UNIVERSITÀ DEGLI
STUDI DI PADOVA

FACOLTÀ DI
INGEGNERIA

CORSO DI LAUREA SPECIALISTICA IN
INGEGNERIA INFORMATICA

TESI DI LAUREA

SVILUPPO DI UN SISTEMA PER L'INTERFACCIAMENTO E LA RICOSTRUZIONE DEI DATI IN RETI DI SENSORI TRAMITE LA TECNICA DI COMPRESSIVE SENSING

RELATORE: *Ch.mo Prof. Michele Rossi*

CORRELATORI: *Ing. Angelo P. Castellani, Ing. Davide Zordan*

LAUREANDO: *Anicet Foba Togue*

Padova, 7 dicembre 2010

Anno Accademico 2010-2011

CORSO DI LAUREA IN
INGEGNERIA
INFORMATICA

DIPARTIMENTO DI
INGEGNERIA
DELL'INFORMAZIONE

Sommario

L'obiettivo principale di questa tesi è lo sviluppo di un sistema di monitoraggio e raccolta dati centralizzata, in grado di interfacciarsi con una rete di sensori basata sul protocollo *6loWPAN*. Il sistema sviluppato permette di usare una tecnica di compressione dei dati chiamata *Compressive Sensing*, che si basa sulla raccolta di una frazione dei dati dai nodi sensore e una successiva ricostruzione del segnale da effettuarsi al punto di raccolta (Gateway).

Il sistema di monitoraggio sviluppato permette inoltre di visualizzare le informazioni raccolte attraverso un'interfaccia grafica (*Web*) realizzata grazie allo sviluppo di un *Plug-in* per *Fandango*, un'applicazione Web implementata dal *Signet group* e usata per interfacciarsi con la rete di sensori del dipartimento di Ingegneria dell'Informazione.

L'applicazione che svolge le funzioni di Gateway è stata implementata nel linguaggio di programmazione *C++* per quanto riguarda il punto di raccolta dei dati, e in *nesC* per la porzione di software che viene eseguita dai nodi sensore.

Indice

Elenco delle figure

1 Introduzione

Nel panorama delle reti wireless si possono distinguere diverse tipologie, sulla base della topologia, delle specifiche dei dispositivi che le compongono e delle funzioni svolte dalla rete. Le reti di sensori in particolare costituiscono una di queste tipologie, con determinate caratteristiche.

Le principali caratteristiche di tali reti si possono riassumere nei seguenti punti:

- **densità:** i nodi sensori vengono posizionati con densità spaziale molto elevata. Nello spazio di pochi metri si possono trovare anche decine o centinaia di nodi di sensori;
- **scalabilità:** il numero di nodi sensori può essere di alcuni ordini di grandezza superiore rispetto ad una rete *ad hoc*;
- **guasti:** i nodi possono essere soggetti a guasti, i quali non devono pregiudicare il funzionamento della rete;
- **topologia:** la topologia di rete può variare nel tempo in modo molto frequente, anche a causa di malfunzionamento di alcuni nodi;
- **capacità:** i sensori hanno limiti stringenti in termini di potenza di calcolo e memoria;
- **comunicazione:** i sensori usano comunicazioni di tipo broadcast, per la natura stessa del canale radio;
- **risorse energetiche:** i nodi dispongono di una fonte di energia esauribile, limitata e solitamente non rinnovabile.
- **gathering point:** una rete di sensori necessita di un punto di raccolta o *sink*, al quale far pervenire i dati raccolti dai nodi, a differenza di una rete *wireless ad hoc*.

Queste specifiche rappresentano dunque i punti caratteristici delle reti di sensori wireless. Nel caso di una rete di sensori cablata, l'utilizzo di cavi consente di trascurare i vincoli energetici poiché laddove sia possibile portare una connessione

cablata per i dati, sarà in generale possibile prevedere anche una o più linee di alimentazione. L'installazione di questi cavi però può essere molto difficoltosa, se non addirittura impossibile (si pensi ad esempio a reti che devono funzionare in ambienti inospitali per l'uomo). Si deve inoltre aggiunge il problema dei costi di installazione e manutenzione della rete, che nel caso di una rete cablata sono molto elevati. L'installazione di ciascun dispositivo infatti richiede manodopera e materiali per le operazioni di cablatura. Infine, una struttura cablata è essenzialmente "rigida" cioè risulta difficile aggiungere nuovi nodi alla rete o modificare la posizione di sensori preesistenti senza riconsiderare l'intera struttura della rete.

Si pensi ad esempio ad un sistema di controllo centralizzato per il condizionamento di una abitazione residenziale: per avere un controllo "ottimale" della temperatura sarà necessario installare dei sensori di temperatura in ogni stanza, collegati ad una unità di controllo centralizzata in grado di decidere in che modo abilitare i condizionatori o termoconvettori presenti nei diversi vani dell'abitazione. È evidente però che per spostare anche uno solo di questi sensori o attuatori è necessario una nuova mapatura dei cavi posti all'interno delle pareti e se non sono state previste in sede di progetto delle guide per cavi aggiuntive risulta necessario ricorrere a nuove opere di muratura. Esempi analoghi possono essere fatti sempre nel campo di *building automation* per quanto riguarda il controllo di illuminazione, la telesorveglianza o il controllo accessi.

L'utilizzo di reti *wireless* risulta l'opzione "ideale" per far fronte a questi problemi ma comporta anche una serie di problematiche da tenere in considerazione per quanto riguarda la propagazione del segnale, le interferenze, la sicurezza e i requisiti di potenza.

Solitamente le reti di sensori vengono impiegate per il monitoraggio di parametri ambientali, come la temperatura, l'umidità, la radiazione luminosa, la presenza di agenti inquinanti o la concentrazione di sostanze tossiche[1]. Questi segnali godono di determinate strutture di correlazione sia nello spazio che nel tempo, che possono essere sfruttate per una compressione dei dati da raccogliere andando così ad alleggerire il carico di lavoro che deve svolgere la rete.

In dettaglio nello sviluppo di questa tesi, l'attenzione è stata rivolta all'analisi di segnali di temperatura, luminosità e umidità rilevati dai rispettivi trasduttori integrati sui nodi della rete del dipartimento.

Il sistema sviluppato integra i seguenti componenti:

- un modulo di comunicazione tra gateway e applicazione Web;
- un modulo di ricostruzione dei dati che applica la tecnica *Compressive Sensing*;
- un database utile a tenere uno storico dei dati raccolti ed essenziale per la fase di ricostruzione.

- un plug-in di interfaccia grafica per la visualizzazione dei dati raccolti e ricostruiti, integrato nella web application *Fandango*[2].

In dettaglio la tesi è strutturata come segue: il capitolo 2 definisce in modo preciso le caratteristiche di interesse per le reti di sensori, le specifiche tecniche dei nodi e la struttura protocollare in essi implementata; nel capitolo 3, viene presentato il linguaggio di programmazione e il sistema operativo scelti per implementare il software in esecuzione sui nodi assieme alle motivazioni che portano a questa scelta; nel capitolo 4 viene descritta l'architettura, i meccanismi e i moduli del sistema sviluppato, punto centrale dell'attività di tesi; il capitolo 5 descrive la tecnica di ricostruzione tramite *Compressive Sensing*[3] e come questa sia stata implementata nel modulo *rebuilder*; il capitolo 6 contiene i risultati ottenuti durante i test effettuati sulla rete di sensori del dipartimento; infine nel capitolo 7 vengono presentate le conclusioni di questo lavoro assieme ai possibili sviluppi futuri.

2 Definizione reti di sensori

Le reti di sensori wireless (*Wireless Sensor Network*, *WSN*) sono una determinata tipologia di rete, caratterizzate da una architettura distribuita, e composte da un insieme di dispositivi elettronici autonomi in grado di prelevare dati dall'ambiente circostante e di comunicare tra loro. Un esempio di WSN è visualizzato nella figura seguente:

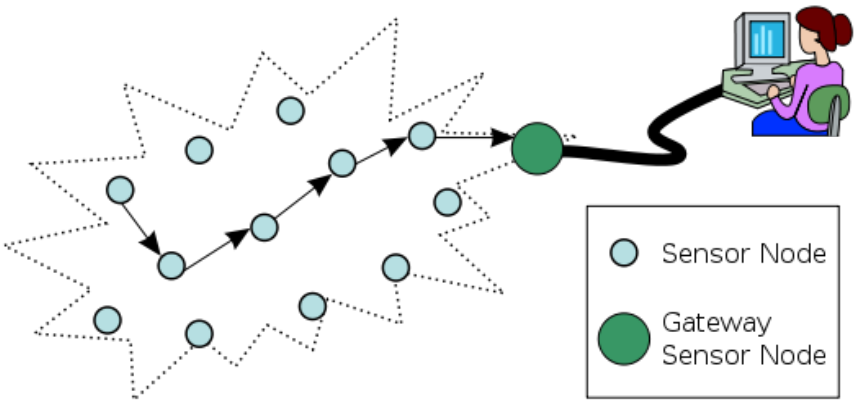

Figura 2.1: Un esempio di rete di sensori wireless.

I dispositivi che compongono la rete, chiamati nodi sensori (*sensor node*), sono formati da componenti in grado di rilevare grandezze fisiche (sensori di posizione, luminosità, temperatura, umidità, ecc.), da microcontrollori per l'elaborazione dei dati e da apparati radio che permettono la comunicazione. In generale i nodi sensore dispongono di una memoria di dimensioni ridotte, devono soddisfare vincoli molto stringenti per quanto riguarda i consumi energetici, la capacità di elaborazione e di comunicazione.

In commercio esistono sensori di diversi produttori, come ad esempio:

- Telos della Moteiv Corporation,
- MicaZ della CrossBow Tech. Inc,

- IntelMote2 della Intel,

- EyesIFX della Ember Corporation.

Va sottolineato che dal punto di vista architetturale, tutte queste famiglie di nodi sono simili tra di loro, essendo di fatto un'evoluzione di una delle prime architetture dei nodi senori sviluppate grazie ad un plug-in di architettura denominato *MICA*[1], progettato presso l'università di Berkeley.

2.1 Architettura dei sensori

Le piattaforme maggiormente utilizzate oggi nella progettazione dei nodi sensori in industria, sono state sviluppate sempre presso Berkeley. Possiamo quindi citare le piattaforme *MicaZ e TelosB* delle compagnie *CrossBow e Moteiv* disegnate con tre principali scopi, ossia:

- il consumo energetico ridotto;

- semplicità di utilizzo;

- la robustezza del software e dell'hardware.

2.1.1 Tecnologia *CrossBowTelos o TmoteSky*

Questo modello di nodo sensore presenta le seguenti caratteristiche:

- Compatibilità con lo standard 802.15.4 [4];

- chip radio CC2420[5] con banda centrata a 2,4GHz e 250 kbps di bitrate per la trasmissione dei dati;

- Microcontrollore TI MSP430 con 10kB di RAM e 48kB di Flash ROM;

- Connvertire ADC a 12-bit e un controllore DMA (Direct Memory Access);

- Antenna integrata sulla scheda;

- Possibilità di raccolta dei dati e programmazione tramite interfaccia USB;

- Sistema operativo *Open-source*;

- Integrazione dei sensori di temperatura, luce e umidità (su diverse range di lunghezze d'onda: $320nm$-$730nm$ e $320nm$-$1100nm$);

Si può vedere nelle figure 2.2 e 2.3 alcune immagini di queste due tecnologie.

[1]Maemo IDE Common Architecture

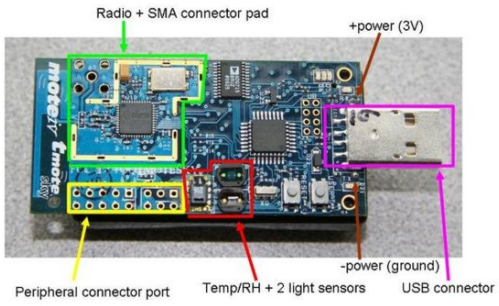

Figura 2.2: Un nodo sensore *TmoteSky*

Figura 2.3: A sensor node *CrossbowTelos*

2.1.2 Alimentazione

I nodi sensore sono stati progettati per poter usufruire di una delle seguenti fonti energetiche:

- due batterie di tipo AA con capacità fino a 3 mAh;
- fonti energetiche alternative (ad esempio dispositivi di energy harvesting quali micro pannelli solari) tramite slot di espansione con connettore a 16-pin, a condizione che la tensione fornita non superi il valore massimo di 3,6 V;
- cavo USB che eroga una tensione media di 3 V.

L'alimentazione via cavo garantisce una disponibilità permanente della risorsa energetica però presenta alcune problematiche come gli elevati costi dei cavi per collegare centinaia di nodi in una *WSN*. Inoltre una rete di sensori cablata perde totalmente la possibilità di avere dei nodi mobili al suo interno.

L'utilizzo delle batterie, invece, consente di potere installare i nodi sensore in ambienti inospitali all'uomo come ad esempio in contesti di guerra, o per tracciare

il movimento delle specie sotto-marrine (ad esempio quelle in via di istinzione), o ancora per rilevare la temperatura nei pressi di un cratere vulcanico[6, 7]. Purtroppo le batterie costituiscono una fonte energetica esauribile e non rinnovabile; inoltre la sostituizione delle batterie in caso di esaurimento può essere molto costoso o addirittura impossibile. La durata della vita di un nodo sensore diventa dunque una questione di massima importanza, per questa ragione è richiesta l'ottimizzazione della gestione della fonte energetica, tramite lo sviluppo di tecniche e protocolli che siano il più efficienti possibile. Per questo motivo, in fase di progetto, sono stati definiti diversi stati di funzionamento per i nodi sensore, con diversi consumi. Ad esempio nella modalità *sleep* un nodo riduce al minimo il consumo di energia andando a spegnere la radio. Si nota, alla luce di varie sperimentazioni, che la durata media della vita di un sensore alimentato a batterie alcaline di tipo AA è di circa 185 ore se la radio viene mantenuta accesa, mentre nello stato inattivo (radio spenta) il consumo totale delle batterie avviene dopo circa 21.000 ore (2 anni 4 mesi). Anche l'accensione e lo spegnimento del nodo sensore devono essere gestiti attentamente in quanto le operazioni di accensione richiedono un apprezzabile quantitativo energetico.

Per questi motivi, un'attenzione particolare deve essere prestata alla scelta del sistema operativo da eseguire sul nodo, le caratteristiche della piattaforma del nodo sensore e l'uso di protocolli ad hoc. In questa tesi verrà descritta una tecnica di monitoraggio che permette di allungare la vita media dei nodi. Utilizzando la tecnica del Compressive Sensing infatti, ogni qualvolta venga richiesta la lettura di un dato alla rete, alcuni dei nodi potranno rimanere inattivi, consentendo un risparmio energetico.

2.1.3 Processore, Memoria, Radio, Interfaccia di I/O e Sensori.

La figura 2.4 mostra uno schema a blocchi dell'architettura dei nodi di sensori. Ogni blocco rappresenta una delle componenti principali di una generica piattaforma dei nodi sensori.

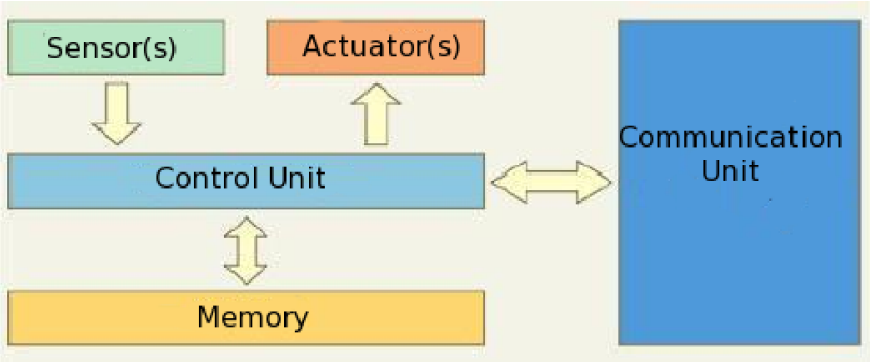

Figura 2.4: Schema a blocchi dell'architettura generale del nodo sensore.

Il blocco **Communication Unit** identifica tutta la componentistica relativa alla radio; il blocco **Control Unit** integra il microcontrollore e le componenti che operano da interfacce verso gli altri blocchi; il blocco **Memory** identifica la memoria flash presente sul nodo; i blocchi **Sensor** e **Actuator** infine descrivono le tipologie di trasduttori e attuatori presenti sulla piattaforma.

Processore

A causa del loro costo elevato, e dato che i sensori devono eseguire dei processi semplici e specializzati, i microprocessori non sono adatti alle *WSN*. I nodi sensore che si trovano in commercio sono equipaggiati con microcontrollori a basso consumo energetico e poca potenza computazionale. Tale integrazione di microcontrollori consente un consumo minore di energia delle *CPU*, e questa è una caratteristica molto importante dato che i sensori wireless sono in generale alimentati da batterie. In aggiunta a questo fatto molto importante, i microcontrollori sono molto adatti alle *WSN*, perchè alcune sue parti funzionali possono essere disattivate se non necessarie, diminuendo ulteriormente il consumo energetico.

Memoria

Oltre ad un'unità di elaborazione, c'è anche la necessità di una *Random Access Memory* (RAM) per memorizzare i dati delle applicazioni, come i valori numerici delle grandezze indicate dai sensori, di una *Read Only Memory* (ROM) per memorizzare in modo permanente il codice del programma, anche quando l'alimentazione viene a mancare. La memoria **RAM** è spesso inclusa nel microcontrollore e limitata a qualche kilobyte. Normalmente i nodi sensori includono anche una EEPROM (*Electrically Erasable Read-Only Memory*) spesso di tipo *FLASH*, per i dati di configurazione. Queste memorie non solo preservano i dati nel caso di interruzione dell'alimentazione, ma possono essere utilizzate come memoria RAM quando necessario. La memoria FLASH è leggibile da dispositivi esterni connessi alla piattaforma attraverso l'interfaccia USB.

Radio

Il nodo sensore dispone di una radio ricevitore-trasmettitore che contiene tutti i circuiti necessari per spedire e ricevere i dati sul canale radio: i blocchi di modulazione e demodulazione, i convertitori analogico-digitale e digitale-analogico, gli amplificatori di potenza e di rumore (*low noise*), i filtri, il *mixer* e l'antenna sono tra le componenti più importanti. La radio solitamente lavora in *half-duplex* cioè la comunicazione non può avvenire contemporaneamente in entrambe le direzioni. In generale le radio montate sui nodi sensori lavorano su tre diversi intervalli di frequenze: 400 *MHz*, 800-900 *MHz* e 2.4 *GHz* o nella banda *ISM* (*Industrial, Scientific and Medical radio bands*). Un aspetto importante da rilevare è che la potenza di trasmissione, gli schemi di modulazione, e la frequenza di trasmissione dei dati variano da produttore a produttore.

Sensori e Attuatori

Sul nodo infine sono integrati alcuni trasduttori e attuatori che permettono il monitoraggio di alcune grandezze fisiche o di eseguire operazioni. Questi componenti si interfacciano con il controllore attraverso degli ADC In commercio esiste un'ampia varietà di sensori, che possono rilevare grandezze quali: temperatura, luminosità, umidità, qualità dell'aria, pressione, posizione, livello sonoro, presenza di campo elettromagnetico.

2.2 Stack protocollare

Per svolgere in maniera efficiente le misurazioni, i nodi sensori supportano al livello fisico lo standard *802.15.4* [4] del modello pubblicato nel 2006 che prevede l'utilizzo di 3 bande di frequenze distinte, quali:

- 250 *kbps* a 2.4 *GHz* (16 canali identificati da un intero k, con $11 \leq k \leq 26$),
- 40 kbps a 915 MHz (10 canali),
- e 20 kbps a 868 MHz (1 canale).

La banda di frequenza di nostro interesse corrisponde alla prima e dispone di 16 canali radio. Lo standard 802.15.4 oltre alle specifiche di livello fisico definisce anche il livello di accesso al mezzo. A livello superiore, il nodo implementa il routing basato sull'indirizzamento *IPv6* [8]. Questo protocollo è la nuova versione di *Internet Protocol* per la futura rete *Internet*, attualmente denominata con expressione: *Internet of Things*. Gli standard *802.15.4 e IPv6* sono interfacciati tramite l'utilizzo di uno strato di adattamento chiamato *6LoWPAN* (*IPv6 over Low power Wireless Personal Area Network*)[9, 10]. Questo livello di adattamento consente di comprime i pacchetti IPv6, di dimensioni troppo elevate per essere utilizzati in una rete di sensori, inoltre aggiunge delle funzionalità al protocollo di routing come la trasmissione in broadcast.

La tabella seguente descrive la struttura degli strati di protocolli che analizzeremo successivamente nel minimo dettaglio.

Layer	*Protocollo*
transport	**UDP**
network	**IPv6**
adaptation	**6LoWPAN**
link	**802.15.4**
physic	

Tabella 2.1: Stack protocollare dei nodi sensore

2.2.1 Il protocollo 6LoWPAN

Lo standard *6LoWPAN* è un protocollo di adattamento che si colloca tra il livello *network* e il livello *link* nelle reti *wireless* consentendo la gestione dell'indirizzamento *IPv6* ovvero *Internet Protocol version 6* [9, 10]. Questo protocollo è stato sviluppato all'interno della *IETF* (*Internet Engineering Task Force*).

Poichè la lunghezza tipica di un pacchetto 802.15.4 è pari a 127 byte, mentre la MTU richiesta da IPv6 è pari a 1280 byte, i due protocolli non possono cooperare. Il livello 6LoWPAN rende le due tecnologie compatibili attraverso la compressione dell'header IP che avviene eliminando tutte le informazioni ridondanti che possono essere recuperate dall'header del livello di link o essere dedotte dal contesto applicativo. Un altra tecnica usata è la frammentazione dei pacchetti IP in più frame ed infine l'inoltro a livello di link attraverso il quale è possibile effettuare l'inoltro dei pacchetti quando nello strato di adattamento è presente l'indirizzo compresso del nodo destinazione.

Protocollo IPv6

IPv6 estende la *Internet Protocol version 4* e adotta campi di 16 *bytes (o octet)* per indirizzare i nodi mentre IPv4 ne possiede soltanto 4. Il formato dell'header dei pacchetti di IPv6 (riportato in figura 2.5), risulta perciò diverso da quello di IPv4, ed è strutturato come segue:

- 4 bit: ***Version***, che ne specifica la versione;

- 8 bit: ***Traffic class***, utilizzato dai schedulatori di traffico(router) per pianificare il traffico nella rete;

- 20 bit: ***Flow label***, destinata ad essere fonte di marcatura dei pacchetti che non richiedono assistenza speciale;

- 16 bit: ***Payload length***, dimensione del pacchetto senza l'intestazione di base;

- 8 bit: ***Next Header***, identifica il tipo dell'intestazione (livello di trasporto o estensione dell'intestazione IPv6);

- 8 bit: ***Hop limit***, determina il tempo di vita del pacchetto come il TTL di IPv4;

- 128 bit: ***Source address***, indirizzo della sorgente;

- 128 bit: ***Destination address***, indirizzo del destinatario.

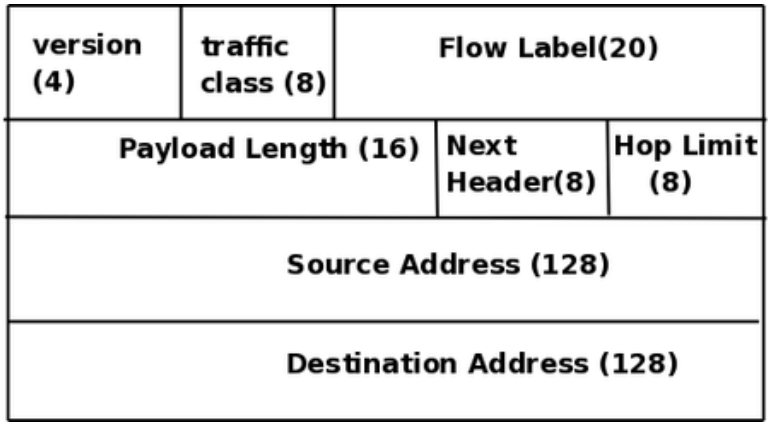

Figura 2.5: Il formato header di *IPv6*

Struttura dell'estensione del header IPv6

In IPv6 sono state definite delle strutture header aggiuntive e opzionali che servono a numerosi scopi. La presenza degli eventuali header aggiuntivi è indicata dal campo *Next Header* dell'header IPv6. Alcune di queste strutture e le relative funzioni offerte sono elencate di seguito:

- ***Hop by hop*** *and* ***Destination*** , vengono utilizzati per includere informazioni aggiuntive sul percorso che deve seguire un pacchetto;

- ***Routing***, può contenere informazioni utili ai nodi intermedi nel percorso tra sorgente e destinazione;

- ***Fragment***, nel caso di frammentazione del pacchetto, in questo header sono contenute tutte le informazioni utili alla ricostruzione;

- ***AH and ESP: IPSec***, possono essere usati per fornire informazioni di autenticazione e per la sicurezza del pacchetto.

Tipi di indirizzi *IPv6*

Il sistema introdotto da IPv6 richiede di distinguere gli indirizzi in tre categorie fondamentali: **unicast, anycast** e **multicast** [11]. Quello che in IPv4 era conosciuto come indirizzo broadcast non esiste più in IPv6.

Un altro particolare interessante riguarda il fatto che solo gli indirizzi che iniziano per FF_{16} (11111111_2) sono di tipo multicast, mentre gli altri sono tutti unicast. Gli indirizzi anycast sono degli indirizzi con caratteristiche uguali a quelli unicast, a cui però è stato attribuito un ruolo differente.

unicast: L'indirizzo unicast riguarda un'interfaccia di rete singola; in altri termini, un indirizzo unicast serve per raggiungere un'interfaccia di rete in modo univoco.

Si è accennato al fatto che tutti gli indirizzi, tranne quelli che iniziano per FF_{16}, sono di tipo unicast (e poi eventualmente tra questi si possono definire degli indirizzi anycast).

La caratteristica più importante degli indirizzi unicast è quella di poter essere aggregati a una maschera di bit continua, simile a quella di IPv4, senza il vincolo delle classi di indirizzi (come avveniva invece con IPv4).

Un nodo IPv6, cioè un componente collocato nella rete che riconosce questo protocollo, può trattare l'indirizzo IPv6 come un elemento singolo (nel suo insieme) oppure come qualcosa formato da diverse componenti, in base al ruolo che questo nodo ha nella rete. In pratica, a seconda del contesto, il nodo IPv6 potrebbe vedere l'indirizzo come un numero composto da 128 bit, oppure potrebbe

128 bit
indirizzo dell'interfaccia

Tabella 2.2: formato standard dell'indirizzo IPv6 di tipo unicast

riconoscere un prefisso relativo a una sottorete:

n bit	128 − **n** bit
prefisso di sottorete	id-interfaccia

Tabella 2.3: formato dell'indirizzo IPv6 di tipo unicast per una determinata sottorete

In questo secondo caso si intende distinguere la parte di indirizzo relativa alla rete in cui si trova collocata l'interfaccia del nodo in questione, rispetto alla parte restante dell'indirizzo, che invece indica precisamente di quale interfaccia si tratti. Ma l'indirizzo unicast può essere visto come il risultato di un'aggregazione molto più sofisticata, dove si inseriscono livelli successivi di sottoreti in forma gerarchica, fino ad arrivare all'ultimo livello che permette di raggiungere la singola interfaccia.

Identificatori di interfaccia: La parte finale di un indirizzo unicast serve a identificare l'interfaccia nel collegamento (link), ovvero la rete fisica in cui si trova. Questa parte dell'indirizzo, definibile come identificatore di interfaccia (interface identifier), deve essere univoca all'interno del collegamento. Eventualmente, potrebbe essere univoca anche in un ambito più grande.

La struttura di indirizzo unicast dipende principalmente dal tipo a cui questo appartiene, in base al prefisso di formato. In molti casi, la parte finale dell'indirizzo destinata a identificare l'interfaccia è di 64 bit (la metà di un indirizzo IPv6) e deve essere costruita secondo il formato IEEE EUI-64. L'identificatore EUI-64 è un numero di 64 bit che serve a identificare il produttore e il «numero di serie» di un'apparecchiatura di qualche tipo. In pratica, un produttore ottiene un numero che rappresenta la sua azienda, e questo viene usato come parte iniziale degli identificatori EUI-64 di sua competenza. Con tale numero potrà «marchiare» le proprie apparecchiature, avendo l'accortezza di utilizzare sempre numeri differenti per ogni pezzo, purché questi inizino tutti con il prefisso che

gli è stato assegnato. In condizioni normali, un identificatore EUI-64 corretto è anche un numero univoco a livello globale.

Nel momento in cui l'interfaccia di rete a cui si attribuisce un indirizzo unicast dispone del numero EUI-64, è facile ottenere l'identificatore di interfaccia; quando questo non è disponibile si utilizzano altre tecniche per generare un numero che gli assomigli. Nel primo caso, si intuisce che il numero utilizzato per l'identificatore di interfaccia è anche univoco a livello globale, mentre negli altri casi questo non può essere vero in assoluto. A questo proposito, lo stesso numero EUI-64 contiene un bit che viene utilizzato per indicare il fatto che si tratti di un identificatore univoco a livello globale o meno. Si tratta del settimo bit più significativo, che così viene sottratto dai valori che può assumere la parte iniziale di 24 bit che identifica l'azienda (company id).

Per la precisione, un indirizzo unicast che termina con l'identificatore di interfaccia composto dall'identificatore EUI-64, inverte il bit che serve a riconoscerlo come univoco a livello globale. La motivazione di questa inversione è molto semplice: si vuole evitare che un indirizzo **non univoco** a livello globale debba avere per forza quel bit a uno, cosa che costringerebbe a una notazione dettagliata dell'indirizzo IPv6 corrispondente.

loopback e link-local: L'indirizzo unicast 0:0:0:0:0:0:0:1 viene usato per identificare l'interfaccia virtuale locale, ovvero l'interfaccia di **loopback**. Come tale, non può essere utilizzato per un'interfaccia fisica reale. In pratica, un pacchetto destinato a questo indirizzo non deve uscire al di fuori del nodo (nella rete fisica esterna); inoltre, un pacchetto destinato a un altro nodo non può indicare come mittente questo indirizzo.

Gli indirizzi **link-local** si riferiscono all'ambito del collegamento in cui si trovano connesse le interfacce di rete. Questi indirizzi rappresentano uno spazio privato che non può essere raggiunto dall'esterno e, di conseguenza, non può attraversare i router. Evidentemente, tali indirizzi servono per scopi amministrativi particolari, legati all'ambito della rete fisica.

La struttura normale di un indirizzo link-local è molto semplice e definito nella tabella 2.4.

64 bit	64 bit
1111 1110 10.....0000	identificatore di interfaccia

Tabella 2.4: formato dell'indirizzo IPv6 di tipo unicast link-local

Come si può vedere, i primi 10 bit servono a definire il formato dell'indirizzo, stabilendo che si tratta del tipo link-local (**FE80::/64**). A metà dell'indirizzo inizia l'identificatore di interfaccia, ottenuto dall'identificatore EUI-64 (già descritto in precedenza), che viene determinato in modo differente a seconda del tipo di interfaccia (**FE80::[id-interfaccia]**).

Dal momento che l'indirizzo link-local deve essere univoco solo all'interno del collegamento fisico in cui si trova, non richiede la distinzione in sottoreti e può essere determinato in modo automatico, eventualmente interrogando la rete stessa. Di solito, in presenza di interfacce Ethernet si utilizza il loro indirizzo MAC trasformandolo secondo la regola già vista a proposito dell'identificatore EUI-48. Per esempio, un'interfaccia Ethernet il cui indirizzo MAC sia:

00:80:ad:c8:a9:81

ottiene l'indirizzo IPv6 link-local seguente:

fe80:0000:0000:0000:0280:adff:fec8:a981

che si può abbreviare come:

fe80::280:adff:fec8:a981.

site-local: Gli indirizzi site-local si riferiscono all'ambito di un sito e si possono utilizzare liberamente senza bisogno di alcuna forma di registrazione. Questi indirizzi rappresentano uno spazio privato che non può essere raggiunto dalle reti esterne al sito in questione.

La struttura normale di un indirizzo site-local è definita nella tabella 2.5.

10 bit	54 bit	64 bit
1111 1110 11	sottoreti	identificatore di interfaccia

Tabella 2.5: formato dell'indirizzo IPv6 di tipo unicast site-local

anycast: L'indirizzo anycast serve per essere attribuito a più interfacce di rete differenti (in linea di principio, queste dovrebbero appartenere ad altrettanti componenti di rete distinti). Si tratta di un indirizzo che ha le stesse caratteristiche esteriori di quello unicast, che però viene attribuito a diverse interfacce di altrettanti nodi, con lo scopo di poter raggiungere semplicemente quello che risponde prima (quello più vicino in base al protocollo di instradamento). Per la precisione, i pacchetti inviati a un indirizzo anycast dovrebbero raggiungere un'unica interfaccia di rete.

multicast: L'indirizzo multicast serve per essere attribuito a più interfacce di rete differenti (in linea di principio, queste dovrebbero appartenere ad altrettanti componenti di rete distinti). I pacchetti inviati a un indirizzo multicast dovrebbero raggiungere tutte le interfacce di rete a cui questo indirizzo è stato attribuito.

Un indirizzo IPv6 multicast serve a identificare e a raggiungere un gruppo di nodi simultaneamente. Gli indirizzi multicast hanno una struttura particolare:

Il prefisso di formato è 11111111_2, ovvero FF_{16}, e a questo seguono 4 bit di opzione. Di questi 4 bit, è stato specificato solo il significato di quello meno

8	4	4	112 bit
1111 1111	000T	scop	identificatore di gruppo

Tabella 2.6: formato dell'indirizzo IPv6 di tipo multicast

significativo, che viene indicato convenzionalmente con la lettera ≪T≫ (gli altri devono essere azzerati fino a che verrà stabilito qualcosa di diverso).

- T = 0 indica un indirizzo multicast assegnato permanentemente dall'autorità globale di Internet;

- T = 1 indica un indirizzo multicast assegnato in modo provvisorio.

I 4 bit successivi rappresentano l'ambito dell'indirizzo multicast (scope). Il significato dei valori che può assumere questo campo sono indicati nella tabella 2.7.

Valore	Significato
0	Riservato
1	Ambito node-local
2	Ambito link-local
3	Non assegnato
4	Non assegnato
5	Ambito site-local
6	Non assegnato
7	Non assegnato
8	Ambito organization-local
9	Non assegnato
A	Non assegnato
B	Non assegnato
C	Non assegnato
D	Non assegnato
E	Ambito globale
F	Non assegnato

Tabella 2.7: Elenco dei valori per definire l'ambito di un indirizzo multicast.

La parte finale dell'indirizzo identifica il gruppo multicast nell'ambito stabilito dal campo scope. Tuttavia, nel caso di indirizzi stabiliti in modo permanente, l'identificatore di gruppo resta uguale per tutti i tipi di ambiti.

Per regola, non si può utilizzare un indirizzo multicast come mittente nei pacchetti IPv6, inoltre questi indirizzi non possono apparire nelle regole di instradamento dei router.

Utilizzo di *ICMPv6*: Nella fase di autenticazione ogni nodo sensore necessita di un meccanismo per pubblicare il proprio indirizzo all'interno della rete *wireless* impostata su un canale radio specifico. Questa operazione avviene grazie

al protocollo *ICMPv6* [2][12] che definisce un protocollo di controllo (e di notifica in caso di errori riscontrati) attraverso scambi di messaggi tra host sulla rete *Internet*. La versione 6 di *ICMP* estende *ICMPv4* con una molteplicità di funzioni, che sono illustrate di seguito:

1 ***Neighbor discovery***: Risoluzione dell'indirizzo del *link-layer* o strato fisico del nodo vicino il quale sostituisce ***ARP*** che sta per *Address Resolution Protocol* del protocollo di controllo *ICMPv4*; Rilevamento di indirizzi duplicati **DAD** (*Duplicate Address Detection*); ***Router discovery*** o localizzazione del router;

2 ***Stateless address configuration:*** configurazione di indirizzi non standardizzati;

3 ***Multicast group management:*** gestione del gruppo *multicast*.

In dettaglio, l'*ICMPv6* esegue le seguenti operazioni:

- scopre il prefisso della rete e autoconfigura l'indirizzo globale Unicast del nodo;

- scopre i routers nella rete e altri parametri della rete quali MTU(*Maximun Transmission Unit*[3]) per la framentazione;

- risolve l'indirizzo (similmente ad *ARP*) del prossimo hop;

- infine rileva gli errori, gli indirizzi duplicati e i vicini non-raggiungibili.

Autoconfigurazione dell'indirizzo link-local di un nodo: Il nodo può determinare, ad esempio, il campo *identificatore dell'interfaccia* mappando a 64 bit il suo indirizzo MAC che è rappresentato dal formato *Ethernet* a 48 bit. Questo identificatore rappresenta il suo EUI-64 (identificatore univoco esteso a 64 bit) che viene associato al prefisso dell'indirizzo link-local noto (FE80::/64) per costituire l'indirizzo completo unicast link-local del nodo.

Successivamente, *ICMPv6 DAD*[12] verifica se l'auto-assegnazione degli indirizzi del nodo siano duplicati nella rete (e li scarta immediatamente). Poi, l'*ICMPv6* effettua l'operazione di scoperta del Router (più vicino) e ritorna al nodo il prefisso di instradamento globale noto al router con cui si collega direttamente al *link* di instradamento globale (autoconfigurato con il suo proprio indirizzo *Unicast*).

Meccanismo di compressione delle intestazioni di IPv6: Notiamo che *IPv6* dispone di una struttura semplice che ne consente un uso facile. Esso aggiunge 40 *Bytes* per ogni sua intestazione base generando un carico maggiore nella framentazione allo strato inferiore. A questo punto interviene il meccanismo di

[2] *Internet Control Message Protocol version 6*

[3] la dimensione massima consentita dalla rete per un pacchetto in trasmissione

compressione implementato da *6LoWPAN* con lo *standard 802.15.4* (adatto per *IPv6*). Nell'incapsulamento del pacchetto *IPv6* operato nello strato inferiore, *6LoWPAN* ne aggiunge altri 3 *Bytes* usati per la compressione.

Il primo byte di essi indicato da ***Dispatch*** segnala che la compressione dell'intestazione viene utilizzato e nello stesso tempo definisce il formato **HC** (*Header Compression*).

Il secondo byte rappresentato da **HC1** è un *bitmap*[4] che definisce quali campi di *IPv6* possono essere azzerati (o meglio assumere il valore zero) oppure essere ricavati dall'intestazione dello *standard 802.15.4* .

Infine, il terzo byte denominato da **HC2** è anch'esso un *bitmap* che specifica quali campi di **UDP** (*User Datagram Protocol*) possono essere ricavati dallo *standard 802.15.4* oppure sono già compressi (cioè *in-line*) nell'incapsulamento del pacchetto *IPv6*[12].

2.2.2 Transport layer: UDP

L'architettura del nodo sensore utilizza il protocollo *UDP* (User Datagram Protocol) per gestire il livello di trasporto, sopra il livello di adattamento 6LoWPAN. Questa scelta al posto dello *standard TCP* (*Transmission Control Protocol*) si giustifica dalla tipologia di connessione implementata dallo standard 802.15.4 .

Questa connessione è di tipo *stateless* a pacchetti e quindi non richiede un controllo di errori sui pacchetti in trasmissione ossia abilita una trasmissione **privo** di affidabilità contrariamente alla caratteristica principale di *TCP*. A questo punto, i pacchetti scambiati dai nodi possono essere dispersi nella rete senza interrompere il meccanismo di trasmissione al contrario di come funziona il protocollo *TCP* (fattore indicando la congestione della rete).

2.3 Layer applicativo: componente *BWS (CoaP)*

Lo strato applicativo definisce il formato e la gerarchia della codifica dell'informazione da pubblicare su un'interfaccia web. Questa componente è utilizzato nell'architettura dei nodi sensori utilizzati nel progetto europeo *Sensei*. Questi nodi implementano al livello applicativo il *Web-service*; il *web service* è una compenente che definisce un insieme di servizi web in grado di consentire una decodifica dell'informazione pubblicata in un formato standard delle applicazioni web, ad esempio tramite un *Browser*.

La funzione principale è quella di codificare i dati dal livello binario (grandezze fisiche dei trasduttori) e tradurli nel formato *RDF* (*Resource Description Format*). Quest'ultimo formato codificherà a sua volta il suo contenuto nel formato

[4] rappresentazione binaria avendo un particolare significato!

EXI-XML (*Efficient Xml Interchange - Extensible Markup Language*) decodificabile dal una qualunque interfaccia *Web* come il *Browser*.

Questa componente **BWS** ovvero *Binary Web Services* utilizza un modello di rappresentazione dei dati denominato *REST* che offre un'architettura software più simplificata per la codifica dei dati. *REST* definisce uno strato di protocollo leggero e implementa molte interfacce uniformi che permettono di indirizzare l'informazione da parte dell'applicazione web. Questa applicazione web individua questa risorsa tramite un unico indirizzo contenente tutte le informazioni per identificarla sul nodo. Tutte queste opzioni permettono al *BWS* di presentare un'organizzazione gerarchica dei contenuti e servizi per indirizzare le risorse in modo uniforme a differenza di RPC (Remote Procedure Call) e SOAP (Simple Object Access Protocol) che usano dei metodi web per indirizzare le risorse. La tabella 2.8 ci mostra una struttura di questo strato applicativo sui nodi sensei e verranno analizzate in seguito.

Layer	*Protocollo*
applicazione	**programma *nesC* eseguibile**
presentazione	**REST o RESTful (CoaP[13])**
sessione	**HTTP**

Tabella 2.8: Struttura del livello applicativo del nodo sensei.

Ora possiamo porsi la domanda del perché *BWS* si crede un modello utile in questo dominio. Ebbene, perchè esso consente una semplice integrazione del protocollo **HTTP** *Host To Transfert Protocol* realizzato seguendo l'architettura dell'ambiente **REST** o meglio *Representation State Transfer*. **REST** è un'ambiente (*Core*) poco rigido cioè *stateless* per rappresentare ipertesti distribuiti nel *World Wild Web* ed definisce un modo uniforme per accedere ai contenuti organizzati secondo una gerarchia predefinita e facilmente stoccabili (*cacheable*). Ricordiamo che l'ultima versione si chiama *RESTful* denominato anche con *CoaP*[13]. Questa versione è stata pubblicata di recente ovvero nel febbraio 2010. Infine, si può riassumere che il modulo *BWS* definisce una versione semplificata di *HTTP* per offrire la possibilità ad una qualunque interfaccia *Web* di visualizzare i dati codificati sul nodo sensore e ci accede interogando la risorsa tramite il suo indirizzo uniforme.

Il modulo CoaP:

- **Agent distributed**, definito dallo strato applicativo SOAP-messages;

- **Pipes and Filters**, tramite SOAP-Security;

- **Client server**, attraverso SOAP-Attachments.

2.3.1 *Interfaccia EXI*

EXI è l'acronimo di *Efficient Xml Interchange* ovvero una versione ottimizzata del formato **XML** (*Extensible Markup Language*) utilizzato in quasi tutti sistemi applicativi come formato standard di codifica e decodifica dei file dati. *EXI* è dunque un formato esteso di codifica dei dati in *XML* sviluppato dal gruppo di lavoro della *World Wide Web Consortium* denominato *W3C*. In breve, Si tratta di uno dei vari contribuiti di ottimizzazione per codificare/decodificare in dati binari i documenti *XML* in alternativa al formato standard testo o *plain-text*.

Va ricordato che il formato *XML* è considerato come una scelta conveniente per il *Web Service* perché essendo:

- standardizzato per le applicazioni web,

- pronto per le comunicazioni tra diversi sistemi operativi attraverso la rete,

- facile da analizzare grazie alle librerie presenti in quasi tutti i linguaggi di programmazione.

Inoltre, *XML* di per sè non è adatto per i dispositivi vincolati come i nodi sensori proprio perché **in primo**, necessita di una compressione per adattarsi alle dimensioni limitate del *payload*[5] e **in secondo** l'analisi del testo che porta ad allocare grandi dimensioni per i programmi assai problematica di fronte ad una limitata capacità della memoria *ROM* dei nodi. Si deduce dunque che il formato *EXI* è una rappresentazione compatta dei simboli o meglio della grammatica di *XML* e quindi diventa ideale per i dispositivi vincolati ovvero con capacità di elaborazione molto limitata.

2.3.2 Componenti *RAI e RPI*

La sigla **RAI** (*Resource Access Interface*) definisce una interfaccia per interrogare le risorse dei nodi. Esso consente l'accesso alla codifica in binario delle risorse ovvero i dati delle grandezze dei sensori integrati sui nodi sfruttando le utilità del servizio binario *XML*. Questo accesso restituisce il dato codificato dal *Binary Web Service* nel formato *exi-xml* disponibile per essere analizzato da un qualunque *Browser* del *World Wild Web* .

Questa procedura viene seguita in modo simile dalla componente **RPI** definito come una *Resource Publication Interface* cioè una interfaccia per la pubblicazione delle risorse o meglio delle informazioni aggiuntive chiamate *meta data* che sono dati descrivendo tutte le informazioni aggreggati ai sensori integrati sul nodo di interesse. Entrambe le componenti *RAI e RPI* sono elaborati dalla componente *BWS* per rispondere alle richieste dell'applicazione *Web*.

[5]sta per carico utile del pacchetto

2.3.3 Risorsa

L'identificazione delle risorse sul nodo avviene tramite l'utilizzo dei programmi *nesC* per interogare le grandezze fisiche definite dai vari *datasheet*[6] dei sensori come quelli descrivendo ad esempio i sensori di temperatura, luce, umidità, leds, batteria, bottone.

Questa identificazione si realizza tramite i rispettivi **URI** (*Universal Resource Identifier*) o meglio identificatori univoci per le risorse indispensabili nel modulo *BWS* da associare alle grandezze fisiche da elaborare. Possiamo ad esempio illustrare alcuni esempi di questi *URI* utilizzati nel codice del programma eseguito sul nodo:

- **/s/temp**, indirizza la temperatura,
- **/s/humy**, per l'humidità,
- **/s/light**, per la luce nello spettro visibile,
- **/s/irlight**, per la luce nello spettro infra-rosso.

A questo punto riassumiamo l'architettura del nodo sensore nella tabella 2.9

layer	*protocollo*
application	**SSL, . . .**
presentation	**XML, REST, RESTful, SOAP**
session	**HTTP**
transport	**TCP, UDP**
network routing	**IPv6**
link	**802.15.4**
physical	**serial tunneling, DSS**

Tabella 2.9: Stack dei protocolli implementati sul nodo sensore.

□

Implementazione di CoaP sul nodo sensore: Questa implementazione viene illustrata dall'architettura nella figura 2.6 e mostra in seguito un'astratto del codice scritto in *nesC*.

[6]documentazione che riassume le caratteristiche del componente elettronico o meccanico.

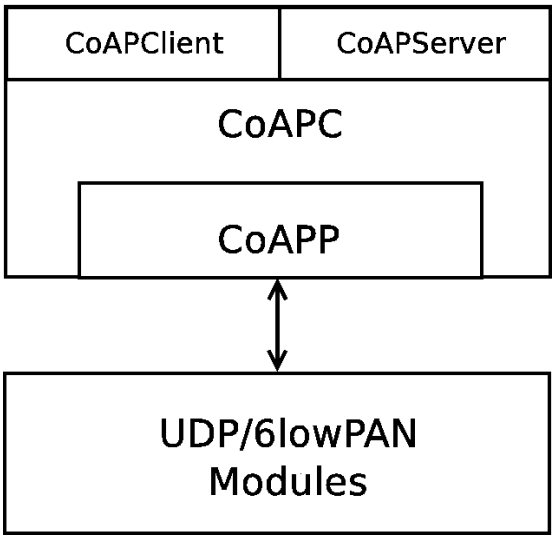

Figura 2.6: Struttura dell'implementazione di **CoaP** su **TinyOS**.

Ora presentiamo definire un'astratto del codice che implementa **CoaP** in nesC e eseguito sui nodi sensori. Come mostrato nella figura 2.6, La componente principale *module* è **CoAPP**, il quale prevede la realizzazione delle interfacce, mentre la componente *configuration* è **CoAPC** che collega i moduli *UDP/6lowPAN*, facilitando il suo utilizzo solo quando il cablaggio delle interfacce **CoAPClient** e **CoAPServer** è necessario. La componente *CoAPClient* è usato quando un nodo vuole inviare richieste a un server e l'interfaccia è strutturata come segue:

```
interface CoAPClient {

command coap_tid_t request (

coap_absuri_t* absuri,

coap_method_t method,

coap_content_t* content,

bool acked

);


event void response (

coap_status_t status,
```

```
coap_content_t* content

);

}
```

Il comando di richiesta (*request*) consente al nodo di fare una richiesta di una specifica risorsa su un server (definito dal parametro *absuri*) con un determinato metodo (definito dal *method*). Questa richiesta può trasportare informazioni passate attraverso i parametri *content*. Il parametro *boolean ACK* permette all'utente di inviare una conferma (**ACK = TRUE**) o meno per la richiesta effettuata.

Ora esaminiamo due strutture dati definiti dai codice seguenti: coap_absuri_t e coap_content_t.

```
typedef struct {

ip6_addr_t* host;

uint16_t port;

coap_uri_t *uri;

} coap_absuri_t;
```

Con la struttura di dati **coap_absuri_t**, una serie completa di informazioni sulla risorsa richiesta è descritta. the IP address of the server (**host**), la porta *UDP* **port** e la **uri** sono forniti.

```
typedef struct {

coap_char_t* data;

coap_length_t len;

coap_contenttype_t format;

} coap_content_t;
```

La struttura dati *coap_content_t* definisce i campi per le informazioni necessarie all'elaborazione del carico utile del messaggio inviato: un puntatore al carico

utile effettivo è presente (dati) con la sua lunghezza (**LEN**) e il suo formato codificato nei tipi *MIME*[7] (vedi la sezione 4.2.2) come prevede la norma sulla codifica dei tipi. *coap_length_t e coap_contenttype_t* sono degli interi senza segno codificato su 8 bit. Se una richiesta di conferma è inviata e la conferma **ACK** arriva, l'evento **response** è segnalato fornendo il codice di risposta (**statut**) e il dato ricevuto (**content**). Per il motivo di completezza si può esaminare l'interfaccia **CoAPServer** nella parte server.

```
interface CoAPServer {

event void request (
coap_rid_t rid,
coap_absuri_t* uri,
coap_method_t method,
coap_content_t* content,
bool toack
);

command error_t response (
coap_rid_t rid,
coap_status_t status,
coap_content_t* content
);
}
```

L'evento **request** è segnalato quando una richiesta arriva, se il parametro **toack** ha valore **TRUE** il server può rispondere alla richiesta utilizzando il comando **response**. Gli altri parametri sono gli stessi visti nella precedente interfaccia con l'unica differenza che il parametro **rid** significa **Request ID**; questo parametro *rid* è una struttura dati che fornisce varie informazioni sulla transazione corrente, come ad esempio **TID** che è l'*IP address* del *client* che ha

[7] *Multipurpose Internet Mail Extensions*

inviato la richiesta.

Queste informazioni sono vitali per la consegna della risposta corretta (se necessario) quindi devono essere fornite al comando **response**. La figura 2.7 ci illustra uno scenario di sincronismo tra Client e Server con uno scambio dei vari codice di risposta sfruttando i metodi del protocollo **HTTP** che vedremo in dettaglio nella sezione (4.2.2).

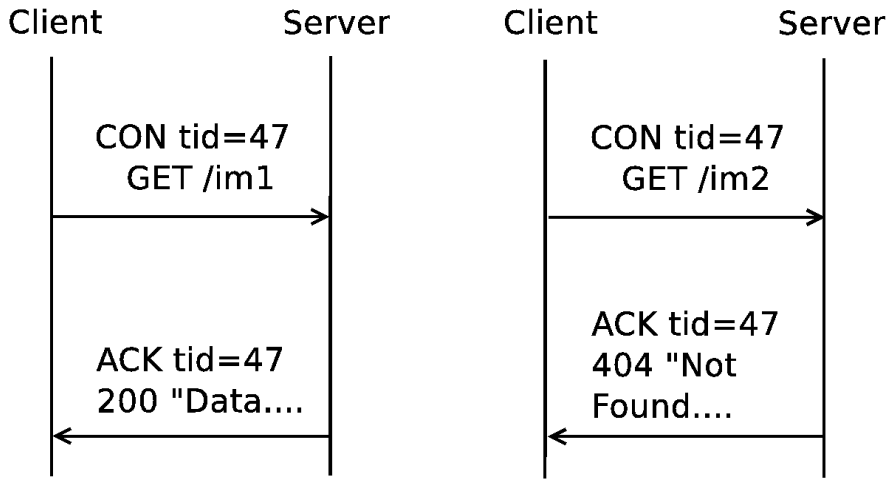

Figura 2.7: Schema di sincronizzazione tra il client e il server.

.

. □

3

Sistema operativo e linguaggio di programmazione adeguato

I requisiti hardware dei dispositivi vincolati ovvero dei nodi sensori costringono quest'ultimi all'uso di un sistema operativo capace di garantire una maggior efficienza nelle prestazioni durante il suo funzionamento. I sistemi operativi *standard*, maggiormente usati a giorno di oggi, nei dispositivi vincolati o *constrained devices* nelle reti wireless sono *TinyOS e Contiki*.

Ci limitiamo in seguito all'analisi di *TinyOS*[14, 15] perché scelto come sistema operativo rispondendo ai requisiti del nostro progetto. *TinyOS* è installato sui nodi sensori che costituiscono la rete wireless di nostro interesse. Sottolineiamo che la differenza sostanziale tra questi due sistemi operativi progettati per le piattaforme con limitate capacità di calcolo e di memoria, sta nel supporto da parte di *Contiki* della funzione di *multi-threading* cioè consente l'esecuzione parallela (simultanea) di più processi o *Thread* mentre *TinyOS* è di tipo *mono-threading* cioè esegue un processo alla volta.

3.1 *Tinyos*

Tinyos è un sistema operativo *open source* sviluppato dall'Università di Berbeley, in California. E' distribuito con la licenza *BSD*[1]. *TinyOS*[14] è un sistema operativo progettato per i dispositivi senza fili a bassa potenza (nodi sensori) ed è utilizzato nei vari progetti applicativi come ad esempio lo sviluppo di reti wireless private, di reti *WSN* per edifici intelligenti (rivelamento di intrusi con la video-sorveglianza, accensione e spegnimento automatico della fonte energetica per le luci, il riscaldamento, ecc.), di reti per il controllo d'allarme in caso di incendio, di reti per assistenza sanitaria a distanza sui pazienti ricoverati a domicilio, di reti per una video-sorveglianza di aree pubbliche dagli agenti di pubblica sicurezza come mostrato nella figura 3.1.

Inoltre, *Tinyos* è un sistema operativo *event-driven* ovvero un sistema operativo programmato dagli eventi con la caratteristica che non consente l'esecuzione

[1]BSD sta per *Berkeley System Distribution* e quindi è una distribuzione del sistemo operativo *Unix* di proprietà del gruppo di ricerca dell'Università di Berkeley.

Figura 3.1: Uno scenario degli apparecchi utilizzando TinyOS.

simultanea di più processi all'opposto di *Contiki* come illustrato precedentemente. Al livello di allocazione di memoria, *Tinyos* occupa uno spazio molto ridotto, condizione chiave per essere adatto a dispositivi con memoria limitata (qualche decina di kbyte). Inoltre, *Tinyos* fornisce dei moduli o componenti software ri-utilizzabili che possono essere abinati per implementare alcune funzionalità di base come ad esempio l'accesso ai registri del micro-controllore, la lettura e scrittura della memoria *EEPROM*[2], la gestione dell'interfaccia radio per la trasmissione e la ricezione dei dati. Non esite un *Kernel* per la gestione delle risorse (che le subdividerebbe in più applicazioni da gestire) ma dispone solo di uno *scheduler*[3] che si limita ad eseguire i *task* secondo una politica *Premptive*[4] ovvero seguendo la logica *FIFO* ossia First In First Out che significa eseguire il primo *task* ricevuto.

Questo meccanismo consente dunque allo stesso scheduler la possibilità di interrompere l'esecuzione del *task* trattenendo la risorsa nel caso si verificasse un evento con priorità magggiore. Infine, lo stesso *scheduler* porta il processore a riposo ovvero nello stato *idle* quando non ci sono più dei *task* in attesa di escuzione.

Si nota che l'allocazione di memoria in *Tinyos* avviene soltanto in maniera statica e quindi non dispone del meccanismo dell'allocazione dinamica di memoria. E' dunque un sistema operativo che non dispone di un sistema di gestione né dell'allocazione dello spazio di memoria né dei processi o meglio dei *task* generati. Infine, *Tinyos* non possiede la memoria virtuale ed tutte le sue operazioni sono non bloccanti cioè non blocca in maniera esclusiva l'utilizzo della risorsa assegnata ai processi.

Comunicazione nella rete: La comunicazione nella rete come implementata da *TinyOS* è basata su *Actives Messages* ovvero dei messaggi attivi. Questi messaggi sono inviati come dei frames[5] 802.15.4 (vedi capitolo precedente) cor-

[2]cioè *Electrically Erasable Read-Only Memory* vedi capitolo precedente.
[3]rappresenta un *task* per l'assegnazione della risorsa in caso di richiesta
[4]cioè consente la sospensione del processo durante la fase di esecuzione
[5]formato del pacchetto a livello fisico

rispondendo ad indirizzi di 16 bit distinti per ogni nodo sensore nella rete di interesse.

Di fatto, *TinyOS* li rappresenta con una struttura a pacchetto di tipo *message_t* che sfrutta a pieno le proprietà di allocazione di memoria per i variabili definite in modo ottimale nel linguaggio di programmazione standard *C*. Questi messaggi sono poi utilizzati dalle funzioni dedicate alla gestione della comunicazione attraverso i moduli di interfaccia indispensabili nell'architettura del linguaggio con cui questi moduli (o struttura a blocchi) vengono programmati: *nesC*[6]. Ricordiamo che lo sviluppo di *TinyOS* ha rilasciato diverse versioni quali le versioni 1.1, 2.0 e 2.1 . C'è da notare un'eccezione importante sulla versione 2.0 cioè che non integra alcune librerie della versione 1.1 mentre la versione 2.1 integra tutte le librerie delle versioni precedenti.

3.2 *nesC*

3.2.1 Definizione

nesC[7] è la denominazione di un'astrazione del linguaggio di programmazione standard C ed inoltre costituisce il linguaggio di sviluppo del sistema operativo *TinyOS*[15]. Infatti *nesC* è stato progettato da un gruppo di ricercatori di *Intel Research, Stanford University, U.C. Berkeley, Harvard University* proprio per sviluppare il sistema operativo *TinyOS*. *NesC* è un linguaggio di programmazione ad eventi, basato su componenti. Di conseguenza, è usato anche per scrivere i programmi applicativi per i *mote* che utilizzano *TinyOS*. Inoltre, *nesC* è un linguaggio statico privo di funzioni per l'allocazione dinamica di memoria ed questa caratteristica vale anche per i collegamenti dinamici.

Questo fatto favorisce un'analisi dell'intero programma in fase di compilazione con una conseguente ottimizzazione sul tempo computazionale. Va notato che il compilatore di *nesC* funziona similmente ad un pre-processore generando un programma eseguibile in linguaggio C come *output* . Tale programma eseguibile è dunque compilato utilizzando un compilatore **GCC**[8] [16] appropriato per la piattaforma su cui è in esecuzione (ad esempio i micro-controllori MSP430-gcc or AVR-gcc integrati sui nodi sensori).

I concetti principali che sono dietro la logica di *nesC* sono:

1 la separazione dell'implementazione dalla composizione delle componenti;

2 la specificazione del comportamento di una componente in termini di un insieme di interfacce;

3 la suddivisione delle funzioni delle interfacce in due gruppi: i comandi, funzioni implementate, da usare, e gli eventi, funzioni da implementare, che gestiscono le situazioni che si verificano di volta in volta nel sistema.

[6] astrazione del linguaggio C

[7] network embedded systems C.

[8] acronimo di *Gnu C Compiler* oppure *Gnu Compiler Collection* nella sua nuova versione.

3.2.2 Architettura del linguaggio

La struttura di *nesC* è basata su due entità principali: *Component e Interface* ossia Componente e Interfaccia. Tutta la logica dietro questo linguaggio (*nesC*) sta nel interagire con queste due entità. L'entità *Component* fornisce e utilizza le *Interfaces* mentre l'entità *Interface* è un insieme di *Comands and Events* ovvero di comandi e eventi; la figura 3.2 descrive lo schema a blocchi di queste entità. Le interfacce fornite rappresentano le funzionalità che una componente fornisce agli utenti; le interfacce usate rappresentano le funzionalità che una componente necessita per effettuare il proprio lavoro.

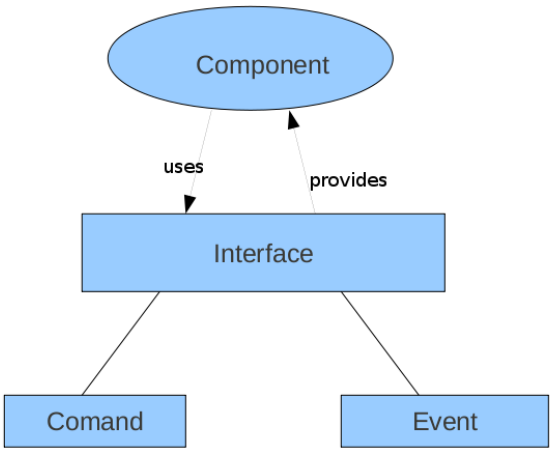

Figura 3.2: Schema a blocchi dell'architettura di nesC.

Component

Nel caso in cui un *Component* fornisce un *Interface*, i *Comands* sono funzioni forniti da questo *Component* e possono essere eseguiti da altri *Components*. Mentre gli *Events* possono essere segnalati da altri *Components* gestiti o implementati dallo stesso *Component* di volta in volta attribuendogli così la funzione di gestore di eventi. Perciò avere sia comandi che eventi consente una comunicazione bidirezionale tra i componenti. Poiché *nesC* è costituito essenzialmente di *Components*, *nesC* abilita quindi un collegamento specifico per connetterli e tutto questo avviene indipendentemente dalla loro implementazione.

I *Components* sono di due tipi[15]: *Configuration and Module* cioè configurazione e modulo. Entrambi i tipi vengono scritti in file separati salvati con l'estensione dei file di *nesC* (.nc) filename.nc

Configuration: il *Component* di tipo *Configuration* stabilisce il collegamento tra i vari *Interfaces* forniti o gestiti mediante una specifica sintassi (−>).

Ad esempio, Il seguente programma per *TinyOS*, chiamato **BlinkNodeAppC**, assembla la componente *BlinkNodeC*, la componente *MainC*, un'istanza della componente *TimerMilliC* e la componente *LedsC*.

```
/* BlinkNodeAppC.nc */

configuration BlinkNodeAppC { }


implementation {

components BlinkNodeC as App, MainC;

components new TimerMilliC() as MilliTimer0;

components LedsC;


....


App.Boot -$>$ MainC;

App.MilliTimer0 -$>$ MilliTimer0;

App.Leds -$>$ LedsC;

}
```

Module: *Module* rappresenta l'altro tipo di *Component* implementando le funzioni per le interfacce (*Interfaces*) che usa o fornisce e di conseguenza implementa tutte le funzioni necessarie per gestire eventi e comandi. Nel programma **BlinkNodeAppC** precedente, la componente *BlinkNodeC* (di tipo *module*) usa l'interfaccia *Boot* fornita da *MainC*, usa l'interfaccia *Timer* fornita da un'istanza della componente *TimerMilliC* e usa l'interfaccia *Leds* fornita dalla componente *LedsC*. Ecco un'implementazione delle funzioni per gestire gli eventi e i comandi.

```
/* BlinkNodeC.nc */


#include <Timer.h>

module BlinkNodeC {

   uses {
```

```
    interface Boot;

    interface Timer<TMilli> as MilliTimer0;

    interface Leds;

    }

}


implementation {

    uint8_t counter = 0;


    event void Boot.booted() {

     call MilliTimer0.startPeriodic(1024);

    }


    event void MilliTimer0.fired() {

       call Leds.set(counter);

       /* counter = (counter+1) mod 8; */

       if (counter == 7)

           counter = 0;

        else

            counter++;

       /***/

    }

}
```

Quindi ogni applicazione *nesC* si collega ad alto livello con la componente di tipo *Configuration* come illustrato nel codice del programma **BlinkNodeAppC**.

Si osserva inoltre che la concorrenza in *nesC* è realizzata tramite l'utilizzo dei *tasks* seguendo la logica già stabilita nel sistema operativo *TinyOS*.

Interface

Come accennato precedentemente, la struttura delle funzioni di *Interface* si definiscono in due gruppi distinti: le funzioni già implementati con la sintassi **comand** e quelle da implementare con la sintassi **signal** per gestire gli eventi che si realizzano di volta in volta nel sistema. Il codice del programma seguente illustra un esempio di sintassi nesC per i file di *Interface* (.nc).

```
/* Timer.nc */

#include "Timer.h"


interface Timer<precision_tag> {


// basic interface


 /**

  * Set a periodic timer to repeat every dt time units.

  * Replaces any current timer settings.

  * The <code>fired</code> will be signaled every dt

  * units (first event in dt units).

  *

  * @param dt Time until the timer fires.

  */

  command void startPeriodic(uint32_t dt);


 /**

  * Set a single-short timer to some time units in

  * the future. Replacesany current timer settings.
```

```
 * The $<$code$>$fired$<$/code$>$ will be signaled
 * when the timer expires.
 * @param dt Time until the timer fires.
 */
command void startOneShot(uint32_t dt);
/**
 * Cancel a timer.
 */
command void stop();
/**
 * Signaled when the timer expires (one-shot) or
 * repeats (periodic).
 */
event void fired();
// extended interface
/**
 * Check if timer is running. Periodic timers
 * run until stopped or replaced, one-shot timers
 * run until their deadline expires.
 * @return TRUE if the timer is still running.
 */
 command bool isRunning();
/**
```

```
   * Check if this is a one-shot timer.

   * @return TRUE for one-shot timers, FALSE

   * for periodic timers.

   */

   command bool isOneShot();

  /**

   * Return the current time.

   * @return Current time.

   */

   command uint32\_t getNow();

}
```

□

In conclusione è importante sottolineare che per compilare i programmi, i file .nc contenenti il codice *nesC* sorgente, viene utilizzato il potente ed estensibile sistema ***make*** dei sistemi *Unix*. Si supponga di essere nella directory che contiene i file ***BlinkNodeAppC.nc, BlinkCntC.nc*** e un ***Makefile*** con il seguente contenuto:

```
COMPONENT=BlinkNodeAppC

include $(MAKERULES)
```

Si può quindi compilare l'applicazione per una determinata piattaforma (**telosb**) e caricare il programma oggetto (file eseguibile) sul nodo sensore (Moteiv della crossbowTelos) digitando nella shell di *Linux* il comando:

```
make telosb install,id\_del\_nodo bsl,port\_serial
```

□

4

Architettura del sistema di interfacciamento con la rete di sensori

Il riferimento strutturale per lo sviluppo di questo sistema coglie la sua architettura da una delle entità di comunicazione con il *WSN* del progetto europeo Sensei[17]. In breve, il nostro sistema di interfacciamento realizza un modulo di comunicazione con un *gateway*, il quale rappresenta il punto di raccolta dei dati dei sensori della rete *WSN*; sul server, questo sistema implementa un modulo di *storage* per i dati raccolti dal *gateway* che vengono raccolti e salvati in modo persistente in appropriate tabelle definite nel *database*. Infine, è stato sviluppato un *plugin* per la visualizzazione grafica delle informazioni sui nodi interogati nel *testbed* a partire da un *browser web*. La figura 4.1 mostra una visualizzazione dei nodi attivi sul testbed.

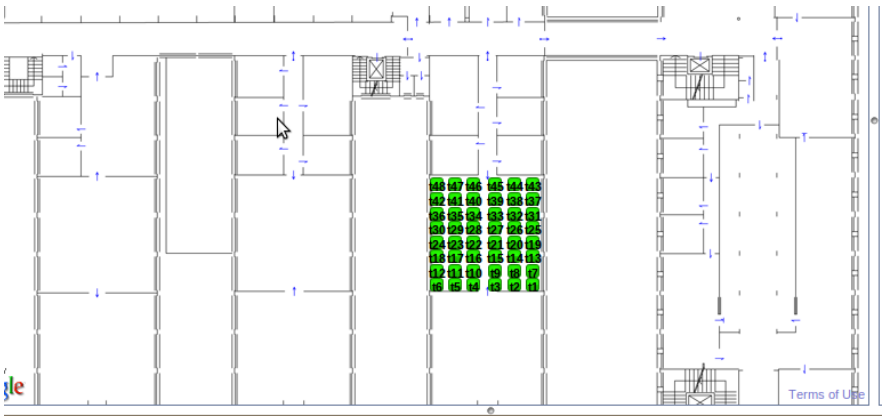

Figura 4.1: Visualizzazione dei nodi sul testbed tramite l'interfaccia grafica (*browser*).

4.1 Una visione del progetto *Sensei*

Il progetto europeo *Sensei* [17] è nato per l'innovazione di ***IoT*** *(Internet Of Things)* ovvero la rete Internet del futuro come illustrato dallo scenario della figura 4.2 e pensato per connettere alla rete Internet ogni entità digitale in grado di fornire determinati servizi web. Questo nuovo concetto di rete dovrebbe essere in grado di dare una visione della società come un insieme di entità digitali connesse sulla rete mondiale *Internet*. La realizzazione di una tale rete implicherebbe una rivoluzione sui principali campi di applicazioni e dei settori industriali come l'energia, i trasporti, l'assistenza sanitaria, contenuti multimediali, programmi di utilità e infine l'ambiente.

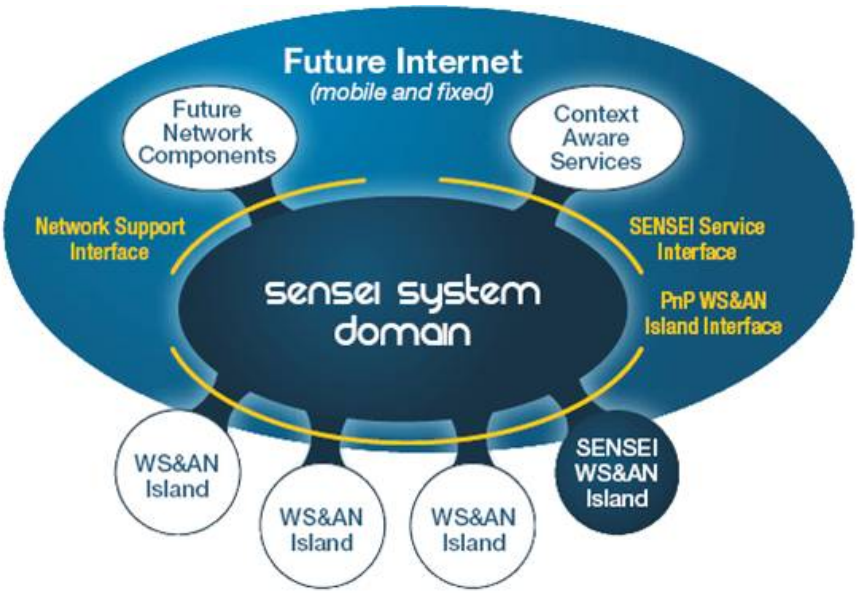

Figura 4.2: Uno scenario del progetto europeo *Sensei*: **Internet Of Things**.

Le trasformazioni tecnologiche di base in questi settori promettono significativi guadagni di produttività, aumentando l'efficienza e l'efficacia dei servizi erogati. Tuttavia, gran parte di questa trasformazione sarà volta a servizi e applicazioni nei vari settori industriali, in grado di disporre delle informazioni grazie alle proprie capacità di interazione con il mondo reale. Va notificato che la conoscenza delle informazioni dal mondo reale può essere effettiva solo se le informazioni riguardanti il mondo fisico sono catturate in modo automatico, idealmente in tempo reale rispetto alle esigenze derivanti.

Il progetto europeo *Sensei* ha affrontato enorme sfide di sviluppi per consentire questa trasformazione del mondo reale attraverso *Internet*. Di fatto, ha sviluppato una architettura con dei *building block*[1] tecnologici che servono come strumenti chiave per descrivere gli ambienti rappresentando i vari settori elencati precedentemente. La figura 4.2 mostra uno scenario di questo progetto *Sensei* dove ad

[1]entità digitale

esempio la componente *WS&AN* che significa *Wireless Sensor and Actuator Network*, rappresenta la componente *software* che sviluppa i fondamenti essenziali per la rete *Internet* del mondo reale. Distribuita ai bordi delle rete, questa componente fornisce un'infrastruttura che consente una scalabilità del mondo reale e interagisce con esso senza la necessità di un intervento manuale. Pertanto si nota che la struttura del dominio del sistema di interfacciamento di *Sensei* è costruita in cima ai strati del servizio di comunicazione dell'attuale rete *Internet* e si collegano ai *WS&AN* per processare le informazioni interrogate dalla corrispondente entità del mondo reale.

Questo approccio ci consente di descrivere la logica concettuale della nostra applicazione di interfacciamento. La figura 4.3 illustra dunque un'astrazione di questa logica progettuale rappresentando i *building block* del progetto europeo *Sensei* con dei nodi sensori di una determinata *WSN* che interagisce con un gateway ovvero il punto di raccolta dati per la rete definita.

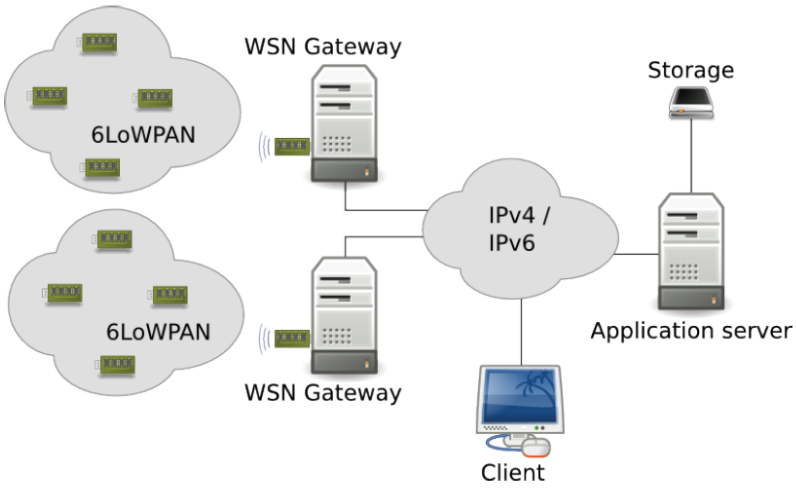

Figura 4.3: Descrizione progettuale del sistema di interfacciamento.

4.2 Gateway

Il Gateway si interfaccia alla rete *WSN* tramite un nodo *sink o BaseStation* che ha il compito principale di raccogliere/interogare le informazioni sulla rete *WSN*; abilita delle funzioni assai semplici nel suo ruolo di *forwarding* o ossia inoltra tutte le informazioni ricevute come indicato nella figura 4.4. Questo comportamento di*forwarding* si realizza, rimanendo in ascolto su una porta *TCP* tramite alcune funzioni principali che individuiamo ad un livello di astrazione con:

- **receiveComm()**, riconosce il comando codificato in *bit* inviato dal server per richieste sulla rete;

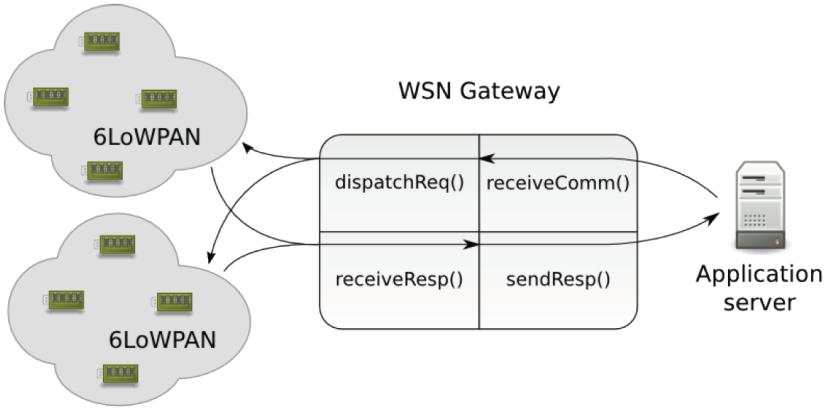

Figura 4.4: Architettura funzionale del *gateway*

- **dispatchReq()**, inoltra la richiesta del server alla rete;
- **receiveResp()**, riceve i dati inviati dai sensori interrogati;
- **sendResp()**, spedisce al server il risultato codificandolo nel formato specificato;

Le funzioni di **dispatchReq()** e **receiveResp()** sono implementati grazie ad alcune funzioni della libreria **curl-7.19.7** [18] scritte in ***C***. Queste funzioni abilitano dei *socket-http*[19] per stabilire la connessione con la risorsa ovvero il nodo sensore da interrogare attraverso il rispettivo indirizzo *URL* o ossia *the Uniform Resource Locator* del nodo interogato.

Inanzitutto notiamo che il nodo *BaseStation* o nodo di raccolta gode di una disponibilità permanente della risorsa energetica perché connesso direttamente ad una macchina di elaborazione fissa o *Personal Computer* in grado di processare, tramite determinate funzioni, varie informazioni e fornire dei servizi digitali all'utilizzatore, il quale rappresenta il *gateway* che può essere collegato sullo stesso *server* oppure su un'altra macchina che accede al server in modalità remota; la figura 4.4 mostra un'architettura concettuale di un gateway. Inoltre, questo nodo *BaseStation* interagisce con la rete *WSN*, quale implementa lo standard 802.15.4 per la comunicazione radio tra i nodi a differenza dell'architettura della normale rete *Internet*. In ogni caso il *gateway*, in esecuzione sul server o in remoto su un *PC*, consente dunque ad un utente *Web* di interagire con i nodi sensori utilizzando un qualunque *Browser Web* tramite il quale effetuare delle richieste e visualizzare in seguito le risposte pubblicate attraverso delle normali pagine web.

Va citata inoltre un'altra componente fondamentale (il *serial tunneling*) abilitata nel stratto fisico tra il *server* e il nodo *BaseStation* senza il quale tutto questo meccanismo di interazione del *gateway* non diventerebbe effettivo. Si tratta del programma *Serial_Tun* che è un programma, in esecuzione sulla stessa porta dell'applicazione *gateway* e, sviluppato nel *toolchain* di *TinyOS* contenuto nella cartella con un *path* simile a:

TOSROOT/tos/support/sdk/c/6lowpan/serial_tun

nel caso in cui il *server* abbia come sistema operativo un ambiente *Unix*.

4.2.1 Collegamento seriale di *TinyOS: Serial tunneling*

Una delle importanti inconvenienze nella comunicazione delle reti di sensori è la difficoltà a scambiare dati sfruttando l'indirizzamento *IP standard*.

Figura 4.5: Un'astrazione funzionale del *serial tunneling*.

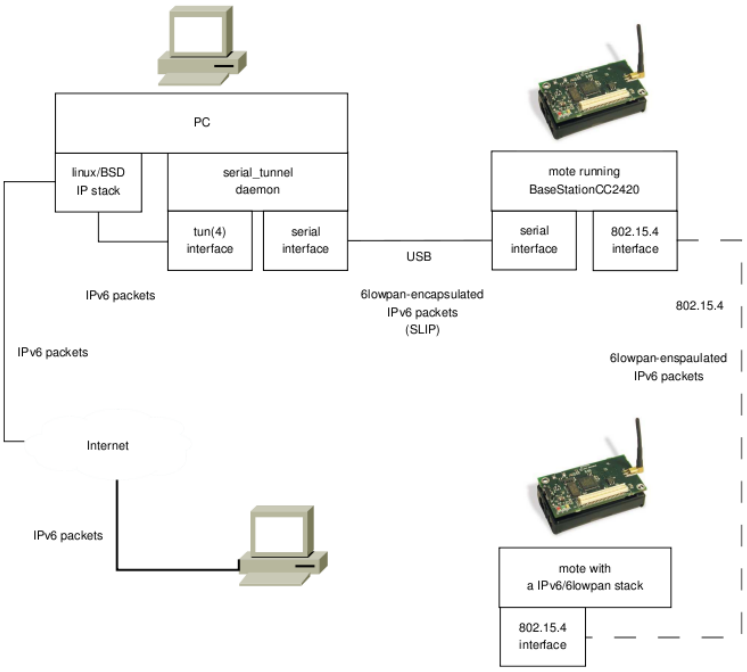

Figura 4.6: Architettura funzionale del *serial_tun* al livello fisico.

Il sistema operativo *TinyOS*, per risolvere questo problema, ci offre una importante funzione implementata nel programma ***serial_tun*** che ha come compito principale quello di incapsulare i pacchetti *6LoWPAN* in normali pacchetti *IPv6* e *viceversa* [9, 10]. Quando il nodo *BaseStation* riceve un pacchetto *6LoWPAN* trasportato su *UDP* e inviato da uno nodo sensore in ascolto sullo stesso canale radio della *WSN*, interpella il programma ***serial_tun*** il quale analizza la sua intestazione di trasporto (*UDP*) e capisce o meno di essere in grado di gestirlo; di conseguenza, questo programma fa scartare il meccanismo che effettua una corretta decompressione del pacchetto *6LoWPAN*. Le figure 4.5 e 4.6 illustrano la logica concettuale alla base dell'architettura funzionale mostrando come si realizza il suo funzionamento durante la comunicazione tra il server, la *WSN* e eventualmente il PC remoto collegato attraverso *Internet*.

Ora si può scambiare i dati nella nella rete *wireless* ma tenendo presente che il contenuto dei pacchetti *UDP* sia attendibile (nel corretto formato di codifica, ecc.) alle richieste della nostra applicazione e poi generare successivamente delle connessioni affidabili con i nodi della *WSN* per mantenere un'interazione assai efficienti. Il tutto avviene sfruttando l'applicazione *Gateway* sviluppata dal **SIGNET Group** che è il gruppo di ricerca di telecomunicazione associato al dipartimento dell'ingegneria dell'informazione presso l'Università degli Studi di Padova. Ora, analizziamo l'architettura principale di quest'applicazione **gateway** focalizzandoci sulla logica concettuale delle funzioni necessarie per il nostro sistema di interfacciamento.

4.2.2 Struttura e funzionamento del *Gateway*

Per risolvere le problematiche osservate nello scambio di dati durante la fase di comunicazione all'interno della rete di sensori, sono stati sviluppati dei ***Plug-in***[2] per l'applicazione *Gateway* in grado di consentire ai nodi di rispondere alle richieste dell'utente *Web* pubblicando le risorse interogate nel formato consentito dai normali pagine *Web*.

Struttura: *Plug-in* e *BWSsocket* Il *gateway* è un'applicazione interamente sviluppata in ***C***; è formato da un insieme di *Plug-in* independenti tra di loro ed ognuno dei quali costituisce un *thread* associato ad una singolare porta *UDP* attraverso il quale il nodo sensore interagirà con il corrispondente *Plug-in* che richiede la sua risorsa. Di seguito, i *Plug-in* generano ulteriori risorse per il *Web* ed è indirizzato con un *URL* specifico:

http://localhost:8000/sensei

Le risorse pubblicate dal *Browser* che chiama questo *URL* saranno utili ai fini di individuare una specifica risorsa assieme al nodo che lo pubblica. Inoltre, la risoluzione di questo *URL* indica che:

[2]accessori software che integrano le funzionalità accessibile tramite una interfaccia come un *Broswer*

- **http://**, il *Browser* usa il protocollo http per stabilire la connessione con il *gateway*;

- **localhost**, corrisponde all'indirizzo della macchina su cui è in esecuzione il gateway;

- **8000**, specifica il numero della port sul quale il *gateway* rimane in ascolto;

- **/sensei**, identifica la risorsa (*file*) del gateway che gestisce tutte le risorse pubblicate dalla rete *WSN*;

Di fatto, l'applicazione *gateway* è in esecuzione sul *PC* locale che può essere la stessa macchina del server; pubblica, grazie ad un *Browser Web*, dei contenuti codificati nel formato standard delle pagine *Web*; questi contenuti non sono altro che delle informazioni specifiche sulle risorse pubblicate che gli vengono inviati dal programma *serial_tun* in esecuzione sul nodo *BaseStation* che si interfaccia con la rete. Siccome tutta l'applicazione è sviluppata con i linguaggi ***C/C++***, ne segue anche lo sviluppo di questi *Plug-in*. Questa scelta dello stesso linguaggio consente un'ottimizzazione dell'intera applicazione.

Per svolgere correttamente questa funzione di interazione, ogni *Plug-in* è composto di due interfacce:

- il ***Plug-in reale*** con tutte le funzioni *standard* e,

- la ***BWSsocket*** ovvero la *Binary Web Service Socket*[3].

La *BWSsocket* elabora tutte le risposte corrispondenti alle rispettive richieste nel formato del pacchetto ricevuto dal nodo *BaseStation* e questo dopo la fase di decompressione eseguita dal programma ***serial_tun*** descritto precedentemente. La funzione principale della *BWSsocket* si riassume nel generare dei processi o *thread* per interagire facilmente con i dati contenuti nelle intestazioni o *header* del pacchetto oppure altri dati utili ad esempio quelli per processare i comandi ricevuti chiamando l'indirizzo *URL* della risorsa del nodo sensore da interogare, indirizzo *URL* digitato nel *browser*.

I nodi sensore possono rispondere alle richieste delle risorse dei vari sensori analizzando lo *URI*[4] della risorsa specificata nella richiesta assieme alle informazioni agguintivi incapsulati nel *Payload* del pacchetto *UDP*. Tutto ciò si realizza correttamente grazie alla *BWSsocket* che distingue queste informazioni gestite all'interno del pacchetto in un formato (*Raw data* [5]) che li rende indistinguibili dopo la decompressione eseguita dal programma *serial_tun* collegato la porta seriale del nodo *BaseStation*.

L'interfaccia *Plug-in reale* dal suo lato dispone di tutte le altre funzioni in grado rispondere alle richieste del *Web Service*.

[3]implementa delle funzioni per avvviare il socket che stabilisce la connessione con il nodo sensore tramite la porta *UDP* assegnata ad ogni *Plug-in* con il quale comunica.

[4]acronimo di *Uniform Resource Identifier ovvero identificatore uniforme della risorsa*

[5]dati grezzi o indistinguibili al livello fisico

Sfruttando dunque il paradigma di programmazione orientato agli oggetti del linguaggio *C++* che consente il concetto di ereditarietà e polimorfismo, si è sviluppato un *Plug-in* specifico di *BWSsocket* chiamato *SenseiBwsSocket* che registra tutte le risorse pubblicate da ogni nodo raggiungibile dal nodo *BaseStation*.

Funzionamento: Quando l'interfaccia *BWSSocket* riceve una richiesta oppure una risposta per una specifica richiesta, invoca un metodo di collegamento del *Plug-in*. Il socket[19] che corrisponde a questo *Plug-in* può accedere all'*URL* e quindi se si sta sviluppando un *Plug-in* complesso per gestire più servizi (ad esempio GET, PUT, POST, DELETE del protocollo *http*) si può, a questo livello, decidere quale sia il servizio richiesto quindi richiamando il metodo corretto del *Plug-in* per accedere a tale servizio. Una volta che la risposta della richiesta è arrivata al *Plug-in*, i dati possono essere inviati all'*browser* utilizzando ad esempio i servizi *HTTP* abiliti dalle funzioni del *browser* di qualunque dispositivo collegato a Internet. Dall'altro lato, l'interfaccia *BWSsocket* è quindi in grado di gestire altretanto le risposte generate dal *Plug-in* consentendolo in questo modo di rispondere alle richieste provenienti in questo caso dal nodo sensore e *così via*.

4.2.3 Metodi e Tipo di contenuto di *HTTP*

Metodi standard di HTTP: Nel paragrafo precedente sono stati accennati dei metodi standard del protocollo *http* ossia:

- **GET**, recupera la descrizione della risorsa identificata dalla *URI*. Se la richiesta ha successo, un codice **200 (OK)** viene inviato con la risposta; nel caso contrario il server restituisce al client, ad esempio, un codice **404 Not Found**;

- **POST**, viene utilizzato per richiedere al server di creare una nuova risorsa identificata con **URI** fornito. Se la risorsa esiste già un codice **200 (OK)** viene inviato in risposta, altrimenti un codice **201** (Creato) viene inviato in risposta. Nella risposta deve esserci una descrizione della risorsa aggiornata di recente;

- **PUT**, chiede l'aggiornamento delle risorse indicate nella **URI** fornita. I dati utilizzati per l'aggiornamento sono contenuti nel corpo della richiesta. Se l'aggiornamento ha successo un codice **200 (OK)** viene restituito, se la risorsa non esiste viene creato e un codice **201** (Creato) viene restituito in risposta.

- **DELETE**, Si chiede la soppressione della risorsa indicata. In caso di successo il codice **200 (OK)** viene inviato in risposta.

.
Inoltre, se un *end-point*[6] riceve un messaggio contenente un metodo sconosciuto genera un messaggio con un codice di risposta **405 Metodo non consentito**. La tabella 4.1 elenca i principali codici di risposta di *HTTP*:

[6]indica un *computer* in remoto.

HTTP Code	Description
100	Continue
200	OK
201	Created
304	Not Modified
400	Bad Request
404	Not Found
405	Method Not Allowed
415	Unsupported Media Type
500	Internal Server Error
502	Bad Gateway
504	Gateway Timeout

Tabella 4.1: Elenco dei codici di risposta di HTTP.

Content Type (MIME): **MIME** sta per *Multipurpose Internet Mail Extensions* il quale è uno standard di *Internet* che definisce il formato delle e-mail. Ricordiamo che MIME nasce con lo scopo principale di definire il formato delle e-mail inviate via il protocollo *SMTP*[7]. Comunque, viene usato anche nel protocollo **HTTP** per definire il tipo dei messaggi trasportati.

Nel *HTTP*, il **content type** è definito con una stringa (ad esempio **application/exi**). Ma inviare questo tipo di dato nei pacchetti può diventare oneroso per il processo di elaborazione da parte dei nodi; perciò viene mappato tramite un codice di *MIME* (**CT**) e un *bit* di identificazione predefinito dallo standard.

Nella tabella 4.2, vengono elencati i tipi supportati da *MIME* e il loro corrispondente codice (**CT**). I valori tra 201 e 255 sono riservate a specifici fornitori e utilizzati anche da applicazioni specifiche o sperimentali mentre i valori tra 0 a 200 sono mantenuti da **IANA**[8] che è il responsabile del coordinamento globale di *Internet Protocol*.

4.3 Architettura del sistema di interfacciamento

Rimanendo sempre nella logica strutturale del *Gateway* che risponde alle esigenze del nostro sistema, è stato sviluppato uno script applicativo in ***C/C++*** che interagisce con il gateway focalizzandomi su alcuni dati correlati misurati dai sensori (di temperatura, humidità, luce e luce_infra_rosso) integrati sul *mote*. Tali dati corrispondono alle rispettive risposte inviate dai singoli nodi avendo ricevuti specifiche richieste tramite il metodo *GET* con i loro rispettivi *URL*.

[7]Simple Mail Transfer Protocol è il supporto di trasporto delle e-mail su *Internet*.

[8]sta per *Internet Assigned Numbers Authority* ovvero l'autorità internationale che assegna gli indirizzi Internet

CT	**MIME Type**
0	plain/text (UTF-8)
1	text/xml (UTF-8)
2	text/csv (UTF-8)
3	text/html (UTF-8)
21	image/gif
22	image/jpeg
23	image/png
24	image/tiff
25	audio/raw
26	video/raw
40	application/link-format
41	application/xml
42	application/octet-stream
43	application/rdf-xml
44	application/soap-xml
45	application/atom-xml
46	application/xmpp-xml
47	application/exi
48	application/x+bxml
49	application/fastinfoset
50	application/soap-fastinfoset
51	application/json

Tabella 4.2: Elenco di tipi definiti da MIME.

4.3.1 implementazione

funzioni della libreria *libCurl*: Ogni singola richiesta del nostro sistema si realizza utilizzando alcune funzioni delle librerie ***libCurl***[9] quali implementano i metodi del protocollo *HTTP* utile per gestire le richieste; nel nostro sistema questa specifica funzione di **libCurl** chiamata **curl_easy_setopt** è definita con il seguente formato:

```
CURLcode curl_easy_setopt(

CURL *handle,
CURLoption option,
parameter

);
```

- **handle**, è un puntatore all'istanza che inizializza la sessione;
- **option**, è la codifica di quale funzione della libreria da chiamare per gestire il parametro indicato.
- **parameter**, indica l'istruzione da compiere dal parametro *option*

L'istruzione seguente descrive concettualmente il formato standard usato dal nostro sistema per indirizzare le risorse tramite il gateway:

```
http://hostname_gateway:porta_gateway/ipv6_nodo/URI_address
```

Nel nostro caso specifico, per interrogare il **gateway** con la funzione **curl_easy _setopt** ci servono tre parametri **option** con altretanto i parametri i quali abilitano dei ruoli distinti che concorrono ad implementare la nostra richiesta di dati. Di fatto, utilizziamo ad questa sequenza di istruzione per fare una **GET** sulla risorsa indirizzata dall'**URL** di cui una parte può essere già disponibile sul **gateway** in ascolto sulla porta 8000 come indicato nell'**URL**. Poi scriviamo il contenuto della risposta ricevuta su un file locale che gestiremo più tardi.

```
.....

/* set URL to get registered nodes */

curl_easy_setopt(handle,CURLOPT_URL,"http://localhost:8000/sensei");

/* no progress meter please */
curl_easy_setopt(handle, CURLOPT_NOPROGRESS, 1L);

/* send all data to this function  */
curl_easy_setopt(handle, CURLOPT_WRITEFUNCTION, write_data);
```

[9]è una nota libreria che implementa funzioni per il trasferimento di dati usando vari protocolli secondo la logica *client/server* noto [18] nella comunicazione *Web*

Di seguito descriviamo la definizione dei vari parametri [18] utilizzati nel nostro codice sorgente.

- **CURLOPT_URL**: è la funzione di *libCurl* che abilita il **GET** sull'**URL**;
- **CURLOPT_NOPROGRESS**: rilascia le librerie <curl/*> in uso; il valore è indicato da *parameter*;
- **CURLOPT_WRITEFUNCTION**: è un puntatore alla funzione indicata dal *parameter* implementato nel codice sorgente.
- **http://localhost:8000/sensei** è l'**URL** della risorsa;
- **1L** definisce il valore di tipo intero lungo che indica il rilascio delle librerie usate;
- **write_data** è la funzione che scrive i dati ricevuto dal metodo **GET**.

Tale richiesta viene inoltrata al *Gateway* (**localhost:8000**) che a sua volta la inoltrerà secondo la risorsa specificata nel **url**. Nel caso in cui la richiesta non gli sia indirizzata come ad esempio la descrizione di tutti gli **url** delle risorse che ha registrato nella rete **WSN**, inoltrerà tale richiesta al ***Plug-in BWSsocket*** che saprà gestire le informazioni in essa contenuta (risolvendo l'indirizzo dell'**url**); dopo avere capito quale risorsa interrogare sulla rete **WSN**, inoltrerà la richiesta, in un pacchetto codificato in binario, al *serial_tun* che successivamente lo impacchetterà seguendo il modello del pacchetto *6LoWPAN* da inviare alla rete. A questo punto, la *BaseStation* riceve la richiesta impacchettata con *6LoWPAN* e lo dissemina nel canale radio della rete da una porta specifica in modalità *Broadcast* e rimane in attesa della risposta da parte del nodo destinatario di cui indirizzo *IPv6* è stato ricavato nel *URL* per il routing. Con l'**url** seguente:

```
http://localhost:8000/0000-0000-fe80-003c/s/temp
```

il **gateway** riceve la richiesta poichè gli è indirizzata (**localhost:8000**) e capisce sempre dall'**url** che si tratta di una richiesta da inoltrare all'indirizzo del nodo noto nel **url** assieme all'**uri** del sensore da interrogare; perciò inoltra la richiesta direttamente a questo nodo sensore tramite il suo **Plug-in** che stabilisce tale connessione con il nodo **BaseStation**. Ovviamente, il nodo deve essere registrato dal *gateway* nel file di descrizione degli **url** pubblicati da tutti i nodi della rete **WSN** altrimenti il *gateway* va in **TIMEOUT** e genera un messaggio di:

```
"Error 404: The requested URI was not found."
```

Quando il nodo destinatario riceve la richiesta, risponde successivamente dopo aver attivato i *Components* **Configuration e Module** che gestiscono l'evento per l'accesso al *datasheet* della risorsa dopo aver decifrato l'indirizzo **uri** della risorsa codificato nell'**url** inviato al gateway da parte del richiedente.

Una volta letto la misura del sensore interogato, il risultato viene incapsulato (o meglio codificato) dal nodo sensore secondo i meccanismi del *BWS* (vedi capitolo precedente) e nel rispetto dei requisiti al livello fisico della rete **WSN**. Dopodicché, il pacchetto generato verrà incapsulato secondo la struttura di **6LoWPAN** e inviato alla sorgente (nodo *BaseStation*). Nel caso non dovesse arrivare nel tempo di attesa prestabilito dal gateway, quest'ultimo genererà un messaggio di errore dovuto alla scadenza del **TIMEOUT** così indicato:

```
"Error 503: The gateway timed out while waiting for the response"
```

Se invece, il nodo *BaseStation* riceve questa risposta entro la scadenza del *timeout*, l'inoltrerà al **serial_tun** che provvederà a decomprimerlo per poi mandare il contenuto al livello superiore ovvero al **Plug-in BWSsocket** che avrà generato precedentemente la richiesta al nodo.

Il **Plug-in reale** decodificherà il contenuto e pubblicherà i dati in formato *MIME* di tipo **exi-xml** sulla porta 8000 da dove è accessibile l'applicazione gateway. Questo valore è il numero di porta predefinito da **SIGNET Group** autore dello sviluppo del codice dell'applicazione *gateway*. Questi dati saranno restituiti alla funzione **curl_easy_setopt** che li scriverà nella funzione **write_data**. Successivamente, questi dati vengono salvati in un file_stream definito localmente.

funzioni della libreria *libxml2*: Attraverso degli *script* implementati in *C/C++* e grazie ad alcune funzioni della libreria **libxml2**[10], viene ricavato il valore della misura e convertito nel formato predefinito **float** per tutte le misure considerate durante la simulazione del nostro sistema.

Una volta che il dato ovvero il valore della misura è salvato su un file locale, viene usato la funzione seguente per effettuare il *parsing*[11]:

```
float retrieveValue(const char* docname)
```

dove il parametro **docname** è il file (*xml*) contenente i dati restituiti con la funzione **write_data**. Questa specifica funzione usa alcune funzioni della libreria **libxml2** per ricavare il dato (formato **string**) ricercato nel file e successivamente convertirlo nel formato **float**. Ora il dato è disponibile per essere salvato nel **database** e usato successivamente nella fase di ricostruzione con l'algoritmo di *CS*. Notiamo che tutte le altre informazioni pubblicate dal gateway vengono recuperate seguendo una elaborazione simile.

4.4 Database

Uno dei requisiti del nostro sistema è lo stoccaggio dei dati raccolti dalla rete dei sensori. Dati che devono mantenere alcune proprietà sul loro stato le quali consistenza e persistenza ovvero essere mantenuti in modo permanente e senza

[10] libreria disponendo delle funzioni standard per il *parsing* o analisi dei file in formato *xml*

[11] cioè per localizzare la stringa corrispondente a questo valore nel file *exi-xml*

alterazioni nel *database*. La loro consistenza cioè validià, precisione, utilizzo e integrità sono gestite tramite la descrizione del formato delle tabelle in cui verranno allocati; assieme alle funzioni che consentono l'accesso al *database o db*.

4.4.1 La libreria MySQL

Ora definiamo il database usando **MySQL**[20] che è un software per la gestione dei database; Come lo indica anche il suo nome, è un sistema che implementa il linguaggio **SQL** ovvero *Structured Query Language* che è un linguaggio di interrogazione per database progettato per leggere, modificare e gestire dati memorizzati in un sistema. La libreria *Mysql* è interamente scritta in **C/C++**[16]. Inoltre, usa delle funzioni **Socket**[19] per stabilire una connessione con il sistema del **database** il quale abilita l'uso dei credenziali per consentire l'accesso al sistema relazione chiamato anche un *RDBMS*[12].

Inoltre, **MySQL** è un sistema di gestione relazionale di archiviazione dati o meglio un *RDBMS* ovvero un *Relational DataBase Management System* composto da un client con interfaccia a caratteri e un server, entrambi disponibili per i sistemi supportando Unix come *GNU/Linux*[21]. Si ricorda che il sistema operativo su cui è stato sviluppato tutta l'applicazione di questa tesi è **Ubuntu 10.04**, un ambiente integrato a *GNU/Linux*. Quindi oltre a questo scopo di retroazione, questo *RDBMS* consente anche la creazione del database con il quale interagisce il nostro sistema applicativo nella fase di stoccaggio dei dati raccolti dal nodo *BaseSation* e di quelli ricostruiti con il modulo applicativo **Rebuilder**[13] [22].

Tuttavia, all'interno del sistema le funzioni della libreria **MySQL** implementa la **sintassi** del linguaggio **SQL**[20] per creare il **database** dentro il quale verranno creati le tabelle che allocheranno i dati da salvare. Eccone alcuni esempi di questa sintassi:

```
->CREATE DATABASE database_name;

->CREATE TABLE table\_name (column_name[,column_name,...]);

->INSERT INTO gateway.Measures (.........) VALUES (......);

->SELECT * FROM gateway.Measures;}
```

Questa sequenza di istruzione si definisce grazie alla funzione di ogni sintassi SQL usata. Ricordiamo quindi che:

- **CREATE DATABASE database_name** , crea il database che allocherà le tabelle;

- **CREATE TABLE table_name** , crea la tabella con la descrizione dei records (campi) delle colonne;

[12]acronimo dei sistemi relazionali per la gestione dei database.
[13]è un'applicazione sviluppata in *C/C++* da *D. Zordan* per la ricostruzione dei dati numerici

- **INSERT INTO table_name VALUES** , inserisce i dati VALUES dentro le colonne specificate della tabella;

- **SELECT * FROM table_name** , seleziona tutte le colonne della tabella;

- **;** , è il delimitatore della sequenza d'istruzione indispensabile per il compilatore del linguaggio.

Ora possiamo dare un'esempio di istruzioni generate dalle funzioni **Mysql** per creare la tabella **Measures** utilizzata nel nostro sistema di stoccaggio dei dati raccolti e utile per istuire l'algoritmo di ricostruzione **CS**.

```
CREATE TABLE ‘Measures‘ (

  ‘idMote‘ int(11) NOT NULL,

  ‘idSensor‘ int(11) DEFAULT NULL,

  ‘value‘ float DEFAULT NULL,

  ‘timer‘ timestamp NOT NULL DEFAULT ’0000\--00\--00 00:00:00’

) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Va notato che il tipo del sistema (**MyISAM**[20]) per la gestione è stato scelto tenendo conto di come verrà utilizzata questa tabella in seguito. **MyISAM** è utilizzato per le tabelle che gestirà maggiormente l'istruzione di inserimento dati rispetto alla loro selezione; per questo motivo **MyISAM** è noto in termine di stabilità della tabella. Mentre il tipo **InnoDB**[20] è noto in termini di velocità ovvero una maggior selezione dei dati e pochi inserimenti di essi con le **query**[14] appropriate.

In breve notiamo che tutte le tabelle create nel database ovvero il **RDBMS** presentano una struttura ben definita nelle funzioni di **Mysql** utilizzate per questo scopo nella nostra applicazione.

Definizione del database e l'uso dei dati.

E' importante implementare un meccanismo retroattivo sui dati ricostruiti negli ultimi istanti precedenti in modo da mantenere sempre una delle proprietà fondamentale nel processo di ricostruzione che richiede ad esempio che i dati richiesti all'istante **k-1** siano completi per consentire la ricostruzione di quelli mancanti all'istante **k** e così via.

A questo punto il nostro sistema *RDBMS* genera la creazione di un *database* ovvero un'entità strutturale dentro la quale verranno create le varie tabelle per

[14] definisce la sequenza di istruzione nel linguaggio SQL.

Tabella	Colonne
testbed	pTX, idGateway, idMote, idSensor, ToM, value, timer
dataMeasured	pTX, idGateway, idMote, idSensor, ToM, value, timer
Measures	idMote, idSensor, value, timer
dataRebuilded	pTX, idMote, idSensor, value, timer

Tabella 4.3: Descrizione delle *table* definite nel database.

archiviare i dati raccolti e ricostruiti. Nel nostro caso, questo database si chiama *gateway* e al suo interno vengono create le tabelle *testbed, dataMeasured, Measures e dataRebuilded*. Tali tabelle sono definite con la descrizione mostrata nella tabella 4.3.

La tabella *testbed e dataMeasured* hanno la stessa e identica definizione dei campi poiché la tabella *testbed*[15] serve a gestire soltanto l'archiviazione dei dati raccolti ad ogni istante mentre quella di *dataMeasured* è per archiviare in modo persistente tutti i dati raccolti dal nodo *sink* ad ogni istante partendo dalla prima istante della fase di inizializzazione del modulo di ricostruzione (*Rebuilder*) ovvero dai dati raccolti nella prima istante di ricostruzione. Perciò ognuna di loro archivia tutti dati oltre a quelli descrittivi del nodo sensore nella rete. Quindi definisce i campi seguenti:

- **pTX** (*float*), probabilità di ritrasmission in questa istante;
- **idGateway** (*int*), identificatore del nodo *sink* nella testbed;
- **idMote** (*int not null*), identificatore univoco del nodo nel testbed;
- **idSensor** (*int*), identificatore numerico del tipo di misura richiesta;
- **ToM** (*char(20)*), descrizione in formato *text* del tipo di misura richiesta;
- **value** (*float*), valore o dato di lettura raccolto dal nodo *sink*;
- **timer** (*timestamp*), istante di tempo per la raccolta dei dati;

Successivamente la tabella *Measures* definisce soltanto i campi necessari per i requisiti del formato del pacchetto da inviare al modulo *Rebuilder*. Quindi definisce i campi *idMote, idSensor, value e timer* identici a quelli della *dataMeasured*. Infine la tabella *dataRebuilbed* definisce altretanto gli stessi campi (per archiviare in modo persistente i dati ricostruiti) in più definisce il campo *pTX* per indicare la nuova probabilità calcolata da utilizzare per la prossima ritrasmissione dei dati (prossima istante di interrogazione dei nodi sensori) sul testbed ovvero nella rete dei sensori definita nel canale radio del nodo **BaseStation**.

[15] individua anche in modo univoco l'area di copertura della *WSN* interrogata

4.5 Funzionamento del modulo Rebuilder

Uno dei punti importante nel funzionamento corretto della nostra applicazione è la fase di ricostruzione dei dati raccolti dal **gateway**. Si nota la sua efficienza quando il **gateway** va in timeout per la manca ricezione della risposta di una specifica richiesta. In breve, il modulo **Rebuilder** riesce a ricostruire questo dato mancante all'istante specifica sfruttando la sua correlazione con i dati precedenti assieme alla stima.

4.5.1 Definizione

L'applicazione del *Compressive Sensing* (**CS**)[3] è rappresentata dal modulo **Rebuilder** nel nostro sistema come lo mostra la figura 4.7. Inoltre, ci consente di avere una visione strutturale del nostro sistema.

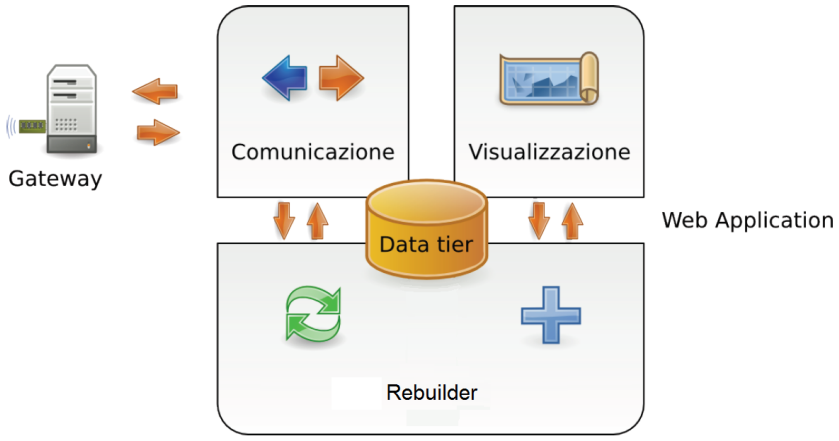

Figura 4.7: Architettura dei moduli del sistema di interfacciamento.

Vediamo la definizione funzionale dei blocchi della figura 4.7.

- **Comunicazione** è il modulo che rappresenta tutti i meccanismi per avviare le richieste delle risorse sul *gateway*;
- **Visualizzazione** definisce l'interfaccia **Web** per visualizzare i nodi della rete con i dati trasmessi;
- **Rebuilder** è il modulo di elaborazione del processo di ricostruzione dei dati mancanti ad un certo istante;
- **Data Tier** rappresenta il nostro sistema relazionale ovvero il database.

Una volta che i dati sono stati raccolti dal modulo di **Comunicazione**, essi vengono salvati nel modulo **Data Tier** o meglio il *database*. In seguito, questi dati verranno inviati, nel formato indicato nella figura (4.7), al modulo **Rebuilder** che procederà alla ricostruzione di quelli mancanti e poi restituire indietro tutti

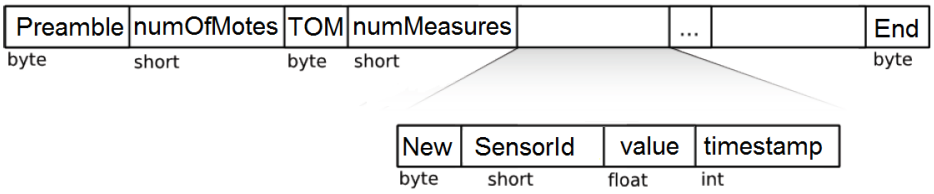

Figura 4.8: Formato del pacchetto richiesto dal **Rebuilder**.

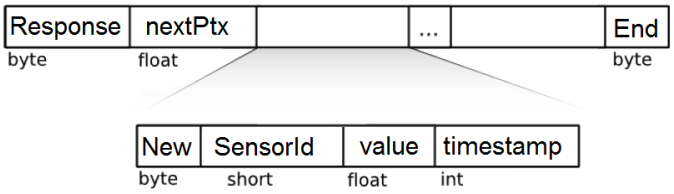

Figura 4.9: Formato del pacchetto ricostruito dal **Rebuilder**.

i dati ricostruiti assieme a quelli raccolti tramite il formato illustrato nella figura (4.8) dove i vari campi hanno il significato seguente:

- **Preamble :** indica l'inizio del pacchetto da ricostruire;
- **numOfMotes :** di tipo *short*, sta per numero di nodi sensori totali nella rete;
- **TOM :** tipo di misura codificato in *byte*;
- **NumMeaasures :** indica il numero di misure (o profondita) per ogni sensore durante la ricostruzione;
- **Response :** è il preambolo di tipo *byte* indicando che si tratta di un pacchetto di risposta alla ricostruzione dei dati raccolti;
- **nextPtx :** di tipo *float*, indica la nuova probabilità di ritrasmissione alla prossima istante (o richiesta dati sulla rete);
- **New :** è un preambolo di tipo *byte* che indica l'inizio dei dati di nodo sensore;
- **SensorId :** indica l'identificatore del nodo sensore di tipo *short*;
- **value :** indica il dato ricostruito di tipo *float*;
- **timestamp :** rappresenta l'istante temporale di ricostruzione;
- **End :** preambolo di fine pacchetto di ricostruzione.

Il punto centrale della ricostruzione si basa su alcuni principi mostrati nella figura 4.10. Si trattano dell'errore di ricostruzione del segnale all'istante corrente e la stima basata sul numero dei dati raccolti negli istanti precedenti salvati nel database.

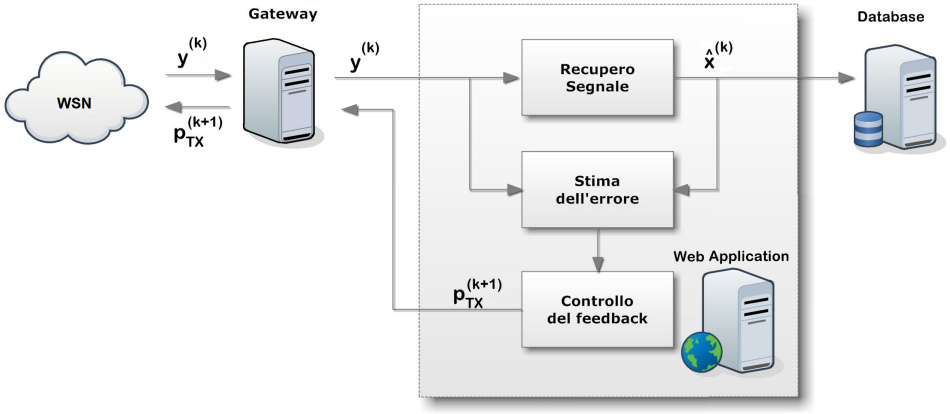

Figura 4.10: Scenario di ricostruzione dei dati raccolti nella **WSN**.

Nel dettaglio, analizziamo come avviene il calcolo della probabilità di ritrasmissione:

- **Recupero del segnale**, ricostruisce $x^{(k)}$ il vettore completo dei dati (compresi quelli raccolti);
- **Stima dell'errore**, determina il confronto tra i dati raccolti e il nuovo vettore dei dati (compresi quelli ricostruiti);
- **Controllo del feedback**, determina il nuovo valore della probabilità di ritrasmissione al prossimo istante calcolata a secondo dell'errore sui dati raccolti all'istante attuale.

In seguito, la probabilità di ritrasmissione verrà utilizzata per determinare quanti nodi sensori da interrogare nella **WSN**; e di conseguenza generare un risparmio sulla gestione della risorsa energetica della rete nel caso in cui tutti nodi siano alimentati soltanto con le batterie.

4.6 Visualizzazione grafica

Il **Plug-in** di visualizzazione grafica è stato sviluppato per osservare , tramite un *browser*, i dati raccolti e ricostruiti (con CS). Queste informazioni appartengono ad ogni singolo nodo sensore nel testbed e visualizzato attraverso **Fandango** una *Web Application* sviluppata dal **SIGNET Group** per la visualizzazione dei **testbed** all'interno del **DEI** (*Dipartimento dell'Ingegneria dell'Informazione*). La figura 4.11 ci mostra una parte del testbed.

Questo **Plug-in** realizza dunque una comoda interfaccia grafica *Web* utilizzata da un utente web per visualizzare il **testbed** di del DEI a partire da un

Figura 4.11: Una mappa dei testbed del **DEI**.

qualsiasi *browser* . L'obbiettivo principale prefissato è quello di implementare un meccanismo per visualizzare sia i dati raccolti dal nodo **BaseStation** (in ascolto sulla *WSN* ovvero sui **testbed** selezionati) e sia quelli ricostruiti dal modulo *Rebuilder* a causa delle perdite dei pacchetti nel canale radio di trasmissione; perdite che possono essere dovute a varie problematiche sulla rete come ad esempio la scadenza del timeout del pacchetto provocato dalle interferenze magnetiche che corrompono i pacchetti originali oppure la congestione al livello dei singoli **hop**[16] noti come dei **FFDs** o meglio dei *Full-Function Devices* per l'instradamento verso il destinatario a differenza dei normali nodi sensori denomiati **RFDs** o meglio *Reduced-Function Devices*, ecc.

In sintesi questa *Web Application* viene implementata seguendo l'architettura *Three-Tier*[17]. Come lo indica la sigla, questa architettura si rappresenta su tre stratti applicativi *Client-Tier, Application-Tier o Business Logic e Data-Tier* come viene illustrato nella figura 4.12.

Si nota che ogni stratto di questa architettura implementa delle funzionalità specifiche che vengono analizzate in dettaglio cui sotto.

4.6.1 *Client-Tier*

Il *Client-Tier* è lo stratto più flessibile e eseguito da un *Browser Web* così da garantire l'accesso a questa visualizzazione da un'ampia gamma di dispositivi quali i *Personal Computer*, apparecchi telefonici o *PDA (Personal Digital Assistant)* a partire da qualsiasi luogo come mostrati nella figura 4.13.

[16]nodo sink che abilita una funzione supplementaria ovvero l'instrademento dei pacchetti.
[17]struttura per rappresentare l'architettura *standard* di sviluppo dei *software* o applicazioni nel mondo del *software-engineering*.

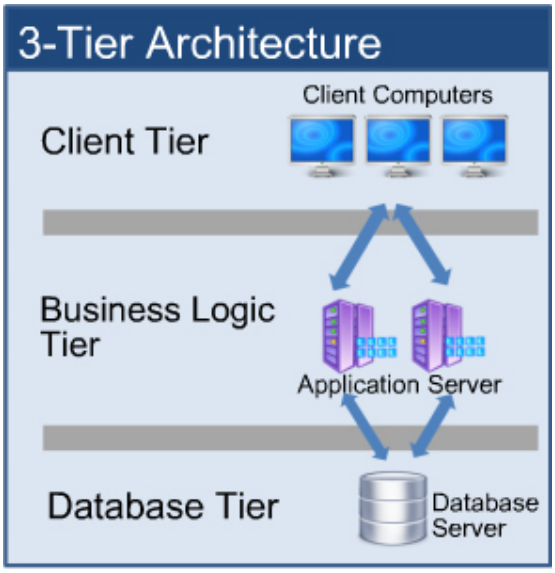

Figura 4.12: Architettura *Three-Tier*.

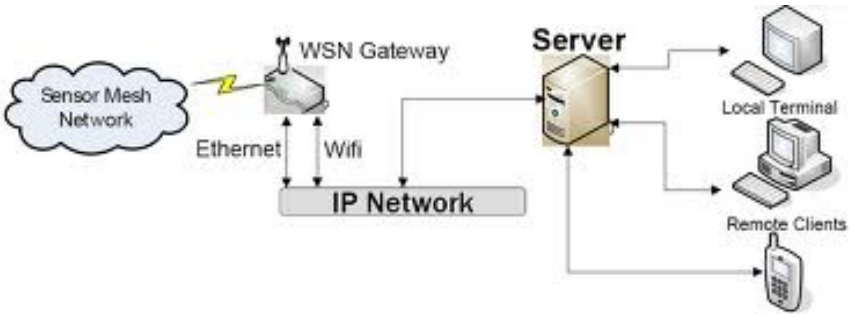

Figura 4.13: Scenario generico di interazione tra server e i dispositivi con *Browser*.

4.6.2 *Application-Tier e Data-Tier*

L'*Application-Tier e il Data-Tier* sono eseguiti sulla stessa macchina (*server*) dove risiede anche il nostro modulo di *Comunicazione* con il *gateway* ma possono eseguirsi altretanto anche su *computer* distinti. Comunque, questa scelta ci consente di ridurre le problematiche legate al tempo computazionale dell'intero sistema se entrambi i *tier* fossero condivisi su più *server* e quindi incrementare certamente e considerabilmente l'andamento temporale dell'intera applicazione.

Application-Tier: Viene chiamato anche *Business logic Tier* e descrive tutta la logica applicativa o l'insieme delle regole di **business** (funzioni) che rende operativa il nostro **Plug-in** di visualizzazione dei dati. Questo stratto progettuale è stato interamente implementato con l'architettura concettuale del *software-engineering* denominato **Java EE**[18] [23] sfruttando vari **tools** come ad esempio il **GWT** ovvero il *Google Web Tools*; il tutto integrato nell'ambiente di svilup-

[18]Java Enterprise Edition

po dei software applicativi **Eclipse**[24] che è stato scelto per i numerosi vantaggi degli strumenti che offre nel mondo della programmazione a tutt'oggi. Sarebbe comunque opportuno, anche per questioni di sicurezza, se questo stratto applicativo fosse installato su un server a sè poiché tutto il nostro sistema relazionale *Data-tier* si esegua sullo stesso *server* dove avviene la maggior parte di elaborazione e scambio di dati nel nostro sistema.

Data-Tier: Il *Data-Tier* rappresenta semplicemente l'entità *database* del nostro sistema come descritto nella sezione precedente. Questo stratto applicativo realizza tutte le regole di accesso al nostro sistema di archiviazione e cioè il nostro *database* **gateway**.

Tuttavia, il *Data-Tier* è realizzato dal sistema relazionale di base di dati **Mysql** utilizzando alcune librerie di *MySQL*[19]; sistema relazione nel quale risiedono i dati raccolti e realizza un sistema di autenticazione sugli accessi attraverso richiesta di credenziali dell'utente. Questo meccanismo introduce un controllo che garantisce una minima protezione sull'accesso al database. Una volta all'interno del database, tutte le istruzioni per maneggiare i dati salvati nelle tabelle sono implementati con *SQL*.

JDBC o Java Database Connectivity

Ci limitiamo qui a definire le tecniche che possono usare i programmi *Java* per connettersi con il database. Il programma *java* implementato genera una **query** corretta e elabora i dati estratti dal database; quest'ultimo avrà il compito di modificare o selezionare questi dati eseguendo la *query SQL* specificata nel programma *java*.

Il *JDBC driver* è un insieme di classi java che consentono a qualunque utente autenticato di interagire facilmente con il database. Esse forniscono all'utente un insieme di metodi standard e una sintassi *SQL* standard per la manipolazione del database. Inoltre, notiamo che *JDBC* è definito [25] in diversi tipi che elenchiamo di seguito.

- **Tipo 1**, usa la tecnologia **JDBC-ODBC**. *ODBC* è un insieme standard di **API**[20] indipendenti dal sistema operativo e dal linguaggio che lo usa. Esso necessita di uno specifico driver del database per operare e questo fatto incrementa la latenza del sistema perché ci sono più di un livello tra applicazione e database.

- **Tipo 2**, questo driver traduce le richieste formulate dall'applicazione in richiesta compiuta usando il database standard **API**. Questo driver è più efficiente del tipo 1 ma necessita della presenza sul computer della API del database, componente con licenza che può essere molto costosa.

[19] ricordiamo che MySQL è un'estensione del linguaggio standard per il database relazionale *SQL* o Structured Query Language

[20] Application Programming Interface

- **Tipo 3**, questo sistema utilizza le connessioni *sockets* e richiede la presenza di un **Database Server**, un programma che gestisce le connessioni in arrivo, elabora le richieste e restituisce il dato richiesto. Questo è il tipo di driver il più comune. La richiesta è inoltrata al database indipendentemente dal protocollo di rete. Esso presenta molti inconvenienti come, ad esempio, conoscere necessariamente l'indirizzo IP del server e la porta sulla quale rimane in ascolto.
- **Tipo 4**, è simile al tipo 3 ma il protocollo di rete utilizzato è specificato dal venditore per ottenere migliori prestazioni.

Per una conoscenza approfondita [20, 25] è un ottimo punto di riferimento. Ora esaminiamo il codice per interagire con il nostro database creato grazie alla libreria MySQL[20]. Primo, dobbiamo installare in memoria il tipo corretto del **JDBC driver**, la stringa necessaria e l'archivio **.jar** contenendo il driver scelto si trova al [20]. Con la seguente istruzione si installa il driver:

```
Class.forName ("com.mysql.jdbc.Driver").newInstance ();
```

Dopo l'installazione del **Driver**, dobbiamo stabilire una connessione con il database. Ricordiamo che è assolutamente necessario che il database *server* stia in esecuzione. Per aver semplicemente un accesso al database, si usa questa istruzione:

```
Connection conn = DriverManager.getConnection (url, user, pwd);
```

dove **url** è l'indirizzo del *database server* mentre **user** è l'*identificatore* dell'utente e **pwd** la sua *password*.

Per manipolare o estrarre dati dal database, dobbiamo istanziare un oggetto **Statement** con questa istruzione:

```
Statement st=conn.createStatement();
```

La classe **Statement** ha due importanti metodi:

```
ResultSet rs = executeQuery(String strQuery);} /*per estrarre i dati*/
```

restituisce un oggetto **ResultSet** [26] che contiene i dati richiesti tramite la *strQuery*. La stringa **strQuery** è una **query** come la sintassi seguente che mostra un tipico esempio:

```
"SELECT * FROM database\_name.table\_name WHERE condition"
```

mentre con quest'altra sintassi utilizzando la funzione *executeUpdate* della libreria JDBC:

```
int up = executeUpdate(String upQuery);
```

la stringa **upQuery** può essere una **query** che manipola i dati oppure genera la struttura del database, ad esempio con questa sintassi SQL:

```
"DELETE FROM database_name.table_name WHERE condition"
```

4.6.3 Fandango

Fandango è un'applicazione *Web* [2] per la gestione dei *testbed* interamente sviluppata dal **SIGNET Group** basata su **GWT**. La sua caratteristica principale è l'interazione con **Google Maps**. Attraverso Fandango si possono visualizzare i nodi sensori installati all'interno del **DEI** sui quali si possono effettuare delle simulazioni come ad esempio l'instradamento dei pacchetti (**routing**), il controllo delle informazioni sui nodi sensori, ecc. La figura 4.14 mostra l'interccia di visualizzazione del testbed signet su Fandango.

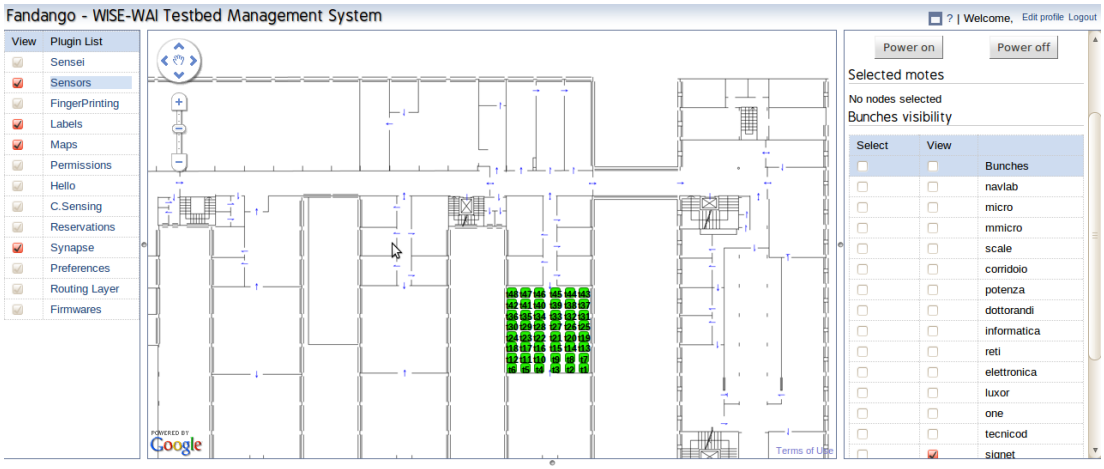

Figura 4.14: Visualizzazione del testbed signet su Fandango.

Per la realizzazione del nostro *Plug-in* di visualizzazione su **Fandango**, ci stiamo limitati solo ad implementare i requisiti necessario per sviluppare la nostra interfaccia grafica. Ricordiamo che su **Fandango**, ogni *Plug-in* consiste in due parti che sono il **layer** e il server.

Layer: E' la parte che crea il **GUI**[21] per l'utente (*Browser*) e visualizza i dati. E' quindi il lato *Client* che abbiamo precedentemente introdotto.

Server: E' il lato che risiede sul *server* e elabora tutte le richieste del *Client* ovvero proveniente dal lato **Layer**.

Layer e **Server** possono comunicare utilizzando lo standard **RPC** ovvero *Remote Procedure Call* implementata nel **GWT** ma su *Fandango* questa comunicazione è realizzata in un modo alternativo chiamato ***Notification Service***. Con **RPC** il *Client* può abilitare delle **actions** per il **server** mentre con ***Notification Service*** sta al *server* avviare le **actions** per il *Client-layer*.

La base di questo sistema è la **NotificationDispatcher**, una classe singola che prende i dati dal *server* e li manda al *Client-layer*. Nel lato *server*, esiste una classe che estende la classe **NotificationPlugin**. Sarà compito di questa classe inviare i dati al *Client-layer* tramite la *NotificationDispatcher*.

[21] *Graphical user interface.*

La prima operazione da eseguire è la registrazione. La classe *NotificationDispatcher* contiene un array per registrare tutti *Plug-in* e una volta completata la registrazione aggiungendo il nostro oggetto di interesse dentro questa *array*, esso potrà quindi inviare i dati al *NotificationDispatcher* seguendo ad esempio queste istruzioni:

```
NotificationDispatcher.getNotifier().notifyData(

CsensingLayerServiceImpl.this,

"csensingNotification",

status

);
```

dove ***CsensingLayerServiceImpl*** è l'oggetto che invia i dati ovvero è esattamente il nostro oggetto registrato precedentemente mentre ***"csensingNotification"*** è la stringa che indica il nome della **Notification** ovvero la *Notifica* e infine **status** è l'oggetto contenendo i dati da inviare al *layer*.

Il *layer* è un'estensione della classe **AbstractLayer** la quale consente al gestore dei *layer* (**layer manager**) di offrire gli strumenti per gestire ogni singolo *layer* che lo estende. Il *layer manager* è dunque l'oggetto che controlla tutti i *layers* e viene passato come argomento al loro costruttore. Nel nostro caso specifico, il *layer* si iscrive al *Notification Service* usando questa istruzione:

```
lm.subscribe(this, "csensingNotification");
```

dove **lm** è il *layer manager* e il metodo **subscribe** consente al *layer* di sottoscriversi ad un particolare *Notification Service*. **this** (indica questa classe) deve appartenere alla classe che estende l'interfaccia **HasNotificationService**.

Quando il *server* esegue il metodo **notifyData()**, il metodo **onNotificationReceived()** implementato nel oggetto *layer* viene invocato tramite la seguente istruzione:

```
public void onNotificationReceived(String name,

ArrayList$<$Datatype$>$ data) {

if(name.equals($"$CsensingNotification$"$))
{

/*elaborazione dei dati*/}}
```

```
}

}
```

La presenza di questo metodo è garantito dall'interfaccia **HasNotification-Service** il quale consente al *layer* di sovrascriverlo. Il parametro **name** è necessario quando il *layer* si registra a più *Notification Services* in quanto tutti i servizi invocano lo stesso metodo *onNotificationReceived()* e ha bisogno di sapere quale *server* ha attivato l'evento o meglio la notifica. Per ulteriore approfondimento sull'API di Fandango, fare riferimento a [2].

GWT - Google Web Toolkit

GWT [27] è un potente strumento per la creazione di complesse applicazioni *web-oriented*. L'idea di base all'interno di GWT è semplice: durante lo sviluppo di applicazioni *web-based* si deve fare i conti con molti linguaggi. Di solito si deve creare una parte *server-side* (codice che viene eseguito sul server) utilizzando tecnologie come ASP (*Active Server Page*), PHP (*Hypertext Preprocessor*), JSP (*JavaServer Pages*)e una parte *client-side* (codice che viene eseguito sul browser), utilizzando AJAX (*Asynchronous Javascript And Xml*). Allora, perché non si può usare lo stesso linguaggio per sviluppare tutte queste cose? GWT permette al programmatore di farlo usando JAVA, poi la parte client-side si traduce in Javascript e HTML automaticamente. La parte *server-side* rimane in Java poiché JSP è una tecnologia supportata dalla maggior parte dei web server. L'unico inconveniente è che quando si sviluppa la parte client non si può utilizzare le classi standard di Java, si può solo utilizzare le classi GWT ma il loro comportamento è simile alle classi standard di Java.

Java Servlet

Una *Servlet*[28] può essere utilizzata in altro ambiente, non solo nel Web con HTTP, in questa breve introduzione ci si riferisce solo al caso Web. Le interazioni tra una Servlet e un client (browser) è illustrata nella figura 4.15. Questa tecnologia può essere usata nella parte *server-side* di un'applicazione GWT.

L'idea principale dietro le servlet è semplice e simile alla filosofia di PHP o ASP: si ha un programma che gira su un server, che risponde alle richieste dei clienti (Request) interagendo con varie risorse (database, file, ecc), attraverso API standard di Java e genera dinamicamente una pagina HTML che viene inviato al client utilizzando il codice *Response*. La forza di questo approccio è che uno sviluppatore con conoscenze di HTML e Java può facilmente sviluppare pagine web dinamiche, in secondo luogo nel programma si è in grado di utilizzare la API standard di Java che ci permette di accedere alle risorse complesso, per esempio database, con semplici comandi. Si può usare una Servlet unica per la generazione di vari tipi di pagine web (si può costruire una pagina web diversa a seconda di quale parametri vengono inviati nella *Response*) e le *Servlet* possono convivere

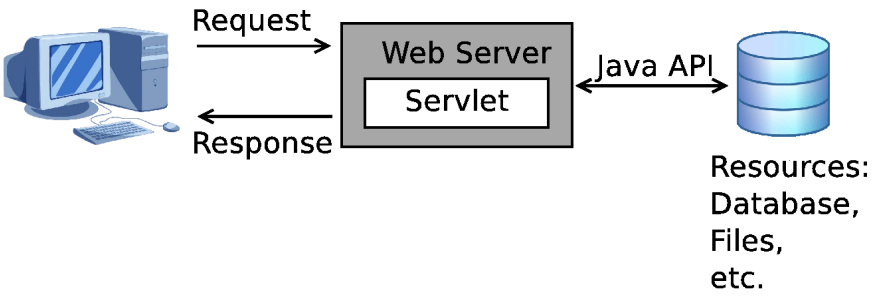

Figura 4.15: Architettura Java Servlet.

con le pagine web normali su un server, ma per semplicità si può pensare che in un web server c'è una Servlet per ogni pagina web che l'utente può visualizzare. Ogni Servlet è una classe Java che eredita il suo metodo dalla classe *HttpServlet*, i metodi importanti per la nostra discussione sono:

```
protected void doGet (
HttpServletRequest request,

HttpServletResponse response
);
```

e

```
protected void doPost (
HttpServletRequest request,

HttpServletResponse response
);
```

Il primo viene richiamato quando il client utilizza il metodo GET, il secondo quando il POST metodo è usato, sono molto simili quindi dovremo analizzare sia allo stesso tempo. I metodi hanno due parametri:

```
HttpServletRequest request
```

che contiene i dati che il cliente ha inviato al server attraverso *Request*, e

```
HttpServletRequest response
```

che contiene i dati che il server vuole inviare al client (di solito un Web pagina). In sostanza il comportamento di una servlet è semplice:

1 Quando una Request viene effettuata *doPost() o doGet()* vengono invocati;

2 Il metodo elabora i dati presenti nella request;

3 L'oggetto Request è riempito con i dati che devono essere inviati al client.

Per accedere ai dati la *request* usa il metodo

```
String getParameter(String name);
```

restituisce il valore associato al parametro *name*, tutti i valori restituiti sono stringhe in modo che per alcuni di questi dati è necessario convertirli nel tipo corretto utilizzando le funzioni di conversione fornite da Java. Ora si ha i dati e si può elaborarli: c'è bisogno di un modo per rispondere al cliente che può essere quello di scrivere la risposta nel oggetto di risposta. In primo luogo c'è bisogno di impostare il tipo di risposta utilizzando il metodo *setContentType()*, se per esempio, vogliamo rispondere utilizzando il formato *text/html* che scriveremo:

```
response.setContentType("text/html");
```

Una volta ottenuta un oggetto *PrintWriter* per scrivere in risposta, questo oggetto fornisce i metodi standard *print() e println()* metodi che vengono utilizzati per lo standard I/O in Java. Dopo l'uso, l'oggetto *PrintWriter* deve essere chiuso con la sintassi seguente:

```
out= resp.getWriter();
out.println(content);
out.close();
```

Quando doGet () o doPost () è terminato il contenuto della risposta verrà inviata alclient. Per informazioni più dettagliate fare riferimento a [28].

5 Applicazione del CS per la ricostruzione dei dati

Una delle problematiche nelle reti *wireless* di sensori è quello della gestione e della compressione dei dati per incrementare l'efficienza della loro raccolta presso un server attraverso il nodo *sink* collegato ad esso grazie al *gateway*.

Uno dei metodi risolutivi determina una soluzione ottimizzata implementando un meccanismo di compressione dei dati successivamente raccolti dal nodo *sink*. Questo metodo denominato *Compressing Sensing*[3, 29, 30] viene definito come una tecnica di compressione innovativa dei dati quale sfrutta la correlazione dei dati dello stesso vettore $\mathbf{x}$ comprimendoli mediante delle matrici rispettando alcune proprietà specifiche. Qual'allora tali matrici dette anche matrice di compressione e i dati di $\mathbf{x}$ negli istanti iniziali rispettano alcune proprietà, si può ricostruire tutti di dati di $\mathbf{x}$ dalla sua versione compressa $\mathbf{y}$ all'istante corrente e tutto questo grazie ad un algoritmo di ottimizzazione convessa (come ad esempio l'algoritmo di *Nesterov*[1] .

5.1 Tecnica del Compressive Sensing

5.1.1 Definizione

Si suppone di avere n nodi sensori casualmente distribuiti in un'area di copertura della rete *wireless* di nostro interesse che misurino dati di un qualche tipo e li rappresentino in un determinato istante un vettore $x \in \mathbb{R}^n$ con n la lunghezza delle vettore $\mathbf{x}$.

Nel server collegato al nodo *sink*, si vuole ricostruire completamente i dati del vettore $\mathbf{x}$ il più fedelmente possibile ad un istante k ricevendo soltanto una frazione dei dati misurati dai nodi sensori dovuta eventualmente alle problematiche nel canale di trasmissione *wireless* provocando perdite di pacchetti. Tale frazione viene rappresentata con il vettore $y \in \mathbb{R}^m$ con $m < n$.

Inoltre, dal sistema di interfacciamento la rete di sensori riceve esattamente m richieste e quindi vengono interrogati soltanto m nodi sensori che invieranno

[1] è un algoritmo di programmazione convessa.

successivamente le loro misure al nodo *sink*. Tutto questo avviene grazie alla probabilità di trasmissione

$$pTX = m/n \tag{5.1}$$

determinato dal modulo *Rebuilder* e utilizzato inseguito dal nostro sistema in esecuzione sullo stesso server per determinare esattamente quanti sensori (e in modo casuale) interrogare nella *WSN* consentendo un'ottimizzazione o meglio un risparmio importante della risorsa di energia disponibile per ogni nodo sensore.

5.1.2 Struttura e ottimizzazione

Si può quindi illustrare dalla relazione seguente come i due vettori $\mathbf{x}$ e $\mathbf{y}$, precedentemente definiti, siano legati:

$$\mathbf{y} = \mathbf{\Phi x} \tag{5.2}$$

dove $\Phi \in \mathbb{R}^{m \times n}$ rappresenta la matrice di routing ovvero la matrice che descrive in quale modo ogni elemento di $\mathbf{y}$ è legato ad un elemento di $\mathbf{x}$.

Nello sviluppo di questa tecnica, viene considerata una particolare matrice $\mathbf{\Phi}$ cioè che descrive il campionamento del vettore $\mathbf{x}$ ovvero una matrice che ha un solo elemento unitario per ogni riga e al più un valore 1 per ogni colonna.

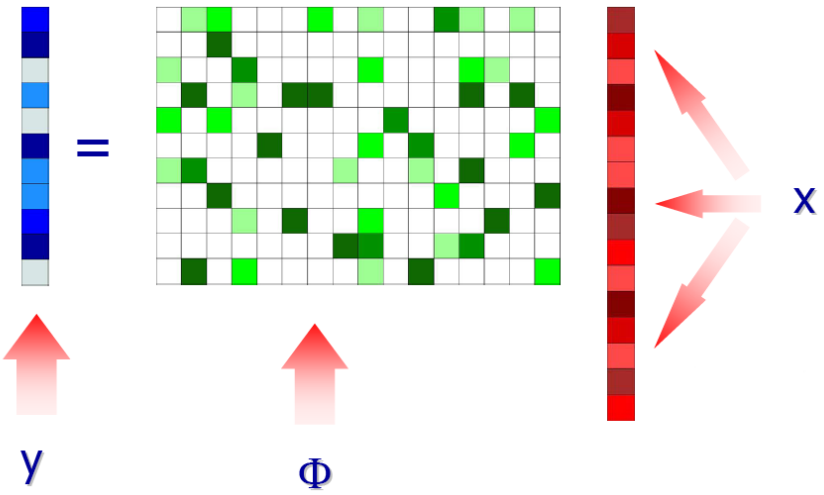

Figura 5.1: Rappresentazione concettuale della relazione tra $\mathbf{y}$ e $\mathbf{x}$

Si supponga ora che esiste una trasformazione invertibile attraverso la quale il vettore $\mathbf{x}$ possa essere reso *sparso*.

***Vettore K-sparso* 5.1.2.1.** *Sia $u \in \mathbb{R}^n$ un vettore di* n *elementi. Si dice che* $\mathbf{u}$ *è un vettore* K-sparso *se e soltanto se K dei suoi* n *elementi sono diversi da zero con $K, n \in \mathbb{N}$.* □

In dettaglio deve esistere una matrice $\mathbf{\Psi}$ invertibile e ortonormale, di dimensione $n \times n$ ovvero si ha $\mathbf{\Psi} \times \mathbf{\Psi}^* = I$ dove I è la matrice Identità di dimensione $n \times n$ tale per cui:

$$\mathbf{x} = \mathbf{\Psi s}, \tag{5.3}$$

dove $\mathbf{s}$ risulta *K-sparso*.

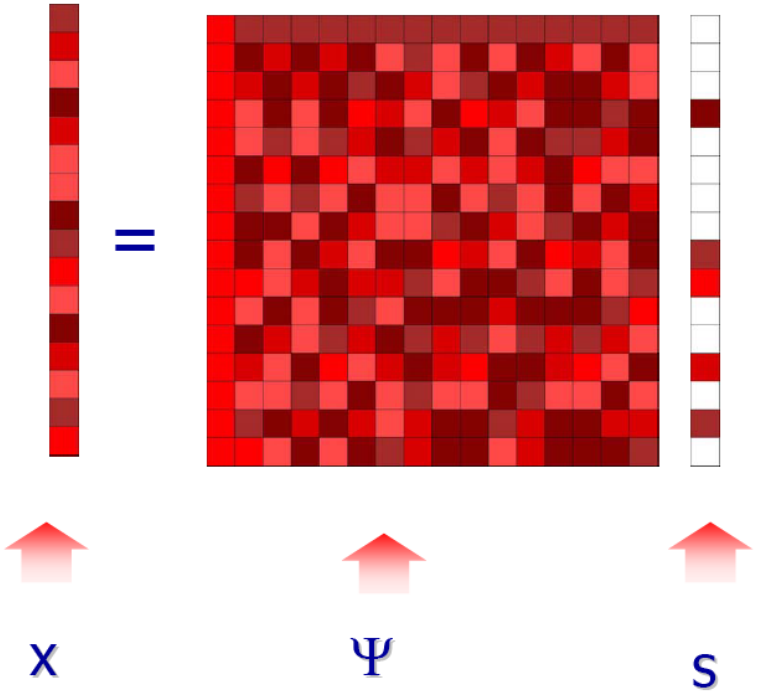

Figura 5.2: Rappresentazione concettuale dell'equazione (5.3), dove le celle di colore bianco elementi nulli o trascurabili.

In generale, non è sempre garantito che tale trasformazione esista. Nella prossima sezione un metodo per determinare una particolare matrice $\mathbf{\Psi}$ a partire dall'analisi della statistica del segnale (elementi del vettore), denominato *Principal Component Analysis (PCA)*.

Ora, associando le equazioni (5.2) e (5.3) si ottiene l'equazione seguente:

$$\mathbf{y} = \mathbf{\Phi x} = \mathbf{\Phi \Psi s} = \mathbf{As}. \tag{5.4}$$

dove il vettore $\mathbf{y}$ e la matrice $\mathbf{A}$ sono noti nel punto di raccolta dei dati sulla *WSN* ovvero definiti al nodo *sink* collegato sul *server* centrale di raccolta e ricostruzione, poi si cerca di determinare il vettore $\mathbf{s}$. Una volta trovato il vettore $\mathbf{s}$, si può ricavare il vettore $\mathbf{x}$ dalla (5.3).

Il sistema definito dalla (5.4) rappresenta un sistema di n equazioni in m incognite quale ammette in generale uno spazio di soluzioni di dimensione n - m. Si nota che ogni elemento in questo spazio può essere utilizzato per ricostruire il vettore $\mathbf{x}$ ma in questo metodo si è interessato alla soluzione $\bar{\mathbf{s}}$ che minimizza l'errore di ricostruzione definito come segue.

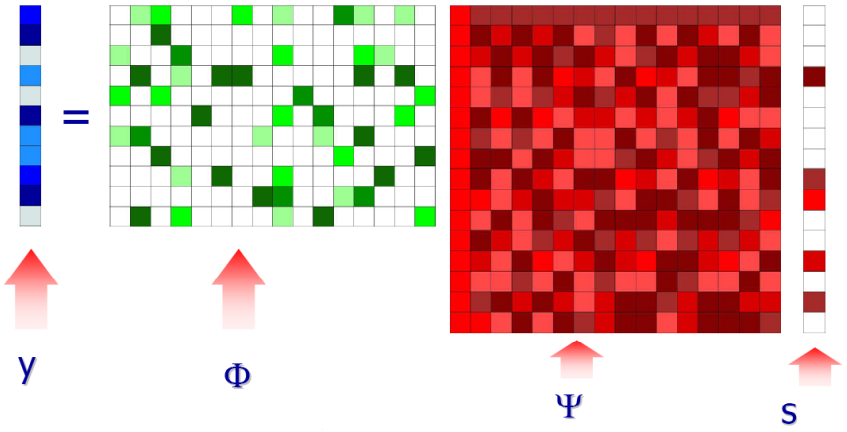

Figura 5.3: Rappresentazione concettuale dell'equazione (5.4).

***Errore di ricostruzione* 5.1.2.1.** *Sia* $\mathbf{x} \in \mathbb{R}^n$ *il vettore contenente le misure di* n *sensori e sia* $\overline{\mathbf{x}}$ *un vettore ottenuto dalla ricostruzione basata sulla tecnica di* CS*, si definisce* **errore di ricostruzione** *indicato con* ξ_R *, la quantità seguente:*

$$\xi_R = \frac{\| \mathbf{x} - \overline{\mathbf{x}} \|_{\ell_2}}{\| \mathbf{x} \|_{\ell_2}} \tag{5.5}$$

dove la funzione $\| \cdot \|_{\ell_2}$ indica la norma ℓ_2 definita di seguito. □

***Norma* ℓ_2 5.1.2.1.** *Sia* $\mathbf{x} \in \mathbb{R}^n$ *un vettore, e* x_i *le sue componenti con* $i \in \{1, 2, \ldots, n\}$*. Si definisce* **norma** ℓ_2 *di* $\mathbf{x}$ *indicata con* $\| \mathbf{x} \|_{\ell_2}$*, tramite la seguente relazione:*

$$\| \mathbf{x} \|_{\ell_2} = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}} \tag{5.6}$$

□

Si dimostra [31, 32] che, dato il vettore *K-sparso* $\mathbf{s}$ della (5.3), se per ogni altro vettore *K-sparso* $\mathbf{t}$ con un numero identico di elementi diversi da zero e per qualche $\delta > 0$, vale:

$$1 - \delta \ \leq \ \frac{\|\mathbf{A}\mathbf{t}\|_{\ell_2}}{\|\mathbf{t}\|_{\ell_2}} \ \leq \ 1 + \delta, \tag{5.7}$$

ossia la matrice $\mathbf{A}$ preserva la lunghezza dei vettori *K-sparsi* allora la soluzione esiste, è unica ed è il vettore $\mathbf{s}$ che soddisfa il sistema (5.4) con il minor numero di elementi non nulli, o equivalentemente il più *sparso*.

Una condizione sufficiente affinché la soluzione sia unica, è che la matrice $\mathbf{A}$ soddisfi la (5.7) per un generico vettore *3K-sparso*. Questa condizione viene denominata *Restricted Isometry Properties (RIP)*. Definiamo ora un'altra condizione equivalente a quest'ultima chiamata *incoerenza delle matrici*. Si definisce dunque la *coerenza* tra due matrici come richiamato qui:

***Coerenza tra matrici* 5.1.2.1.** *Siano* $\mathbf{\Phi}$ **e** $\mathbf{\Psi}$ *due matrici. Siano* $\{\Phi_k\}$ *le righe di* $\mathbf{\Phi}$ *e* $\{\Psi_j\}$ *colonne di* $\mathbf{\Psi}$*, entrambe di lunghezza n. Si definisce la* coerenza *tra* $\mathbf{\Phi}$ **e** $\mathbf{\Psi}$*, il valore* μ *determinato dall'uguaglianza seguente:*

$$\mu\left(\mathbf{\Phi}\ ,\mathbf{\Psi}\right) = \sqrt{n} \cdot \max_{1\leq k,j\leq n} |(\{\Phi_k\},\{\Psi_j\})| \tag{5.8}$$

□

In pratica questo valore μ determina la massima correlazione tra gli elementi delle due matrici $\mathbf{\Phi}$ **e** $\mathbf{\Psi}$ e quindi ricavare una *coerenza* alta significa ottenere elementi molto correlati tra di loro.

Segue da alcuni sviluppi algebrici che $\mu\left(\mathbf{\Phi}\ ,\mathbf{\Psi}\right) \in [1,\sqrt{n}]$ [29]. La condizione di incoerenza richiede dunque che la massima correlazione tra gli elementi di $\mathbf{\Phi}$ **e** $\mathbf{\Psi}$ sia piccola, o equivalentemente che non sia possibile rappresentare le righe $\{\Phi_k\}$ di $\mathbf{\Phi}$ tramite una combinazione lineare sparsa delle colonne $\{\Psi_j\}$ di $\mathbf{\Psi}$ e *viceversa*. Usando una matrice $\mathbf{\Phi}$ ottenuta da un campionamento casuale, sono soddisfatte, con alta probabilità, sia la *RIP* che l'*incoerenza* con $\mathbf{\Psi}$.

L'algoritmo di ricostruzione del segnale deve dunque ricavare gli n campioni del segnale $\mathbf{x}$, a partire dagli m campioni di $\mathbf{y}$ e dalla matrice $\mathbf{A} = \mathbf{\Phi\Psi}$. Come già accennato, le soluzioni della (5.3) sono tutte contenute nello spazio $\mathcal{H} = \mathcal{N}(\mathbf{A})+\mathbf{s}$ di dimensione (n - m), dove $\mathcal{N}(\mathbf{A})$ è il nucleo della matrice $\mathbf{A}$.

L'approccio più semplice per determinare una soluzione di un problema di questo tipo è di ricavare la soluzione con *norma* ℓ_2 minore nel nucleo di $\mathbf{A}$ traslato in $\mathbf{s}$ risolvendo il problema di minimizzazione seguente:

$$\overline{\mathbf{s}} = \arg\min \|\mathbf{s}\|_{\ell_2} \qquad , \qquad \textit{con } \mathbf{y} = \mathbf{As}. \tag{5.9}$$

Questa forma di ottimizzazione ha una forma chuisa semplice ma purtroppo restituisce la soluzione quasi mai *sparsa*. Per questa nuova problematica, si va a definire altre forme di *norma* quali ℓ_0 *e* ℓ_1:

***Norma* ℓ_0 5.1.2.1.** *Sia* $\mathbf{x} \in \mathbb{R}^n$ *un vettore, e* x_i *le sue componenti con* $i \in \{1,2,\ldots,n\}$*. Si definisce* **norma** ℓ_0 *di* $\mathbf{x}$ *indicata con* $\| \mathbf{x} \|_{\ell_0}$*, tramite la seguente relazione:*

$$\| \mathbf{x} \|_{\ell_0} = \sum_{i=1}^{n} 1\{x_i \neq 0\}, \tag{5.10}$$

dove la funzione $1\{\cdot\}$ è la funzione *indicatrice*. □

***Norma* ℓ_1 5.1.2.1.** *Sia* $\mathbf{x} \in \mathbb{R}^n$ *un vettore, e* x_i *le sue componenti con* $i \in \{1,2,\ldots,n\}$*. Si definisce* **norma** ℓ_1 *di* $\mathbf{x}$ *indicata con* $\| \mathbf{x} \|_{\ell_1}$*, tramite la seguente relazione:*

$$\| \mathbf{x} \|_{\ell_1} = \sum_{i=1}^{n} |x_i| \tag{5.11}$$

□

Risolvendo il problema di minimizzazione (5.9) sostituendo alla *norma* ℓ_2 con la *norma* ℓ_0, che conta il numero di elementi non nulli, si ottiene la soluzione più *sparsa* ovvero quella con il maggior numero di zeri:

$$\bar{\mathbf{s}} = \arg\min \|\mathbf{s}\|_{\ell_0} \qquad , \qquad con\ \mathbf{y} = \mathbf{As}. \tag{5.12}$$

Sfortunatamente il problema (5.12) è un problema *NP-completo* e numericamente instabile. Per problemi di grande dimensione è quindi computazionalmente impossibile da risolvere. Una soluzione ideale è quella di risolvere la (5.9) utilizzando un algoritmo di minimizzazione di *norma* ℓ_1:

$$\bar{\mathbf{s}} = \arg\min \|\mathbf{s}\|_{\ell_1} \qquad , \qquad con\ \mathbf{y} = \mathbf{As}. \tag{5.13}$$

Se la matrice $\mathbf{A}$ soddisfa la *RIP*, la soluzione del problema (5.12) è la stessa del problema (5.13) a differenza che si ottiene un costo computazionale trattabile, anche per problemi di grandi dimensioni. Perciò, al nodo *sink* è dunque necessaria l'implementazione di un metodo che ricerca la soluzione con norma ℓ_1 minore tra le possibili soluzioni di (5.4).

L'ipotesi sull'esistenza della matrice $\mathbf{\Psi}$ è un'ipotesi molto forte e non sempre verificata quando si sta lavorando con segnali reali. Per causa di questa problematica, viene utilizzato un metodo statistico, la *PCA*, che sfrutta la correlazione spaziale e temporale delle componenti del vettore $\mathbf{x}$.

5.1.3 *Principal Component Analysis*

La *Principal Component Analysis (PCA)* è una procedura matematica che consente la rappresentazione di un insieme di variabili, eventualmente correlate, con un minor numero di variabili non correlate chiamate componenti principali. Per fare ci viene richiesta la conoscenza della statistica del primo ordine delle variabili di partenza. A seconda del campo di applicazione, la *PCA* è anche chiamata Karhunen-Loève Transform, Hotelling Transform o Proper Orthogonal Decomposition. In questo contesto useremo la *PCA* per costruire una matrice di trasformazione che renda sparso un segnale di dimensione n in un qualche dominio.

Il risultato teorico che sta alla base della *PCA* è il teorema di Ky Fan, definito di seguito:

***Teorema di Ky Fan* 5.1.3.1.** *Sia $\mathbf{\Omega} \in \mathbb{R}^n$ una matrice simmetrica, siano $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ i suoi autovalori e $\mathbf{a_1}, \ldots, \mathbf{a_n} \in \mathbb{R}^n$ gli autovettori associati (che senza perdita di generalità vengono assunti ortonormali). Dati $\mathbf{m}$ autovettori ortonormali $\mathbf{b_1}, \ldots, \mathbf{b_m} \in \mathbb{R}^n$, con $\mathbf{m} \leq \mathbf{n}$, vale l'uguaglianza seguente:*

$$\max_{b_1,\ldots,b_m} \sum_{j=1}^{m} (\mathbf{b_j^T \Omega b_j}) = \sum_{i=1}^{m} \lambda_i, \tag{5.14}$$

e il *massimo* si ottiene quando si ha $\mathbf{b_i} = \mathbf{a_i},\ \forall i$. □

Si definisca il vettore $\mathbf{x}^{(k)} \in \mathbb{R}^n$ come il vettore delle letture dei n sensori all'istante k. Si supponga di collezionare i dati relativi a tutti i sensori per K istanti di tempo successivi e di calcolare il vettore delle medie $\hat{\mathbf{x}}$ e la matrice delle covarianze ∇ su questo insieme di dati.

$$\hat{\mathbf{x}} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{x}^{(k)} \ , \ \nabla = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{x}^{(k)} - \hat{\mathbf{x}})(\mathbf{x}^{(k)} - \hat{\mathbf{x}})^T. \tag{5.15}$$

Da un punto di vista geometrico si può considerare $\mathbf{x}^{(k)}$ come un punto in uno spazio a $\mathbf{n}$ dimensioni e cercare il piano M-dimensionale più vicino a $\mathbf{x}^{(k)}$. Come afferma il teorema di Ky Fan, la trasformazione lineare che conserva maggiormente l'informazione contenuta nel segnale $\mathbf{x}^{(k)}$ può essere costruita dagli autovettori della matrice di covarianza. Se si definisce $\mathbf{U_M}$ come la matrice formata dagli M autovettori corrispondenti agli M autovalori maggiori di ∇, la proiezione M dimensionale di $\mathbf{x}^{(k)}$ può essere scritta come:

$$\mathbf{x}^{(k)^\wedge} = \hat{\mathbf{x}} + \mathbf{U_M}\mathbf{U_M}^T(\mathbf{x}^{(k)} - \hat{\mathbf{x}}). \tag{5.16}$$

Se si definisce $\mathbf{s} = \mathbf{s_n} = \mathbf{U_n}^T(\mathbf{x}^{(k)} - \hat{\mathbf{x}})$, si può notare che per come è costruita la matrice $\mathbf{U_N}^T$, gli elementi del vettore $\mathbf{s}$ risultano ordinati in modo non crescente. Se per $i > M$ si ha che $s(i) \ll s(j)$, con $j = 1, ..., M$, allora $\mathbf{x}^{(k)}$ è ben approssimato da (5.16) attraverso l'utilizzo di $M < \mathbf{n}$ coefficienti, cioè quasi tutta l'energia è concentrata nelle prime componenti. Il che equivale a dire che il punto $\mathbf{x}^{(k)} \in \mathbb{R}^n$ viene trasformato nel punto $\mathbf{s_M} \in \mathbb{R}^M$ in questo modo:

$$\mathbf{s_M} = \mathbf{U_M}^T(\mathbf{x}^{(k)} - \hat{\mathbf{x}}). \tag{5.17}$$

Il risultato è che $\mathbf{s} = \begin{bmatrix} \mathbf{s}_M \\ \mathbf{0}_{M-N} \end{bmatrix}$ è un vettore M-sparso. Inoltre la matrice $\mathbf{U_M}$ è ortonormale, e quindi $\mathbf{U_M}\mathbf{U_M}^T = \mathbf{I_N}$ che è la matrice identità. Si è quindi in possesso di una trasformazione che rende sparso il segnale d'ingresso $\mathbf{x}^{(k)}$ in un altro dominio, inoltre questa trasformazione è invertibile e ortonormale. Sulla base dei risultati visti in precedenza è immediato verificare come si possa utilizzare la matrice $\mathbf{U_N}$ come matrice di trasformazione. Si ha infatti:

$$\mathbf{x}^{(k)} - \hat{\mathbf{x}} = \mathbf{U_N}\mathbf{s}^{(k)} = \mathbf{\Psi}\mathbf{s}^{(k)} \tag{5.18}$$

□

5.2 Funzionamento

Ora possiamo definire un metodo, chiamato *CS+PCA* [33], che sfrutta la matrice di trasformazione calcolata attraverso la *Principal Component Analysis* e i risultati teorici della tecnica del *Compressive Sensing* per poter ricostruire il segnale $x^{(k)} \in \mathbb{R}^n$ (vettore contenente le misure dei n sensori all'istante $\mathbf{k}$) a partire dalla sua versione campionata $y^{(k)} \in \mathbb{R}^n$.

Usando il metodo *CS+PCA* si ottengono risultati migliori (in termini di errore di ricostruzione) rispetto a quanto si ottiene con *CS* affiancato da altre trasformate note come visto nella sezione precedente [33]. A questo punto, si possono distinguere due procedure con i quali applicare il metodo *CS+PCA*:

- Il *primo* metodo prevede l'alternarsi di due regimi di funzionamento della rete dei sensori. Durante il primo regime tutti i sensori trasmettono con probabilità $pTX = 1$ e il nodo *sink* colleziona tutti i dati e li utilizza per costruire le statistiche necessarie. Successivamente la probabilità di trasmissione viene ridotta a ($pTX<1$) e il nodo *sink* applica la tecnica appena vista per ricostruire il segnale completo. In questo modo le statistiche vengono rinnovate di tanto in tanto e usate per gli istanti successivi.

- Il *secondo* metodo prevede una fase iniziale in cui si studiano le statistiche del segnale durante K *round* in cui vengono collezionati i dati di tutti i sensori ($pTX = 1$). Dopo l'inizializzazione, le statistiche vengono aggiornate utilizzando di volta in volta il segnale ricostruito. Ad ogni *round* di questa fase la probabilità con cui i sensori trasmettono è $pTX \leq 1$ e al nodo *sink* vengono ricalcolate le statistiche sugli ultimi K campioni di $x^{(k)}$. Usando questo metodo si può inoltre introdurre un controllo della qualità del segnale ricostruito che va ad agire sulla probabilità di trasmissione pTX dei sensori in modo dinamico.

Quest'ultimo metodo corrisponde dunque alla tecnica scelta e implementata nel modulo di ricostruzione della nostra applicazione e utilizzata durante il processo di simulazione.

6 Simulazioni e prestazioni

Per testare l'efficienza e le prestazioni del sistema sviluppato, sono state effettuate alcune raccolte di misure sul testbed del DEI o *the Dipartment Of Information Engineering* dell'Università degli Studi di Padova. La figura 6.1 mostra una mappa dei nodi sul testbed del DEI.

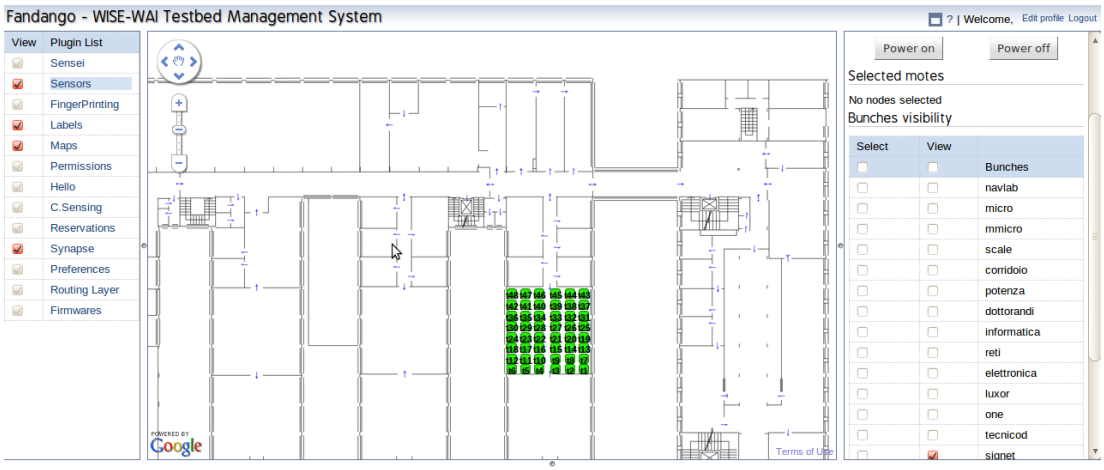

Figura 6.1: Una mappa dei nodi sensori del testbed del DEI.

Le simulazioni si sono limitate ad un sottoinsieme di nodi che rilevano, tramite i trasduttori di cui dispongono, le misure di temperatura, di umidità e di luce, sia nello spettro del visibile che in quello dell'infra-rosso. Queste misure presentano diversi gradi di correlazione spaziale e temporale; perciò vengono classificate secondo la loro sensibilità: luce, luce infra-rosso, umidità e temperatura. Consideriamo ad esempio la luce per cui i dati rilevati sono poco correlati tra di loro ovvero hanno la massima correlazione molto piccola, condizione sufficiente per definire l'incoerenza tra la matrice di trasformazione che rende molto *sparso* il vettore dei dati da ricostruire e il metodo con cui questi dati vengono raccolti. La tecnica del CS+PCA utilizza questo coefficiente di incoerenza per valutare l'errore di ricostruzione sui dati raccolti e calcola dunque la probabilità con cui determinare il numero dei dati da collezionare al prossimo *round* di raccolta.

Il metodo che il sistema implementa, per applicare la CS+PCA nella ricostruzione dei dati, prevede l'alternarsi di due regimi di funzionamento della rete dei sensori.

1- Durante il primo regime tutti i sensori trasmettono con probabilità $p_{TX} = 1$ e il sistema colleziona tutti i dati tramite il gateway e li utilizza per costruire le statistiche necessarie.

2- Successivamente ossia nel secondo regime la probabilità di trasmissione viene ridotta a $p_{TX} < 1$ e il sistema interroga soltanto un sottoinsieme dei nodi nel testbed applica la tecnica del Compressive Sensing con Principal Component Analysis per ricostruire il segnale completo.

Questi regimi di funzionamento del sistema di interfacciamento salva nel database tutte le informazioni pubblicate dai nodi interogati. La tabella testbed raccoglie i dati in funzione della probabilità calcolata dal CS nel round precedente. Ecco ad esempio i dati raccolti sul testbed ad un determinato tempo con $p_{TX} = 0.25$ (3 nodi su un totale di 15 nodi attivi nel testbed). Queste informazioni sono salvate nella tabella *dataMeasured* del database.

```
0.25,12,0000-0000-fffe-00c5,7,1,Light,100,2010-12-01 09:08:32

0.25,12,0000-0000-fffe-00e4,9,1,Light,98,2010-12-01 09:08:32

0.25,12,0000-0000-fffe-00c4,4,1,Light,112,2010-12-01 09:08:32
```

Si nota $p_{TX} = 0.25$, il numero dei nodi interogati per questo round è di conseguenza pari a 3 (su un insieme di 15 nodi) ed uguale al numero delle righe nella tabella confermando l'efficienza della trasmissione nel canale radio (nessuna perdita di pacchetti).

Precisiamo che i nodi sensori su cui è stata effettuata la campagna di raccolta dei dati si localizzano tutti nella stessa stanza (routing multihop non ancora implementato); giustificando così l'efficienza della trasmissione dei pacchetti al nodo *sink* (instradamento diretto ossia senza hop, punto critico nel routing nelle WSN) che è collegato al gateway.

Ora, analizziamo le informazioni presenti nella tabella *dataMeasured*; ad esempio, il nodo (nome: t32) con indirizzo IPv6 **0000-0000-fffe-00c5** è mappato nella tabella con l'identificatore intero 7 e la sua misura di tipo **Light** ossia luce corrisponde a valore 100 raccolto all'istante **2010-12-01 09:08:32**. Il plug-in implementato per la visualizzazione grafica su Fandango mostra tutte queste informazioni del nodo 7 tramite la figura 6.2.

Nel caso in cui un nodo non sia stato interogato nell'istante corrente, la tecnica del CS ricostruisce comunque la sua misura sfruttando le statistiche sulle precedenti misure della rete. La figura 6.3 mostra, grazie al plugin implementato, le

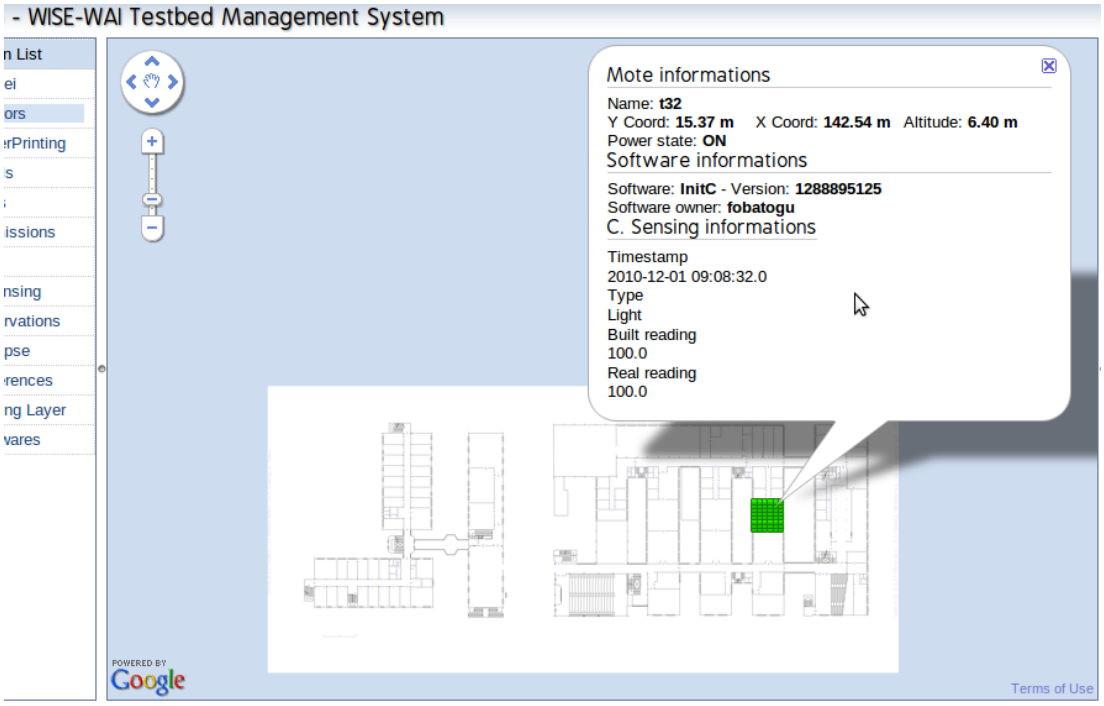

Figura 6.2: Visualizzazione grafica delle informazioni raccolte sul nodo 7 (nome: t32).

informazioni del nodo 1 (nome: t31) non interogato (con misura reale fittizia pari a -60000) nello stesso istante di quello del nodo 7 della figura 6.2; la sua misura ricostruita è pari a 96 sempre di tipo **Light**. Ecco, ad esempio, le informazioni ricostruite per i nodi attivi nel testbed:

```
0.25,1,1,96,2010-12-01 09:08:32

0.25,2,1,102,2010-12-01 09:08:32

0.25,3,1,75,2010-12-01 09:08:32

0.25,4,1,112,2010-12-01 09:08:32

0.25,5,1,86,2010-12-01 09:08:32

0.25,6,1,80,2010-12-01 09:08:32

0.25,7,1,100,2010-12-01 09:08:32

0.25,8,1,116,2010-12-01 09:08:32

0.25,9,1,98,2010-12-01 09:08:32

0.25,10,1,91,2010-12-01 09:08:32
```

```
0.25,11,1,89,2010-12-01 09:08:32

0.25,12,1,86,2010-12-01 09:08:32

0.25,13,1,82,2010-12-01 09:08:32

0.25,14,1,98,2010-12-01 09:08:32

0.25,15,1,80,2010-12-01 09:08:32
```

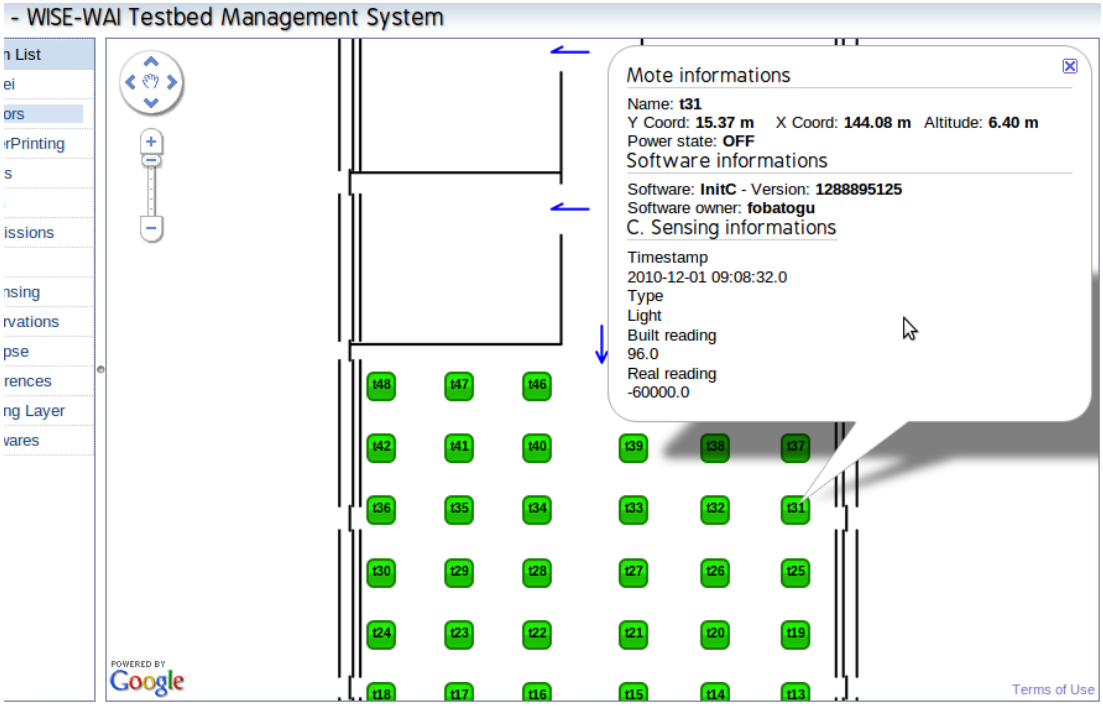

Figura 6.3: Visualizzazione grafica delle informazioni raccolte sul nodo 1 (t31).

Sottolineiamo che il modulo di ricostruzione opera con prestazioni migliori a secondo della sensibilità del segnale da ricostruire, fare riferimento alle simulazioni effettuate durante la sua implementazione [22]. Le figure 6.4, 6.5 e 6.6 mostrano come la probabilità p_{TX} calcolata e l'errore di ricostruzione presentano una notevole variazione a secondo della sensibilità del segnale (tipo del segnale) considerato. Queste variazioni affermano dunque l'importanza dell'ipotesi sulla correlazione temporale e spaziale dei segnali da ricostruire.

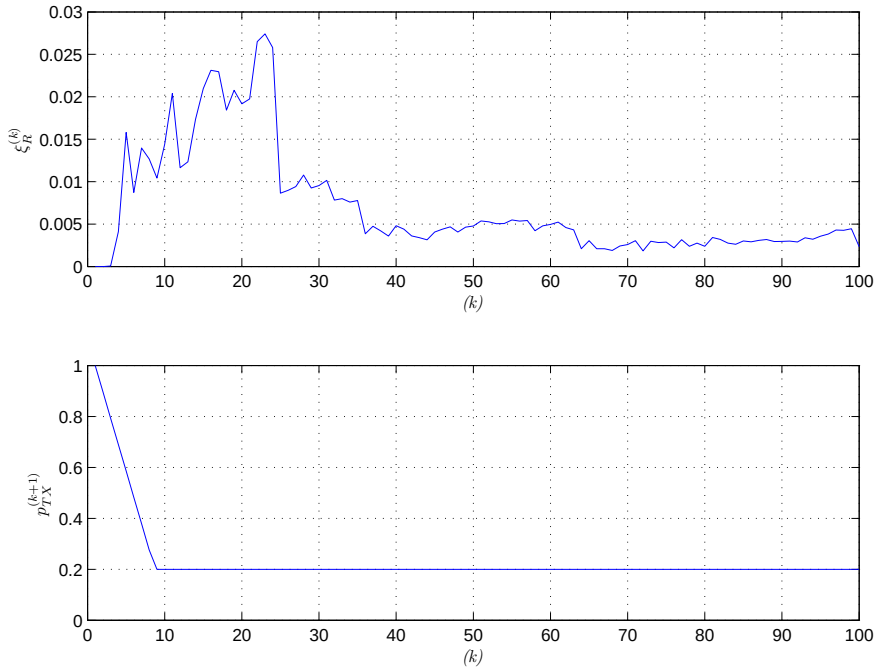

Figura 6.4: Segnale di temperatura.

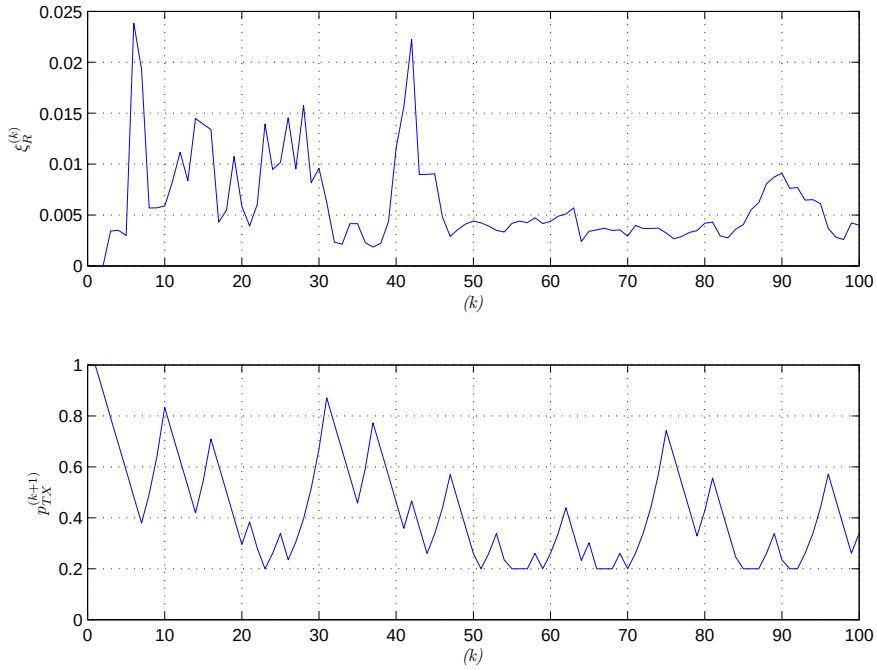

Figura 6.5: Segnale di humidità.

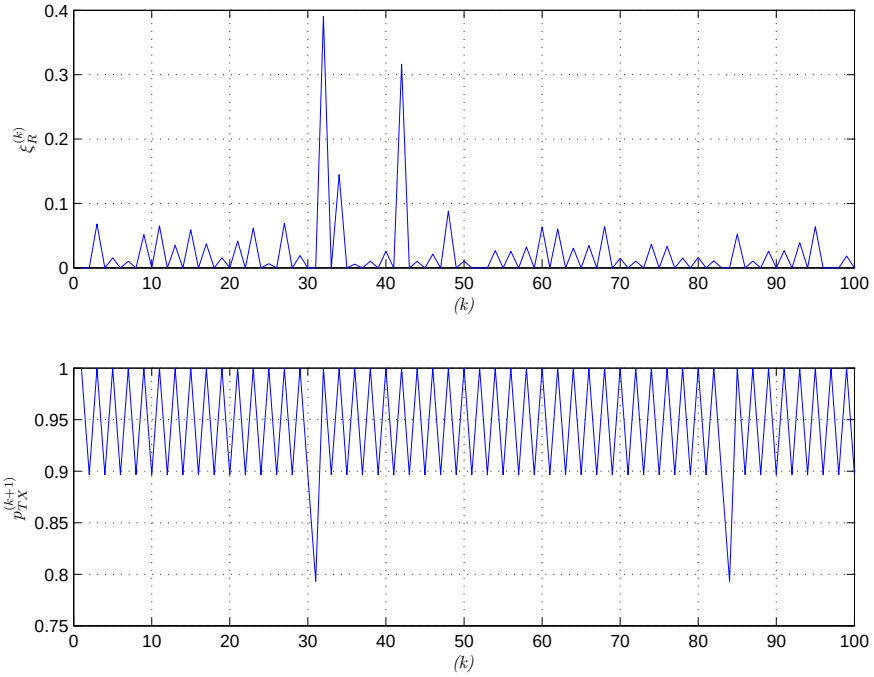

Figura 6.6: Segnale di luminosità.

Si osserva inoltre che la determinazione dell'indirizzo dei nodi da interogare durante questo test è sempre realizzata nel sistema di interfacciamento.

La figura 6.7 mostra un confronto dell'andamento dei numeri di nodi (in funzione di p_{TX}) interogati sul testbed ad ogni istante di richiesta dati. Applicando CS, si nota che durante la fase iniziale, il sistema interoga tutti i nodi (con probabilità massima ossia $p_{TX} = 1$) e decrescono fino a raggiungere il minimo numero di nodi da interogare in funzione del tempo; il numero dei nodi rimane stabile giustificando l'efficienza del canale di trasmissione ovvero non ci sono perdite di pacchetti durante la trasmissione.

In presenza di un algoritmo di *routing o multi-hop* (che sarebbe oggetto per i sviluppi futuri), si noterebbe certamente una perdita dei pacchetti durante l'intrastradamento verso il destinatario; Questo problema potrebbe nascere dalla collisione dei pacchetti nei hop durante l'instradamento oppure dalla presenza di interferenza in grado di corrompere i pacchetti nel canale radio.

Si può infine accennare un'ipotesi sul risparmio energetico osservando, con l'utilizzo di questo sistema di interfacciamento, che un calo del consumo energetico può risultare notevole se si dispone di un algoritmo di *power-off* per spegnere i nodi che non sono interogati ad ogni istante di richiesta dei dati nella rete.

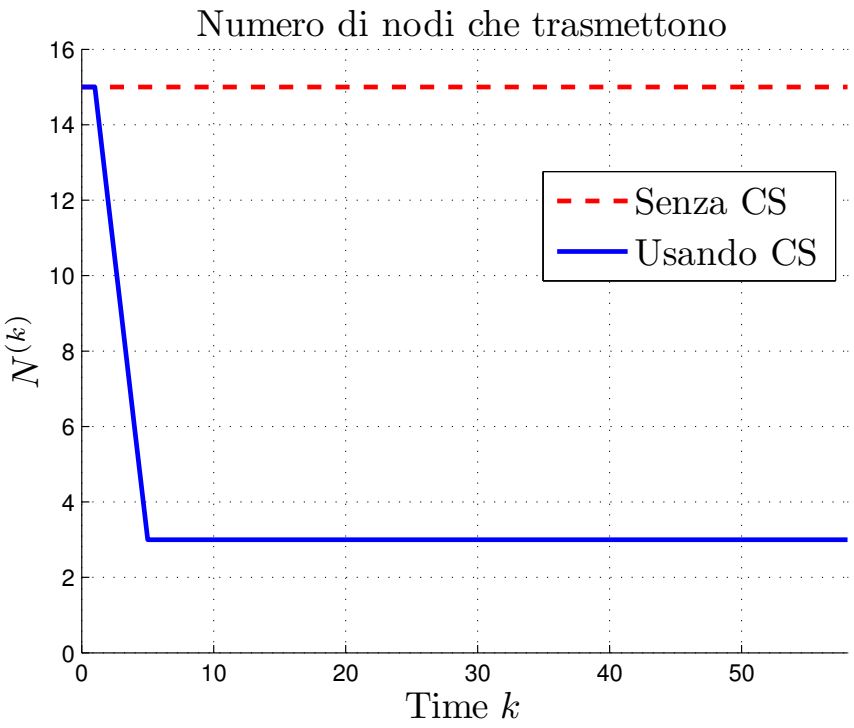

Figura 6.7: Numero di nodi per il sistema applicato senza CS.

7

Conclusioni e integrazioni future

La realizzazione di questo sistema di interfacciamento è stata implementata per raccogliere, ricostruire e visualizzare dati o misure di sensori provenienti da una rete di sensori. Esso, grazie alla componente di comunicazione con il gateway e alla tecnica del Compressive Sensing, permette di raccogliere dati, ricostruire quelli persi per errori di trasmissione e di visualizzare il tutto in un interfaccia grafica web attraverso un plugin di visualizzare integrato su Fandango.

Il sistema è stato sviluppato su un sistema operativo Linux (Ubuntu 10.04) utilizzando Eclipse come ambiente di sviluppo del plugin di visualizzazione che viene implementato grazie a JDBC, Servlet e GWT; utilizza varie funzioni delle librerie *open-source* per implementare sia il modulo di comunicazione con il gateway che raccoglie i dati sulla rete di sensori, sia per implementare il database che contiene tutte le tabelle necessarie per tenere traccia dei dati raccolti e ricostruiti.

I dati salvati nel database vengono utilizzati dal modulo *rebuilder*, che usa la tecnica del Compressive Sensing, per ricostruire i dati persi durante la raccolta nel gateway che interoga un sottoinsieme dei nodi sensore secondo una probabilità determinata durante la ricostruzione all'istante precedente. Una volta salvati nel database sia i dati raccolti che quelli ricostruiti, il plugin usa JDBC per accedere al database e manipulare le informazioni disponibili attraverso le query SQL. Questo plugin restituira le seguenti informazioni per ogni nodo sensore visualizzato sul testbed tramite Fandango:

1- il tempo di raccolta del dato;

2- il tipo della misura effettuata;

3- il valore della misura raccolta sul nodo sensore;

4- il valore della misura ricostruita dal CS;

Inoltre la probabilità restituita dal modulo *rebuilder* può risultare determinante per valutare ad esempio il consumo energetico nella rete wireless soprattutto quando i nodi sensore sono alimentati solamente con batterie che ha una durata

di vita limitata. Infatti, quando questa probabilità sarà minore di 1 il sistema interogherà soltanto un sottoinsieme di nodi come osservato nei test illustrati nel capitolo 6. In presenza di un opportuno meccanismo di *switch off* dei nodi, ossia quando tutti i nodi sono sincronizzati sullo stesso intervallo di tempo, è possibile spegnere la radio dei nodi che non vengono interogati dal gateway. Sotto questa condizione, si otterrebbe un risparmio energetico nella rete determinata dal fatto che tutti nodi con radio spenta (perché non trasmettono) risparmiano il consumo dell'energia fornita dalle batterie e di conseguenza il un tempo di vita medio viene prolungato di una buona parte.

Risulta che le prestazioni del modulo *rebuilder* si noterebbe maggiormente in presenza di un algoritmo di *routing* ovvero un algoritmo di multi-hop consentendo di raggiungere un raggio di disseminazione molto elevato ossia una dimensione della rete wireless più grande. Queste prestazioni sono determinate dall'errore di ricostruzione dei dati persi per causa di collisione (congestione) sugli *hop* che li instradano oppure per causa di interferenze nel canale di comunicazione wireless.

Si può auspicare che questi algoritmi sia di *routing (multi-hop)* sia *switch off* saranno progettati in futuro per rendere più effettivo al livello di prestazione il sistema realizzato dall'attività di questa tesi.

Bibliografia

[1] R. Kay and F. Matter, "The design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, 2004.

[2] M. Visona', "Fandango - user and developer manual."

[3] D. Donoho, "Compressed sensing," *IEEE Trans. on Information*, vol. 52, no. 4, pp. 4036–4048, 2006.

[4] I. C. Society, "Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans)," *IEEE 8 September 2006*, 2006.

[5] Chipcon, "Cc2420 datasheet." [Online]. Available: http://inst.eecs.berkeley.edu/~cs150/Documents/CC2420.pdf

[6] T. Haenselmann, *Sensornetworks*. GFDL Wireless Sensor Network textbook, 2006.

[7] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Comput. Networks*, vol. 38, 2002.

[8] "Ipv6 – internet protocol version 6 – rfc 2460." [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[9] J. W. Hui and D. E. Culler, "Extending ip to low-power, wireless personal area networks," *Internet Computing*, vol. 12, no. 4, pp. 37–45, 2008.

[10] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of ipv6 packets over ieee 802.15.4 networks," *IETF RFC 4944*, 2007.

[11] "Ipv6 – introduzione a ipv6." [Online]. Available: http://www.infis.univ.trieste.it/AppuntiLinux/a2124.html

[12] "Icmpv6 – internet control message protocol version 6 – rfc 4443." [Online]. Available: http://tools.ietf.org/search/rfc4443

[13] D. S. Z. Shelby, B. Frank, "Constrained aplication protocol (coap) draft-ietf-core-coap-01." [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-coap/

[14] B. Research group of the University of Califonia, "Tinyos official web site." [Online]. Available: http://tinyos.net/

[15] D. G. P. Levis, "Tinyos programming," *Cambridge University Press 2009.*

[16] "Gnu c compiler documentation." [Online]. Available: http://gcc.gnu.org/

[17] "Progetto europeo sensei." [Online]. Available: http://www.sensei-project.eu/

[18] "Library curl official web site." [Online]. Available: http://curl.haxx.se/libcurl/

[19] "Sockets library official web site." [Online]. Available: http://www.alhem.net/

[20] "Mysql official web site." [Online]. Available: http://www.mysql.com/

[21] "Gnu/linux documentation." [Online]. Available: http://www.gnu.org/gnu/linux-and-gnu.html

[22] D. Zordan, "Applicazione dell'algoritmo di nesterov alla tecnica di compressive sensing in reti di sensori radio," *Tesi di Laurea Specialistica – Universita' di Padova*, 2010.

[23] "Piattaforma Java EE." [Online]. Available: http://java.sun.com/javaee/

[24] "Eclipse official web site." [Online]. Available: http://www.eclipse.org/

[25] "Sun developer network tutorial on jdbc technology." [Online]. Available: http://java.sun.com/products/jdbc/driverdesc.html

[26] "Java api, resultset object documentation." [Online]. Available: http://download-llnw.oracle.com/javase/1.4.2/docs/api/java/sql/ResultSet.html

[27] "Google web toolkit rpc introduction." [Online]. Available: http://code.google.com/intl/it-IT/webtoolkit/doc/1.6/DevGuideServerCommunication.html

[28] "Java servlet tutorial." [Online]. Available: http://www.jsptube.com/servlet-tutorials/simple-servlet-example.html

[29] E. J. Candes and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, no. 21, 2008.

[30] R. Masiero, G. Quer, D. Munaretto, M. Rossi, J. Widmer, and M. Zorzi, "On the interplay between routing and signal representation for compressive sensing in wireless sensor networks," *Workshop on Information Theory and Applications*, vol. ITA, no. San Diego, 2009.

[31] R. G. Baraniuk, M. Davenport, R. DeVore, and M. B. Wakin, "A simple proof of the restricted isometry principle for random matrices," *Constructive Approximation*, 2007.

[32] E. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. on Information Theory*, vol. 52, no. 2, pp. 489–509, 2006.

[33] R. Masiero, G. Quer, D. Munaretto, M. Rossi, J. Widmer, and M. Zorzi, "Data acquisition through joint compressive sensing and principal component analysis," *IEEE GLOBECOM 2009*, vol. Honolulu Hawaii, 2009.

Ringraziamenti

Vorrei esprimere un sincero ringraziamento a tutti coloro che mi hanno sostenuto in questi anni, in particolare al mio padre e mia madre residenti in Camerun, alle mie sorelle e fratelli; al prof Michele Rossi per avermi dato l'opportunità di svolgere questa attività presso il laboratorio SIGNET del dipartimento dell'ingegneria dell'Informazione, dell'Università degli Studi di Padova; infine, ai ricercatori del SIGNET che mi hanno guidato durante lo svolgimento di questa attività, i quali: gli Ing Andrea Bardella, Nicola Bressan, Nicola Bui, Riccardo Manfrin, Matteo Danieletto, Moreno Dissegna, Mattia Gheda, Davide Zordan e Angelo P. Castellani co-relatori di questa tesi.

A tous, merçi infiniment!!!

Anicet Foba Togue