipologie:

- storico dell'energia scambiata dal nodo, intesa come energia prodotta e consumata;
- storico dell'energia immagazzinata nei dispositivi di storage;
- storico delle potenze istantanee con la granularità del ciclo di gestione;
- storico delle accensioni e spegnimenti dei carichi asincrono rispetto al ciclo di gestione.

3.2 Ciclo di lavoro

Tutte le operazioni eseguite dai nodi, siano essi *energy gateway* o *energy interface*, devono essere sincronizzate. Allo scopo, si sono definite le seguenti cinque fasi:

- **aggiornamento stato:** l'*energy gateway*, controllato lo stato effettivo dei carichi e della generazione, aggiorna il proprio stato interno;
- **comunicazioni:** i dati raccolti durante l'aggiornamento vengono scambiati tra i nodi adiacenti e inviati all'*energy interface*;
- **analisi:** i nodi analizzano i dati ricevuti per redarre un opportuno protocollo di azioni da intraprendere;
- **azione:** gli *energy gateway* attuano le operazioni pianificate al passo precedente;
- **sincronizzazione:** i nodi sincronizzano i propri cicli di lavoro.

Il ciclo di lavorogestione attribuisce ad ogni operazione un preciso istante temporale e un tempo limite oltre il quale, in caso di mancata esecuzione, si passa alla fase successiva. I parametri di funzionamento t_0, $\mathbf{T}$, visibili in figura 3.2, forniscono rispettivamente il riferimento temporale iniziale e l'ampiezza complessiva dei un ciclo di lavoro. In particolare t_0 è determinato dall'*energy interface* e considerato come tempo zero da tutta la rete. Come si può intuire

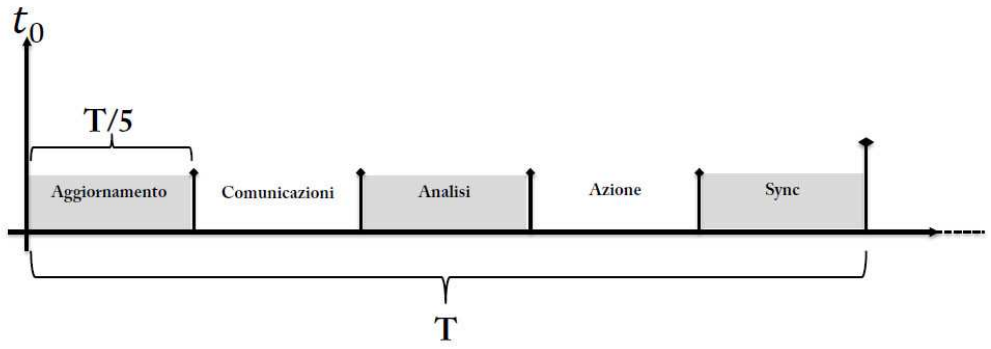

Figura 3.2: Schema del ciclo di lavoro implementato in tutti i nodi

le due tipologie di nodo presenti non sfruttano tutte le fasi del ciclo di lavoro, *l'energy gateway* sarà sempre impegnato ad effettuare i compiti nelle diverse finestre temporali, mentre *l'energy interface* avrà delle finestre libere come, per esempio, la fase di aggiornamento stato e di azione; tuttavia anche se il nodo non esegue operazioni, tali intervalli si potranno sfruttare per eseguire nuovi compiti non ancora previsti durante la stesura di questo elaborato.

3.3 Simulazione di funzionamento

Per verificare che il sistema risponda in modo adeguato, è nata la necessità di creare un semplice simulatore che dia la possibilità di configurare in automatico e dinamicamente tutti i parametri e gli stati che possono influenzare il comportamento della federazione. Devono essere inoltre disponibili delle opzioni per la raccolta e l'aggregazione dei dati prodotti dagli applicativi durante le sessioni di test in modo da essere interpretabili sia da un utente sia da programmi di elaborazione come *Matlab* o *Excel*. Il modulo di simulazione quindi una volta instaurata la connessione con i nodi della federazione carica da dei propri file le istruzioni seguendo l'ordine temporale impostato e le sottopone agli Energy Gateway. Tale simulatore deve poter essere funzionante sia in nodi che partecipano attivamente alle simulazioni (*Energy Gateway o Energy Interface*) sia su Pc di controllo connessi alla stessa sottorete.

I tipi di comandi che il simulatore fornisce ai nodi sono gli stessi che un operatore potrebbe fare agendo sui carichi quindi è previsto che tale applicativo gestisca:

1. accesione/spegnimento dei carichi;
2. accensione/spegnimento impianto generazione;
3. variazione di potenza prodotta in percentuale.

Lo scopo è quello di preparare accuratamente degli scenari di funzionamento della rete e sottoporli ad una verifica sul campo in modo preciso e veloce, assicurando nel contempo una produzione di dati utili per le analisi. Le prime due azioni, infatti, simulano il comportamento di utenti nella rete, e la terza imita le variazioni nella potenza prodotta in impianti di microgenerazione come, ipotizzando che il nodo abbia a disposizione un impianto fotovoltaico, il passaggio di nubi o se presente una turbina eolica, variazioni nella velocità del vento.

3.4 Gestione caduta nodi

In un sistema in cui sono presenti diverse entità che interagiscono il problema della caduta di qualche nodo deve essere gestito efficacemente, quindi, sia gli *energy gateway* che gli *energy interface* devono poter accorgersi di tale evenienza senza creare eccezioni. Possono verificasi due tipi di errori:

- ***caduta di un energy gateway:*** in questo caso tutti i nodi, siano essi le adiacenze o il nodo *energy interface*, semplicemente chiudono le comunicazioni con tale nodo. Le adiacenze si troveranno a far funzionare il proprio ciclo di lavoro con un entità in meno. Si deve inoltre gestire dal lato *energy interface* la caduta di tutti i nodi della federazione;
- ***caduta dell'energy interface:*** in tale evenienza i singoli nodi della rete devono chiudere tutte le comunicazioni con le adiacenze e mantenere attivi i servizi di base.

La gestione combinata di queste due eccezioni porta a risolvere efficacemente i problemi di rete, in quanto in caso di caduta dei sistemi di comunicazione, tutti i nodi si troverebbero isolati passando ad una modalità di funzionamento singola.

Capitolo 4

Sistema Embedded per lo sviluppo

Il presente capitolo elenca i moduli hardware e software scelti per la realizzazione degli obiettivi. Il linguaggio di programmazione scelto è il ***C***, già conosciuto dallo sviluppatore e che ha il vantaggio, essendo un linguaggio compilato, di avere una elevata velocità di esecuzione. E' inoltre possibile interagire con l'hardware del sistema embedded con relativa facilità.

4.1 FoxBoard G20

Il sistema Embedded preso in considerazione in questo lavoro era già conosciuto nelle versioni precedenti dai collaboratori del progetto ed è stato scelto in quanto piattaforma versatile che si presta a diverse tipologie di applicazioni. L' hardware della ***FoxBoard G20*** viene prodotto e reso disponibile sul mercato dall'italiana ***ACME SYSTEMS srl*** con sede a Roma. Si tratta di un sistema basato su architettura ARM[1] fornito con una versione Debian Linux [2] come sistema operativo.

Le principali caratteristiche sono elencate nel seguito:

- Processore ARM9 @ 400Mhz CPU;
- 256KB di memoria flash per bootloader;
- lettura di microSD card fino a 16GB;
- due porte USB 2.0;

[1] L'architettura ARM (*Advanced RISC Machine*) indica una famiglia di microprocessori RISC a 32-bit sviluppata da ARM Holdings e utilizzata in una moltitudine di sistemi *embedded*.

[2] Debian GNU/Linux è un sistema operativo che utilizza un *kernel Linux* e programmi di provenienti dal progetto GNU il cui scopo è la creazione e il mantenimento di un sistema operativo composto esclusivamente da software libero chiamato.

- porta Ethernet 10/100;
- 5VDC power supply;
- real time clock con batteria tampone;
- linee GPIO(3.3v);
- dati di consumo: 80 mA @ 5V (0.4 Watt) senza microSD, ethernet,o device USB;

L'interazione con tale hardware è stato principalmente effettuato attraverso SSH[3] per la gestione dei comandi, ed attraverso FTP per il trasferimento dei dati o file necessari; i parametri per l'utilizzo di tali protocolli si trovano nel capitolo 9, e tutta da documentazione si può trovare sul sito dell'azienda produttrice [4]. Il sistema Fox, quindi, con tali caratteristiche fornisce un'ottima potenza di calcolo che assieme alle possibilità offerte dalla connessione Ethernet, dall'ampia memoria disponibile, ed alle precedenti esperienze di utilizzo in altri lavori lo ha portato ad essere un rifermento per tutto il lavoro di tesi.

In particolare si sono presi in considerazione i lavori di tesi di Francesco Sandri [5] e di Thomas Scremin [6] entrambi sviluppati su versioni precedenti della piattaforma *FoxBoard* e che riguardavano la sincronia in rete di sistemi *embedded*. Un sistema *embedded* basato su un sistema operativo Linux ci ha permesso inoltre di testare quanto un' applicazione di questo tipo potesse essere sviluppata su tali dispositivi. Si favorisce inoltre la portabilità tra diversi SOC che potrebbero in futuro rivelarsi più prestanti, sia a causa dell'obsolescenza naturale sia a causa di diversi criteri di scelta che potrebbero nascere con lo sviluppo dei lavori riguardanti le SmartGrid.

Durante la fase di sviluppo e di simulazione dell'algoritmo di gestione energetica si sono utilizzate cinque FoxBoard, una dedicata alla gestione del software dell'Energy Interface e le rimanenti incaricate alla simulazione degli Energy Gateway con l'applicazione di due moduli hardware sempre forniti dalla ACME SYSTEMS srl, descritti nel seguito. Tali moduli hanno lo scopo di creare una interfaccia e dare una visualizzazione del corretto funzionamento dell'algoritmo di gestione all'utente. Tutte le schede sono state connesse alla rete del dipartimento attraverso uno switch, così che tutte avessero un indirizzo IP statico che ha facilitato la connessione con le stesse. Inoltre la connessione alla rete dipartimentale e quindi alla rete Internet ha permesso di installare facilmente tutti i software necessari e l'utilizzo come test di server NTP esterni al dipartimento.

[3]SSH-Secure SHell è un protocollo di rete che permette di stabilire una sessione remota cifrata tramite interfaccia a riga di comando con un altro host di una rete

Figura 4.1: FoxBoard G20 senza involucro protettivo

4.2 GPIO e Moduli DAISY per simulazione

Per lo sviluppo del progetto si è reso necessario, come primo passo, creare degli applicativi software che permettessero una simulazione di consumo e fornissero una risposta visiva all' utente. Si è quindi scelto di utilizzare dei moduli a corredo delle FoxBoard per effettuare tale interazione, in particolare è stato installato su ogni scheda preposta alla simulazione di un Energy Gateway il modulo **DAISY-1** in figura4.2. Il modulo mette a disposizione un insieme di porte GPIO direttamente connesse con gli interrupt hardware del microprocessore e facilita l'installazione di periferiche esterne. Al DAISY-1 sono stati connessi i moduli **DAISY-5** (figura4.3) e **DAISY-11** (figura4.4), rispettivamente una schedina con otto tasti e una con otto led. In questo modo si hanno a disposizione otto interruttori che simulano l'accensione o lo spegnimento dei carichi. Un' opportuna accensione dei led del secono modulo indica all'utente lo stato del carico. Durante le fasi di test è stato definito che i primi sette led indicassero un carico e l'ottavo indicasse un dispositivo di produzione di energia; la configurazione di questi e i valori, tuttavia, si può modificare attraverso uno specifico file di configurazione che verrà descritto nel seguito.

Come già accennato in precedenza il simulatore creato può attivare da remoto in qualsiasi momento i carichi ricreando, nei nodi interessati, una pressione dei

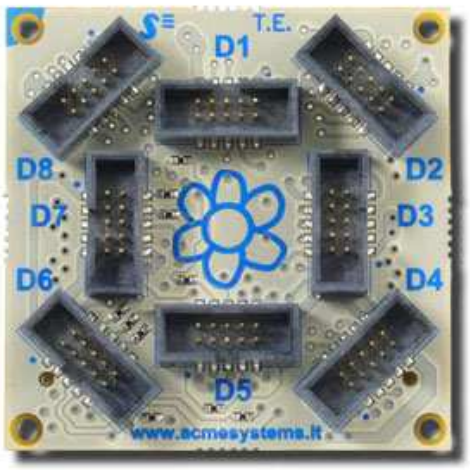

Figura 4.2: Modulo Daisy-1

Figura 4.3: Modulo Daisy-5

tasti sul modulo **DAISY-5** in modo totalmente trasparente all'applicativo di gestione sviluppato.

Per la connessione dei moduli sono stati usate le porte D2 per il modulo led e D5 per il modulo dei tasti in quanto queste mettono a disposizione tutti pin con un Kernel ID, indispensabile per la gestione degli interrupt provenenti principalmente dal modulo tasti **DAISY-5**.

Figura 4.4: Modulo Daisy-11

Pin sul Modulo	KernelIDFunzione
D2.1	3.3 V (alimentazione)
D2.2	63
D2.3	62
D2.4	61
D2.5	60
D2.6	59
D2.7	58
D2.8	57
D2.9	94
D2.10	GND

Tabella 4.1: Porta D2: pin e riferimenti Kernel

Pin sul Modulo	KernelIDFunzione
D5.1	3.3 V (alimentazione)
D5.2	76
D5.3	77
D5.4	80
D5.5	81
D5.6	82
D5.7	83
D5.8	84
D5.9	85
D5.10	GND

Tabella 4.2: Porta D5: pin e riferimenti Kernel

4.3 Network Time Protocol

In un sistema di comunicazione e scambio dati come quello sviluppato è molto importante avere un riferimento temporale preciso e aggiornato, per questo in tutte le FoxBoard utilizzate negli esperimenti è stato installato un protocollo di sincronizzazione dei timer interni, che ha permesso di tenere aggiornate la data e l'ora del sistema in modo automatico.

Lo scopo del protocollo NTP è quello di trasmettere le informazioni relative alla gestione del tempo di alcuni nodi principali ad altri server o client attraverso Internet ed effettuare una verifica incrociata dei clock per mitigare gli errori nei riferimenti locali, inoltre esso si basa sul rilevamento dei tempi di latenza dei pacchetti sulla rete ed è in grado di sincronizzare gli orologi di diversi computer su una WAN entro un margine di 5 millisecondi in presenza di sistemi embedded con scarse capacità [7]. Il protocollo prevedere diversi server NTP organizzati un una struttura gerarchica dove, riferendosi alla figura 7.2, lo strato 1 è sincronizzato su una fonte temporale quale un orologio atomico e diffonde queste informazioni verso gli strati più bassi. I nodi di ogni strato si sincronizzano confrontando il proprio orologio con quello dei sistemi allo strato superiore ed effettuano opportune modifiche.

Nella rete utilizzata, tuttavia, non era necessaria una realizzazione di server Ntp, ma dei soli client, è stata quindi installata una versione ridotta chiamata ***NTPdate*** che richiede solamente la data e l'ora a server definiti dall'utente e setta di conseguenza il timer interno. Nptdate effettua un aggiustamento del clock in due modi, se trova una differenza di tempo superiore a 0.5 secondi chiama la routine di sistema *settimeofday()* aggiornando il clock time ad un nuovo valore, in caso invece l'errore temporale fosse inferiore al mezzo secondo allora utilizza il metodo *adjtime()*che permette una correzione graduale del clock, accelerando o rallentando, che porta ad una diminuzione dell'errore tra i tempi rilevati tra il server e il client.

Per meglio capire il funzionamento del protocollo NTPdate, nonchè del PTP che verrà successivamente descritto, è utile introdurre alcune definizioni che riguardano da vicino i riferimenti temporali :

1. **stabilità del clock:** indica quanto si riesce a mantenere una frequenza costante;

2. **accuratezza:** è maggiore quanto più il tempo tende ai valori assunti dagli standard nazionali;

3. **precisione:** indica in che modo stabilità e accuratezza possono essere mantenute nel tempo senza la presenza di un sistema di cronometraggio;

4. **offset:** è la differenza in tempo che intercorre tra due clock;

5. **skew:** differenza in frequenza tra due clock;

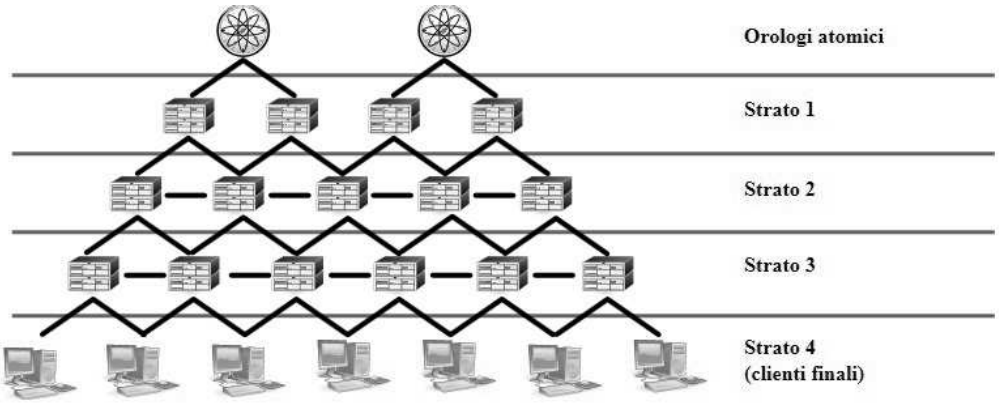

Figura 4.5: Architettura NTP

6. **drift:** è la variazione di skew che si manifesta negli orologi reali

NTPdate quindi si basa sul calcolo di tre grandezze: *clock offset, roundtrip delay e dispersion*, valutate relativamente ad un clock di riferimento comune:

- Clock offset: rappresenta l'ammontare dell'aggiustamento che il clock locale deve subire per ritrovarsi in corrispondenza con quello di riferimento;
- Roundtrip delay: fornisce una stima del tempo necessario ad un messaggio inviato per arrivare a destinazione;
- Dispersion: fornisce il valore dell'errore tra il clock locale ed il riferimento.

4.4 Precision Time Protocol - IEEE 1588

Durante lo sviluppo del lavoro ci si è resi conto che il semplice protocollo NTP non garantiva una sufficiente sincronizzazione tra i clock delle FoxBoard. Si è valutato quindi un meccanismo che permettesse una accurata gestione dello skew e dell'offset tra in clock dei dispositivi. Durante le prove,infatti, i nodi già dopo alcuni minuti di funzionamento, non rispettavano le finestre temporali imposte dal ciclo di lavoro. Si è quindi preso in considerazione l'uso di PTP-D trovando una efficace implementazione sotto licenza GNU GPL[4]. Basato sullo standard IEEE 1588 esso definisce un protocollo in grado di far ottenere una precisa sincronizzazione dei clock dei dispositivi.

Un volta installato il demone PTP-D sui nodi e fatto avviare, il primo passo che compie l'algoritmo è quello di identificare tra tutte le entità il clock di riferimento attraverso un algoritmo di elezione chiamato **Best Master Clock**

[4]Licenza che rende disponibile il codice sorgente e la possibilità di modifica in base alle proprie esigenze

Algorithm, ottenuto tale risultato tra i nodi vengono scambiati dei *Sync Message* che permettono a tutti i nodi di convergere verso un identico riferimento temporale e di mantenere con esso un offset limitato. Al fine di ottenere una miglior sincronizzazione è richiesta dallo standard la seguente importante ipotesi, ***ritardo della rete tra master e slave simmetrico***. Tale ipotesi tuttavia non è vera nel contesto applicativo studiato ma anche considerando una diversa latenza nella gestione dei pacchetti da parte del sistema operativo nelle FoxBoard dovuto a diverso carico computazionale, lanciando il processo PTPD si ha un netto miglioramento dell'offset tra i nodi. Dai dati sperimentali raccolti in precedenti lavori risulta come tale approccio mantenga l'offset tra i clock un ordine di grandezza inferiore a quello necessario [*ms*] nella sincronizzazione del ciclo di lavoro.[6] [8]

Capitolo 5

Le basi, Socket e Thread

La descrizione del codice creato nel prossimo capitolo necessita la spiegazione di alcune basi offerte dal linguaggio ***C***, verranno qui introdotte le principali primitive che sono state largamente usate nel lavoro di tesi.

5.1 Socket

In questo lavoro un fondamento importante nello sviluppo delle comunicazioni sono i ***Socket*** una particolare astrazione software che permette ai processi messi in comunicazione di inviare e ricevere dati all'interno dello stesso nodo o verso nodi in rete attraverso un descrittore. I *socket* sono nati intorno agli anni '80, e la struttura base di funzionamento è di tipo Client/Server.

Esistono quattro tipologie di *socket*:

- *socket* che utilizzano i protocolli di Internet come TCP e UDP;
- gli *Unix Domain Socket*, usati in ambienti POSIX per la comunicazione in locale dei processi;
- *socket* che utilizzano i protocolli di Xerox Network System[1] ;
- *socket* che utilizzano i protocolli della International Standard Association (riferimento al modello ISO/OSI).

Nel linguaggio ***C*** la libreria da importare è <***sys/socket.h***>, che offre i metodi per la gestione dei *socket* , per la definizione delle porte e per la creazione di connessioni di diverso tipo tra le parti. In aggiunta a tali metodi si possono utilizzare diverse funzioni per la manipolazione delle comunicazioni, quelle più utilizzate in questo lavoro sono principalmente:

[1] Xerox Network Services (XNS) è un protocollo sviluppato da Xerox che prevedeva anche funzioni come l' integrità di pacchetto e chiamate *remote procedure call*. XNS ha preceduto e influenzato lo sviluppo dello standard Open Systems Interconnect (OSI).

- inet_aton_HOSTNAME: riceve una stringa contenente il nome di un host e la traduce in una stringa di 4 byte, in pratica se riceve un argomento *"prova.it"* restituisce l'indirizzo IPV4 di tale hostname, in caso non si riesca ad effettuare la conversione la stringa di ritorno e indefinita;
- inet_ntoa_IP_ADDRESS: riceve una stringa e la converte nel classico formato di rete;
- INADDR_BROADCAST: restituisce in formato IPV4 l'indirizzo di broadcast della sottorete al quale siamo connessi.

Segue una breve descrizione delle principali funzioni dei *socket* utilizzate nel progetto utili per spiegare come questi sono stati utilizzati e come questi interagisco lato client e server. Sono state inserite quindi le righe di codice utili alla comprensione e necessarie all'avvio di un *socket*.

```
int socket(int dominio, int tipo, int protocollo)
```

con questa istruzione si crea un *socket* bloccante e se ne restituisce un descrittore di tipo ***int*** che sarà utilizzato per la comunicazione in ogni istruzione che permette lo scambio di dati. Il campo dominio indica la famiglia di protocolli che viene usata dalle comunicazioni, nel lavoro svolto è stato scelto di utilizzare la famiglia AF_INET che sono i protocolli usati in internet. Nel campo ***tipo***, si definisce la tipologia di comunicazione , cioè se comunicazione è di tipo TCP o UDP (nel caso di scelta della famiglia di protocolli internet), nello specifico, nel lavoro svolto sono stati utilizzati entrambi i metodi, il primo per la comunicazione tra FOX e FOX o tra FOX e PC per il impartire ordini diretti, il secondo per la comunicazione in *broadcast* a tutta la rete di messaggi o dati.

```
int bind(int descrittore_socket, struct sockaddr* indirizzo, int
                    lunghezza_record_indirizzo)
```

La funzione ***bind*** assegna un indirizzo al *socket* e una porta di ascolto specificati nella struttura sockeaddr che nel caso di comunicazione con i protocolli internet cambia tipologia in sockaddr_in.

```
int close (int descrittore_socket)
```

la funzione *close*, chiude il *socket* di comunicazione il cui descrittore è passato come parametro

```
int connect (int descrittore_socket, struct sockaddr*
  indirizzo_server, int lunghezza_record_indirizzo)
```

la funzione cerca di effettuare un connessione tra il socket creato e un secondo socket in ascolto ad un determinato indirizzo passato con struct sockaddr* indirizzo_server

```
int listen (int descrittore_socket, int dimensione_coda)
```

se il *socket* è in ascolto, può ricevere delle richieste di connessione; mentre viene servita una di queste richieste, ne possono arrivare altre e vengono messe in una coda di attesa ***listen*** si occupa di definire la dimensione massima di questa coda.

```
int accept (int descrittore_socket, struct sockaddr* indirizzo,
                int* lunghezza_record_indirizzo)
```

in caso di *server socket* una volta creato, collegato e messo in ascolto, ***accept*** prende la prima connessione disponibile sulla coda delle connessioni pendenti, crea un nuovo socket con le stesse proprietà di quello rappresentato da descrittore_socket e restituisce un descrittore di connessione che identifica univocamente la comunicazione.

```
int send (int descrittore_socket, const void* buffer, int
        lunghezza_messaggio, unsigned int opzioni)
```

Con ***send*** è possibile inviare messaggi dal *socket* rappresentato dal descrittore con cui è connesso. Questa funzione può essere usata solamente in presenza di una connessione. Il parametro buffer contiene il messaggio e deve avere una dimensione non inferiore a lunghezza_messaggio.

```
int recv (int descrittore_socket, const void* buffer, int
         dimensione_buffer, unsigned int opzioni)
```

recv serve a ricevere messaggi da un *socket* e può essere usata solamente in presenza di una connessione. Il risultato della chiamata a questa funzione, in caso di errore, è -1, altrimenti è il numero di caratteri ricevuti. Il messaggio ricevuto è contenuto nella memoria puntata da buffer.

5.2 Thread e organizzazione

L'organizzazione del lavoro è stata effettuata tramite la creazione di diversi ***Thread*** ognuno con un compito specifico, nello sviluppo infatti è nata la necessità di generare processi velocemente e con un basso overhead sulle risorse hardware, con minor utilizzo di memoria e la possibilità di condividere i dati durante esecuzione.

I vantaggi nell'uso dei thread in ambiente Linux sono:

- sono un flusso ***indipendente*** di istruzioni;
- possono ***condividere risorse*** con altri thread dello stesso processo;

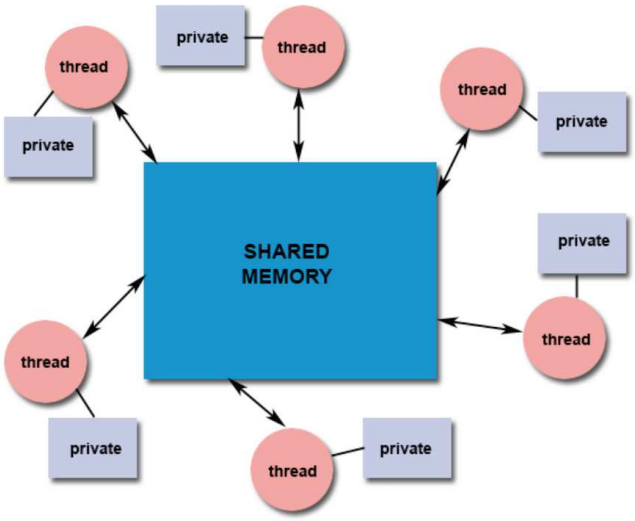

Figura 5.1: Struttura dei Thread a memoria condivisa

- sono più ***leggeri*** della primitiva *fork()*[2];
- condividono lo stesso spazio di indirizzamento;
- i cambiamenti di un thread sono visibili dagli altri dello stesso processo
- è possibile ***sincronizzare*** l'esecuzione dei compiti.

Il principale modello di programmazione multithreaded scelto rispecchia sostanzialmente la logica *pipeline*, dove il lavoro da compiere è suddiviso in una serie di operazioni ciascuna delle quali è gestita in modo concorrente da thread diversi. Per la condivisione dei dati da parte dei thread si è scelto il modello a memoria condivisa, dove tutti accedono alla stessa area di memoria condivisa come mostrato in figura 5.1.

L'interfaccia di programmazione dei thread in *UNIX* è stata standardizzata da *IEEE* nel 1995 con lo standard *POSIX*, e tutte le implementazioni fedeli a questo standard si chiamano POSIX threads o *pthreads*, la libreria che definisce un set di tipi in ***C*** e primitive può essere importata dall'*header pthread.h*.

5.3 Creazione e cancellazione di un Thread

La creazione di un nuovo thread viene effettuata tramite l'istruzione

[2]La primitiva *fork()* nel linguaggio C trasforma un singolo processo in due processi identici riconoscibili come processo padre e processo figlio per questa operazione inoltre viene creato un nuovo spazio di indirizzamento e i due processi possono proseguire la loro esecuzione indipendentemente l'uno dall'altro.

```
int pthread_create(thread,attr,start_r,arg)
```

che inoltre lo rende eseguibile; i parametri per l'inizializzazione sono:

- thread(output): di tipo pthread_t, è un identificatore del thread creato;
- attr(input): di tipo pthread_attr_t, serve a settare gli attributi del thread;
- start_r (input): puntatore alla funzione ***C*** (starting routine) che verrà eseguita una volta che il thread è creato;
- arg(input): argomento che può essere passato alla funzione (obbligatorio il casting a void *).

La chiusura di un thread invece può essere effettuata dal thread stesso o dal thread che l'ha inizializzato, nel primo caso si usa l'istruzione :

```
int pthread_cancel(pthread_t thread)
```

che invia un segnale di terminazione al thread coinvolto, tuttavia la terminazione non è sincrona con il valore di ritorno di tale istruzione e viene effettuata appena è possibile, di solito appena un thread entra in uno stato di attesa o richiede accesso ad un semaforo.

Nel secondo caso, cioè quanto un thread termina la propria esecuzione si usa l'istruzione:

```
int pthread_exit(void *value_ptr)
```

eseguita immediatamente ed informa gli eventuali altri thread in attesa passando il parametro **value_ptr*.

5.4 Sicronizzazione tra Thread

Per accedere in modo efficace ad un area di memoria condivisa, i thread hanno bisogno di sincronizzare i loro accessi tramite i ***mutex*** che permettono di organizzare le modifiche e le letture dei dati. I mutex quindi sono utili per mantenere un integrità delle variabili condivise e non perdere modifiche dovute a letture e scritture casuali.

I sono mantenuti da una struttura ***pthread_mutex_t*** che va allocata e inizializzata con le seguenti istruzioni :

```
pthread_mutex_t c = PTHREAD_MUTEX_INITIALIZER int
pthread_mutex_init( pthread_mutex_t mutex, const
          pthread_mutexattr_t attr);
```

dove *attr* è NULL per caricare i parametri di default. I mutex si usano richiedendo, ottenendo e rilasciando le risorse, l'istruzione che serve per il blocco dei un mutex è

```
int pthread_mutex_lock (pthread_mutex_t mutex);
```

o

```
int pthread_mutex_trylock (pthread_mutex_t mutex);
```

dove la prima blocca il thread che la esegue fino a che non gli viene dato accesso alla sezione critica, la seconda tenta una di occupare la risorsa e se già un altro thread ha bloccato il mutex non si blocca, salta la sezione critica e continua la propria esecuzione.In entrambi i casi un thread dopo aver bloccato un mutex deve rilasciarlo con l'istruzione

```
int pthread_mutex_unlock (pthread_mutex_t mutex).
```

L'utilizzo si questi mutex deve essere molto preciso per evitare che durante l'esecuzione un thread vada in stallo richiedendo un mutex già bloccato da se stesso e mai rilasciato.

La sincronizzazione fatta in questo modo è di tipo passiva, tuttavia esiste un modo far attendere un thread su un mutex fino al verificarsi di una condizione, in questo modo si possono modificare i dati di un'area di memoria lanciare un segnale al thread in attesa perchè effettui operazioni su tali dati prima che altri thread li modifichino nuovamente. Questa procedura si può effettuare con le ***condition variable***, ed il loro utilizzo prevede che precedentemente si richieda il blocco di un mutex e poi si esegua l'istruzione:

```
int pthread_cond_wait(pthread_cond_t cond, pthread_mutex_tmutex);
```

che rilascia il mutex ma mantiene il thread in attesa. Un secondo processo entrato nella sezione critica può quindi risvegliare il thread in attesa con un signal:

```
int pthread_cond_signal(pthread_cond_t cond,
          pthread_mutex_t*mutex);
```

per successivamente rilasciare il mutex ed uscire dalla sezione critica.

Capitolo 6

Moduli software

Il presente capitolo spiega in dettaglio i moduli software sviluppati e fornisce una definizione passo passo delle operazioni svolte. Tuttavia tale descrizione non vuole addentrarsi a spiegare il codice riga per riga ma delinea in modo semplice i criteri utilizzati e le scelte fatte durante la stesura. L'ordine di presentazione sarà diviso per tipo di nodo e seguirà quello utilizzato nello sviluppo, dapprima cioè saranno introdotti gli applicativi più vicini all'hardware ed in seguito quelli di gestione del ciclo di lavoro. I moduli sviluppati si dividono in due gruppi, quelli definiti per lo sviluppo degli *energy gateway* e quello definiti per l'*energy interface*, oltre a questi è presente il codice del simulatore indipendente dagli altri che nelle prove è stato di solito eseguito nel nodo *energy interface*. In appendice sono riportate le istruzioni per la compilazione e la gestione dei file del programma.

6.1 I metodi per gestire i Socket

I metodi per la gestione diretta dei socket sono stati i primi ad essere creati e sono un gruppo di funzioni che permettono la creazione di:

- server broadcast;
- client broadcast;
- metodi per l'invio di messaggi in rete;
- metodo di aggiornamento dei servizi trovati.

Gli header di riferimento sono *ss_for_eg.h* e *sc_for_eg.h*.

6.1.1 Broadcast_Server & Client

Il metodo deve essere richiamato quando un qualsiasi thread attiva un server socket, esso riceve in ingresso una struttura che contiene i parametri per la

creazione del broadcast server e il messaggio che questo dovrà diffondere nella rete. Nel lavoro sviluppato questo tipo di server vengono usati per annunciare alla federazione che un nodo attivo ha un determinato servizio in attesa su una porta, così che nuovi entranti possano conoscere velocemente i nodi attivi e quindi instaurare le connessioni. Un nodo remoto può riceve questi messaggi attraverso un thread *broadcast_client*, che metterà a disposizione su dei file temporanei tutti i server visti con riportate le seguenti informazioni:

tipo_servizio, mac_address, porta_ascolto, indirizzoIP, broadcastIP

Tale informazioni vengono poi utilizzate per i thread del nodo che in questo modo possono collegarsi a qualsiasi servizio.

6.1.2 Metodi invio di messaggi

I metodi per l'invio di messaggi attraverso i socket sono di due tipi, il primo per l'invio semplice di un messaggio tra client e server chiamato *SpedisciMSG*, il secondo per l'invio da server a client *SpedisciMSGserver*. Nel secondo caso infatti oltre che al descrittore del socket è necessario conoscere anche il descrittore della connessione altrimenti il messaggio non verrà recapitato causando un errore[1]. I metodi servono principalmente per gestire gli errori e nel caso succedessero ritentano l'invio entro pochi millisecondi. Entrambi ritornano un valore intero che identifica il descrittore del socket in caso di successo ed l'intero -1 in caso di insuccesso dell'invio.

6.2 I Thread dell'Energy Gateway

L'organizzazione del software negli *energy gateway* è mostrato in figura 6.1, e mostra come dal main vengano lanciati in esecuzione otto diversi thread, che sincronizzandosi attraverso semafori e condizioni di attesa eseguono tutte le operazioni necessarie per soddisfare gli obiettivi di progetto. La presentazione seguirà la figura 6.1 da sinistra verso destra rispecchiando l'ordine di avvio dei thread importante per una gestione efficace dei semafori, evitando di creare all'avvio delle situazioni di stallo, ad eccezione del thread Logger che per motivi di chiarezza verrà lasciato per ultimo.

6.2.1 Il metodo MAIN

Il Main ha il compito di inizializzare tutte le strutture e le variabili che sono condivise tra i thread esso non segue un file di configurazione in quanto saranno i singoli processi a caricare dai propri file i parametri standard di funzionamento. Dopo una prima fase di avvio quindi il Main si prende in carico di sostenere le funzioni di base dei nodi con l'ausilio dei thread che simulano l'hardwar. Infatti

[1] la funzione `send()` ritorna il valore -1 invece della lunghezza in byte del messaggio inviato

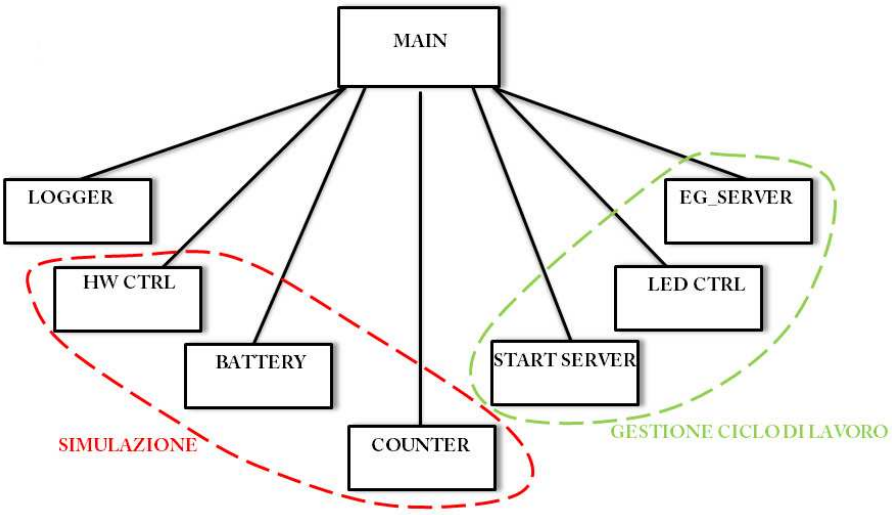

Figura 6.1: Illustrazione dell'organizzazione dei Thread

appena avviato un nodo *energy gateway* non è connesso a nessuna adiacenza e a nessun nodo *energy interface*, il main quindi mantiene attivi i thread di controllo che sono in attesa di connessioni esterne. In questo modo il nodo presenta attive tutte le funzioni di alimentazione dei carichi, gestione della batteria, attivazione della microgenerazione anche se il funzionamento di questi non è influenzato dalle dinamiche della rete. Si sottolinea che i thread creati sono concorrenti e la loro esecuzione è simultanea, o comunque schedulata dal sistema operativo.

6.2.2 HW_CONTROL

Il thread HW_CONTROL ha il compito di gestire i moduli GPIO e gestisce gli interrupt che arrivano dall' hardware Daisy. Gestisce quindi tutti gli aspetti legati all'interazione dell'utente con il sistema embedded.

Come si nota dalla figura 6.2 del flusso di operazioni, come prima azione l'HW_CONTROL inizializza le porte GPIO per la corretta gestione dei moduli Daisy, tale inizializzazione viene fatta semplicemente aprendo per ogni porta un descrittore file e impostando ad un opportuno valore il registro associato. In particolare le porte connesse al modulo dei led vengono inizializzate tramite queste istruzioni:

```
fopen(sysclassgpioexport, ab);
sprintf(set_value, %d, int interrupt);
```

dove l'interrupt è il valore del pin nel microprocessore mostrati in tabella 4.1. Dopo tale inizializzazione di passa alla definizione del tipo di direzione che ha la porta, nel caso dei led la porta è out, tale operazione di esegue con le seguenti istru-

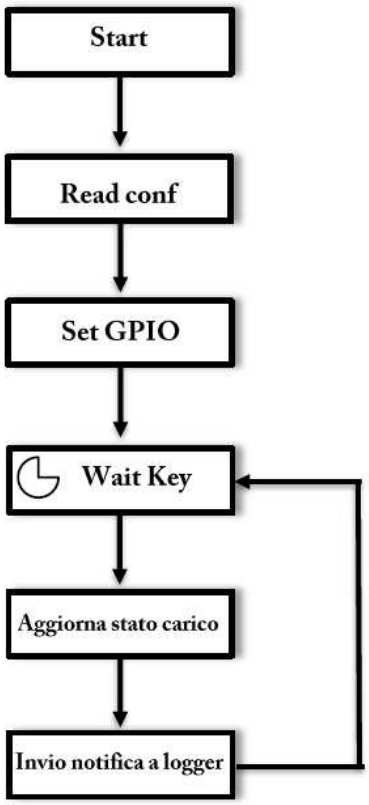

Figura 6.2: Flusso di lavoro del thread HW_CONTROL

zioni:

```
fopen(/sys/class/gpio/gpio%d/direction, rb+);
strcpy(set_value, out);
fwrite(&set_value, sizeof(char), 3, fp);
```

dove si nota come la direzione vine impostata su **out**.

Per il modulo che gestisce i tasti si eseguono le stesse operazioni con la sola modifica che la direzione è in ingresso e che si deve definire quale fronte considerare alla pressione del tasto, si è deciso di considerali entrambi e via software gestire quale azione intraprendere.

Ad ogni ciclo inoltre i descrittori dei file aperti si inseriscono in un array che servirà alla libreria *poll.h*. I metodi di tale libreria vengono sfruttati per mettere in attesa il thread dopo ogni pressione dei tasti, in questo modo il thread non carica eccessivamente la CPU ma viene risvegliato solamente quando si esegue un interrupt. Ad ogni risveglio quindi l'HW_CONTROL prende il

tempo di sistema, va a vedere quale tasto è stato premuto e aggiorna nella memoria condivisa attraverso l'uso dei semafori lo stato dei led, in particolare aggiorna un semplice array di interi che rispecchia lo stato del carico (0 spento, 1 acceso). Nel caso sia attivo il logger, si invia una notifica di scrittura e si fornisce la stringa contenente i dati dell'evento accaduto. Fatte queste operazioni l'HW_CONTROL si rimette in attesa della pressione di altri tasti sul modulo Daisy-5.

6.2.3 BATTERY

Il thread BATTERY, come da nome, simula il comportamento di un eventuale modulo di storage presente nel nodo, il suo funzionamento è semplice, dopo una prima fase di inizializzazione legge i parametri della batteria nel file di configurazione */etc/battery.conf*, se tale file esiste segue le indicazioni contenute se non esiste o le indicazioni sono errate, ignora i dati e di conseguenza il nodo non avrà un modulo storage disponibile. La sua funzione principale è aggiornare la quantità di carica disponibile, la risoluzione di tale operazione è del secondo, allo scoccare di ogni secondo il thread si risveglia e calcola in base ai valori richiesti di potenza, in entrata o uscita, l'energia residua controllando che questa non superi la carica massima o la minima. In pratica al superamento di tali soglie vengono impostati dei flag che non permettono più la carica o se lo storage non ha energia, non lo rendono più disponibile al nodo. Se il *logger* è attivo, anche questo thread lo risveglia periodicamente inviandogli lo stato della batteria.

6.2.4 COUNTER

Il thread COUNTER contabilizza l'energia prodotta e consumata dal nodo, esso crea un file permanente sullo storage del nodo ogni secondo e indipendentemente dalla durata del ciclo di lavoro, vengono riportati i valori di energia consumati basandosi sulla potenza media scambiata. Tale thread si basa sul controllo, nell'area di memoria condivisa, di quali carichi sono accesi e associando ad ogni carico i valori di potenza istantanea con la convenzione degli utilizzatori, calcola le energie consumate in wattora. Il file creato si trova in */var/log/energy_node*.

6.2.5 START_SERVER

Questo è il primo *thread* descritto che utilizza i *socket* per le comunicazioni di rete, ed è la base di partenza per la creazione di ulteriori altri *thread* figli. Il suo funzionamento si basa sui parametri passati in ingresso dal main durante la creazione, è infatti un *thread* che utilizza i metodi per i *socket* server definiti in precedenza. Di tale *thread*, durante l'esecuzione esistono due istanze. La prima, inizializzata opportunamente, attende alla porta 6010 la connessione del simulatore, ed una seconda che attende alla porta 6020 la connessione delle adiacenze.

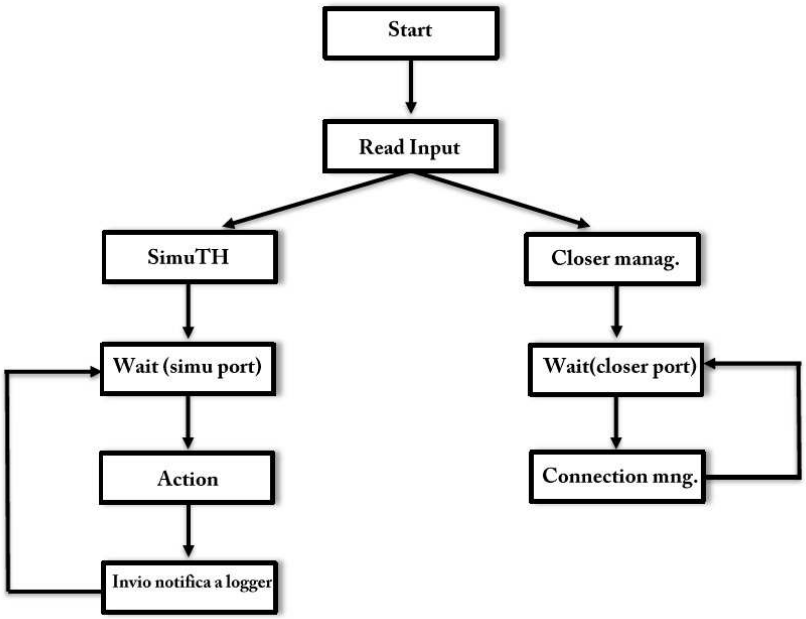

Figura 6.3: Flusso di lavoro del thread start_server

L'istanza che gestisce la connessione con il simulatore una volta instaurata la connessione richiama il metodo *simuTH* che riceve in ingresso le istruzioni del simulatore la cui descrizione si trova nel seguito del capitolo. L'istanza che invece gestisce la connessione con le adiacenze rimane semplicemente in attesa di connessioni da altri *energy gateway*, e quindi salva in una coda i descrittori dei *socket* e il descrittore della connessione corrispondete al nodo remoto. Tale lista è utilizzata dal ciclo di lavoro quando deve comunicare i dati alle adiacenze. In pratica, l' istanza di *start_server* aggiorna ad ogni nuova connessione esterna la lista delle adiacenze. In particolare, se tale metodo viene inizializzato con i parametri di gestione della simulazione il nome con il quale il *main* lo identifica è *threadSERV*, se è inizializzato per la gestione delle adiacenze il main lo identifica con in nome *threadEGCLOSER*.

6.2.6 EG_SERVER

Il funzionamento di tale thread è abbastanza complesso e coinvolge nell'esecuzione anche un metodo indipendente chiamato *eg_work* incaricato di eseguire il lavoro del ciclo di gestione. Se si scorre il codice di *eg_work* ci si rende subito conto che rispecchia la struttura del ciclo di lavoro che era stato definito negli obiettivi del progetto.

Partendo dalla descrizione di *EG_SERVER*, dopo una prima fase di inizializzazione delle variabili interne al thread si va ad aprire il file di configurazione

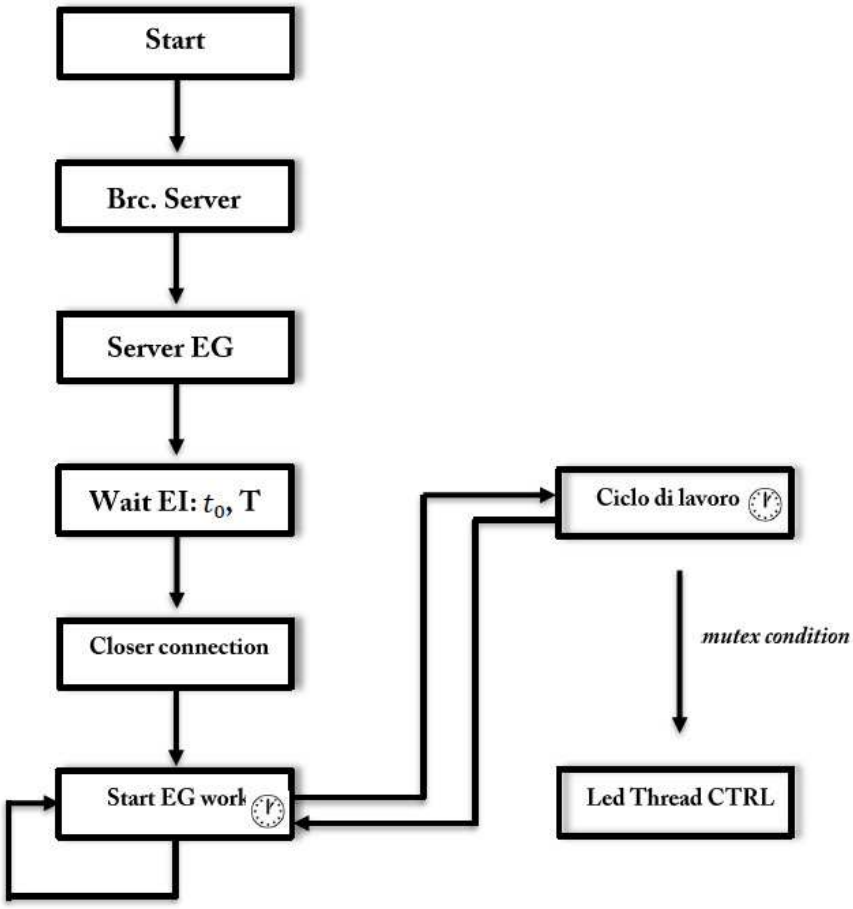

Figura 6.4: Flusso di lavoro del thread eg_server

contenente i parametri dei carichi, in questo file si trova, con la convenzione degli utilizzatori, il valore della potenza istantanea consumata o prodotta assieme alle priorità dei carichi. Una volta caricati i valori da tale file li memorizza in un array condiviso tra tutti i thread. Successivamente crea un socket TCP per la comunicazione ed un socket UDP per l'annuncio in rete della presenza del nodo. Il messaggio di broadcast UDP sarà recepito dall'*energy interface* che ne sfrutterà i parametri contenuti per instaurare le comunicazioni. La connessione con l'*energy interface* fa procedere il thread che altrimenti rimane in attesa sul socket. Una volta che *l'energy interface* richiede la connessione tale thread dopo una fase si autenticazione [2], analizza i primi messaggi in arrivo sul socket della connessione. L'ordine dei messaggi è irrilevante ma il thread si aspetta di

[2]Nelle prime versioni per prova è stato creato un metodo per l'autenticazione dei nodi che si basa sul mac address, tale autenticazione è stata mantenuta anche se non pertinente con il lavoro. In appendice se ne trova una breve descrizione.

ricevere due messaggi, uno contenente la lista delle adiacenze alle quali si dovrà collegare, un messaggio contenente i parametri del ciclo di lavoro t_0 e $\boldsymbol{T}$ e un messaggio che arriva per ultimo quando cioè *l'energy interface* è pronto a ricevere nella federazione il nuovo nodo che sostanzialmente informa il nodo di continuare con le operazioni. Quindi si esegue la connessione alle adiacenze e si prendono i riferimenti locali di data e ora[3].

Si impostano quindi i segnali di sistema, in particolare si gestisce l'arrivo di futuri segnali dal timer con l'istruzione `signal(SIGALRM, eg_work)` che all'arrivo di un segnale SIGALRM richiama il metodo ***eg_ work***. Il segnale SIGALRM viene generato da un allarme ogni $\boldsymbol{T/5}$ millisecondi e quindi richiama le cinque fasi del ciclo di gestione. Purtroppo non è possibile passare parametri all'avvio del metodo e questo ha come conseguenza che tutte le variabili che devono essere gestite tra il thread e il metodo richiamato devono essere definite globali; l'accesso a tali variabili è quindi gestito da semafori. Il richiamo del metodo permette ad ogni iterazione di entrare in una specifica fase del ciclo di lavoro, un flag ricorderà al metodo in quale fase entrare e quindi eseguire le operazioni opportune. L' EG_WORK quindi è strutturato in cinque parti dove la quarta è la ***fase di controllo e azione***, tale fase è la più importante e ne verrà in seguito descritto il funzionamento nei particolari.

6.2.7 LED_CTRL_THREAD

Questo thread fa da tramite tra le decisioni prese dal ciclo di gestione e l'azione di spegnimento effettivo dei carichi, il funzionamento si basa sulla sincronizzazione tramite semafori e *condition*. Il thread come prima operazione richiede il blocco di un semaforo e subito si mette in attesa di un segnale per proseguire il lavoro liberando allo stesso tempo le risorse condivise per il ciclo di lavoro. Ad ogni decisione di aggiornamento dei carichi nel ciclo di lavoro tale thread riceve un segnale e prosegue accedendo alle variabili condivise dove trova le azioni da intraprendere. In particolare per tutti il carichi connessi controlla che abbiano il premesso di rimanere accesi (permesso modificato dal ciclo di gestione), e nel caso fossero attivi li spegne e memorizza che il carico è stato spento dal sistema e non dall'utente. Questa operazione serve per riabilitare i carichi scollegati nel qual caso il ciclo di lavoro decidesse che le condizioni sono favorevoli ad un aumento del consumo del nodo. La riaccensione tuttavia è lenta, ad ogni ciclo di gestione solamente un carico, se necessario, viene riattivato, questo per limitare le oscillazioni nei consumi che un controllo di questo tipo può innescare.

Anche questo metodo interagisce con il logger e ad ogni modifica dello stato dei carichi comunica attraverso le *condition* appena citate passando il riferimento temporale e l'azione appena compiuta.

[3] Al lancio dell'applicativo di gestione, data e ora sono già sincronizzati con gli altri nodi presenti attraverso PTPD e NTP.

6.2.8 LOGGER

Il logger è un semplice thread che sfruttando le *condition* dei semafori attende che qualche altro thread del processo lo avverta che c'è un nuovo messaggio da riportare. Esso crea un file in */var/log/logger_file#* il simbolo # sta ad indicare che ad ogni nuova simulazione chiude il file e ne crea un successivo. In questo modo andando a prelevare i log si possono osservare tutte le azioni compiute nei carichi con un *timestamp*. La creazione di questo thread ha permesso a tutti gli altri di sgravarsi dalle operazioni di scrittura su disco velocizzandone l'esecuzione.

6.3 Ciclo di gestione nell'Energy Gateway

Il ciclo di gestione segue l'ordine mostrato in figura 3.2 le parti centrali sono quelle di azioni e di analisi dei dati. L'analisi in particolare permette di decidere in base alle comunicazioni con le adiacenze in che stato si trova il nodo. Dalle adiacenze infatti arrivano i dati già bilanciati sulla produzione e sul consumo del nodo, tale valore può essere maggiore,minore o uguale a zero. Il valore rispecchia sostanzialmente la potenza istantanea disponibile al nodo, quindi quanto il nodo può cedere alla rete. Se il valore è zero il nodo è all'equilibrio, se è inferiore a zero il nodo assorbe potenza dalla rete, se il valore è positivo il nodo consuma meno di quello che produce e la sua energia può essere utilizzata dalle adiacenze. La fase di analisi basa le proprie decisioni sul totale che assume la somma della potenza disponibile tra le adiacenze e la propria, i possibili stati sono:

- ***down***: stato del ciclo di gestione che si verifica quando il consumo della federazione è superiore al consumo massimo imposto *dall'energy interface* oppure quando il singolo nodo supera il consumo massimo imposto al singolo pur non avendo uno sforamento da parte del totale delle adiacenze;
- ***prod+cons***: stato in cui è rilevato che il nodo con le proprie adiacenze ha una produzione superiore al consumo interno;
- ***can_consume***: in questo stato il controllo entra se rileva che le adiacenze e lo stesso nodo nel quale è eseguito il controllo non superano il massimo imposto *dall'energy interface*;
- ***ok***: stato in cui le adiacenze sono all'equilibrio, l'insieme tutti i nodi ai quali si è connessi scambia potenza nulla o è in linea con le richieste dell'*energy interface*;
- ***go***: fase in cui per errori di calcolo o di comunicazione l'algoritmo non è stato in grado di definire uno degli stati precedenti;
- ***batt_d***: in questo stato si può entrare solamente se il nodo riceve i comandi *dall'energy interface* e non è connesso alle adiacenze, il nodo semplicemente

riceve il comando di utilizzare al massimo lo storage, questo per affrontare dinamiche di rete che si possono verificare.

Lo stato successivamente viene utilizzato per definire le azioni da eseguire, esse considerano tutti i parametri del nodo come energia residua e potenza massima uscente dallo storage, potenza prodotta dai moduli di microgenerazione e potenza consumata in quel momento dai carichi, oltre alla quantità di potenza istantanea resa disponibile dall'energy interface.

Nel seguito si elencano per ogni stato le diverse azioni intraprese dall'algoritmo:

- ***Stato down:*** il primo controllo si esegue sulla quantità di potenza istantanea assorbita dal modulo storage, se è maggiore di zero per cercare di bilanciare velocemente il carico del nodo viene portata a zero smettendo di fatto di caricare lo storage. Si controlla anche se la disponibilità dello storage o dell'impianto di generazione del nodo o delle proprie adiacenze riesce a sopperire alla richiesta, se è possibile rientrare nel valore massimo di potenza assorbita tramite lo storage allora si richiede la quota parte di potenza necessaria fino al massimo offerto in uscita dal dispositivo. A questa potenza viene sommata quella generata in quel momento dalle adiacenze e dal nodo. Questa scelta è fatta in quanto lo spegnimento dei carichi deve essere l'ultima azione da intraprendere per bilanciare la rete. Se anche considerando storage e microgenerazione non si scende al valore richiesto ci si appresta a spegnere i carichi basandosi sulle priorità che si trovano nel file di configurazione caricato all'avvio. Lo spegnimento viene fatto memorizzando le richieste di quali carichi sono stati spenti per riattivarli non appena sia possibile.

- ***Stato prod+cons:*** la gestione di questo stato si basa principalmente sul cambiamento di potenza richiesta alla batteria, se si entra in questo stato e lo storage non è utilizzato gli si inietta potenza per ricaricarlo, la potenza è tuttavia proporzionale a quella disponibile, se invece lo storage stava fornendo energia, significa che rispetto al ciclo di lavoro precedente i carichi o la microgenerazione hanno subito un brusco cambiamento, l'algoritmo simulando lo storage si trova in uno stato inconsistente, e diminuisce l' energia richiesta alla batteria.

- ***Stato can_ consume:*** in tale stato si entra quando il consumo del nodo sommato a quello delle adiacenze è inferiore al limite imposto *dall'energy interface* quindi l'algoritmo non determina grandi cambiamenti si va a controllare solamente che lo storage non stia fornendo energia e nel caso lo si stacca ma tale eventualità in questo stato non si dovrebbe verificare in quanto il limite imposto *dall'energy interface* se non è superato è già gestito in modo efficente dallo stato di go descritto in seguito.

- ***Stato ok:*** in questo stato non viene intrapresa nessuna operazione in quanto il gruppo delle adiacenze è all'equilibrio con le richieste dell'energy interface, si sono quindi equilibrati produzione, generazione e storage per raggiunge l'obiettivo fissato.

- ***Stato go:*** ci sono stati errori di calcolo o di comunicazione, il singolo nodo fa del suo meglio per raggiungere l'obiettivo *dell'energy interface*, attiva quindi lo storage o usa la potenza fornita dal proprio impianto di generazione se presente. Arriva a staccare i carichi nel caso le azioni intraprese non permettessero di arrivare all'obiettivo *dell'energy interface* precedentemente comunicato.

6.4 I Thread dell'Energy Interface

I moduli sviluppati per l'energy interface sotto il profilo dello comunicazioni sono gli stessi che si trovano nell'energy gateway, a volte si utilizza lo stesso metodo, per esempio i metodi creati per la gestione dei socket, a volte si ha che i thread o i metodi eseguono un lavoro che è speculare a quello già spiegato.

L'energy interface non presenta quindi tutti i moduli per la gestione dell'harware e per la simulazione dello storage e della microgenerazione, ed è composto sostanzialmente dal main, dal ciclo di lavoro, dal logger identico a quello definito in precedenza e da un metodo scan che cerca nella rete nuovi servizi e nuovi *energy gateway* e sfrutta i metodi creati per la gesitione dei socket anch'essi identici a quelli già precedentemente spiegati.

6.5 Ciclo di gestione nell'Energy Interface

Il ciclo di gestione dell'*energy interface* è stato il primo ad essere sviluppato in quanto nelle prime versioni tale algoritmo non era distribuito tra i nodi, ma tutte le informazioni venivano raccolte e analizzate e alla fine della attuale fase di analisi gli *energy gateway* attendevano la comunicazione dello stato di funzionamento dall'*energy interface*. L'algoritmo quindi non è diverso quindi da quello già esposto , solamente basa i calcoli sui dati ricevuti dall'intera federazione. Nella fase di comunicazione questi dati vengono diffusi in quanto può accadere che qualche nodo non disponga di adiacenze attive e quindi si basa su tali messaggi per eseguire il proprio ciclo di lavoro.

6.6 Gestione degli errori di rete

La gestione della caduta di un nodo è stata implementata seguendo la logica più semplice, ovvero se un *energy gateway* non risponde nella fase di comunicazione per un determinato numero di cicli di controllo esso viene espulso sia dall'*energy interface*, sia dalle adiacenze in modo indipendente, semplicemente viene chiusa

Secondi	Numero nodo	azione	Led coinvolti
10	0	on	all
10	1	off	all
11	2	on	1
11	3	on	1 2 3
15	0	on	7
15	0	pwr_factor	0.7

Tabella 6.1: Struttura dei file per comandare il simulatore

la connessione corrispondente e le comunicazioni continuano solamente con i superstiti. Nel caso l'*energy interface* rilevasse che tutti i nodi della federazione non rispondono, allora esso attua la seguente tecnica: entra in uno stato in cui vengono chiuse tutte le comunicazioni e tutti i thread vengono terminati e riavviati ripetendo le operazioni del flusso di avvio fino a ritornare in attesa di connessioni. Tale scelta è sembrata la più logica in quanto permette al nodo un totale reset delle funzioni, e inoltre, permette la riattivazione dei socket in caso la caduta fosse causata da problemi di rete. Infatti se al nodo non viene riassegnato un IP si tenta si rieseguire il *bind* finché non va a buon fine.

Nel caso la caduta fosse dell'energy interface o se un energy gateway si trovasse isolato viene riavviato solamente il thread ***eg_ server*** che gestisce il ciclo di lavoro, in questo modo il main torna a gestire le funzionalità di base e il thread ***eg_ server*** torna in attesa della connessione da parte dell'energy interface.

6.7 Simulatore

Il simulatore è stato creato implementando i metodi client dei socket, infatti al suo avvio parte un thread per scovare gli *energy gateway* nella rete e si procede alla connessione tramite socket con il thread remoto che fa da server per le comunicazioni. Il main prende in ingresso dalla linea di comando il numero del file da caricare per la simulazione che dovrà essere contenuto nello stesso percorso con il nome di ***simu_ register***#, dove il simbolo # sta ad indicare il numero della simulazione. Sempre da linea di comando riceve in ingresso quante volte deve ripetere le azioni. All'avvio quindi registra il tempo e lo comunica a tutti i nodi, questi lo utilizzeranno per la creazione di log partendo da tale riferimento. Instaurate quindi le connessioni il simulatore passa alla fase di lettura delle istruzioni, queste sono fatte in modo da recare come primo campo i secondi dall'inizio della simulazione ai quale inviare i comandi, in questo modo è più semplice la stesura dei file che hanno la struttura mostrata in tabella 6.1.

Nella tabella si nota come il file informa il simulatore a quale nodo connettersi e le azioni, nell'ultima riga della tabella si mostra inoltre un tipo di comando che

Secondi	P ist. consumata	P ist. prodotta	P batt	J in storage
5	700	0	0	5000
6	500	100	0	5000
11	300	100	0	5000

Tabella 6.2: Struttura log di simulazione

serve per definire il fattore di potenza dell'impianto di generazione del nodo al quale sarà inviato, grazie a quel parametro si definisce una graduale diminuzione della potenza prodotta, per esempio per simulare interferenze nuvolose o altri cali nella produzione come già spiegato negli obiettivi del progetto. Alla fine della simulazione si invia un messaggio di **end_work** che fa in modo che i singoli nodi coinvolti raccolgano i dati dei log e producano dei log basati sulla simulazione. Tali log hanno la struttura mostrata in tabella 6.2,

che indica per come primo campo i secondi trascorsi dall'inizio della simulazione, le potenze misurate sul nodo, la potenza assorbita dalla batteria e la quantità di energia immagazzinata nello storage in quell'istante. A questi log si sommano quelli prodotti dagli eventi e l'unione di entrambi permette di analizzare il comportamento del sistema.

6.8 Metodi di supporto

Il presente paragrafo descrive i diversi metodi non descritti fin'ora che sono di supporto ai diversi thread, essi gestiscono per esempio la chiusura delle connessioni o vengono usati dal ciclo di lavoro per aver i dati dello stato dei carichi.

- ***Key_state***: questa funzione si richiama dopo aver aggiornato lo stato dei led, semplicemente va a rilevare effettivamente con un controllo sulle porte GPIO quali carichi sono accesi e quali spenti aggiornando le strutture corrispondenti;

- ***Resync***: permette la risicronizzazione con il ciclo di lavoro, viene richiamata dopo un numero di cicli definiti dall'utente e fa ripartire la successiva fase di aggiornamento all'intero successivo delle finestre temporali, l'operazione che esegue è quella di confrontare il tempo di sistema con il t_0 fornito dall'*energy interface* e calcolare al millisecondo in quale istante ripartire. Tale metodo non è molto utile nelle simulazioni in laboratorio in quanto gli shift tra i diversi nodi non sono elevati, tuttavia in caso di sessioni di lavoro prolungate si ha la sicurezza di mantenere i cicli di lavoro sincronizzati;

- ***Counter_for_comm***:tale metodo raggruppa i dati del modulo di storage, dei carichi e della generazione e li rende disponibili hai thread che lo richiamano;

- ***exit_func***: la sua unica funzione, è quella di catturare il segnale di terminazione SIGINT associato al comando da tastiera Ctrl+C e terminare in modo sicuro il logger. Infatti, se si termina il programma di gestione senza che il logger sia avvertito, il file di log prodotto non è leggibile perché non chiuso correttamente. In questo modo se un utente termina volutamente il programma, tutti i file saranno a sua disposizione e leggibili.

Capitolo 7

Simulazioni e analisi dei log

Nel presente capitolo saranno riportati e descritti i dati raccolti durante le simulazioni effettuate, per ognuna sarà presente una parte introduttiva che descriverà i parametri utilizzati, un grafico riassuntivo ed una breve descrizione dei dati raccolti. Tutte le simulazioni hanno una durata temporale che va dai quaranta secondi ai due minuti sufficienti per comprendere l'andamento del controllo dell'energia. I dati raccolti sono di tre tipi:

- sincroni con il ciclo di lavoro: riportano i valori relativi alla potenza consumata e allo storage visti dal lato *energy interface*;
- sincroni con la cadenza del counter: principalmente dati riassuntivi delle potenze istantanee di ogni nodo;
- asincroni: ovvero dati relativi ai soli *energy gateway* nei quali sono riportati gli eventi con i relativi istanti temporali; i tipi di evento disponibili sono accensionespegnimento carico o storage in carica o in scarica.

Tutti i log sono prodotti durante le simulazioni, i primi due però si basano su dati che comunque verrebbero raccolti in quanto sono relativi alla contabilità dei diversi nodi, alla fine di ogni simulazione i vari *energy gateway* li aggregano e li memorizzano nella cartella di esecuzione del programma.

7.1 Esempi di log

I log, come mostrato nei capitoli precedenti seguono sostanzialmente la stessa struttura già presentata, questa sezione vuole descrivere brevemente come essi sono organizzati.

- ***Log lato Energy Interface***: sono log sincroni con il ciclo di gestione, ed i campi sono divisi da virgole. Nel primo campo contengono il riferimento temporale, nel secondo la potenza istantanea imposta dall'*energy interface* espressa in Watt, nel terzo la potenza disponibile della federazione con

la convezione degli utilizzatori in Watt, la potenza prodotta nell'istante di raccolta sempre espressa in Watt e l'energia totale negli *storage* della federazione in Joule. Qui sotto un esempio di come risulta un tale file.

```
10:03:13:556,0,0,0,64749600
10:03:14:56,0,-3000,3000,64749600
10:03:14:556,0,-1400,3000,64748400
10:03:15:56,0,-500,3000,64748400
10:03:15:556,0,-200,3000,64748400
10:03:16:56,0,-1600,4000,64748400
10:03:16:556,0,-1300,4000,64747300
```

Tale log è stato creato durante una simulazione con un $\boldsymbol{T}$ pari a 500ms, si può notare come ogni riga riporti un incremento temporale pari a $\boldsymbol{T}$.

- ***Log lato Energy Gateway***: sono log di simulazione con la risoluzione del secondo, utili per la creazione veloce di grafici o per osservare l'andamento della potenza istantanea nei nodi sempre con la convenzione degli utilizzatori. La struttura è divisa da spazi e il primo campo riporta i secondi trascorsi dall'inizio della simulazione, il secondo riporta la potenza consumata del nodo, il terzo la potenza istantanea, quindi la potenza in uscita dallo *storage* e l'energia immagazzinata. Le unità di misura sono Watt per le potenze e Joule per l'energia.

```
7 700 0 700 16200000
8 700 0 700 16198600
9 700 0 700 16197900
10 700 0 700 16197200
11 700 0 700 16197200
12 700 0 700 16195800
13 700 0 700 16195100
```

- ***Log asincroni***: tali log riportano l'evento accaduto esattamente nell'istante temporale rilevato, l'ordine dei campi è il seguente: ora dell'evento con risoluzione dei millisecondi, potenza scambiata, indirizzo IP del nodo, numero identificativo del carico, tipo di evento e parametro. L'esempio seguente ne mostra la struttura:

```
10:03:12:3,700,147.162.14.141,1,STORAGE,discarge
10:03:13:7,700,147.162.14.141,1,STORAGE,discarge
10:03:13:459,-1000,147.162.14.141,7,GEN,on
10:03:14:13,400,147.162.14.141,1,STORAGE,discarge
10:03:15:17,0,147.162.14.141,1,STORAGE,discarge
10:03:16:21,0,147.162.14.141,1,STORAGE,discarge
10:03:17:25,300,147.162.14.141,1,STORAGE,carge
```

Tipo Nodo	Indirizzo IP	MacAddress
Energy Interface	147.162.14.176	00:04:25:28:21:f9
Energy Gateway 1	147.162.14.87	00:04:25:28:17:76
Energy Gateway 2	147.162.14.141	00:04:25:28:17:77
Energy Gateway 3	147.162.14.173	00:04:25:28:22:2a
Energy Gateway 4	147.162.14.180	00:04:25:28:17:6c

Tabella 7.1: IP assegnati all'hardware durante le prove e MAC delle FoxBoard

Un associazione temporale al ciclo di gestione in questo caso non è possibile in quanto i comandi che creano il log arrivano in modo asincrono dal simulatore o dai thread del programma.

7.2 Ambiente di prova

Per effettuare le prove le FoxBoard sono state connesse alla rete del dipartimento attraverso uno switch e la connessione alla rete internet ha permesso di avere disposizione il collegamento con i server NTP per il riferimento temporale. I dati dello switch utilizzato sono:

- Marca: D-LINK
- Modello: DES-1008D
- Numero di serie: PW171A5007560
- Versione HW: J1

Gli allindirizzi IP statici forniti da rete del DEI sono mostrati in tabella 7.1.

7.3 Prova del funzionamento

Come prima prova per il funzionamento dell'algoritmo è stata effettuata una semplice simulazione che ha previsto l'accensione di tutti i carchi nello stesso istante con le impostazioni della rete elencate nelle tabelle 7.2 e 7.3.

Impostazione Energy Interface	Parametro
Consumo federazione	800[Watt]
Ciclo di gestione	500 msec

Tabella 7.2: Parametri prova di funzionamento dell'*energy interface*

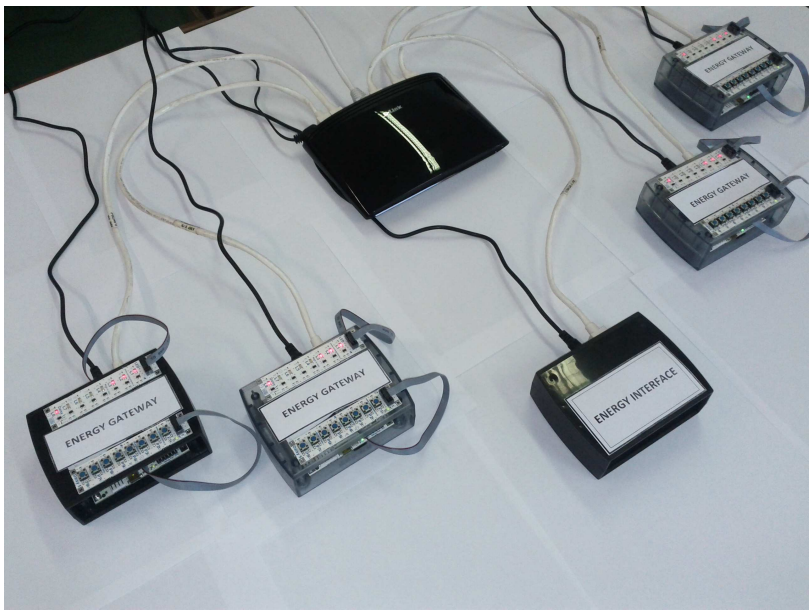

Figura 7.1: L'hardware connesso alla rete dipartimentale durante una sessione di prova

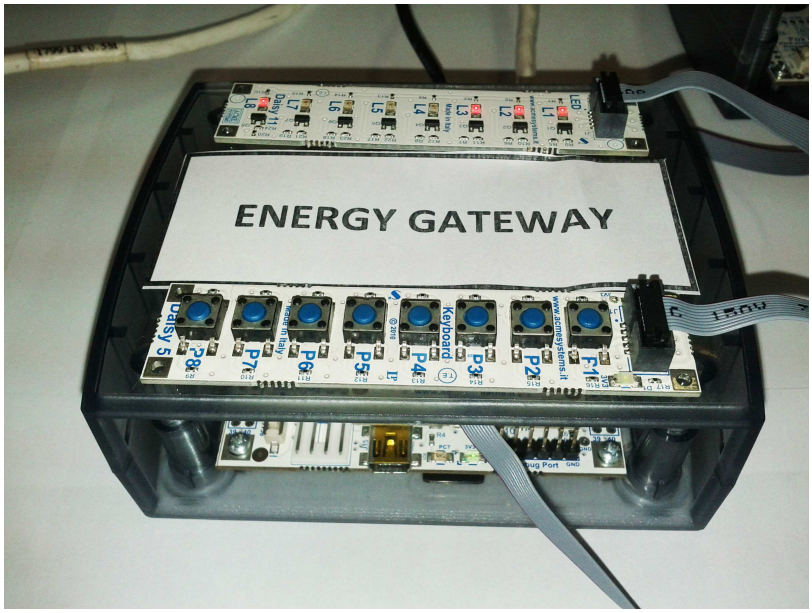

Figura 7.2: Particolare che mostra come i moduli dei tasti e i moduli dei led sono stati installati sulle FoxBoard che simulavano un Energy Gateway

Impostazione Energy Gateway	Parametro
Potenza carichi	100 per carico[Watt]
Potenza microgenerazione	0 per nodo [Watt]
Storage	non attivo

Tabella 7.3: Parametri prova di funzionamento dell'*energy gateway*

Ci si aspetta quindi che dopo un transitorio la rete si assesti verso il valore di potenza massima imposta dall'*energy interface* e lo mantenga. Dal grafico mostrato in figura 7.3 si nota come questo avvenga e come non appaia del *ripple* nei consumi, infatti l'algoritmo così come è stato costruito funziona al meglio se l'*energy interface* da la possibilità di consumare una quantità minima di energia tale da mantenere acceso almeno un carico sul nodo. La decisione presa, indipendentemente dalle direttive aggiuntive dettate dall'*energy interface*, è stata quella di far consumare al massimo per ogni nodo e per le proprie adiacenze 200 Watt. L'algoritmo distribuito per la gestione della potenza quindi dimostra, in questo caso base, la propria capacità nel controllare i consumi.

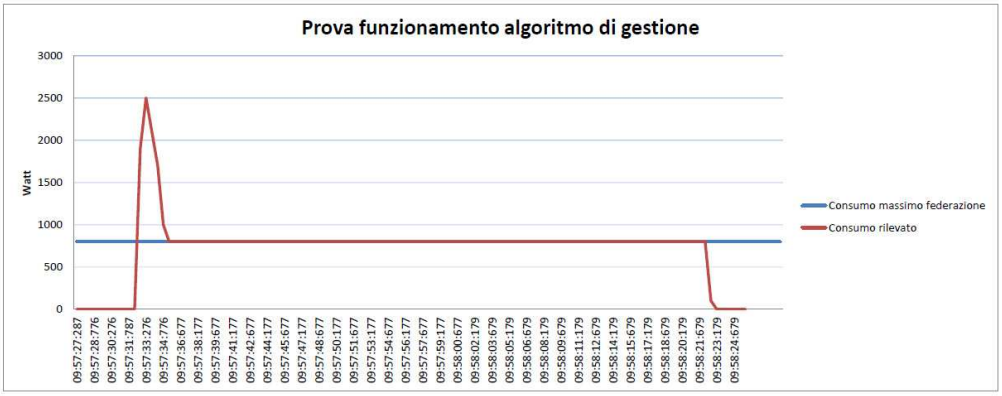

Figura 7.3: Test dell'algoritmo distribuito di gestione della potenza, si nota come dopo un transitorio la comunicazione con le adiacenze permetta di mantenere i consumi sotto controllo

7.4 Primo scenario

Il primo scenario che è stato studiato per vedere il comportamento della rete in presenza della sola microgenerazione, è stato quindi definito che la potenza scambiata all'*energy interface* fosse nulla e la potenza necessaria ai nodi fosse prodotta dagli stessi o dalle adiacenze. Nelle tabelle 7.4 e 7.5 l'elenco dei parametri utilizzati.

Impostazione Energy Interface	Parametro
Consumo federazione	0[Watt]
Ciclo di gestione	500 msec

Tabella 7.4: Parametri primo scenario dell'*energy interface*

Impostazione Energy Gateway	Parametro
Potenza carichi	100 per carico[Watt]
Potenza microgenerazione	1000 per nodo [Watt]
Storage	non attivo

Tabella 7.5: Parametri primo scenario dell'*energy gateway*

L'andamento della simulazione ha seguito i passi elencati in seguito, espressi in secondi dall'avvio della simulazione:

```
10 0 on all  //accensione tutti i carichi sul nodo 0 al decimo secondo
10 1 on all  //accensione tutti i carichi sul nodo 1
10 2 on all
10 3 on all
15 0 on 7  //accensione generazione sul nodo 0
15 1 on 7
15 2 on 7
15 3 on 7
45 0 off 7  //spegnimento generazione
45 1 off 7
45 2 off 7
45 3 off 7
50 0 off all //spegnimento carichi
50 1 off all
50 2 off all
50 3 off all
//fine simulazione
```

I comandi appena elencati fanno si che nella federazione tutti i nodi accendano i carichi nello stesso istante ma si trovino con gli impianti di generazione spenti, l'algoritmo di gestione quindi fa del suo meglio per diminuire la potenza assorbita come si vede nella prima fase della figura 7.4. Successivamente il simulatore comanda di accendere la generazione a tutti i nodi e appena l'algoritmo di gestione nota questa nuova disponibilità energetica comincia a riattivare i carichi che erano stati in precedenza spenti. Tale procedura richiede alcuni secondi in quanto per minimizzare le oscillazioni nelle accensioni il controllo riaccende

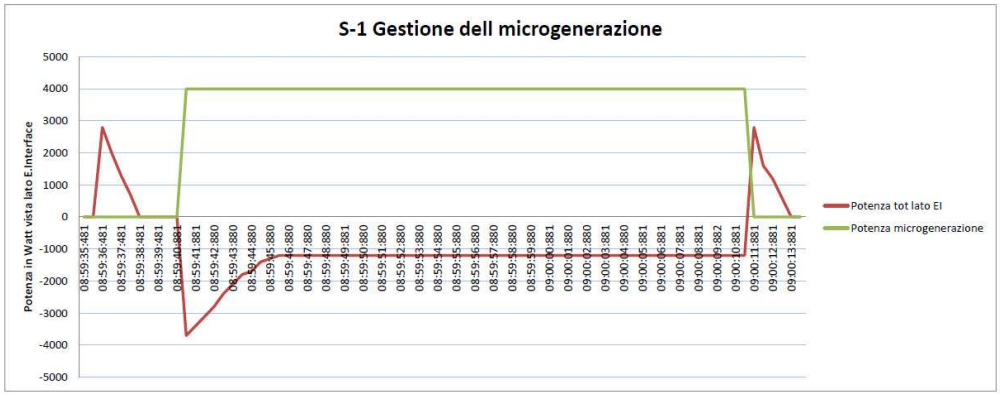

Figura 7.4: Andamento della potenza assorbita dalla federazione lato E.Interface. Nell'asse delle ascisse è presente il tempo mostrato in ore,minuti,secondi e millisecondi, nelle ordinate la potenza espressa in Watt e con la convenzione degli utilizzatori.

Impostazione Energy Interface	Parametro
Consumo federazione	1400[Watt]
Ciclo di gestione	500 msec

Tabella 7.6: Parametri secondo scenario dell'*energy interface*

solamente un carico per ciclo di lavoro. Verso la fine della simulazione vengono prima spenti gli impianti di generazione e poi i carichi per verificare che il controllo ripetesse anche in questo caso la procedura di spegnimento.

Il sistema ha quindi risposto in modo adeguato alle sollecitazioni in questo semplice scenario dimostrando il funzionamento dei meccanismi di base in presenza della generazione distribuita.

7.5 Secondo scenario

Questo secondo scenario simulato ha come caratteristica quella di cambiare impostazioni all'*energy interface* ed in particolare è stato impostato un consumo massimo per nodo di 350 Watt. Tale valore è stato appositamente scelto in quanto è la metà esatta del consumo massimo per ogni nodo (100 x 7 carichi) tuttavia è impossibile da mantenere in quanto i nodi presentano salti di consumo sui carichi di 100 Watt, quindi si possono innescare delle oscillazioni nei consumi. La tabella 7.6 riporta le nuove condizioni dell'*energy interface*, invece i parametri degli *energy gateway* sono rimasti identici ai valori mostrati in tabella 7.5.

Il file di simulazione è il seguente:

```
10 0 on all  //accensione tutti i carichi sul nodo 0
```

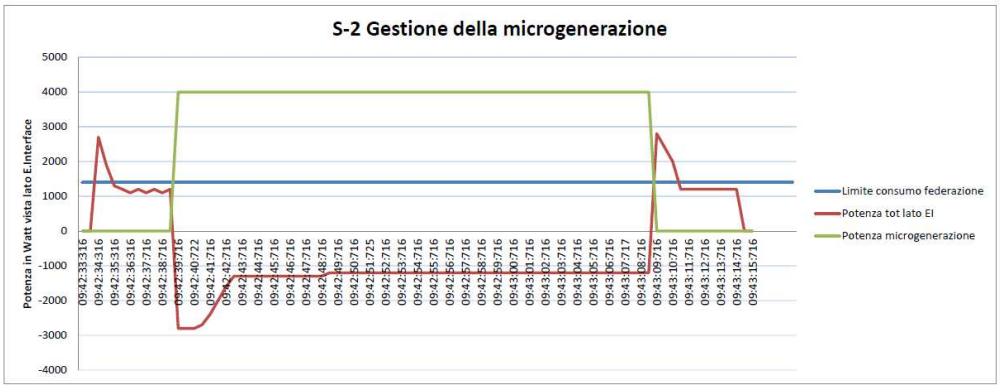

Figura 7.5: Andamento della potenza assorbita dalla federazione lato E.Interface. Nell'asse delle ascisse è presente il tempo mostrato in ore,minuti,secondi e millisecondi, nelle ordinate la potenza espressa in Watt e con la convenzione degli utilizzatori.

```
10 1 on all  //accensione tutti i carichi sul nodo 1
10 2 on all
10 3 on all
15 0 on 7  //accensione generazione sul nodo 0
15 1 on 7
15 2 on 7
15 3 on 7
45 0 off 7  //spegnimento generazione
45 1 off 7
45 2 off 7
45 3 off 7
50 0 off all //spegnimento carichi
50 1 off all
50 2 off all
50 3 off all
//fine simulazione
```

Nella figura 7.5 viene mostrato come la rete ha reagito. In particolare, si nota come prima dell'accensione della microgenerazione tutti i nodi abbiano subito uno spegnimento controllato dei carichi e siano rientrati entro il valore limite comunicato. L'accensione della microgenerazione in tutti i nodi ha fornito l'energia necessaria alla riaccensione dei carichi, energia che è stata in parte assorbita nel consumo interno e in parte ceduta alla rete esterna alla federazione. Successivamente lo spegnimento della microgenerazione ha costretto l'algoritmo di controllo ad intervenire nuovamente e spegnere carichi per riequilibrare il bilancio.

Nella prima parte della simulazione si nota come il sistema sia entrato in

uno stato oscillatorio dovuto forse al passaggio (virtuale) di energia tra nodi delle adiacenze, tale fenomeno non si ripete nella successiva fase di spegnimento e tuttavia in entrambi i casi i consumi sono rimasti inferiori a quelli dettati dall'*energy interface*.

7.6 Terzo scenario

Il terzo scenario simulato ha previsto l'attivazione negli energy gateway dello storage, esso è stato impostato in modo da riuscire a far fronte al consumo totale del nodo per tutto il periodo della simulazione. I parametri della simulazione sono mostrati nelle tabelle 7.7 e 7.8 ed il file per il simulatore è lo stesso dei precedenti casi. In figura 7.6 si può vedere l'andamento della potenza vista dall'*energy interface*.

Impostazione Energy Interface	Parametro
Consumo federazione	0[Watt]
Ciclo di gestione	500 msec

Tabella 7.7: Parametri terzo scenario dell'*energy interface*

Impostazione Energy Gateway	Parametro
Potenza carichi	100 per carico[Watt]
Potenza microgenerazione	1000 per nodo [Watt]
Storage	attivo
Pot.massima storage in/out	1000[Watt]

Tabella 7.8: Parametri terzo scenario dell'*energy gateway*

A differenza della simulazione precedente, nella quale i nodi non disponevano dello storage si nota come appena attivata la generazione la quota parte di energia non utilizzata dai carichi venga utilizzata per la ricarica dello storage. Nelle fasi in cui la microgenerazione non è attiva, nella parte sinistra della figura 7.6 il consumo dei carichi viene subito bilanciato dallo storage portando a zero lo scambio di energia sull'*energy interface*. I picchi che si notano sono dovuti al tempo di reazione dell'algoritmo e dipendono dall'ampiezza del ciclo di lavoro, tuttavia si può affermare come anche in questa simulazione l'algoritmo di gestione controbilanci efficacemente le richieste di energia limitando i picchi sia in produzione che in consumo.

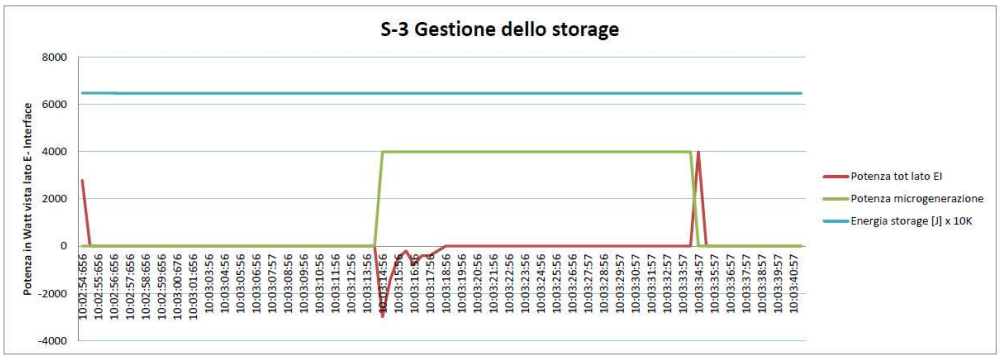

Figura 7.6: Simulazione con i moduli di storage attivi, si nota come i consumi e la produzione vengono livellati a zero, il consumo dell'energia immagazzinata non si nota in quanto lo storage è stato sovradimensionato

7.7 Quarto scenario

In questo scenario si è cercato di condensare un ipotetico andamento reale dei consumi e della produzione con un opportuna simulazione che aumentasse gradualmente le richieste dai carichi dei nodi e successivamente attivasse, sempre in modo graduale la microgenerazione.

Impostazione Energy Interface	Parametro
Consumo federazione	0[Watt]
Ciclo di gestione	500 msec

Tabella 7.9: Parametri quarto scenario dell'*energy interface*

Impostazione Energy Gateway	Parametro
Potenza carichi	100 per carico[Watt]
Potenza microgenerazione	1000 per nodo [Watt]
Storage	attivo
Pot.massima storage in	1000[Watt]
Pot.massima storage out	400[Watt]

Tabella 7.10: Parametri quarto scenario dell'*energy gateway*

Come si nota nella tabella 7.10 lo storage è stato attivato e la potenza in uscita che riesce a fornire non è sufficiente per coprire il fabbisogno del nodo. Quindi la simulazione ha previsto le seguenti fasi:

1. attivazione graduale del carico fino al massimo;

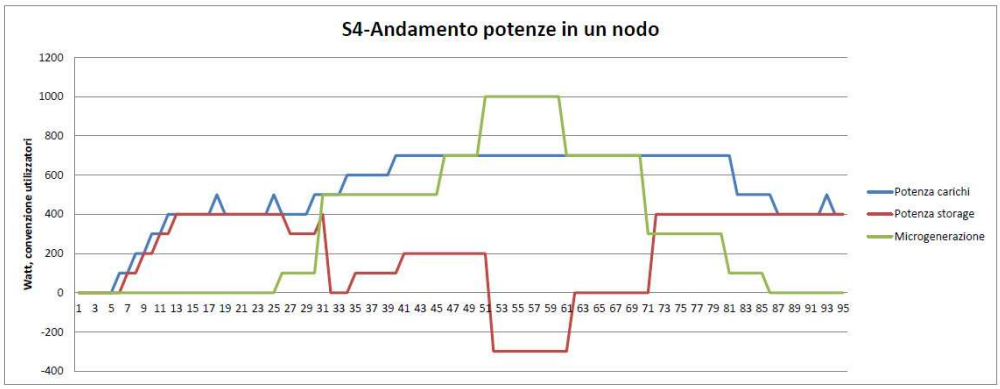

Figura 7.7: Il grafico illustra come sono gli andamenti delle potenze in uno dei nodi della federazione, sull'asse dei tempi troviamo i secondi dall'inizio della simulazione

2. attivazione graduale della microgenerazione di ogni nodo;

3. spegnimento graduale della microgenerazione di ogni nodo.

In questo modo si sono osservati gli scambi di energia tra tutte le entità, cioè carichi, storage e generazione. Si nota dai grafici riportati come sia l'andamento delle potenze all'interno dei nodi e al contempo come sia lo scambio energetico a livello *energy interface*. Prendendo in considerazione il grafico 7.7 si nota come l'andamento della potenza richiesta dai carichi sia in crescendo finché non si raggiunge il limite massimo di potenza attiva che lo storage può fornire. I consumi quindi si assestano sul valore di 400 W e a meno di un ripple causato dall'algoritmo di gestione rimangono costanti. Successivamente la generazione viene accesa e l'aumento della potenza fornita è graduale, in questa fase che va dal venticinquesimo fino al quarantanovesimo secondo di simulazione lo scambio di energia tra le varie entità è variabile. Infatti, il controllo comincia a riaccendere i carichi spenti in precedenza a mano a mano che la potenza dalla generazione locale aumenta, e allo storage viene richiesta sempre meno energia. Si arriva anche alla situazione in cui la generazione riesce,in autonomia, a far fronte al consumo interno e l'eccesso viene assorbito dallo storage che si ricarica. Dal sessantunesimo secondo di simulazione il processo è inverso, la generazione diminuisce e comincia ad essere richiesta maggior energia allo storage, alla fine della simulazione i nodi si ritrovano nella situazione iniziale, sono cioè costretti a non scambiare energia con l'*energy interface* e quindi il controllo spegne i carichi bilanciando i consumi.

Per completare l'analisi, si riporta in figura 7.9 il profilo di potenza visto dall'*energy interface* senza l'esecuzione dell' algoritmo di gestione. Confrontando tale grafico con la figura 7.8, si può notare come l'andamento della potenza sia molto diverso e come gli scambi energetici siano minori, in particolare nel

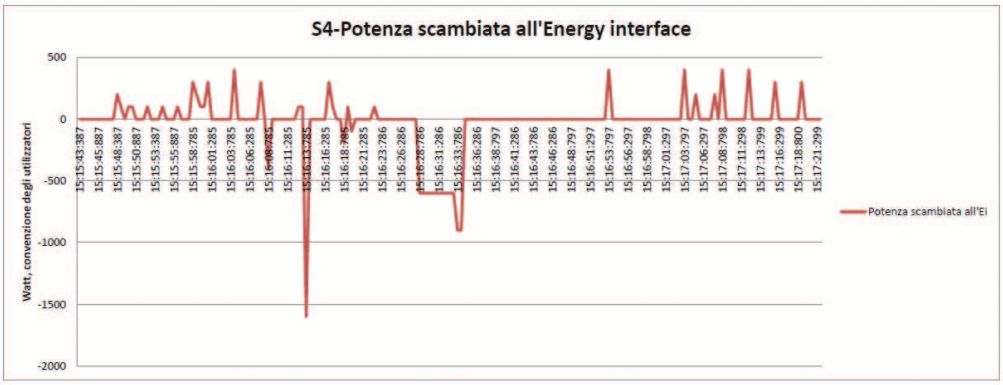

Figura 7.8: Grafico che illustra gli scambi di energia all'energy interface durante la simulazione del quarto scenario

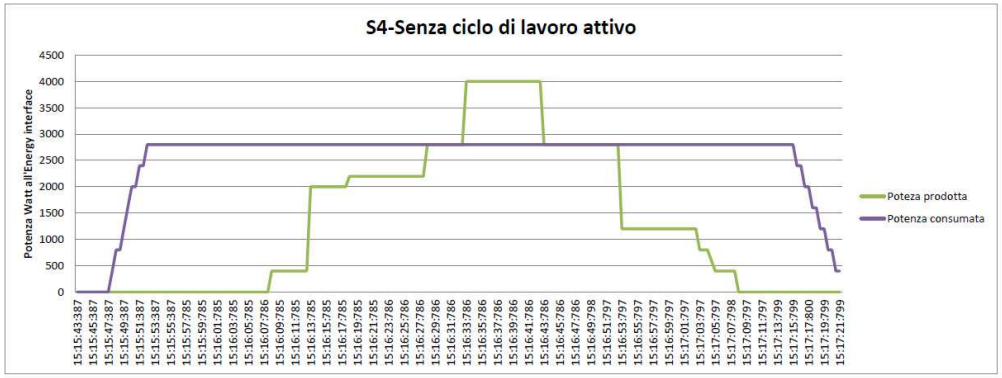

Figura 7.9: Grafico che illustra gli scambi di energia all'energi interface durante la simulazione del quarto scenario, senza l'attivazione della gestione distribuita.

grafico 7.9 oltre alla potenza consumata dai nodi in linea viene immessa anche tutta la produzione, questo comporta un ulteriore sovraccarico nella linea a differenza dello scenario con il controllo attivo dove i picchi, pur essendo presenti, sono limitati sia in ampiezza sia in durata temporale. Questo dimostra anche se in modo sperimentale i vantaggi che l'algoritmo di gestione, così come è stato pensato, può portare nella gestione delle linee elettriche, intesi come *power quality* e come *hosting capacity*.

Capitolo 8

Conclusioni

Una volta definiti i protocolli di comunicazione e coordinazione dell'hardware, il lavoro di tesi si è concentrato sulla stesura e sul successivo affinamento del codice dei thread che partecipano attivamente alla gestione dell'energia.

Al fine di rendere più robusta e affidabile la struttura dei thread, si sono dotati i metodi preposti alla comunicazione di opportune procedure per l'individuazione e la gestione degli errori. In particolare, i singoli nodi sono in grado di rilevare e risolvere in modo autonomo gli errori causati da altri energy gateway non funzionanti.

Particolare attenzione è stata dedicata allo sviluppo e all'ottimizzazione del ciclo di lavoro. In quest'ottica, si sono presi in considerazione numerosi casi particolari, cercando di esplorare una gran varietà di configurazioni operative. Tale modo di procedere ha consentito di individuare gli aspetti più critici dell'algoritmo di gestione, evidenziando le reali esigenze del sistema.

Nell'ottica di risolvere tali problematiche, sono stati introdotti dei ben precisi stati di funzionamento che permettono di classificare in modo semplice ed esclusivo la situazione in cui ci si trova ad operare. Mediante questo approccio, è stato possibile approntare degli appositi protocolli di azione, volti al medesimo obbiettivo, ma determinati dalle particolari condizioni della rete.

Un'opportuna temporizzazione delle diverse fasi del ciclo di lavoro si è rivelata utile in termini di gestione dell'energia e delle comunicazioni.

Nel primo caso, infatti, l'accurata sincronizzazione consente che i singoli nodi si trovino ad operare su dati riferiti al medesimo istante temporale. In particolare, ad ogni ciclo di lavoro ogni nodo acquisisce un'istantanea delle proprie adiacenze e ne ricava le azioni da intraprendere.

Nel secondo caso, invece, riservare appositi slot temporali alla trasmissione dei dati porta vantaggi anche sotto l'aspetto della sincronizzazione tra i nodi. Infatti, è così possibile instradare i messaggi di sincronizzazione in quelle fasi temporali meno appesantite dalle comunicazioni. Tali considerazioni possono apparire ininfluenti nel caso di una rete veloce quale quella di test, ma diventano basilari nel caso si adottino protocolli più lenti per le connessioni tra i nodi.

Le simulazioni effettuate confermano la bontà delle prestazioni dell'algoritmo, che soddisfa appieno i requisiti imposti in sede di formulazione del problema. L'analisi dei grafici evidenzia come i picchi della domanda vengano attenuati in modo rapido e nel contempo si favorisca lo sfruttamento dell'energia prodotta localmente.

La piattaforma hardware ha dimostrato una notevole versatilità e una capacità di elaborazione addirittura superiore a quella necessaria. Simili prestazioni suggeriscono di adottare tale tecnologia per gestire moli ben maggiori di dati. Ampliando ulteriormente la prospettiva con cui si guarda al problema, è possibile impiegare questi device per realizzare una prima versione concreta e operativa del concetto di *home gateway*.

Nel complesso, il lavoro di tesi ha gettato le basi per un sistema di gestione di una Smart Microgrid e ne ha proposto un'efficiente implementazione.

Appendice A

Istruzioni per installazione e gestione componenti

I seguenti paragrafi spiegano come installare o come impostare i parametri dei moduli esterni utilizzati, tutte le procedure sono state testate nell'hardware attuale delle FoxBoard G20.

A.1 Connessione SSH e FTP

La connessione con le schede Fox è stata effettuata tramite ***ftp***[1] Tale metodo ha come prerequisito quello di conoscere l'indirizzo IP con il quale l'hardware è collegato alla rete, si possono usare diversi modi in base al sistema operativo che si ha a disposizione. In WINDOWS® si può utilizzare l' applicativo freeware *ipscan.exe* che scansiona un insieme di indirizzi nella sotto-rete alla quale il computer è connesso e individua tutti gli *host* attivi. In Linux invece si può far riferimento al sito `http://www.acmesystems.it/` dove è riportato in dettaglio come accedere alle FoxBoard, in seguito viene comunque riportato uno script utile per scovare gli host nella rete:

```
#!/bin/sh
echo "Usage: $0 "
i=1
while [ "$i" -lt 254 ]
do
  ping -c 1 -W 1 "$1.$i" > /dev/null
    if [ "$?" -ne 1 ]
    then
        echo "$1.$i SUCCESS !"
    else
```

[1] File Transfer Protocol, è un protocollo per la trasmissione di dati tra host basato su TCP.

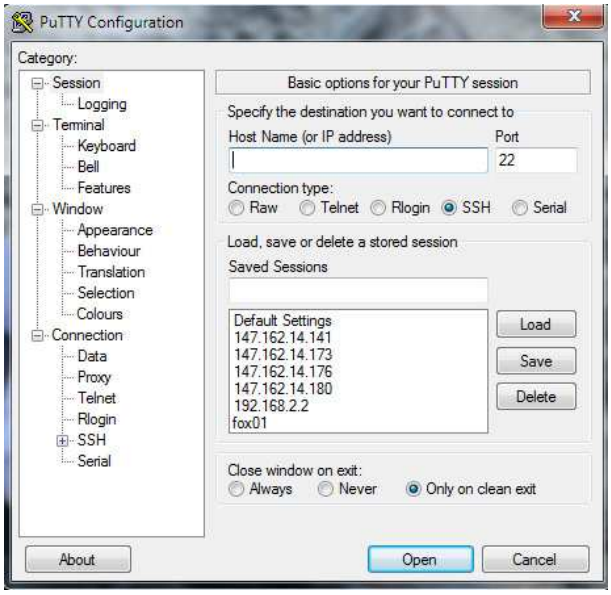

Figura A.1: Schermata principale di *putty.exe* in WINDOWS®

```
        echo "$1.$i fail"
    fi

    i=$(( $i + 1 ))
done
```

Si salva questo file come *scanip.sh* e si abilita all'esecuzione con *chmod +x scanip.sh* quindi si può lanciarlo fornendo la radice degli ip della propria Lan. Per esempio:

./scanip.sh 192.168.1

Ora che si è a conoscenza dell'indirizzo della FoxBoard si può procedere alla connessione, in ambiente WINDOWS® si può facilmente utilizzare *putty.exe* e connettersi con l'indirizzo ip precedentemente trovato e porta 22. Su Linux invece basta aprire un terminale e digitare:

ssh root@IP

In entrambi i casi verrà richiesta la password che di default è: ***netusg20***

Per effettuare il trasferimento dei file tra il pc sul quale si sviluppa e le FoxBoard si può usare ***Filezilla*** sotto Linux e ***WinSCP*** su WINDOWS®, entrambi richiedono IP, e l'inserimento della porta (solitamente 22), nome utente (*root*)e password(*netusg20*).

A.2 Installazione NTPD

Si può installare il client NTP digitando da terminale:

apt-get update

e successivamente

apt-get install ntpdate

a questo punto avete già il client funzionante, per aggiornare l'ora basterà eseguire il comando ntpdate seguito dall' indirizzo del server NTP scelto. Il server NTP scelto per il lavoro di tesi come riferimento è *ntp1.inrim.it* ma non è obbligatorio usare questo. In tal caso il comando da digitare da terminale sarà:

ntpdate ntp1.inrim.it

Naturalmente utilizzato così ha poco significato, il protocollo NTP diventa veramente utile se mantiene la data e l'ora aggiornata. In questo caso ci viene in aiuto il demone Ntpd, installabile da terminale mediante il comando:

apt-get install ntp

Il demone sarebbe già utilizzabile, ma si consiglia di aggiungere al file **/etc/ntp.conf** le voci:

server 3.it.pool.ntp.org

server 1.europe.pool.ntp.org

server 3.europe.pool.ntp.org

A questo punto si riavvia il servizio relativo ad NTP mediante il comando:

/etc/init.d/ntp restart

Dopo pochi istanti il servizio NTP ritorna attivo e ad ogni riavvio successivo del sistema sarà riattivato.

A.3 Installazione PTPD

L'installazione di PTPD prevede di scaricare il pacchetto contenente i file del programma dal sito `http://ptpd.sourceforge.net/`. Una volta scaricato il pacchetto che sarà nel formato *tar.gz* lo si estrae in una cartella e si esegue il comando *make*. In pochi secodi si ha a disposizione il demone ptpd che può essere avviato in diverse modalità elencate nella tabella A.1.

Parametro avvio	Descrizione
-c	run in command line (non-daemon) mode
-f FILE	send stats to FILE
-S	send output to syslog
-T	set multicast time to live
-d	display stats
-D	display stats in .csv format
-R	record data about sync packets in a file
-x	do not reset the clock if off by more than one second
-O	do not reset the clock if offset is more than NUMBER nanoseconds
-M	do not accept delay values of more than nanoseconds
-t	do not adjust the system clock
-a	NUMBER,NUMBER specify clock servo P and I attenuations
-w	NUMBER specify one way delay filter stiffness
-b	NAME bind PTP to network interface NAME
-u	ADDRESS also send uni-cast to ADDRESS
-l	NUMBER,NUMBER specify inbound, outbound latency in nsec
-o	NUMBER specify current UTC offset
-e	NUMBER specify epoch NUMBER
-h	specify half epoch
-y	NUMBER specify sync interval in 2ÑUMBER sec
-m	NUMBER specify max number of foreign master records
-g	run as slave only
-p	make this a preferred clock
-n	NAME specify PTP subdomain name

Tabella A.1: Possibili parametri di avvio per PTPd

A.4 Autenticazione

Ad ogni connessione di un client verso un server nel sistema che è stato creato è presente una fase di autenticazione che non permette ad estranei di partecipare alle comunicazione. Tale fase è stata prevista per uno sviluppo futuro dove le comunicazioni e le connessioni saranno protette da crittografia e metodi di scambio chiavi come Diffie-Hellman [2] [3] [9]. In questo progetto tuttavia il metodo di autenticazione prevede un semplice confronto del *mac address* del nodo che richiede la connessione con una lista di mac permessi memorizzata nei nodi. Se non presente l'autenticazione fallisce, il server chiude il socket e termina le comunicazioni. La lista con i *mac address* si deve memorizzare in */etc/mac_permission* che è semplicemente un file dove per ogni riga c'è un identificativo.

A.5 Gestione dei file di progetto

In allegato al presente elaborato si trova un DVD contenete il materiale prodotto nel lavoro della tesi e i risultati delle simulazioni sotto forma di log raccolti. La cartella ***Programma*** contiene due sotto cartelle contenenti rispettivamente il codice per l'energy gateway la prima e per l'energy interface e il simulatore la seconda. La cartella ***Configurazione*** contiene tutti i file da inserire in etc ognuno descritto sommariamente all'interno. La cartella ***Risultati*** invece contiene i log delle simulazioni riportate nel capitolo 7. Per effettuare prove pratiche è sufficiente copiare il contenuto delle cartelle Energy_Gateway e Energy_Interface nei nodi designati alla simulazione e compilare il programma con il seguente comando:

- `cc -w -o fox_ei fox_ei.c -lpthread` per l'energy interface;
- `cc -w -o fox_eg fox_eg.c -lpthread` per l'energy gateway;
- `cc -w -o simulatore simulatore.c -lpthread` per il simulatore;

Si creeranno quindi, gli eseguibili *fox_ei* e *fox_eg* nei rispettivi nodi, i quali accettano all'avvio il comando *-l*. Se presente si produrranno i log e tutti i file utili per il controllo dei risultati delle simulazioni, altrimenti tali file non verranno creati sgravando le Fox Board da alcuni compiti che potrebbero rallentare le operazioni importanti. L'eseguibile del simulatore ha bisogno di due parametri come già accennato nei capitoli precedenti, il primo è il numero della simulazione, il secondo il numero dei volte che tale simulazione deve essere ripetuta. I file di configurazione da inserire in */etc* come già anticipato riguardano i dati di accesso, i parametri dello storage (*battery.conf*)e i valori dei carichi negli energy

[2] Lo scambio di chiavi Diffie-Hellman è un metodo che consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione insicuro senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza

gateway (*power_cons.conf*). Negli energy interface invece, dovrà essere copiato il file *fox_ei.conf* che definisce i parametri del ciclo di lavoro e delle adiacenze. Il file contente i mac address permessi nelle comunicazioni dovrà essere copiato in entrambi i tipi di nodo e modificato nel caso si usi nuovo hardware per le prove.

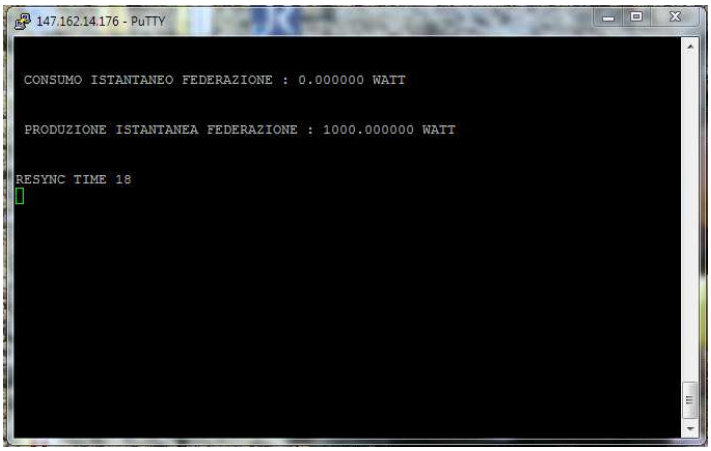

Figura A.2: Schemata mostrata dall'Energy Interface durante l'esecuzione del controllo. Viene anche mostrato il contatore per il resync, utile per verificare se l'esecuzione procede, si aggiorna infatti ogni ciclo di controllo.

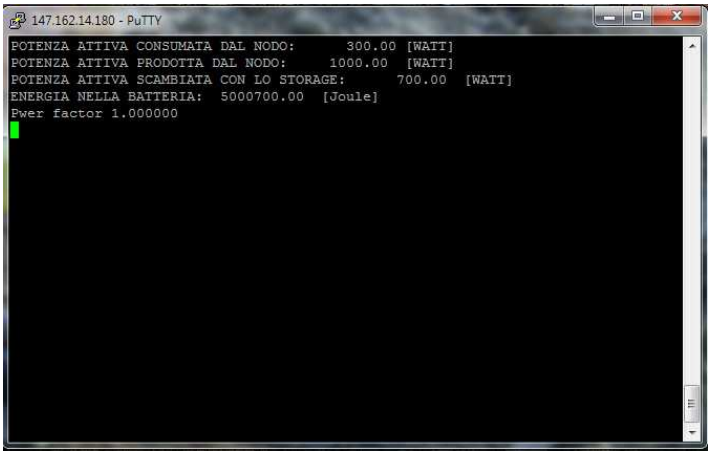

Figura A.3: Schemata mostrata da un Energy Gateway durante l'esecuzione del controllo, l'aggiornamento avviene ogni tre secondi.

L'esecuzione del programma inoltre produce ulteriori file:

- */var/log/energy_node* contenente il totale della energia scambiata di ogni energy gateway;

- */tmp/registro_ consumi* contenente i consumi con la risoluzione del secondo utili in caso di produzione dei log e cancellati ad ogni avvio.

Tali file comunque non vengono creati se si lancia il programma senza il parametro *-l*, è possibile tuttavia seguire l'andamento dell'algoritmo in quanto sul terminale della connessione saranno mostrati dei dati come nelle figure A.2 e A.3.

Bibliografia

[1] D. D. Zenobio, "Smart grids e metering, università degli studi di padova," 15 Febbraio 2012.

[2] M. Bertocco, G. Giorgi, and C. Narduzzi, "Networking in an internet-like smart microgrid," 2011.

[3] P. Tenti, "Corso: Smart grids-dipartimento di ingegneria dell'informazione," 2012.

[4] "Acme systems srl." http://www.acmesystems.it/, Marzo 2012.

[5] F. Sandri, "Realizzazione, mediante fpga, di un clock hardware ad alta risoluzione per la sincronizzazione di sistemi embedded," Master's thesis, Università degli studi di Padova, 2008.

[6] T. Scremin, "Sincronizzazione in reti di sistemi embedded: Implementazione ed analisi delle prestazioni del processo di timestamping," Master's thesis, Università degli studi di Padova, 2008.

[7] C.-C. Chao, S.-P. Huang, and H.-L. Hung, "Embedded system on ntp," *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, 2009.

[8] G. Giorgi, C. Narduzzi, and M. Stellini, "A test bed for synchronization in heterogeneous network environments," in *Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE*, pp. 612 –617, may 2008.

[9] G. Schäfer, *Security in Fixed and Wireless Networks: An Introduction to Securing Data Communications*. Wiley, 2003.

[10] K. Correll, N. Barendt, and M. Branicky, "Design consideration for software only implementation of the ieee 1588 precision time protocol," 2005.

[11] H. Gharavi and B. Hu, "Multigate communication network for smart grid," *Proceedings of the IEEE*, vol. 99, pp. 1028 –1045, june 2011.

[12] V. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. Hancke, "Smart grid technologies: Communication technologies and standards," *Industrial Informatics, IEEE Transactions on*, vol. 7, pp. 529 –539, nov. 2011.

[13] A. S. Tanenbaum, *Distributed Systems: Principles and Paradigm*. 2007.

[14] C. Lo and N. Ansari, "The progressive smart grid system from both power and communications aspects," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1 –23, 2011.

[15] S. Nowak, F. Schaefer, M. Brzozowski, R. Kraemer, and R. Kays, "Towards a convergent digital home network infrastructure," *Consumer Electronics, IEEE Transactions on*, vol. 57, pp. 1695 –1703, november 2011.

[16] Y. Yan, Y. Qian, H. Sharif, and D. Tipper, "A survey on cyber security for smart grid communications," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1 –13, 2012.

Elenco delle tabelle

Elenco delle figure

Ringraziamenti

Quando si arriva a questi traguardi ci si guarda indietro e quasi con nostalgia si ripensa a tutti i momenti, positivi o negativi, che hanno permesso di raggiungere una tappa così importante e desiderata. Ogni gradino affrontato mi ha infatti permesso di vedere i successivi sempre con maggiore lucidità e impegno, e di affrontarli confidando sulle mie capacità e sull'importante aiuto delle persone che mi sono state vicine. Tutte, a modo loro, hanno contribuito a rendere la mia esperienza universitaria unica e indimenticabile.

Per primi desidero ringraziare i miei genitori e mio fratello Denis, che supportandomi moralmente e anche economicamente, mi hanno dato quella forza e quella determinazione a non mollare anche quando tutto sembrava perduto.

Desidero ringraziare Laura, l'amore della mia vita, che con la sua presenza ha allietato e reso meno pesanti le ore di studio a casa, definendo oltre al cammino universitario un grande obiettivo che spero di raggiungere al più presto assieme a lei.

Da ricordare inoltre gli *amici del Dei*, compagni che hanno lasciato un segno particolare nel percorso di studi. Tutte le ore passate a fare progetti per i diversi esami e a studiare assieme mi ha permesso di affrontare con serenità anche le sfide più difficili, conscio che loro erano sempre pronti a condividere consigli ed esperienze. Ringrazio quindi Diego che mi ha insegnato a far coesistere studio e svago ricavando per entrambi la giusta quantità di tempo. Ringrazio Piera, per avermi mostrato cosa vuol dire essere determinati verso una meta, Mauro, per la sua profonda capacità di analisi che a volte sfiora l'immaginazione e che mi ha insegnato ad essere lungimirante nelle mie scelte. Francesco, che con le sue idee *"open-source"* mi ha portato a vedere il mondo sotto diversi punti di vista. Michele e Simone, due persone molto preparate su aspetti tecnici e sempre un passo avanti a tutti con la loro capacità di pianificazione ed infine Guglielmo che ha allietato con la sua compagnia gli ultimi mesi della mia avventura universitaria.

Desidero inoltre ringraziare il Professor Matteo Bertocco, che con la sua preparazione ha saputo sapientemente consigliarmi la giusta strada da intraprendere durante i mesi di sviluppo della tesi.