Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria Elettronica, Elettrotecnica e delle Telecomunicazioni

Studio e implementazione di soluzioni WSN avanzate per l'IoT e applicazioni a sistemi reali

Study and development of advanced WSN solutions for IoT applications on real systems

Tesi di Dottorato di:
Incipini Lorenzo

Tutor:
Prof. Paola Pierleoni

Co-Tutor:
Marco Filipponi

Coordinatore del Curriculum:
Prof. Francesco Piazza

XXXII ciclo - nuova serie

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria Elettronica, Elettrotecnica e delle Telecomunicazioni

Studio e implementazione di soluzioni WSN avanzate per l'IoT e applicazioni a sistemi reali

Study and development of advanced WSN solutions for IoT applications on real systems

Tesi di Dottorato di:
Incipini Lorenzo

Tutor:
Prof. Paola Pierleoni

Co-Tutor:
Marco Filipponi

Coordinatore del Curriculum:
Prof. Francesco Piazza

XXXII ciclo - nuova serie

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Facoltà di Ingegneria
Via Brecce Bianche – 60131 Ancona (AN), Italy

A Marina e
alla mia Famiglia...

Ringraziamenti

La stesura dei ringraziamenti probabilmente rappresenta al meglio il percorso di studi portato a termine negli ultimi tre anni di dottorato, poiché in queste poche righe vengono raccolti gli incontri e le esperienze significative avute, e che ora fanno parte del mio bagaglio personale.

In *primis* voglio ringraziare la Prof. Paola Pierleoni, Marco Filipponi (Filmar Energy srl) e la Regione Marche per avermi dato la possibilità iniziare questo viaggio che mi ha fatto analizzare e affrontare le innovative e interessanti tematiche dell'*Internet of Things*. In questi anni, grazie al tema di ricerca assegnatomi, ho avuto modo di analizzare sotto diversi aspetti la realizzazione e la gestione delle reti di sensori *wireless* e la gestione degli ambienti *server* per il controllo della rete e dei valori misurati. Vorrei ringraziare il Prof. Mohamad El Mehtedi, l'Ing. Massimiliano Pieralisi e l'Ing. Tommaso Mancia, del DIISM, per avermi concesso la loro assistenza e l'accesso al loro laboratorio, per avermi sopportato durante l'installazione del mio sistema IoT e durante il monitoraggio delle loro macchine in quest'ultimo anno di dottorato.

Un sentito ringraziamento va al Prof. Ennio Gambi e all'Ing. Adelmo De Santis, e ai loro sforzi per rendere possibile la collaborazione con uno dei grandi *leader* internazionali nel settore delle telecomunicazioni, e dei momenti indimenticabili passati insieme a loro e ai compagni di squadra Danny Pigini e Mattia Silvestrini durante le le finali mondiali a Shenzhen (Cina). Un ulteriore ringraziamento è dovuto all'Ing. De Santis per aver dedicato molto del suo tempo alla formazione degli studenti che hanno mostrato il desiderio di intraprendere questo percorso formativo ricco di valore aggiunto.

Ringrazio anche il Prof. Massimo Conti che mi ha invitato a partecipare al progetto DAAD, dove ho potuto conoscere i colleghi tedeschi Maksym e Daniel, con i quali ho potuto passare piacevoli serate a Costanza in compagnia dei colleghi e amici dell'*Open Space* Laura, Adriana, Sara, Luca e Davide.

Voglio ringraziare anche Linda Senigagliesi per aver organizzato un fantastico viaggio per me, Danny, Paolo e Federico, dove abbiamo passato 3 fantastici giorni a spasso per l'Irlanda alla ricerca del famoso *festival* della musica.

Vorrei anche ringraziare Roberto Concetti per gli sforzi sostenuti nell'organizzare un così interessante viaggio d'istruzione a Monaco nello scorso periodo di Ottobre per scoprire i segreti della fermentazione e degustazione del malto tostato.

Ringrazio e saluto tutti i miei colleghi della sala dottorandi del Dipartimento di Ingegneria dell'Informazione e anche quelli emigrati all'estero alla ricerca di fortuna

e gloria.

Sono doverosi i ringraziamenti a tutti i miei amici. Susanna e Daniele, Annie e Federico, Piero e Giuditta, Tommaso e Giulia, Mattia e Linda, Stefano M., Giordano, Margherita, Stefano D., Daniele A., e anche ai piccolissimi Vittorio e Naima, che mi hanno accompagnato supportato e motivato durante tutti gli anni passati all'università.

Infine voglio ringraziare la mia Famiglia, e la mia compagna Marina, che mi hanno sempre appoggiato e sostenuto nelle scelte che ho fatto.

Grazie a tutti a Voi.

Ancona, Marzo 2020

Incipini Lorenzo

Prefazione

Durante il periodo di dottorato, condotto presso il Dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche, ho avuto il piacere di collaborare con il gruppo di ricerca guidato dalla Prof.ssa Paola Pierleoni.

Durante i tre anni di ricerca sono stati studiati i protocolli e i dispositivi per la realizzazione di *Wireless Sensor Networks* per applicazioni nell'ambito dell'*Internet of Things*. In particolare si sono indirizzati le risorse verso le tematiche del monitoraggio energetico, dell'illuminazione e del monitoraggio sismico e strutturale.
Sono stati portati avanti anche temi inerenti allo studio delle reti *enterprise* grazie ai corsi di alta formazione tenuti dall'Ing. Adelmo De Santis e da K Labs s.r.l presso i locali della Facoltà d'Ingegneria dell'Università Politecnica delle Marche. I corsi seguiti mi hanno permesso di ottenere 3 certificazioni Huawei[1] e di classificarmi al 2° posto a livello Europeo alla competizione internazionale Huawei ICT Skill Competition 2017/2018. Si è anche avuta la possibilità di partecipare all'*hackaton* LighHacker nel Luglio 2017, organizzato da iGuzzini presso la loro sede di Recanati, come consulente esperto per il *team* FunnyGain con cui ci si è aggiudicati il 2° posto alla competizione.

Inoltre, durante questi anni ho avuto la possibilità di collaborare con aziende come la iGuzzini Illuminazione, la Filmar Energy e con l'Istituto Nazionale di Geofisica e Vulcanologia, sui temi del monitoraggio utilizzando le *Wireless Sensor Networks*.

Parte del lavoro di questa tesi è stato incluso nelle seguenti pubblicazioni scientifiche:

- **Incipini, L.**, Belli, A., Palma, L., Concetti, R., & Pierleoni, P. (2019, May). MIMIC: a Cybersecurity Threat Turns into a Fog Computing Agent for IoT Systems. In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 469-474). IEEE.

- **Incipini, L.**, Belli, A., Palma, L., Concetti, R., & Pierleoni, P. (2019, May). Databases Performance Evaluation for IoT Systems: the Scrovegni Chapel Use Case. In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 463-468). IEEE.

[1] *HCNA Routing&Switching, HCNA WLAN, HCNA Cybersecurity.*

- **Incipini, L.**, Belli, A., Palma, L., Ballicchia, M., & Pierleoni, P. (2017). Sensing Light with LEDs: Performance Evaluation for IoT Applications. Journal of Imaging, 3(4), 50.
- **Incipini, L.**, Mancia, T., El Mehtedi, M., & Pierleoni, P. (2019, September). IoT Network for Industrial Machine Energy Monitoring. In 2019 AEIT International Annual Conference (AEIT) (pp. 1-6). IEEE.
- **Incipini, L.**, Palma, L., Belli, A., Raggiunto, S., & Pierleoni, P. (2019, June). Performance Evaluation of a Full IPv6-based Internet of Things Wireless Sensor Network. In 2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT) (pp. 333-338). IEEE.
- Pierleoni, P., Conti, M., Belli, A., Palma, **L., Incipini**, L., Sabbatini, L., ... & Concetti, R. (2019, June). IoT Solution based on MQTT Protocol for Real-Time Building Monitoring. In 2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT) (pp. 57-62). IEEE.
- Pierleoni, P., Belli, A., Palma, L., Valenti, S., Raggiunto, S., **Incipini, L.**, & Ceregioli, P. (2018). The Scrovegni Chapel Moves Into the Future: An Innovative Internet of Things Solution Brings New Light to Giotto's Masterpiece. IEEE Sensors Journal, 18(18), 7681-7696.

Durante questi anni oltre ai lavori focalizzati sul tema del monitoraggio con le WSN, che hanno portate all'elaborazione degli articoli sopracitati, sono stati condotti anche altri studi paralleli che sono stati raccolti nelle seguenti pubblicazioni:

- Pierleoni, P., Belli, A., Gentili, A., **Incipini, L.**, Palma, L., Valenti, S., & Raggiunto, S. (2018, July). A eHealth System for Atrial Fibrillation Monitoring. In Italian Forum of Ambient Assisted Living (pp. 229-241). Springer, Cham.
- Pierleoni, P., Belli, A., Palma, L., **Incipini, L.**, Raggiunto, S., Mercuri, M., ... & Sabbatini, L. (2019, June). A Cross-Protocol Proxy for Sensor Networks Based on CoAP. In 2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT) (pp. 251-255). IEEE.
- Baldi, M., Chiaraluce, F., **Incipini, L.**, & Ruffini, M. (2019). Code-based physical layer secret key generation in passive optical networks. Ad Hoc Networks, 89, 1-8.

Abstract

The research topic carried out during the three years of the PhD course is mainly focused on the study and development of *Internet of Things* solutions for monitoring and energy efficiency. These solutions consist of networks of smart wireless devices, on which appropriate sensors are installed for sampling electrical quantities. The research topic, is not only limited to energy monitoring, but also to monitoring environmental parameters such as: lighting, temperature, humidity, construction and structural parameters. Data measured by the different types of sensors are sent by the network to a designated cloud platform, which takes care of their storage and use through a management system developed within the research period. The latest technologies and protocols related to the Internet of Things have been studied and implemented for the development of the monitoring network. Starting from the physical and data link layer of the ISO/OSI protocol stack, implemented with the IEEE 802.15.4 standard, proceed with the network layer using the IPv6 protocol, then with the transport described by the TCP and UDP protocols, up to application layer where HTTP and CoAP protocols are used.

Using the developed monitoring system, experiments on the field were conducted in different application scenarios, not only in the context of industrial energy monitoring but also in the lighting and structural monitoring context. In order to evaluate the realized *Internet of Things* system, specific tests were conducted to analyze the performance obtained.

Sommario

Il tema di ricerca sviluppato durante i tre anni del corso di dottorato è principalmente focalizzato sullo studio e sviluppo di soluzioni *Internet of Things* per il monitoraggio ed efficientamento energetico. Tali soluzioni sono costituite da reti di dispositivi *wireless* intelligenti, sui quali, vengono installati opportuni trasduttori per il campionamento delle grandezze elettriche. Nell'ambito del tema di ricerca, non ci si é solo limitati al monitoraggio energetico, ma anche al monitoraggio di parametri ambientali quali: illuminazione, temperatura, umidità, parametri costruttivi e strutturali.

I dati misurati dai diversi tipi di trasduttori vengono inviati dalla rete verso una designata piattaforma Cloud, che si occupa della loro memorizzazione e fruizione tramite un sistema gestionale sviluppato nell'ambito del tema di ricerca. Mentre per lo sviluppo della rete di monitoraggio sono state studiate le più recenti tecnologie e protocolli relativi all'*Internet of Things*. Partendo dal livello fisico e di collegamento dello *stack* protocollare ISO/OSI, implementati con lo standard IEEE 802.15.4, si procede con il livello di rete utilizzando il protocollo IPv6, poi con il livello di trasporto descritto dai protocolli TCP e UDP, fino al livello applicazione dove vengono utilizzati i protocolli HTTP e CoAP.

Utilizzando il sistema di monitoraggio sviluppato, sono state condotte delle sperimentazioni sul campo in diversi scenari applicativi, non solo nel contesto monitoraggio energetico industriale ma anche in quello dell'illuminazione e strutturale. Al fine di valutare il sistema *Internet of Things* realizzato, sono stati condotti dei test specifici per l'analisi delle prestazioni ottenute.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli scripts

Glossario

6LoWPAN IPv6 *over Low-Power Wireless Personal Area Networks.* xxi, xxiii, 9, 22, 24, 43, 71–76

ACS Acqua Calda Sanitaria. 52

AI *Artificial Intelligence.* 10

API *Application Programming Interface.* 9, 39, 77, 79

APN *Access Point Name.* 50

ARP *Address Resolution Protocol.* 77

BACS *Building & Automation Control System.* 55

CCA *Clear Channel Assessment*. 14

CoAP *Constrained Application Protocol.* xiii, xv, xxv, 27, 59, 61, 62, 72

CSK *Color Shift Keying*. 20

CSMA-CA *Carrier Sense Multiple Access - Collision Avoidance.* 15

CSMA-CD *Carrier Sense Multiple Access - Collision Detection.* 15

CSS Cascading Style Sheets. 49

Cu *Copper*. 30

d.lgs. Decreto Legislativo. 51–53

DAG *Direct Acyclic Graph.* 25

DAO *Destination Advertisement Objects.* 25, 26

DB *DataBase.* 9, 91

DBMS *DataBase Management System.* xxiii, 9, 78–80

DII Dipartimento di Ingegneria dell'Informazione. ix, 39, 71

DIISM Dipartimento di Ingegneria Industriale e Scienze Matematiche. ix, 38, 71, 91

DIO DODAG *Information Object*. 25

DIS *DODAG Information Solecitation.* 26

DNS *Domain Name System.* 50, 77

DODAG *Directed Oriented Direct Acyclic Graph.* 25

ED *Energy Detection.* 14

EGE Esperto in Gestione Energia. 52

ENEA Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile. 52, 53

ESC *Energy Service Company*. 52

ESN *Environmental Sensor Network*. 10

FFD *Full Function Device.* 13

GARR Gruppo per l'Armonizzazione delle Reti della Ricerca. 1

GPRS *General Packet Radio Service.* xxv, 50

HC1 *Header Compression 1.* 24

HTTP *HyperText Transfer Protocol.* xiii, xv, 42, 45, 46, 48, 49, 80

I/O *Input/Output*. 42

IEEE *Institute of Electrical and Electronic Engineers.* xiii, xv, 13–15, 22, 24, 78

IETF *Internet Engineering Task Force.* 25

IFS *Inter Frame Space.* 15

INGV Istituto Nazionale di Geofisica e Vulcanologia. 2, 64

IoE *Internet of Everything*. 6, 7

IoT *Internet of Things.* ix, xi, xiii, xv, xxi, 1, 3–9, 11, 13, 17, 22, 44, 45, 51, 54, 58, 71, 72, 76–80, 84, 85, 91, 92

IP *Internet Protocol.* 1, 3, 22, 46, 49, 77, 79

ISCT *International Symposium on Consumer Technologies.* 69

ISM *Industrial Scientific Medical.* 14

IT *Information Technology.* 8

JSON *Java Script Object Notation.* xxv, 79, 80, 84–86

LAN *Local Area Network.* 15

LED *Light Emitting Diode.* xix, 8, 16–18, 20, 22, 55

LLN *Low power and Lossy Network.* 25

LQI *Link Quality Indication.* 14

LR *Linear Regression.* 87, 89

LR-WPAN *Low Rate Wireless Personal Area Network.* 13

M2M *Machine to Machine.* 3, 49

MAC Medium Access Control. 8, 13, 14, 16

MIM *Man-In-The-Middle.* 76, 77

MIMIC *Man-In-the-Middle Iot Computing.* xxi, 76–79

ML *Machine Learning.* 10, 11, 85, 91

MTS *Material Test System.* 63, 91

MTU *Maximum Transmission Unit.* 22

NFV *Network Function Virtualization.* 8, 9

NoSQL *Non Structured Query Language.* 9, 79

OF *Objective Function.* 25

OF0 *Objective Function Zero.* 26, 75

OWC *Optical Wireless Communication.* 16

P&G Procter & Gamble. 3

PAN *Personal Area Network.* 13–15

PDU *Protocol Data Unit.* 14

PPDU *Physical Protocol Data Unit.* xix, 14

PPP *Point-to-Point Protocol.* 50

PSDU *Physical Service Data Unit.* 14, 78

QoS *Quality of Service.* 8, 77

RFC *Request For Comment.* 26

RFD *Reduced Function Device.* 13

ROLL *Routing Over Low power and Lossy networks.* 25

RPL *Routing Protocol for Low power and Lossy Network.* 25, 75

RTT *Round Trip Time.* xxi, 72, 73, 75

SD_WSN6Lo *Wireless Sensor Network framework for 6LowPAN.* 8, 9

SDN *Software Defined Network.* 8

SDR *Software Defined Radio.* 8

SFV *Sensor Function Virtualization.* 9

SHMS *Self-learning Home Management System.* 11

SIM *Subscriber Identity Module.* 49, 50

SLIP *Serial Line Internet Protocol.* 43

SOSUS *SOund SUrveillance System.* 5

SQL *Structured Query Language.* 9, 79

TCP *Transmission Control Protocol.* xiii, xv, 1, 30

TIA *Transimpedance Amplifier.* xix, 17

UDP *User Datagram Protocol.* xiii, xv

URI *Uniform Resource Identifier.* 77

URL *Uniform Resource Locator.* xxv, 45, 62

VLC *Visible Light Communication.* 8, 16, 22

VM *Virtual Machine.* 9

WAN *Wide Area Network.* 42, 78

WC *Working Group.* 25

WPAN *Wireless Personal Area Network.* 13

WSGI Web Server Gateway Interface. 48, 49

WSN *Wireless Sensor Network.* xi, xii, 5, 6, 8–11, 13, 14, 41, 42, 54, 77, 78, 84, 85, 91, 92

Capitolo 1

Introduzione

Al giorno d'oggi viviamo immersi in ambienti e luoghi monitorati dai più diversi dispositivi intelligenti connessi tra di loro, in modalità cablata o *wireless*, che acquisiscono dati dall'ambiente per generare conoscenza. Le misure effettuate dai dispositivi vengono inviate, attraverso un opportuno mezzo trasmissivo, ad un sistema centrale per la loro analisi e memorizzazione. Il numero dei nuovi oggetti *smart* sta crescendo rapidamente, e infatti Cisco prevede che ci saranno più di 50 miliardi di dispositivi connessi alla rete globale nel 2020. Ovviamente questo fenomeno sta attirando l'interesse delle industrie, che vedono un importante settore in espansione, e allo stesso tempo i centri di ricerca sono attratti dalle possibilità di creare innovazione nel mondo dell'*Internet of Things*, intuendo la necessità di nuove soluzioni per i problemi emergenti nella gestione di questa enorme quantità di dispositivi interconnessi alla rete.

In questo lavoro di tesi vengono studiate e analizzate le reti di sensori *wireless*, in particolare quelle reti adatte a realizzare le architetture dell'*Internet of Things*. Per consentire a questi oggetti intelligenti di accedere alla rete globale, implementando il paradigma dell'*Internet of Things*, viene usato lo *stack* protocollare TCP/IP sui singoli *smart devices* che compongono l'architettura della rete di sensori. Nello specifico, per questo lavoro di ricerca, è stato adoperato il protocollo IPv6 per l'instaurazione della rete di comunicazione e, grazie al supporto del Gruppo per l'Armonizzazione delle Reti della Ricerca, è stato possibile utilizzare indirizzi IPv6 pubblici per la realizzazione di alcuni progetti. La scelta dell'utilizzo del protocollo IPv6 è dovuta all'estensione del suo spazio di indirizzamento. Rispetto al protocollo IPv4 si passa infatti dall'indirizzamento a 32 bit all'indirizzamento a 128 bit, che corrisponde alla possibilità utilizzare più di 340 miliardi di miliardi di miliardi di miliardi di indirizzi IP.

Vengono inoltre trattati i protocolli di comunicazione utilizzati per consentire ai nodi della rete di sensori *wireless* di realizzare il paradigma dell'*Internet of Things*, e sono stati condotti severi *tests* per valutare le prestazioni dei protocolli implementati. Alla rete *wireless* creata si è proceduto ad aggiungere un *border router* per la memorizzazione dei dati raccolti all'interno di un *database*, ed è stato realizzato un *server*

web in Python per la visualizzazione dei dati presenti nel *database* e per la gestione della dei nodi della rete.

Nel quinto capitolo vengono mostrati casi d'uso reali del sistema di monitoraggio sviluppato nel corso dei tre anni di dottorato. Il sistema così composto è stato installato, in collaborazione con l'azienda iGuzzini Illuminazione, presso la Cappella degli Scrovegni di Padova per monitorare l'illuminazione dei famosi affreschi di Giotto. Inoltre una seconda versione dello stesso sistema di monitoraggio è stata installata presso i laboratori della Facoltà di Ingegneria dell'Università Politecnica delle Marche, per rilevare i consumi energetici delle macchine industriali presenti nel laboratorio. A chiudere questo capitolo, viene descritto un prototipo per un sistema di monitoraggio di eventi sismici, realizzato in collaborazione con l'Istituto Nazionale di Geofisica e Vulcanologia, e composto da un dispositivo *embedded* connesso tramite protocollo seriale all'unità di *sensing*.

Nell'ultimo capitolo di questo elaborato, vengono introdotti i temi del *Machine Learning*, che saranno utilizzati per l'analisi delle misure effettuate nel corso degli anni dal sistema di monitoraggio realizzato e installato presso la Cappella degli Scrovegni.

Capitolo 2

Lo stato dell'*Internet* delle cose

L'internet delle cose o più propriamente l'*Internet of Things* (IoT) è divenuta una tecnologia rivoluzionaria che ha suscitato moltissimo interesse, sia in ambito industriale che in ambito accademico. Il termine IoT venne introdotto per la prima volta nel 1999 da Kevin Ashton che lo utilizzò come titolo per una presentazione presso Procter & Gamble (P&G). Tra le molte definizioni di IoT che sono presenti nella rete viene qui riportata quella dello stesso Ashton:

> *The Internet of Things means sensors connected to the Internet and behaving in an Internet-like way by making open, ad hoc connections, sharing data freely and allowing unexpected applications, so computers can understand the world around them and become humanity's nervous system.*
>
> ***Kevin Ashton***.

L'idea dell'*iot* nasce quindi come soluzione per consentire ai computers di interagire con l'ambiente e di condividere i dati raccolti tra di loro. Ma ancora più importante è la possibilità di creare sempre nuovi servizi e applicazioni che prima non erano nemmeno concepibili. Diventa necessario consentire ai calcolatori di poter percepire e operare sull'ambiente circostante, e ciò diventa possibile mediante l'utilizzo di trasduttori e attuatori. I primi vengono utilizzati per convertire grandezze fisiche (come la temperatura, umidità, accellerazione, luminosità, *etc.*) in un segnale elettrico proporzionale, che può essere interpretato da una macchina. Invece gli attuatori convertono un segnale elttrico in un'azione (come i relé, motori elettrici, valvole, *etc.*). Queste due famiglie di strumenti permettono quindi ai sistemi IoT di acquisire informazioni e di interagire con l'ambiente.

Oltre a raccogliere dati e intervenire in modo attivo, i dispositivi IoT sono caratterizzati da connettività IP (v4 o v6), che permette loro di essere identificati nella rete e di poter essere raggiunti per accedere alle loro funzionalità. Infatti una caratteristica chiave dei dispositivi IoT è la possibilità di accedere a *Internet* [1]. Nasce così una nuova era di *Internet* dove non soltanto le persono possono comunicare e condividere dati attraverso la rete, ma dove anche oggetti intelligenti si scambiano informazioni tra di loro (*Machine to Machine* (M2M)) nel mondo digitale. La continua

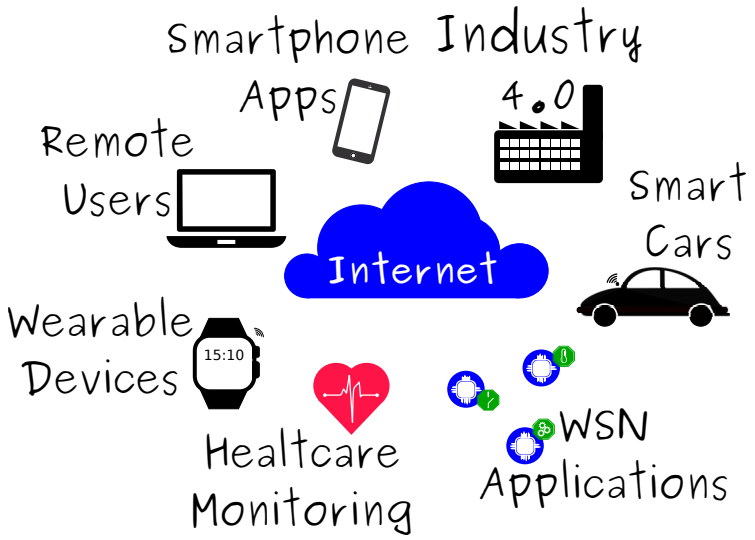

Figura 2.1: Visione dell'*Internet of Things*, oggetti e sistemi connessi ad *Internet*.

rischiesta di conoscere l'andamento delle caratteristiche dell'ambiente, delle macchine, o delle persone, ha portato ad un forte aumento della distribuzione di dispositivi IoT sul territorio. Infatti Intel ha predetto che nel 2020 ci saranno più di 200 miliardi di dispositivi connessi alla rete [2], ovvero circa 26 oggetti per ogni uomo. Di fatto questo valore è plausibile se si pensa ai possibili scenari applicativi dei sistemi IoT, che vanno dagli oggetti indossiabili in ambito *healt-care* fino alle nuove *smart cars*. Di seguito si ha un elenco dei possibili scenari applicativi del paradigma IoT:

- *smart cities* [3, 4, 5]
- *smart cars* [6, 7, 8]
- *smart energy* [9, 10, 11]
- *smart home* [12, 13, 14]
- *healt-care* [15, 16, 17]
- agricoltura [18, 19, 20]
- industria 4.0 [21, 22, 23]
- *early warning system* [24, 25, 26]
- *etc.*. [27]

L'elenco sopra riportato rappresenta un sottoinsieme dei possibili scenari applicativi dell'IoT, le tecnologie sono in continua evoluzione e possono nascere continuamente nuovi casi di applicabilità per questi dispositivi *smart* connessi alla rete globale.

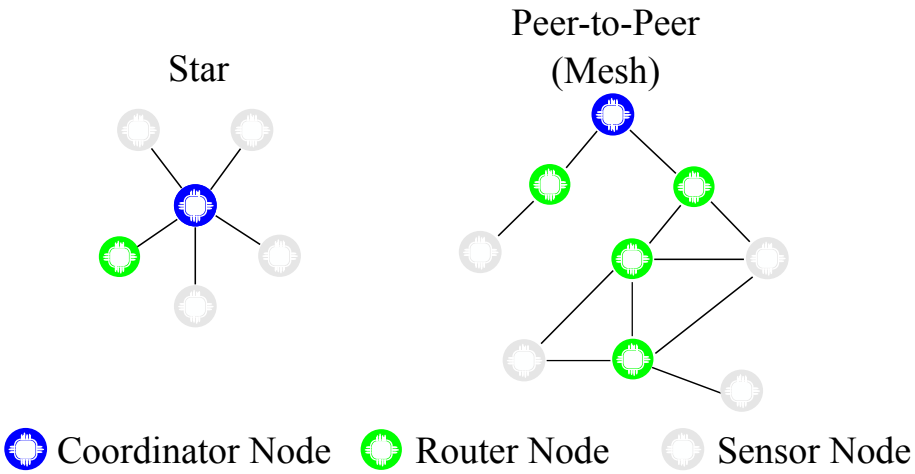

Figura 2.2: Topologie tipiche delle WSN. Da sinistra a destra: rete a stella, rete *Peer-to-Peer*.

2.1 IoT vs WSN

A fianco al concetto e al paradigma di IoT si hanno le cosiddette *Wireless Sensor Network* (WSN), ovvero le reti di dispositivi muniti di unità di *sensing* e di connettività *wireless*. Le reti di sensori non sono un concetto nuovo, infatti, la prima rete di sensori che può essere accomunata ad una moderna WSN è il sistema *SOund SUrveillance System* (SOSUS), sviluppato nel 1950 dall'esercito USA.

Oggigiorno una rete WSN risulta essere composta di un elevato numero di nodi, anche dell'ordine delle migliaia, a basso consumo energetico (generalmente alimentati a batteria), di piccole dimensioni, poco invasivi e a basso costo. Questi nodi andranno a costituire una rete che può assumere differenti topologie (Fig. 2.2) a seconda delle necessità dello specifico caso d'uso. I nodi di queste reti hanno il compito di monitorare una o più grandezze fisiche, e di inviarle al nodo collettore (*sink node*) che si occuperà di elaborare i dati ricevuti e di compiere le opportune decisioni sulla base delle specifiche di progetto. A prescindere dalla topologia adottata per costruire la WSN (Fig. 2.2), la rete è in grado di raccogliere dati (*i.e.* temperatura, umidità, accellerazione, *etc.*) e di trasmetterli al coordinatore che si occuperà della loro elaborazione. Una rete di questo tipo non ha la necessità di avere a disposizione una connessione verso *Internet*. Tuttavia è evidente che una WSN può essere dotata di connettività *Internet* grazie all'utilizzo di un apposito nodo *gateway*, che permette l'inoltro dei pacchetti dalla rete globale alla WSN e viceversa. Quindi si può affermare che la connettività nativa verso la rete globale per i dispositivi IoT risulta essere la differenza sostanziale tra il concetto di IoT e di WSN [1, 28]. Invece quando una WSN dispone di connettività remota, la si può caratterizzare come un'applicazione

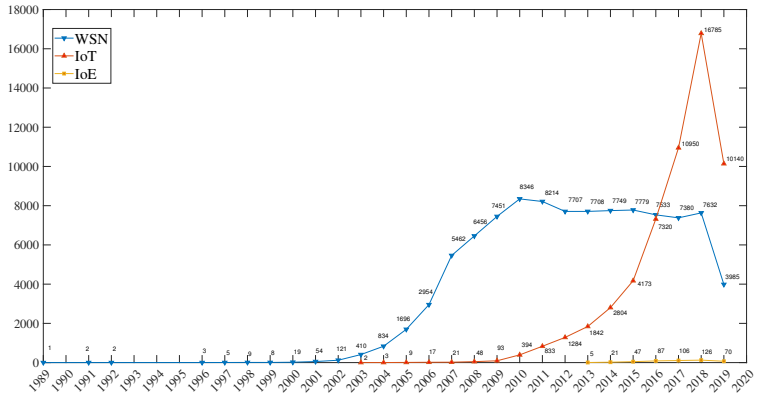

Figura 2.3: Andamento delle pubblicazioni scientifiche dal 1989 al 2019 per WSN, IoT e IoE. **Fonte: Scopus**.

IoT, dove i sensori sono in grado di comunicare tra di loro (M2M) attraverso una connessione wireless.

Per comprendere meglio il legame che sussiste tra IoT e WSN, vengono mostrati in Fig. 2.3 il numero di pubblicazioni scientifiche relative all'IoT, le WSN e l'*Internet of Everything* (IoE). Queste tre curve sono state ottenute cercando sul *database* Scopus le seguenti parole chiave: "*wireless sensor network*", "*internet of things*" e "*internet of everything*", e il numero di pubblicazioni per anno é stato messo a confronto. Come mostrato dalla Fig. 2.3, a partire da circa il 2010, si ha una crescita esponenziale delle pubblicazioni inerenti il tema dell'IoT, e allo stesso tempo un calo delle pubblicazioni riguardo alle WSN. Ciò non significa che i ricercatori stanno riducendo l'attenzione alle WSN, ma piuttosto che l'IoT ha assorbito le WSN diventandone parte integrante [28, 29, 30], e di conseguenza indirizzando le risorse e l'interesse della comunità scientifica verso il settore dell'IoT. Insieme a queste due tecnologie inizia a comparire il concetto di IoE, ovvero una rete che connette tutti gli oggetti e le persone del mondo.

2.2 L'*Internet del tutto*

L'IoE è una nuova visione, o più semplicemente una naturale evoluzione [31], del concetto di IoT. Nel 2013 Cisco definisce in un documento [32] il concetto di IoE come:

> The Internet of Everything (IoE) brings together people, process, data, and things to make networked connections more relevant and valuable than ever before - turning information into actions that create new capabilities, richer experiences, and unprecedented economic opportunity for businesses, individuals, and countries.
>
> ***Cisco, 2013.***

Cisco getta così le basi per l'IoE, un nuovo paradigma per le tecnologie che si basa su quattro pilastri fondamentali [33]:

1. persone (*people*)
2. processi (*process*)
3. dati (*data*)
4. oggetti (*things*)

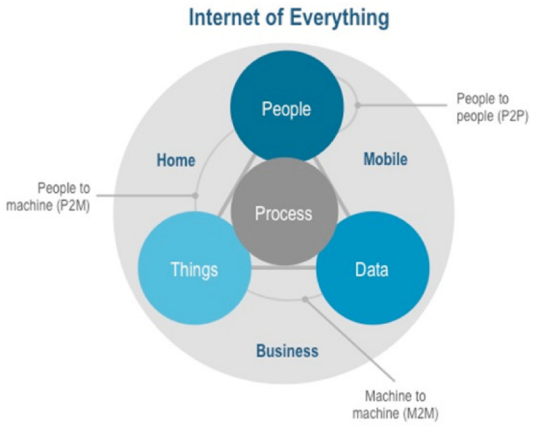

Figura 2.4: *Internet of Everything.* **Fonte: Cisco IBSG, 2012.**

dove l'IoT è uno dei quattro pilastri (4. *things*) che costituiscono l'IoE. Con questo nuovo paradigma si possono analizzare in tempo reale dati di milioni di oggetti, sensori e persone, al fine di generare conoscenza attraverso processi automatizzati per aiutare le persone nella vita quotidiana (Fig 2.4, [34]).

2.3 *Related Works*

Durante la fase di studio e, successivamente di sviluppo del progetto di ricerca, sono stati valutati numerosi lavori prodotti da colleghi della comunità scientifica. In questa sezione saranno riportati i lavori tenuti in considerazione durante la fase di analisi e progettazione del sistema IoT sviluppato.

2.3.1 *Visible Light Communication*

La tecnologia delle *Visible Light Communication* (VLC) permette di inviare dati attraverso segnali ottici *wireless* invece che sulle più classiche fibre ottiche e molti ricercatori stanno studiando i livelli fisici (PHY) e MAC per valutarne le capacità trasmissive e aumentarne l'efficienza. In particolare per l'analisi dello stato dell'arte del VLC si è data maggior importanza alle comunicazioni *LED-to-LED*, ovvero sulla trasmissione e ricezione di segnali ottici attraverso l'utilizzo di soli LED come *transceiver*. Quest'analisi ha portato a studiare i lavori della Disney Research [35, 36], che sono indirizzati all'utilizzo del canale *LED-to-LED* per il telecontrollo dei giocattoli. Infatti in [35] Schmid *et. al.* hanno utilizzato i LED come trasmettitori e ricevitori di segnali ottici per comunicazioni *wireless* a breve distanza, e avere la possibilità di utilizzare lo stesso LED come trasmettitore e ricevitore risulta essere un notevole vantaggio. Per ottenere questo risultato, in [35], sono stati studiati e analizzati il livello fisico (PHY) e il livello Medium Access Control, ed è stata realizzata e successivamente valuta la loro implementazione del CSMA-CA sul canale *LED-to-LED*. Il *testbed* utilizzato per verificare le funzionalità del sistema descritto è costituito da 7 *boards* Arduino, in cui su ognuna di esse è montato un LED rosso da 5mm trasparente con montaggio a foro passante. Con questo semplice *setup* sono state valutate le prestazioni del sistema VLC realizzato.

Un'analisi più generale sullo stato dell'arte della tecnologia VLC è stata condotta nel *survey* [37], in cui si descrivono le sorgenti e i ricevitori utilizzabili (LED), il modello del canale e le tecniche per il suo accesso e le tecniche di modulazione del segnale adatte alle comunicazioni ottiche.

Gli articoli scientifici presentati costituiscono una *summa* adatta ad inquadrare il comportamento dei LED come *transceivers* per il VLC e i risultati dello studio sono stati pubblicati in [38].

2.3.2 Virtualizzazione delle risorse

Le tecnologie di virtualizzazione si sono diffuse in molti ambiti dell'IT, come ad esempio le soluzioni *Cloud* offerte dalle aziende più note (Amazon, Google, Microsoft, IBM, *etc...*), o come le tecnologie di virtualizzazione delle risorse di rete, ovvero la *Software Defined Network* (SDN) e la *Network Function Virtualization* (NFV), o addirittura con la virtualizzazione dell'*hardware* per le comunicazioni radio utilizzando la *Software Defined Radio* (SDR).

Nell'ambito di questo progetto di ricerca, cioè nella progettazione di sistemi IoT per il monitoraggio di parametri ambientali ed elettrici, è stata presa in considerazione la virtualizzazione come strumento per aumentare l'efficienza della QoS delle trasmissioni delle WSNs implementate [39, 40, 41], e in letteratura sono stati valutati diversi approcci nei confronti della virtualizzazione applicata all'IoT. Infatti, basandosi sul paradigma delle SDN, Lasso *et.al.* hanno proposto il *Wireless Sensor Network fra-*

mework for 6LowPAN (SD_WSN6Lo) per il controllo dell'instradamento dei pacchetti all'interno della rete 6LoWPAN [42]. I risultati ottenuti dalle simulazione effettuate con il *tool* Cooja, hanno permesso di determinare un risparmio energetico di circa il 15% dei consumi dei nodi della rete. Il tema del risparmio energetico dei nodi delle reti 6LoWPAN è stato affrontato anche in altri lavori scientifici, come in [43], dove sono state spostate parte delle funzionalità del livello applicativo su risorse virtuali. In questo modo è stato ottenuto un considerevole risparmio energetico pari a circa il 40%.

Uno studio molto interessante e con degli aspetti che lo rendono in parte a simile a quanto proposto in [44], è stato presentato da Van den Abeele *et.al.* in [45] e riguarda l'*exploiting* delle NFV nell'ambito delle reti di sensori. Nel contesto delle WSN, l'NFV viene rivista e modificata nel nuovo paradigma della *Sensor Function Virtualization* (SFV). Questa soluzione proposta viene principalmente gestita su una piattaforma *Cloud* installata su un *server* remoto, mentre un punto chiave sottolineato nel loro studio, e rimarcato in [44], riguarda la trasparenza della soluzione proposta per il *client* o per l'utente remoto che utilizza la risorsa virtualizzata. Ciò significa che non devono essere introdotte nuove API per poter accedere alle risorse virtualizzate, viene così notevolmente semplificata la gestione dell'accesso ai dati raccolti dai dispositivi IoT.

2.3.3 *Databases* per IoT

In letteratura, e più in generale nel *web*, si possono trovare moltissimi *tool* di *benchmarking* per la valutazione delle prestazioni dei *databases*. Per la scelta del *DataBase Management System* (DBMS) da utilizzare nel progetto di ricerca, sono stati analizzati diversi lavori scientifici focalizzati sull'analisi delle prestazioni dei *databases* nel contesto delle WSN e dell'IoT. Questa scelta è dovuta alla potenza di questi strumenti, infatti i *tests* effettuati dai migliori *tool* di *benchmarking* disponibili sul mercato, vanno a stressare il *server* che ospita il *database* con milioni di operazioni al secondo. Nel sistema sviluppato, dato che la densità di nodi è inferiore alle 10 unità, e il campionamento dell'ambiente avviene ogni 5 min, non si ha la necessità di valutare il sistema utilizzando questi *tool* che effettuano milioni di *query* al secondo.

Come termine di paragone per l'analisi condotta sui diversi *databases* considerati nei *test*, è stato scelto principalmente il lavoro di Van Der Venn [46] che mette a confronto le prestazioni di un insieme di 3 differenti *databases*, tra cui un DB SQL e due DB NoSQL, valutati sia su un *server* che su *Virtual Machine* (VM) nel contesto dei dati generati dalle reti di sensori. I *clients* utilizzano connettori in Java per effettuare le operazioni sui singoli DB, e vengono eseguite operazioni di scrittura/lettura di un singolo dato e di scrittura/lettura di blocco di dati, sfruttando le API messe a disposizione dai differenti DBMS.

2.3.4 Prestazioni IPv6

Per valutare le prestazioni della rete IPv6 di sensori *wireless* realizzata per il monitoraggio dell'illuminazione e delle grandezze elettriche nei rispettivi progetti di ricerca, è stato considerato principalmente il lavoro presentato in [47, 48]. In questi due contributi viene descritto il progetto di una *Environmental Sensor Network* (ESN), cioè di una rete IPv6 di sensori *wireless* per il monitoraggio di parametri ambientali nelle montagne scozzesi. Di questa installazione Bragg *et. al.* hanno valutato le prestazione di rete in termini di latenza, pacchetti persi e *throughput*.

Altre soluzioni IP per il monitoraggio in ambito WSN sono state proposte, e alcune di queste utilizzano soluzioni di *natting* 6-4 sfruttando le capacità del NAT64 dei *gateway*, o in alternativa, si utilizzano *proxy* per l'inoltro delle richieste da reti IPv4 a reti IPv6.

2.3.5 IoT per il monitoraggio energetico

La continua e sempre più rapida evoluzione tecnologica ha permesso di realizzare dispositivi *smart* più piccoli ed economici, che trovano facile impiego in soluzioni di monitoraggio delle più diverse grandezze fisiche. In particolare, nel contesto dell'Industria 4.0, questi dispositivi vengono utilizzati per il controllo e la gestione dei processi industriali. Tra queste tipologie di controllo e gestione rientrano i sistemi di monitoraggio dei parametri energetici, legati al consumo di corrente delle macchine industriali, e dei parametri ambientali (temperatura, umidità, illuminazione) per la gestione dei sistemi di riscaldamento, ventilazione e illuminazione.

Con la ricerca condotta in [49] è stato realizzato un *framework* per adattare i cicli di produzione dell'azienda con le fasce orarie a basso costo del contratto del fornitore energetico, basando il *framework* sul modello della domanda e dell'offerta.

2.3.6 *Machine Learning* per l'IoT

Il *Machine Learning* è un settore della ricerca nell'ambito dell'*Artificial Intelligence* con lo scopo di creare macchine non soltanto intelligenti ma anche capaci di apprendere autonomamente. La prima definizione del ML la si può attribuire ad Arthur Samuel:

> *Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.*
>
> ***Arthur Samuel.***

Con questo approccio al problema della programmazione, Arthur Samuel addestrò un computer facendogli giocare migliaia di partite di scacchi, trasformandolo in un giocatore imbattibile [50, 51]. Quest'anno gli algoritmi di ML sono stati utilizzati per addestrare una macchina a giocare al popolare *videogame* Starcraft 2, facendogli

giocare decine di migliaia di partite, e i risultati ottenuti dal modello sono stati eccezionali [52], il *computer* è stato in grado di battere i migliori campioni internazionali di Starcraft 2.

Gli algoritmi di *Machine Learning* non sono solo applicati al contesto della videoludica ma in tutte quelle applicazioni dove si hanno a disposizione grandi quantità di dati (*Big Data*), e trovano ovviamente anche uso nell'ambito delle WSN e dell'IoT. In letteratura si trovano molti contributi scientifici relativi all'impiego di algoritmi di ML nel contesto dell'IoT. In [53] viene presentato un dettagliato *survey* che descrive gli algoritmi di ML, le loro applicazioni in vari ambiti tra cui l'IoT e un'interessante caso d'uso per classificare le pubblicazioni che trattano di IoT e ML.

Nell'ambito del monitoraggio domestico (*Smart Home*) è stato sviluppato da *Li et.al.* in [54] un sistema definito *Self-learning Home Management System* (SHMS) per la classificazione dei consumi domestici con il fine di ridurre i consumi e i costi, attivando le utenze domestiche nelle fasce orarie ottimali. Lo studio riportato in [55] descrive un nuovo metodo per scoprire le relazioni che sussistono tra le misurazioni effettuate da sensori eterogenei di un sistema IoT.

2.4 Obiettivi della ricerca

Con il lavoro di ricerca condotto all'interno delle tematiche dell'IoT e delle WSN, vengono studiati ed analizzati i protocolli di comunicazione, le scelte *hardware* e *software* per la realizzazione dei nodi sensori, le soluzioni *front-end* e *back-end* per l'implementazione di server per la gestione dei dati raccolti e gli algoritmi di *Machine Learning* (ML) per l'elaborazione e generazione di conoscenza.

Tutti i lavori della comunità scientifica riportati sono stati tenuti in considerazione per migliorare il sistema IoT sviluppato durante il progetto di ricerca, e dove possibile i risultati di questi lavori sono stati utilizzati come confronto per le prestazioni ottenute dalla rete di sensori *wireless* utilizzata nei diversi casi d'uso realizzati.

Capitolo 3

Architettura del sistema IoT

In questo capitolo vengono descritti i protocolli che sono stati impiegati per realizzare il sistema di monitoraggio IoT basato su WSN. Si inizierà la descrizione a partire dai livelli protocollari più bassi dello *stack* ISO/OSI fino ad arrivare al livello applicazione.

3.1 Lo *standard* IEEE 802.15.4

Lo *standard Institute of Electrical and Electronic Engineers* (IEEE) 802.15.4 [56] definisce le specifiche per i livello fisico (PHY), e le specifiche del sottolivello di accesso al canale (MAC[1]) che appartiene al livello di collegamento dello *stack* ISO/OSI. Queste specifiche sono studiate per le *Low Rate Wireless Personal Area Network* (LR-WPAN), cioè quelle reti composte da dispositivi a basso costo, ridotti consumi energetici e a basso *throughput*. Due tipologie di dispositivi possono costituire una rete IEEE 802.15.4: i *Full Function Device* (FFD) e i *Reduced Function Device* (RFD). I primi sono in grado di operare come coordinatori della *Personal Area Network* (PAN), mentre gli altri sono utilizzati per applicazioni semplici che richiedono il minimo delle risorse e di capacità di calcolo del dispositivo. Due dispositivi dotati i interfaccia radio che implementa lo *standard* IEEE 802.15.4, uno dei quali è un FFD, costituiscono una *Wireless Personal Area Network* (WPAN). Utilizzando queste due tipologie di dispositivi si possono andare a creare delle reti molto semplici, come le reti punto-punto, composte da solo due nodi, a reti molto dense composte da centinaia o migliaia di nodi. Lo *standard* prevede che questi dispositivi possono formare due topologie di reti differenti, le reti *Peer-to-Peer*[2] e le reti a stella (Fig. 2.2), e in entrambi i casi le connessioni vengono instaurate tra i dispositivi e il coordinatore della PAN. Questi dispositivi vengono identificati all'interno della PAN tramite un indirizzo univoco detto (*extended address*). Una rete *Peer-to-Peer* consente inoltre di instradare i messaggi tra i diversi dispositivi che compongono la rete stessa, creando così delle topologie molto complesse come la topologia *mesh*. Tuttavia la funzionalità di inoltro dei messaggi

[1] *Medium Access Control*.

[2] Si tratta di una tipologia di rete dove ogni dispositivo é in grado di comunicare con qualunque altro purché sia in copertura radio.

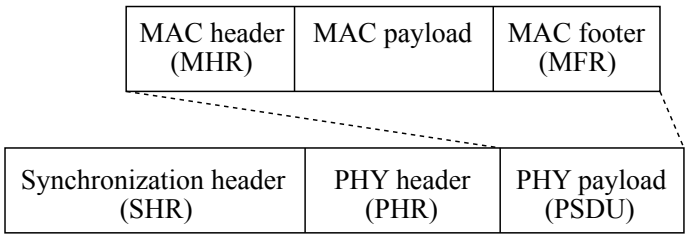

Figura 3.1: Rappresentazione schematica della *Physical Protocol Data Unit* (PPDU) e del *frame* MAC.

deve essere implementata ai livelli superiori dello *stack* protocollare, e quindi non rientra in questo *standard*, così come la topologia *mesh*.

3.1.1 Livello Fisico (PHY)

Il livello fisico è il livello più basso dello *stack* protocollare, e si occupa della gestione del *transceiver* radio, *Energy Detection*, *Link Quality Indication* (LQI), *channel selection*, *Clear Channel Assessment* (CCA), *ranging*, e infine trasmissione/ricezione di pacchetti.

Un parametro importante che appartiene al livello fisico è il *aMaxPhyPacketSIze*, ovvero, la dimensione massima della *Physical Service Data Unit* (PSDU) che i dispositivi conformi allo *standard* IEEE 802.15.4 devono essere grado di ricevere. Lo *standard* supporta l'utilizzo di PSDU di dimensioni pari a 127 ottetti (Fig. 3.1). Per questo progetto di ricerca le WSNs utilizzate operano nella banda ISM[3] alla frequenza sub-GHz di 868MHz con una *rate* massima di $20kb/s$.

3.1.2 Sottolivello *Media Access Control*

Il sottolivello Medium Access Control (MAC) fornisce due servizi, il *data service* e il *management service*, in cui il primo si occupa della trasmissione di *Protocol Data Unit* (PDU) con il livello fisico. Inoltre, il livello MAC, offre una serie di funzionalità per la gestione del canale radio, come ad esempio: la gestione dei *beacons*, l'accesso al canale, la validazione dei *frames*, l'*assosiacion* e la *disassociation* e implementa meccanismi di sicurezza.

Lo *standard* definisce inoltre per il sottolivello MAC una struttura detta *superframe* limitata da due *beacon*, che hanno lo scopo di sincronizzare i dispositivi, identificare la PAN e di descrivere la struttura stessa del *superframe*. Nelle reti conformi allo *standard* IEEE 802.15.4, il coordinatore della PAN può utlizzare o meno il *superframe*, in qual caso verrà anche disabilitata la trasmissione del *beacon*.

[3] *Industrial Scientific Medical.*

Di fatto i dispositivi IEEE 802.15.4 possono tramettere e ricevere messaggi in due modalità: *beacon-enable* e *nonbeacon-enabled*, cioè con la prensenza o meno dei *beacons* e del rispettivo *superframe*. Si possono poi distinguere tra casistiche di scambi di messaggi:

1. un dispositivo trasmette al coordinatore
2. il coordinatore trasmette al dispositivo
3. un dispositivo tramette ad un altro dispositivo

Nei casi presentati, i dispositivi possono appartenere a una PAN *beacon-enabled* o meno, di conseguenza si avranno delle modalità differenti di accesso al canale.

Quando un dispositivo che appartiene a una rete *beacon-enable* vuole trasmettere un messaggio al coordinatore, ascolterà il canale per cercare un *beacon* così da sincronizzarsi con il *superframe*. Dopodiché nella finestra temporale appropriata invierà il *Data Frame* al coordinatore. Nel caso invece di una rete *nonbeacon-enabled*, il dispositivo trasmetterà il *Data Frame* al coordinatore. Quando è il coordinatore che vuole trasmettere un messaggio in una rete *beacon-enabled*, indicherà nel *beacon*[4] che un messaggio è in attesa di trasmissione e il dispositivo, che lo ascolta periodicamente, invierà al coordinatore una *Data Request* per ricevere il messaggio. In una PAN *nonbeacon-enabled*, il coordinatore attenderà la ricezione di una *Data Request* da parte del dispositivo a cui deve mandare il messaggio. Infine, nel caso *peer-to-peer*, se i dispositivi sono sincronizzati, quando si acquisisce l'accesso al canale si trasmette il *Data Frame*.

Bisogna ora specificare come un dispositivo ottiene l'accesso al mezzo trasmissivo, cioè al canale radio. Lo *standard* IEEE 802.15.4 prevede l'utilizzo di 6 metodi di accesso, dei quali verranno descritti l'*Unslotted Carrier Sense Multiple Access - Collision Avoidance* (CSMA-CA) per reti *beacon-enabled* e lo *Slotted* CSMA-CA per reti *nonbeacon-enabled*.

Il CSMA-CA è una tecnica che viene utilizzata nelle reti *wireless*[5] per evitare che vi siano collisioni dei messaggi inviati dai dispositivi. Questa tecnica prevede una fase di ascolto del canale per controllare se è libero in modo tale da consentire al dispositivo di trasmettere, e in caso positivo aspetta un intervallo di tempo detto *Inter Frame Space* (IFS), controlla di nuovo il canale e poi trasmette. Nel caso di PAN *beacon-enabled* si utilizza lo *Slotted* CSMA-CA che prevede l'allineamento del'inizio del periodo di *backoff* del dispositivo con l'inizio del *beacon* trasmesso dal coordinatore della rete. Al contrario, nel caso *nonbeacon-enabled*, il periodo di *backoff* non è allineato con l'inizio del *beacon*.

[4] All'interno del campo *pending address* del *beacon* sono indicati gli indirizzi dei dispositivi che hanno un messaggio in attesa. Il numero massimo di indirizzi in attesa dovrebbe essere limitato a 7.

[5] Per il caso *wired*, *i.e.* lo *standard* IEEE 802.3 per *Local Area Network* (LAN), viene utilizzato il *Carrier Sense Multiple Access - Collision Detection* (CSMA-CD).

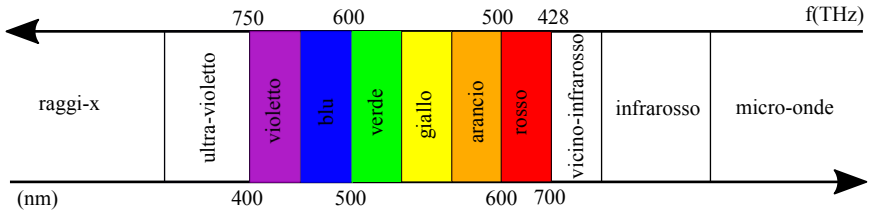

Figura 3.2: Spettro elettromagnetico.

3.2 Illuminare il livello fisico

In questa sezione verrà introdotto un altro *standard* che è stato studiato durante il periodo di dottorato per consentire la trasmissione di segnali *wireless* utilizzando come mezzo trasmissivo non più le onde elettromagnetiche ma la luce, cioè i fotoni. La VLC è la tecnologia che permette di utilizzare proprio la radiazione luminosa nello spettro del visibile (da 400 nm a 700 nm oppure da 430 THz a 700 THz, Fig. 3.2) per trasmettere i dati all'interno di un ambiente attraverso il sistema di illuminazione. Questo particolare mezzo trasmissivo *wireless* offre molteplici vantaggi quando impiegato per la trasmissione di dati. Ad esempio il segnale luminoso trasmesso è confinato all'interno dell'ambiente in cui è presente la sorgente luminosa, la luce non supera le pareti come accade per le onde elettromagnetiche, di conseguenza si ha una riduzione dell'interferenza con altri sistemi VLC presenti e, si viene a creare una sicurezza a livello fisico[6] che non è possibile ottenere con la tecnologia Wi-Fi. Inoltre se il sistema di trasmissione VLC coincide con il sistema di illuminazione si ha un notevole risparmio in termini di consumi energetici per la trasmissione delle informazioni. L'ultimo vantaggio da considerare è la possibilità di impiegare questa tecnologia nei luoghi dove non è consentita la trasmissione di onde radio, come ad esempio in alcuni reparti degli ospedali. Lo *standard* IEEE 802.15.7 [57] fornisce le specifiche tecniche per i livello fisico (PHY) e per l'accesso al canale (MAC) per le *Short-range Optical Wireless Communication* (OWC).

Durante il periodo di dottorato è stata valutata la possibilità di utilizzare i *Light Emitting Diode* (LED) siamo come sorgente luminosa che come ricevitori ottici, per realizzare la cosiddetta *LED-to-LED Communication*. Come è ben noto, questi dispositivi a semiconduttore sono progettati per la trasmissione luminosa, e quando sottoposti a una differenza di potenziale si viene a creare un corrente elettrica che porta gli elettroni dalla banda di conduzione alla banda di valenza. Se il salto tra queste due bande è sufficiente, l'elettrone rilascia energia sotto la forma di un fotone che viene emesso dal LED. Essendo il LED un dispositivo basato su semiconduttore è altre sì vero che è in grado, come tutti i semiconduttori, di assorbire l'energia di un fotone incidente e usarla per generare una coppia lacuna-elettrone. Se si considera

[6] Physical Layer Security.

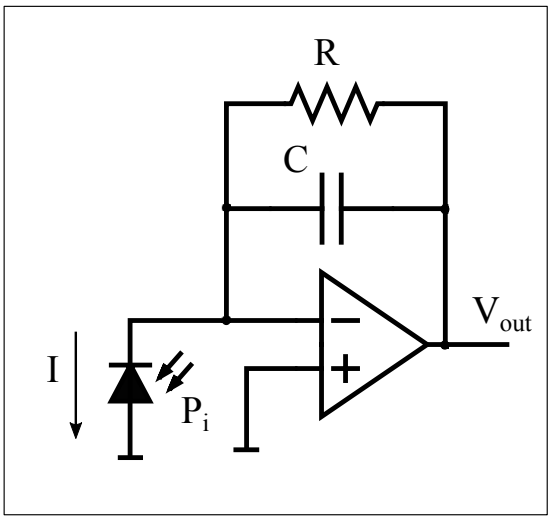

Figura 3.3: Amplificatore a transimpedenza, o *Transimpedance Amplifier* (TIA).

la potenza ottica incidente P_i si ottiene una foto-corrente $I = R \cdot P_i$, dove R è la *responsivity* del materiale.

Utilizzando un appropriato circuito si può andare a misurare la corrente elettrica generata dalla potenza ottica incidente, come mostrato dal circuito di conversione corrente-tensione di Fig. 3.3. In alternativa all'amplificatore mostrato in Fig. 3.3, che permette di ottenere una tensione V_{out} proporzionale alla potenza ottica incidente P_i, è possibile misurare il tempo che impiega la capacità di giunzione del LED di scaricarsi una volta caricata in *Reverse Bias*. Nella figura seguente (Fig. 3.4) vengono mostrate le due fasi necessarie a misurare questo tempo di scarica, dove nella prima fase viene caricata la capacità, e nella seconda fase viene scaricata a causa della potenza ottica incidente P_i. Le fasi di carica e scarica della capacità di giunzione del LED sono pilotate da un micro-controllore, che inizialmente carica il LED una tensione V e successivamente si riconfigura il *pin* del micro-controllore in modalità *input*, che fa scaricare la capacità sull'impedenza Z del micro-controllore stesso. Misurando il tempo che intercorre tra la tensione sulla capacità pari a $5V$ e alla sua scarica pari che tende a $0V$, è possibile terminare la presenza di un segnale ottico che incide sul LED. A seconda del valore di tensione V si modifica la dimensione della capacità di giunzione del LED [58] e di conseguenza il tempo necessario a scaricarla, e si modifica anche il guadagno di foto-corrente [59]. Questa particolare caratteristica del LED permette di utilizzare questo semplice dispositivo come *transceiver* per realizzare una comunicazione bidirezionale (LED-to-LED) per sistemi IoT.

In [38] è stata studiata la capacità del LED di ricevere la luce. Questi dispositivi sono caratterizzati da una particolare lunghezza d'onda λ (nm) di emissione di luce, la quale dipende dall'energia E persa dall'elettrone in banda di conduzione che si

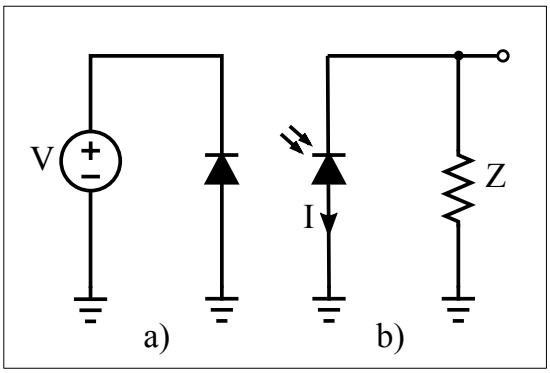

Figura 3.4: Fasi della misura sul LED: a) il LED viene polarizzato in inversa dalla tensione V e si carica la capacità di giunzione; b) si misura la durata della tensione sull'impedenza Z del micro-controllore che viene scaricata dalla corrente fotoelettrica i.

Modello	**Produttore**	**Colore**	λ
L-7104RSC-E	Kingbright	rosso	660 nm
WP710A10SGC	Kingbright	verde	557 nm
WP710A10VBC/D	Kingbright	blu	460 nm

Tabella 3.1: Caratteristiche dei tre LEDs utilizzati per le prove.

ricombina in banda di valenza con una lacuna. Questa energia si deve conservare, pertanto viene convertita in energia termica E_T e in un fotone di energia $E' = hf$, dove f corrisponde alla frequenza (o lunghezza d'onda) di emissione del fotone. Per ottenere questo risultato, il semiconduttore del LED viene drogato in modo tale da avere una banda proibita tale da generare un fotone del colore richiesto ($E' \geq E_g$). Per analizzare questo comportamento sono state valutate le capacità di ricezione di tre diversi LEDs (rosso, verde e blue) (Tab. 3.1) illuminati da diverse lunghezze d'onda (660 nm, 557 nm, 460 nm). Questo perchè quando il colore (lunghezza d'onda o frequenza) del LED ricevente non coincide con quello del LED trasmittente, si viene a generare una minore foto-corrente. Ciò dipende dai livelli energetici delle bande nel semiconduttore e dai colori (lunghezza d'onda o frequenza) dei fotoni considerati.

Nella Fig. 3.5 viene mostrata la ricezione da parte di un LED rosso (Tab. 3.1) di un segnale ottico trasmesso da un secondo LED rosso. Dalla figura si possono distinguere chiaramente due andamenti differenti, uno relativo alla scarica della capacità di giunzione quando il LED sta ricevendo il segnale, e l'altro relativo alla scarica quando è presente la sola luce ambientale, e andando a misurare la durata temporale delle scariche ($T_{Sr} \approx 6.5$ ms) è possibile determinare quando il LED sta ricevendo dei dati. Al contrario invece, nella Fig. 3.6 viene mostrato il diverso comportamento del LED

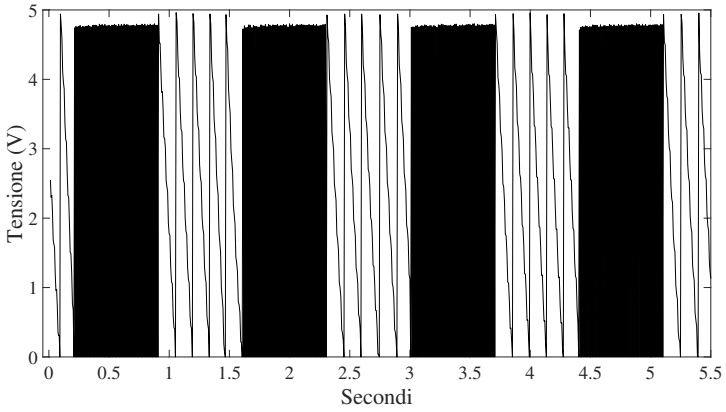

Figura 3.5: Andamento della carica e scarica della capacità di giunzione del LED durante una comunicazione LEDrosso - LEDrosso.

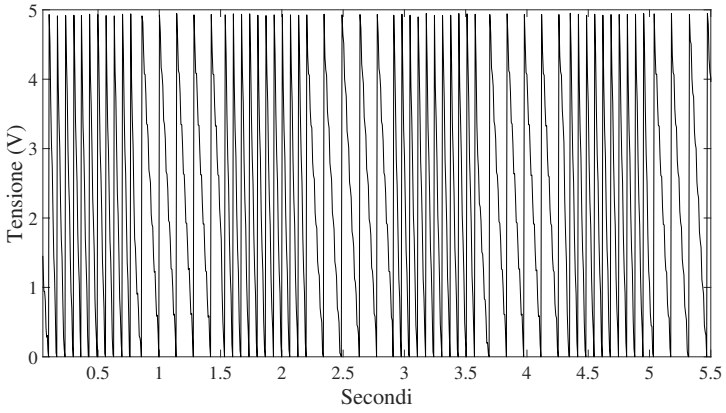

Figura 3.6: Andamento della carica e scarica della capacità di giunzione del LED durante una comunicazione LEDblu - LEDrosso.

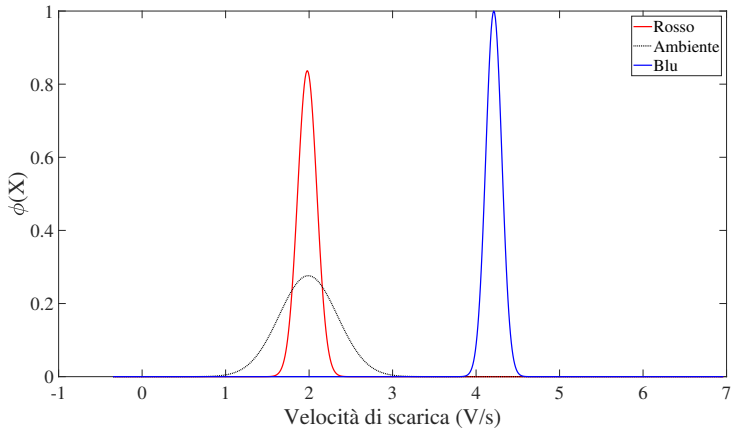

Figura 3.7: Distribuzioni delle velocità di scarica della capacità di giunzione del LED blu quando illuminato da un LED rosso, BLU, o dalla luce ambientale.

rosso quando riceve il segnale ottico inviato da un LED blu. I fotoni blu hanno una diversa lunghezza d'onda, o in altri termini una differente energia $E_B = hf_B \neq E_R$, di conseguenza si ha una minore probabilità che il fotone incidente generi una coppia elettrone-lacuna nel semiconduttore e la capacità impiegherà un tempo maggiore per scaricarsi ($T_{Sb} \approx 61$ ms). Quindi i LEDs oltre a essere in grado di rilevare la luce, possono essere utilizzati anche per distinguerne il colore come mostrato nelle Figg. 3.5 e 3.6. Per valutare questa capacità del LED di generare foto-correnti di valori diversi quando illuminato da fotoni di lunghezza d'onda differente, sono state misurate le medie e deviazioni *standard* dei tempi di scarica della capacità per le seguenti combinazioni di colori in trasmissione e ricezione: rosso, blu e ambientale come mostrato in Fig. 3.7. Da questi risultati ottenuti si è poi passati allo studio di una soluzione per consentire la trasmissione di segnali ottici con tecniche di modulazione più evolute come la *Color Shift Keying* (CSK) [60]. Per condurre questa analisi sono state ripetute le prove precedentemente descritte, ma con un LED RGB come ricevitore del segnale ottico al posto di un LED monocromatico. In Fig. 3.9 viene mostrato l'andamento della scarica di tensione delle tre capacità di giunzione sul LED RGB mentre riceve un segnale trasmesso da un LED rosso. Le tensioni relative ai canali verde e rosso, non risentono significativamente del segnale ottico (rosso) incidente, oltretutto il canale blu non è in grado di scaricarsi completamente sia quando illuminato dal LED rosso che dalla sola luce ambientale. Al contrario invece il canale rosso del LED RGB è in grado di rilevare la presenza della trasmissione scaricando più velocemente la tensione sulla giunzione.

I risultati ottenuti dalle differenti prove mettono in evidenza la possibilità di utilizzare il LED, pensato e progettato per l'illuminamento, come dispositivo per la trasmissione bidirezionale. E' stato dimostrato [38] che la scarica della capacità di giun-

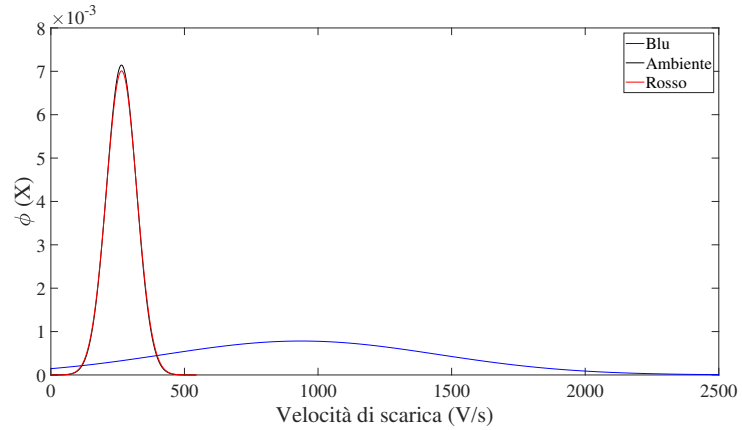

Figura 3.8: Distribuzioni delle velocità di scarica della capacità di giunzione del LED verde quando illuminato da un LED rosso, BLU, o dalla luce ambientale.

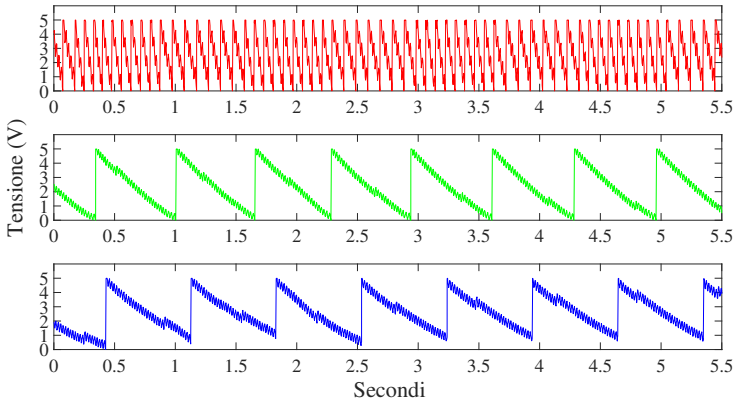

Figura 3.9: Andamento della carica e scarica delle tre capacità di giunzione del LED RGB durante una comunicazione LEDrosso - RGB.

zione del LED dipende dalla quantità di luce incidente e anche dal suo colore. In particolare questa abilità del LED può essere utilizzata per realizzare sistemi VLC.

3.3 Protocolli di rete

Nel modello dello *stack* protocollare ISO/OSI al di sopra del livello fisico e del livello collegamento è collocato il livello di rete, che si occupa dell'identificazione degli *host*[7] e dell'instradamento dei pacchetti[8] da un *host* all'altro scegliendo il percorso ottimale usando algoritmi di *routing*. In questa sezione verranno introdotti i protocolli di rete utilizzati per consentire lo scambio di pacchetti tra i nodi IoT e verso la rete globale.

3.3.1 IPv6

Il numero di dispositivi connessi alla rete globale sta crescendo continuamente, e l'Intel ha predetto che per il 2020 ci saranno più di 200 miliardi di dispositivi connessi alla rete [2]. Ognuno di questi dispositivi (*computer*, *smartphone*, *tablet*, *smart car*, nodi IoT, *etc.*.) avrà bisogno di un indirizzo IP per poter scambiare dati. La versione 4 del protocollo IP (IPv4) è quella attualmente utilizzata per la comunicazione con i servizi *web*, e le tecniche[9] fino ad ora utilizzate per estenderne la durata non saranno più sufficienti per l'enorme richieste di indirizzi che si avrà nei prossimi anni. Questa necessità di uno spazio di indirizzamento più ampio sarà garantita dalla nuova versione del protocollo IP, il protocollo IPv6, che estenderà lo spazio degli indirizza da 32 bit a 128 bit. Dallo spazio di indirizzamento IPv4 che mette a disposizione circa 4,3 miliardi di indirizzi ($4,3 \cdot 10^9$), si passa allo spazio IPv6 che fornisce circa "340 miliardi di miliardi di miliardi di miliardi" di indirizzi ($3,4 \cdot 10^{38}$), cioè si avranno $6.6 \cdot 10^{22}$ indirizzi per metro quadro di superficie terrestre. Oltre all'estensione dello spazio degli indirizzi, rispetto all'IPv4, il protocollo IPv6 semplifica l'*header* del pacchetto (Fig. 3.10), introduce l'utilizzo di *header* aggiuntivi con l'*Extension Header* e permette l'assegnazione di un *Flow Label* per gestire particolari flussi di traffico. La dimensione minima dell'*header* IPv6 è di 40 Byte a cui vanno aggiunti eventuali *Extension Headers* e il *payload*. Di fatto la dimensione del pacchetto IPv6 non è idonea alla trasmissione su reti *Low Power* basate sul protocollo IEEE 802.15.4, perché da specifiche questo pacchetto necessità di *links* in grado di supportare *Maximum Transmission Unit* (MTU) di 1280 Byte. Invece il protocollo IEEE 802.15.4 ha a disposizione 127 Byte a livello fisico a cui vanno sottratti 25 Byte massimi di *overhead* (Fig. 13.11), lasciando così per il *frame* solo 102 Byte. Per consentire l'incapsulamento di pacchetti IPv6 all'interno di *frame* IEEE 802.15.4 viene utilizzato un livello di adattamento definito dall'IPv6 *over Low-Power Wireless Personal Area Networks* (6Lo-

[7] Terminali.
[8] La PDU del livello di rete prende il nome di pacchetto.
[9] Si pensi al *subnetting* e al *natting*.

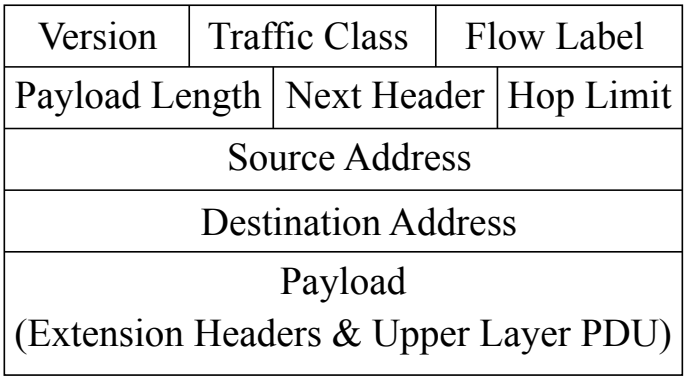

Figura 3.10: Formato del pacchetto IPv6.

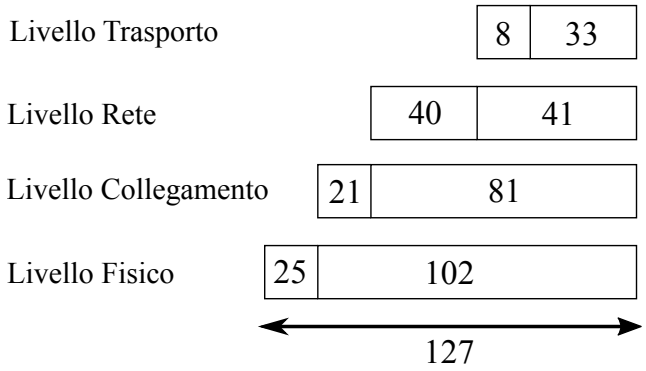

Figura 3.11: Dimensioni in ottetti delle PDUs per i primi quattro livelli dello *stack* ISO/OSI per lo *standard* IEEE 802.15.4. Vengono considerati i massimi *overhead* per ogni livello protocollare.

WPAN) [61, 62, 63], per consentire di inviare pacchetti IPv6 su *links* con vincoli di trasmissione così stringenti.

3.3.2 IPv6 per *Low-Power Wireless Personal Area Networks*

Lo *standard* 6LoWPAN [61, 62, 63] definisce le regole per il livello *Data Link*[10] per la trasmissione di pacchetti IPv6 su reti *low power* IEEE 802.15.4 specificando come devono essere realizzati:

- il formato del *frame*
- il formato dell'indirizzo IPv6
- l'*header compression*
- la frammentazione dei *frame*
- le regole per l'inoltro di pacchetti in reti *mesh*

Il *frame* 6LoWPAN è quindi realizzato dalla composizione di uno o più dei seguenti *headers*: *mesh*, *broadcast* e *fragment*, in questo ordine specifico. Il primo *header* viene utilizzato per indicare informazioni utili per l'inoltro di pacchetti all'interno di una rete *mesh* nello *standard* IEEE 802.15.4. Il secondo *header* viene utilizzato per la trasmissione di messaggi *broadcast*[11], infine il terzo *header* viene usato per la gestione della frammentazione dei pacchetti IPv6 all'interno di una rete 6LoWPAN.

Dato che un pacchetto IPv6 richiede una MTU di 1280 Byte, mentre, per un *frame* IEEE 802.15.4 sono disponibili solo 81 Byte (Fig. 3.11). E' quindi indispensabile utilizzare una tecnica di compressione per riuscire a trasmettere i pacchetti IPv6 all'interno delle reti *low power*. Nel caso specifico delle reti 6LoWPAN si hanno delle informazioni note e condivise tra i dispositivi che appartengono a queste reti, e in questo modo non sarà necessario l'utilizzo di una vera e propria tecnica di compressione, ma piuttosto di rimozione di informazioni ridondanti. Si pensi infatti alla struttura del pacchetto IPv6 rappresentato in Fig. 3.10, è triviale che la versione del pacchetto sarà sempre la stessa (6), oppure che il *Payload Length* può essere determinato dal *Frame Length* del messaggio IEEE 802.15.4. Operando in questo modo si costruisce un nuovo *header* che prende il nome di *Header Compression 1* (HC1), e avrà una dimensione di 2 ottetti contro i 40 ottetti dell'*header* IPv6. I due ottetti dell'HC1 sono dati dalla concatenazione dell'HC1 *Encoding* (1 Byte), cioè le informazioni compresse dell'*header* IPv6 e del campo *Hop Limit* (1 Byte) che non è possibile comprimere.

[10] Livello collegamento dello *stack* ISO/OSI

[11] Si vuole ricordare che il protocollo IPv6 non permette l'invio di pacchetti *broadcast*, permesso invece con l'IPv4, ma consente solamente il *multicast*. Qui il *broadcast* appartiene al livello collegamento e viene usato per inviare pacchetti IPv6 *multicast*, poichè lo standard IEEE 802.15.4 non prevede messaggi *multicast*.

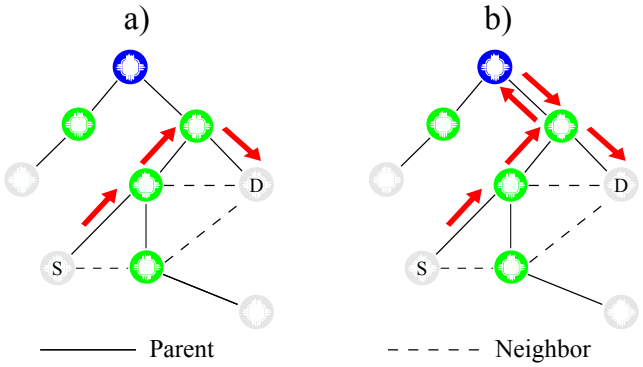

Figura 3.12: Sulla sinistra (a) è raffigurato un DODAG in modalità *non-storing*, mentre sulla destra (b) un DODAG in modalità *storing*. I *link* a tratto continuo indicano che il nodo di *rank* superiore è un *parent*, mentre il tratto discontinuo indica che i nodi sono in rapporto di vicinato (*neighbor*).

3.3.3 Protocollo di *Routing*

Le reti 6LoWPAN hanno la caratteristica di essere auto-configuranti e di essere resistenti ai *failure* dei singoli nodi. Questa capacità è dovuta all'utilizzo di protocolli di *routing*[12] che calcolano i percorsi ottimali per raggiungere il *gateway* della rete. Utilizzando i protocolli di *routing*, al primo avvio della rete, vengono subito generate le rotte per ogni singolo nodo per poter raggiungere il coordinatore della PAN (auto-configurazione), o in caso di *failure* di uno o più nodi vengono ricalcolate le rotte per mantenere l'operatività della rete. Per studiare e sviluppare protocolli di *routing* adatti all'utilizzo in reti *Low power and Lossy Network* (LLN), l'*Internet Engineering Task Force* (IETF) ha istituito un *Working Group* (WC) denominato ROLL[13], e tra i suoi lavori il *Routing Protocol for Low power and Lossy Network* (RPL) [64] è stato utilizzato per controllare lo stato della rete nelle varie installazioni realizzate durante questo progetto di ricerca.

L'RPL si basa sulla costruzione di grafi aciclici diretti (*Direct Acyclic Graph* (DAG)), e realizza topologie di rete come *Directed Oriented Direct Acyclic Graph* (DODAG), cioè un grafi senza cicli con percorsi che portano ad un nodo radice (*root*), *i.e. sink node*, *border router*. Il protocollo RPL fornisce ai nodi della rete le rotte verso il nodo *root*, che vengono calcolate con delle apposite funzioni dette *Objective Function* (OF). Queste funzioni indicano quali sono le metriche utilizzate all'interno dell'istanza del DODAG, e sono utilizzate per calcolare il *rank* del nodo, e come il nodo sceglie il proprio *parent*. Queste informazioni vengono inviate ai nodi tramite appositi messaggi chiamati DODAG *Information Object* (DIO), che hanno lo scopo di permettere ai nodi di determinare le rotte in *upwards* verso il nodo *root*. Invece i messaggi *De-*

[12] Protocolli di instradamento di pacchetti all'interno di reti IP.
[13] *Routing Over Low power and Lossy networks.*

stination Advertisement Objects (DAO) vengono utilizzati all'interno del DODAG per determinare le rotte in *downward*. Sono previste due modalità di traffico in *downward* dal protocollo RPL, la modalità *storing* e la modalità *non-storing*, come mostrato nella (Fig. 3.12). L'ultimo tipo di messaggio inviato dall'RPL è il *DODAG Information Solecitation* (DIS), che è di fatto l'analogo del *Router Solicitation* del protocollo IPv6.

Questi tipi di messaggi, scambiati in modalità *link-local unicast* o *multicast*[14], sono tutti incapsulati all'interno degli ICMPv6 *Control Message*, a cui viene assegnato il valore 155 nel campo *Type* dell'*header* ICMPv6, mentre il campo *Code* indica il tipo di messaggio di controllo. L'RFC[15] 6550 [64] definisce quali codici sono associati ai rispettivi messaggi di controllo dell'RPL:

- 0x00: DODAG Information Solicitation
- 0x01: DODAG Information Object
- 0x02: Destination Advertisement Object
- 0x03: Destination Advertisement Object Acknowledgment
- 0x80: Secure DODAG Information Solicitation
- 0x81: Secure DODAG Information Object
- 0x82: Secure Destination Advertisement Object
- 0x83: Secure Destination Advertisement Object Acknowledgment
- 0x8A: Consistency Check

Ai fini della realizzazione della rete IoT per questo progetto di ricerca, è stata scelta come funzione per la selezione delle metriche e della selezione dei *parents* dei nodi, la *Objective Function Zero* (OF0) [65]. Questa funzione viene utilizzata per cercare il più vicino nodo *root grounded*[16], e se non è possibile trovarlo, viene create un DODAG *floating*[17]. Per ogni nodo del DODAG, la OF0 seleziona un *parent* preferito e un *parent* di *backup* sulla base del *rank* dei nodi vicini al fine di raggiungere il nodo *root* con la metrica migliore.

3.4 Protocolli livello applicazione

In questa sezione si andrà a descrivere l'ultimo livello dello *stack* protocollare ISO/OSI, facendo riferimento ai protocolli che appartengono a questo specifico livello e utilizzati durante questo progetto di ricerca. Questi protocolli sono implementati

[14] L'indirizzo *multicast* (*all-RPL-nodes*) è l'indirizzo ff02::1a.
[15] *Request For Comment.*
[16] Si tratta di un nodo *root* che offre connettività verso altre reti al fine di soddisfare le necessità dell'applicazione.
[17] Si tratta di una rete RPL che consente lo scambio di messaggi solo al suo interno.

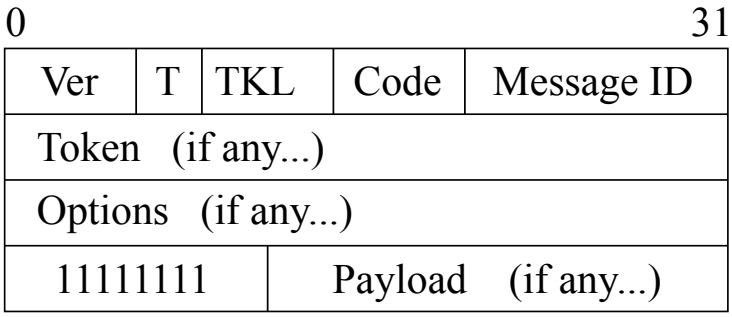

Figura 3.13: Formato del pacchetto CoAP.

da applicazioni, generalmente usate dall'utente, per scambiare dati con altri terminali della rete su un modello di comunicazione *client/server*.

3.4.1 Constrained Application Protocol

Il *Constrained Application Protocol* (CoAP) [66, 67, 68] è un protocollo basato sul modello architetturale *client/server*, ed è particolarmente adatto per essere utilizzato su dispositivi *constrained*, ovvero su dispositivi con poca memoria, poche risorse di calcolo, a banda limitata e generalmente alimentati a batteria. Si tratta di un protocollo pensato per permettere a questi dispositivi che compongono i sistemi IoT di poter avere a disposizione un'architettura *REST*[18] in ambiente *web* per lo scambio di dati, sia per ricevere richieste dagli utenti sia per instaurare comunicazioni M2M. Utilizzando il CoAP è possibile scambiare messaggi tra un client e un server in modo asincrono, incapsulando le richieste e le risposte all'interno di un datagramma UDP[19]. Il primo campo del pacchetto *CoAP* (Fig. 3.13 descrive la versione (VER) del formato del messaggio. E' composto da 2 *bit* e attualmente l'unico valore ammesso è $1_{10} = 01_2$, altri valori sono destinati per usi futuri. Il secondo campo (T) composto da 2 *bit* indica il tipo di messaggio che viene trasmesso. I tipi di messaggi che possono essere scambiati dal protocollo CoAP sono quattro:

0. Confirmable
1. Non-confirmable
2. Acknowledgement
3. Reset

[18] *REpresentational State Transfer* (REST): modello architetturale per il *World Wide Web* (WWW) definito da Fielding in [69]. In questo documento sono specificati i principi sui quali si dovrebbe basare un sistema con architettura *REST*, ovvero: modello *client-server*, interazioni *stateless*, funzionalità di *caching*, interfaccia uniforme, sistema a *layers*, *Code-On-Demand*. Un tale sistema che implementa tutti questi principi viene detto *RESTful*.

[19] *User Datagram Protocol* (UDP): protocollo di livello trasporto della *suite internet* di tipo *connectionless*.

c_{10}	c_2	Significato
0	000	*Request*
1	001	*Reserved*
2	010	*Success response*
3	011	*Reserved*
4	100	*Client error response*
5	101	*Server error response*
6	110	*Reserved*
7	111	*Reserved*

Tabella 3.2: Bit di classe (c) del campo *Code* nel pacchetto CoAP.

$Code_{10}$	$Code_2$	Significato
0.01	00000000	*GET*
0.02	00000001	*POST*
0.03	00000010	*PUT*
0.04	00000011	*DELETE*

Tabella 3.3: Codici dei metodi *CoAP* nel formato decimale c.dd e binario.

Un messaggio di tipo *Confirmable* richede da parte del ricevitore l'invio obbligatorio di una risposta, cioè di un *ACK*. Il protocollo *CoAP* permette di incapsulare la risposta direttamente nel messaggio di *ACK*, andando a definire così una *Piggybacked Response*. In questo modo non servono ulteriori messaggi, oltre all'*ACK*, per ottenere la risorsa richiesta dal *client*. Tuttavia non è possibile in tutti i casi inviare la risorsa tramite una *Piggybacked Response*, perché ad esempio, il *server* potrebbe impiegare molto ad ottenere tale risorsa. Per evitare che il *client* invii ulteriori richieste, il server può inviare immediatamente un *ACK* e successivamente la risorsa tramite un messaggio *confirmable*. Questa soluzione permette ai dispositivi *constrained* di ridurre il numero di messaggi trasmessi e di conseguenza di ridurre l'occupazione del canale radio, e il consumo energetico del *transceiver*. Il terzo campo (TKL) di 4 *bit* indica la dimensione del campo *Token*. Con il quarto campo (*Code*) (Tab. 3.2) del pacchetto CoAP (Fig. 3.13) si possono definere messaggi di *request* o di *response*. Gli 8 *bit* che costituiscono il campo *Code* vengono divisi in due gruppi, e vengono indicati rispettivamente con tre cifre decimali nella seguente forma: *c.dd*. Il primo gruppo è detto "classe" (c) ed è rappresentato dai primi 3 *bit*, mentre il secondo gruppo composto dagli ultimi 5 *bit* è chiamato "dettagli" (dd). Impostando correttamente i *bit* del campo *Code* si può definire un metodo (Tab. 3.2) per far interagire il *client* con la risorsa contenuta all'interno del *server CoAP*: Il quinto, ed ultimo campo (*Message ID*) dell'*header* del pacchetto CoAP, è una sequenza di 16 *bit* utilizzata per individuare eventuali messaggi duplicati, o per associare messaggi di *ACK/Reset* al rispettivo messaggio *Confirmable/Non-confirmable*. A seguire l'*header* ci sono il *Token* e le *Options*,

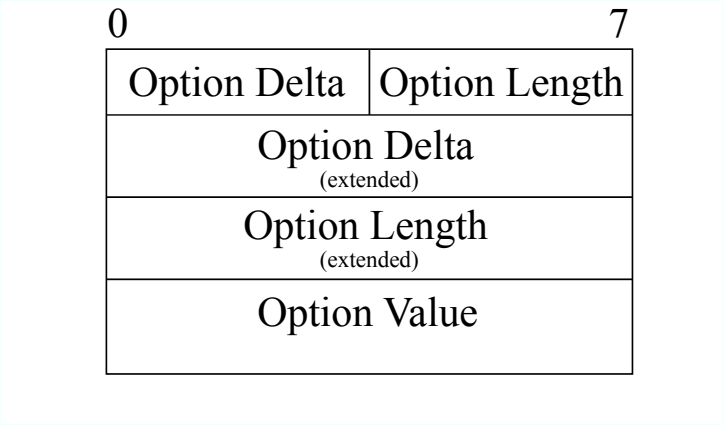

Figura 3.14: Formato delle *Options* nel pacchetto CoAP.

e se presenti, ci sarà anche un *Byte* per indicare la presenza del *Payload* opzionale. Il *Token* è composto da 0 a 8 *Byte* e viene utilizzato per associare la *request* alla rispettiva *response*. Infine l'*option* prende la forma mostrata in Fig. 3.14. Le *Options* sono distinte dall'*Option Number*, che viene calcolato nel seguente modo:

$$Option\_Number_i = Option\_Number_{i-1} + Option\_Delta \tag{3.1}$$

dove per la prima *Option* ($i = 1$) del pacchetto CoAP si considera:

$$Option\_Number_{i-1} = 0 \tag{3.2}$$

Nel caso di istanze multiple della stessa *Option* si utilizza $Option\_Delta = 0$. All'interno dell'*Option Value* è presente il *payload* della *Option*, *i.e.* il percorso della risorsa all'interno del *server* CoAP. L'elenco completo delle *Options* è descritto dettagliatamente in [70].

Questo protocollo prevede inoltre una particolare *Option* per estendere la funzionalità del metodo *GET*. L'*Option* indicata con l'$Option\_Number = 6$, è l'opzione di *Observe* [71] che viene inviata tramite una *GET CoAP* per ottenere una rappresentazione della risorsa e richiedere inoltre gli *updates* della risorsa stessa. Questa funzionalità si basa sul modello descritto in [72], in cui un *observer* si registra su una specifica risorsa detta *subject*, di cui è interessato a ricevere notifiche del cambio di stato. Fino a che l'*observer* rimane registrato sulla risorsa, continuerà a ricevere notifiche. L'*observer* può cancellare la registrazione sulla risorsa in due modi, può dimenticarsi della risorsa osservata oppure cancellare la registrazione. Nel primo caso, alla ricezione di una osservazione con un *Token* non riconosciuto, l'*observer* (*client*) invierà un messaggio di *Reset*. Risulta invece più efficiente cancellare la registrazione con un messaggio di *deregister*, rilasciando così le risorse sul *server*. Questa funzionalità

del protocollo CoAP risulta essere particolarmente adatta per ricevere aggiornamenti in tempo reale dai dispositivi IoT muniti di sensori per il monitoraggio.

3.4.2 Implementazione in Python

Un problema del protocollo CoAP è la difficoltà nel suo utilizzo lato *client*. I *browser* per la navigazione *web*, utilizzati per inviare e ricevere messaggi *HTTP*, sono basati sul protocollo TCP[20] e non sono in grado di comunicare utilizzando il protocollo UDP. Una soluzione per ovviare a questo problema è data dal *plug-in Copper* (Cu) [73] per il *browser* Firefox. Questo *plug-in* permette al *browser* di creare un *socket*[21] UDP per consentire la comunicazione con dispositivi CoAP, e fornisce un'interfaccia per inviare le *request* e visualizzare il contenuto delle *response* direttamente sul *browser*.

Per poter accedere ai dati ricevuti dai messaggi CoAP al fine di salvarli su un *database* e di visualizzarli in *real-time* su grafici, è stata realizzata un'applicazione in Python per la trasmissione e ricezione dei pacchetti CoAP su *socket* UDP. L'applicazione è composta da due classi, *coappacket* e *coapclient*, che si occupano rispettivamente della gestione del pacchetto (creazione e lettura) e della gestione del *socket* UDP per la trasmissione e ricezione dei messaggi. Nella Fig. 3.15 viene mostrato il diagramma UML[22] che descrive le relazioni tra le due classi utilizzate per realizzare il *client* CoAP. Una singola istanza della classe *CoapClient* (*m_client* in Fig. 3.15) viene utilizzata per trasmettere e ricevere i pacchetti CoAP, definiti dalla classe *CoapPacket*, utilizzando i metodi *send()* e *handle_response()*.

3.4.3 WebSocket

Si tratta di un protocollo [74] incapsulato all'interno del TCP che viene utilizzato per consentire alle applicazioni *web* di poter avere a disposizione un canale di comunicazione bidirezionale, tra *client* e *server*, senza dover aprire connessioni HTTP multiple. Il WebSocket si propone di risolvere il problema del *polling*[23] HTTP [75], creando proprio una connessione TCP bidirezionale.

La connessione tra il *client* e il *server* viene stabilita a seguito di un *handshake* che viene effettuato a livello applicazione dal protocollo HTTP. Il *client* inizia la fase di *handshake* inviando una richiesta di HTTP Upgrade all'*host* remoto, inserendo nell'*header* HTTP le informazioni necessarie per aprire un websocket come mostrato in Fig. 3.16. Una volta che la fase di *handshake* viene conclusa con successo, cioè se

[20] *Transmission Control Protocol*: protocollo di livello trasporto della *suite internet* di tipo *connection oriented*.

[21] Il *socket* è un'interfaccia definita dalla coppia (*ip_address*, *port_number*) per consentire la comunicazione tra due terminali su una rete IP, oppure, per consentire la comunicazione tra due processi sulla stessa macchina passando per le interfacce di *loopback*.

[22] *Unified Model Language*

[23] Il protocollo HTTP prevede che sia il *client* a iniziare la connessione verso il *server* per inviare richieste, e non il contrario. Di conseguenza il *server* non è in grado di trasmettere al *client* eventi asincroni, e il *client* è costretto a interrogare ripetutamente il *server* per ricevere al più presto gli eventi

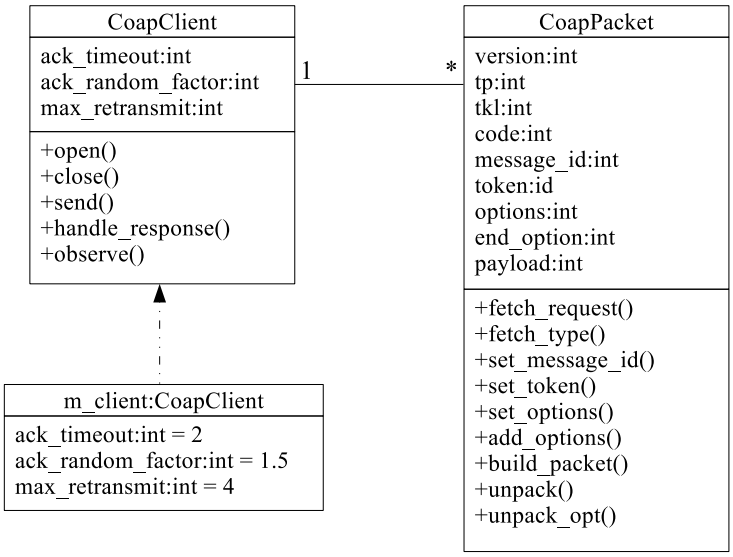

Figura 3.15: Diagrammi delle classi utilizzate per realizzare il *client* CoAP in Python.

HTTP Upgrade Request	**HTTP Response**
GET /chat HTTP/1.1	HTTP/1.1 101 Switching Protocols
Host: <remote_address>	Upgrade: websocket
Upgrade: websocket	Connection: Upgrade
Connection: Upgrade	Sec-WebSocket-Accept: hash(<key>)
Sec-WebSocket-Key: <key>	Sec-WebSocket-Protocol: chat
Sec-WebSocket-Protocol: chat, superchat	
Sec-WebSocket-Version: 13	

Figura 3.16: Messaggi HTTP scambiati tra il *client* (sinistra) e il *server* (destra) durante l'*handshake* per l'instaurazione della connessione WebSocket.

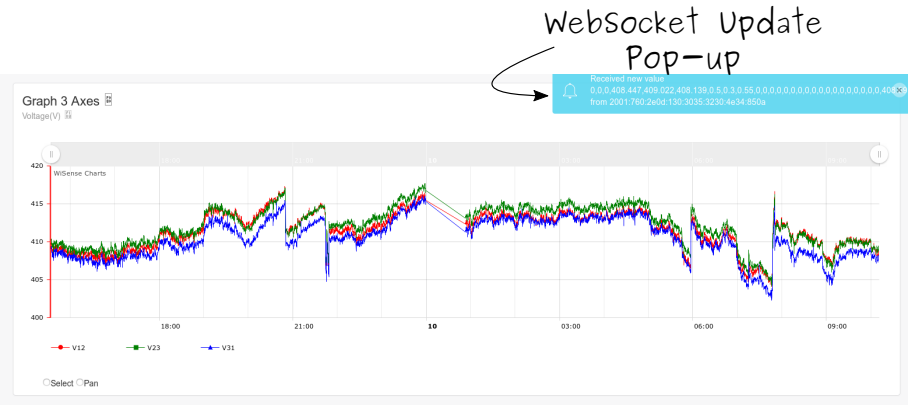

Figura 3.17: *Screenshot* del *pop-up* di un messaggio ricevuto tramite il protocollo WebSocket all'interno dell'applicazione *web* per il monitoraggio della tensione trifase di una macchina industriale.

l'*host* remoto risponde inserendo nell'*header* HTTP la *Web-Socket-Accept* con il *digest*[24] della chiave inviata dal *client* calcolata con l'algoritmo SHA-1, allora i due *hosts* possono iniziare a scambiarsi messaggi. Una volta terminato lo scambio di messaggi, *client* e *server* possono chiudere la connessione websocket utilizzando il *closing handshake* inviandosi messaggi con *opcode* 0x8. Una volte che sono stati ricevuti i messaggi di *Close*, gli *hosts* possono procedere con la chiusura della connessione TCP sottostante.

Nella Fig. 3.17 viene mostrato l'utilizzo del WebSocket sull'applicazione *web* utilizzata per mostrare in *real-time* la tensione della linea trifase che alimenta una pompa idraulica per prove di trazione. Nell'angolo in alto a destra della figura, viene mostrata all'utente la ricezione di un messaggio tramite il protocollo WebSocket, e allo stesso tempo la nuova misura viene concatenata al grafico.

3.5 Contiki OS

Contiki OS è un sistema operativo *open-source* progettato per dispositivi *embedded* con microcontrollori a limitata memoria (2 kB RAM e 40 kB ROM rappresentano una configurazione tipica) e per applicazioni basate su WSN. Contiki OS è stato sviluppato da Adam Dunkels *et. al.* e presentato in [76], dove è stato definito come un sistema operativo leggero ad alta portabilità basato su un sistema ibrido ad eventi con *multi-threading*. In particolare questo sistema operativo per dispositivi *constrained* fornisce la libreria necessaria per consentire la comunicazione sui protocolli IPv4 e

[24]Il *digest* è il risultato dell'algoritmo di *hashing*. Si tratta di algoritmi basati su operazioni non invertibili che servono per "mescolare" la stringa di *input*.

IPv6, cioè si tratta dell'implementazione dello *stack* TCP/IP con la libreria μIP [77]. Inoltre fornisce lo *stack* Rime [78] per ridurre i consumi delle comunicazioni radio.

All'interno di questo sistema operativo vanno definiti i processi che saranno utilizzati per eseguire le operazioni necessarie per la raccolta delle misure dai sensori, l'attivazione di eventuali attuatori, e altre operazioni che possono essere utili per il monitoraggio ambientale. Contiki OS permette l'utilizzo di processi attraverso una libreria di *threading* chiamata Protothreads [79].

```
1 #include "contiki.h"
2
3 PROCESS(test_proc, "Processo di Prova");
4 AUTOSTART_PROCESSES(&test_proc);
```

Codice 3.1: Definizione di un processo in Contiki OS.

Nelle righe di codice sopra riportate, viene mostrato come si definisce un processo con all'interno del sistema operativo Contiki OS con la macro PROCESS(), passandogli la variabile che indica il processo e la stringa che ne definisce il nome (riga 3). Successivamente viene utilizzata la macro AUTOSTART_PROCESS() (riga 4) per indicare al sistema operativo di avviare al termine del *boot* il processo passatogli come argomento. Ora occorre definire le operazioni che devono essere svolte dal processo "Processo di Prova", e per fare ciò si utilizza la macro PROCESS_THREAD(), come mostrato nelle seguenti righe.

```
1 PROCESS_THREAD(test_proc, ev, data){
2     PROCESS_BEGIN();
3     ...
4     istruzioni da eseguire
5     ...
6     PROCESS_END();
```

Codice 3.2: Struttura di un processo in Contiki OS.

Nella prima riga viene identificato il processo, e gli argomenti successivi (ev, data), sono contenitori per valori passati da eventuali eventi che attivano il processo. La macro PROCESS_BEGIN() indica dove il processo ha inizio, e a seguire si inseriscono le istruzioni che devono essere eseguite dal processo. Al termine delle operazioni che si vogliono far eseguire, si chiude il processo con PROCESS_END(). Il processo mostrato nel codice precedente, viene schedulato in modo cooperativo, cioè è il processo stesso che deve restituire il controllo della CPU al sistema operativo, in modo tale da consentire ad altri processi di essere eseguiti. Il programmatore deve quindi evitare di creare processi troppo lunghi che possono andare ad occupare le risorse limitate del dispositivo. Al fine di evitare problemi di occupazione eccessiva delle risosrse, si può fare ricorso ad una serie di eventi messi a disposizione dal sistema operativo che consentono l'interruzione del codice e la sua successiva ripresa. Riprendendo l'esempio precedente, si può inserire un evento basato su un *timer*:

```
1  PROCESS_THREAD(test_proc, ev, data){
2      static struct etimer et;
3
4      PROCESS_BEGIN();
5      etimer_set(&et, 1000);
6      while(1){
7          PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
8          ...
9          istruzioni da eseguire
10         ...
11         etimer_reset(&et);
12     }
13
14     PROCESS_END();
```

Codice 3.3: Utilizzo dei *timer* per schedulare i processi di Contiki OS.

Il codice riportato mostra come viene definito (riga 2) e inizializzato (riga 5) il *timer* "et", utilizzato per interrompere il processo e restituire il controllo allo *scheduler* del sistema operativo. Allo scadere del *time* "et" il processo riprendà la sua esecuzione, eseguirà le istruzioni programmate e infine resetterà (riga 11) il *timer* al suo valore originale di 1 secondo. Utilizzando questo metodo si possono andare a eseguire sullo stesso dispositivo più processi concorrenti che vengono interrotti e riavviati da eventi messi a disposizione da Contiki OS.

Oltre agli eventi, sono previste altre due modalità per interrompere un processo e restituire così il controllo allo *scheduler* del sistema operativo, ovvero le macro PROCESS_PAUSE() e PROCESS_YIELD().

```
1  PROCESS_THREAD(test_proc, ev, data){
2      PROCESS_BEGIN();
3      ...
4      istruzioni da eseguire
5      ...
6      PROCESS_PAUSE();
7      ...
8      istruzioni da eseguire
9      ...
10     PROCESS_END();
```

Codice 3.4: Controllo dei processi di Contiki OS con le pause.

Infatti utilizzando la pausa fornita dalla macro di riga 6, il processo restituisce volontariamente il controllo allo *scheduler*, che assegnerà le risorse del dispositivo ad altri eventi in coda, per poi restituire il controllo al processo in pausa che terminerà il suo lavoro. Al contrario, con l'utilizzo della macro PROCESS_YIELD(), lo *scheduler* non restituirà il controllo al processo ma aspetterà che un altro processo in corso invochi una process_poll() passando come argomento il puntatore al processo in attesa (*i.e.* process_poll(&test_proc);). Per completezza si riportano gli eventi messi a disposizione dal sistema operativo:

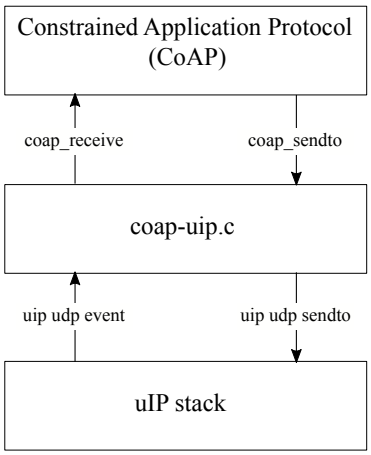

Figura 3.18: Implementazione del protocollo CoAP all'interno del sistema operativo Contiki OS.

- PROCESS_EVENT_NONE
- PROCESS_EVENT_INIT
- PROCESS_EVENT_POLL
- PROCESS_EVENT_EXIT
- PROCESS_EVENT_SERVICE_REMOVED
- PROCESS_EVENT_CONTINUE
- PROCESS_EVENT_MSG
- PROCESS_EVENT_EXITED
- PROCESS_EVENT_TIMER
- PROCESS_EVENT_COM
- PROCESS_EVENT_MAX

Utilizzando i processi di Contiki OS sono state implementate sui dispositivi IoT le operazioni necessarie per raccogliere i dati dai sensori connessi e per trasmettere le informazioni al *server* che si occupa della loro memorizzazione e fruizione.

3.5.1 CoAP *Engine*

Contiki OS fornisce anche un'implementazione del *server* CoAP attraverso il CoAP *Engine* basato sull'implementazione Erbium di Mattias Kovatsch [80]. Il CoAP *Engine* (Fig. 3.18) di Contiki OS è un modulo che si occupa della gestione delle richieste

CoAP e delle risorse CoAP che il *server* mette a disposizione. Le componenti principali dell'implementazione CoAP in Contiki OS sono: CoAP *Engine*, CoAP *Handler*, CoAP *Endpoint*, CoAP *Transport*, CoAP *Messages* e il CoAP *Timer*; e questi moduli si occupano della gestione del *server*, delle risorse disponibili nel dispositivo, della gestione dei messaggi e del (de/in)capsulamento all'interno del protocollo di trasporto UDP.

Con il seguente blocco di istruzioni si va attivare il CoAP *Engine* e a definire alcune risorse all'interno del sistema.

```
1  #include "contiki.h"
2  #include "rest-engine.h"
3
4  extern resource_t res_energy;
5
6  PROCESS(er_example_server, "CoAP Example Server");
7  AUTOSTART_PROCESSES(&er_example_server);
8  PROCESS_THREAD(er_example_server, ev, data){
9    PROCESS_BEGIN();
10     rest_init_engine();
11     rest_activate_resource(&res_energy, "test/energy");
12
13     PROCESS_END();
14 }
```

Codice 3.5: Inizializzazione del *server* CoAP in Contiki OS.

Nella riga 2 si importa il modulo che gestisce il CoAP *Engine* e a seguire, nella riga 10, questo modulo viene avviato all'interno di un processo eseguito al termine del *boot* del dispositivo IoT. Dopo di che si possono andare ad attivare le risorse CoAP che il sistema vuole rendere disponibili per gli utenti remoti, e per fare questo, nella riga 11 viene utlizzata la funzione *rest_activate_resource(coap_resource_t *resource, const char *path)* che prende come argomenti il puntatore alla risorsa e l'URI che le si vuole associare.

Bisogna ora definire come deve essere fatta la risposta CoAP della risorsa attivata. Per fare ciò, si vanno a definire la risorsa e il suo *handler* all'interno di un *file* C dedicato. Nel seguente esempio viene mostrato come si definisce la risorsa CoAP e come si realizza il suo *handler*:

```
#include "rest-engine.h"

static void res_get_handler(void *request, void *response,
    uint8_t *buffer, uint16_t preferred_size, int32_t *offset);

RESOURCE(res_energy, "title=\"Energy\";rt=\"Text\"",
    res_get_handler, NULL, NULL, NULL);

static void res_get_handler(void *request, void *response,
    uint8_t *buffer, uint16_t preferred_size, int32_t *offset){
    ...
```

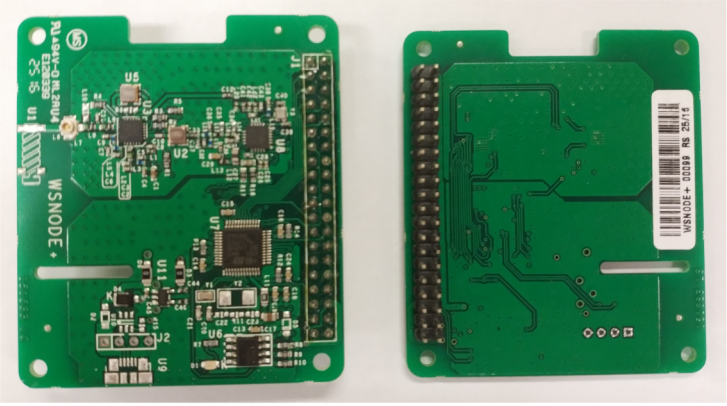

Figura 3.19: Dispositivo IoT utilizzato per realizzare la rete di sensori nel progetto di ricerca.

```
9       istruzioni da eseguire
10      ...
11      REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
12      REST.set_header_etag(response, (uint8_t *)&length, 1);
13      REST.set_response_payload(response, buffer, length);
14  }
```

Codice 3.6: Definizione della risorsa CoAP e del suo *handler*.

All'interno di questo *file* viene dichiarata la funzione di *handler* che si occuperà della ricezione del messaggio CoAP e della creazione della rispettiva risposta (riga 3). Nella riga 5 viene associata alla risorsa la rispettiva funzione di *handler*, e viene anche inserita una stringa che verrà visualizzata dai *clients* che inviano una richiesta di *disovery* a questo *server* CoAP. Infine si ha la definizione della funzione di *handler* (righe 7-14), al cui interno saranno presenti le istruzioni che si occupano della creazione della *response*, e a seguire della le istruzioni che preparano il messaggio CoAP che sarà inviato al *client*.

3.6 Il dispositivo IoT

Il dispositivo Ipv6 utilizzato per realizzare molti dei progetti IoT portati a termine durante il periodo di dottorato è mostrato in Fig. 3.19. Si tratta di un nodo *constrained*, di nome WSNODE+, progettato dal nostro gruppo di ricerca [81] per realizzare reti di sensori *wireless* per applicazioni ad ampio spettro. Questo dispositivo è composto principalmente da:

- microcontrollore STM32L
- *transceiver* SPIRIT1

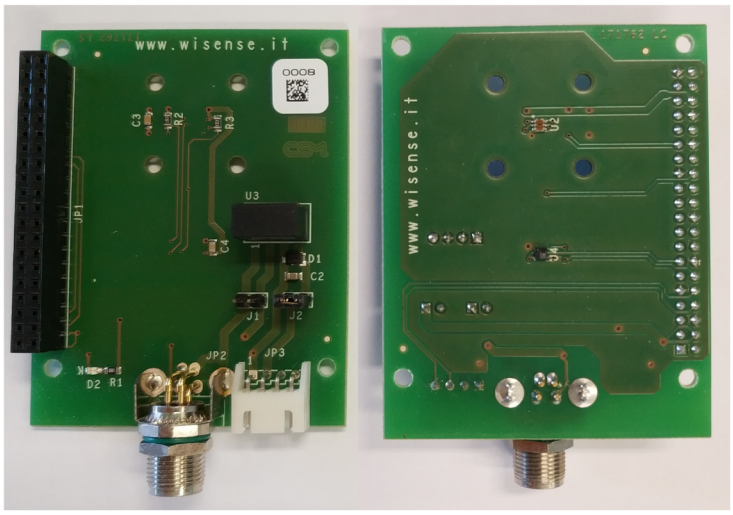

Figura 3.20: Unità di *sensing* installata sul dispositivo IoT.

- 32 kB RAM
- 256 kB di memoria Flash
- antenna ceramica

Inoltre il dispositivo è munito di un *pin header* (Fig. 3.19) per poter aggiungere moduli con funzionalità estese, come ad esempio una *board* munita dei sensori o attuatori necessari per uno specifico progetto di monitoraggio ambientale. Infatti in [82] è stato installato sul dispositivo il modulo munito di sensori di illuminamento, temperatura e umidità, mostrato in Fig. 3.20. Sono state inoltre disegnate, e poi progettate grazie all'aiuto del Dipartimento di Ingegneria Industriale e Scienze Matematiche (DIISM) dell'Università Politecnica delle Marche, le *cover* per questi nodi sensori come mostrato in Fig. 3.21 e Fig. 3.22. In questo modo si realizza un nodo IoT altamente scalabile che si adatta di volta in volta alle specifiche richieste dal progetto da sviluppare. Su questo dispositivo viene poi caricato il *file* binario del sistema operativo Contiki OS, che contiene tutte le istruzioni necessarie per l'avvio del nodo, la configurazione delle rete IPv6 e la lettura dei dati prodotti dai sensori presenti.

3.7 La rete *mesh* IoT

Utilizzando i dispositivi illustrati nella sezione precedente si possono realizzare reti *wireless* (Fig. 2.2) più o meno complesse a seconda delle esigenze applicative. Per realizzare una rete 6LoWPAN *grounded* è necessario che uno dei nodi della rete IoT

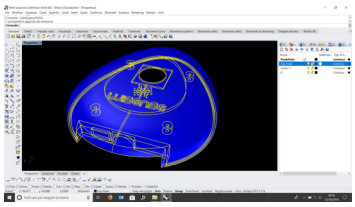

Figura 3.21: Progetto della *cover* per il nodo sensore a forma di goccia.

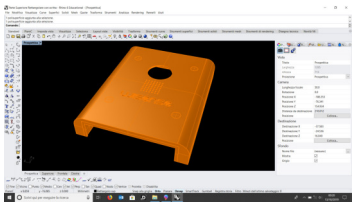

Figura 3.22: Progetto della *cover* per il nodo sensore a forma di parallelepipedo.

sia configurato come nodo *root* del DODAG, e per far questo si utilizza una specifica API[25] fornita da Contiki OS:

```
/* Richiedo il prefisso */
  while(!prefix_set) {
    etimer_set(&et, 1000);
    request_prefix();
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
  }
  dag = rpl_set_root(RPL_DEFAULT_INSTANCE,(uip_ip6addr_t *)dag_id
    );
  if(dag != NULL) {
    rpl_set_prefix(dag, &prefix, 64); /* Creato nuovo DAG */
  }
```

Codice 3.7: Inizializzazione del nodo *root* della rete *mesh* IPv6.

I nodi che ora si accendono o si trovano in attesa dell'avvio del nodo *root* ricevono i *beacon* contenenti le informazioni della relative rete 6LoWPAN, e possono richiedere di far parte di tale rete. Il protocollo di *routing* RPL si prende carico di creare e gestire la rete, di inserire nuovi nodi, e ricalcolare le rotte quando un nodo non è più raggiungibile.

Nella seguente figura viene mostrata una rete 6LoWPAN installata all'interno del Dipartimento di Ingegneria dell'Informazione (DII) della facoltà di Ingegneria dell'Università Politecnica delle Marche. Questa rete IPv6 pubblica[26], identificata dal *network ID* 2001:760:2e0d:130::/64, è stata utilizzata nel corso del progetto di dottorato per testare le diverse funzionalità dei nodi che dovevano essere sviluppate per i diversi progetti portati a termine, e al meglio delle nostre conoscenze, si tratta della prima WSN Italiana completamente IPv6 [40]. Nella Fig. 3.23 indicato con la stella blu il nodo *root* identificato dall'indirizzo globale 2001:760:2e0d:130:3035:3230:7134:890c, con i *link* blu i nodi che si trovano a solo 1 *hop* di distanza dal *root*, e con i *link* verdi i nodi a 2 *hop* dal *root*.

[25] *Application Programming Interface.*

[26] Si tratta di una rete IPv6 pubblica fornita dal Gruppo per l'Armonizzazione delle Reti della Ricerca (GARR).

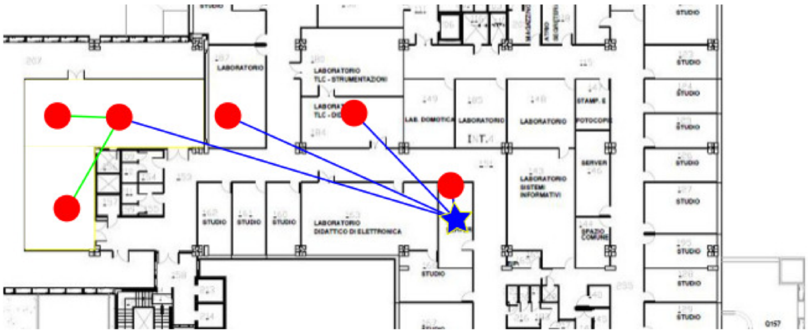

Figura 3.23: Rete di dispositivi *wireless* installata presso la facoltà di Ingegneria del'Università Politecnica delle Marche.

Capitolo 4

Casi d'uso

In questo capitolo verranno presentati i lavori per la realizzazione dei sistemi di monitoraggio realizzati durante i tre anni del dottorato di ricerca.

4.1 Monitoraggio dell'illuminazione

Nel 2017, in occasione di un progetto che ha visto protagonista iGuzzini Illuminazione in stretta collaborazione coi dottorandi e ricercatori dell'Università Politecnica delle Marche, è stato possibile studiare e progettare un sistema di monitoraggio dell'illuminazione volto a rendere maggiormente fruibili gli affreschi trecenteschi della cappella di Santa Maria della Carità di Padova. Il celebre edificio, meglio noto come "cappella degli Scrovegni" dal nome della famiglia che ne commissionò la decorazione a Giotto, è considerato un capolavoro dell'arte occidentale medievale. Interamente affrescata, compreso il soffitto a volta, ospita episodi dipinti della vita di Cristo, e nella parete occidentale che corrisponde all'ingresso, è collocato il celebre affresco del *Giudizio Universale*.

La distribuzione asimmetrica delle finestre (sei sulla navata sud, un rosone con due finestre sull'abside e una grande finestra aperta sopra il *Giudizio Universale*) è la causa di una forte alternanza di luci e ombre, che altera la percezione delle tonalità cromatiche. Si è quindi proceduto a realizzare un sistema di monitoraggio per misurare le variazioni di illuminazione, al fine di controllare dinamicamente il sistema di illuminamento a LED, installato all'interno della cappella dalla iGuzzini. In questa sezione verrà descritto il lavoro svolto per realizzare parte del sistema di monitoraggio attualmente in uso a Padova.

4.1.1 Architettura di rete

La WSN installata presso il sito culturale sopra citato è stata realizzata utilizzando l'architettura (Cap. 3.7) e i nodi (Cap. 3.6) descritti nel capitolo precedente, che vanno a formare una rete *mesh* composta di 6 nodi IPv6 (Fig. 4.1). Tra questi nodi, 5 sono utilizzati per il monitoraggio dei parametri ambientali, mentre uno ha la funzione di

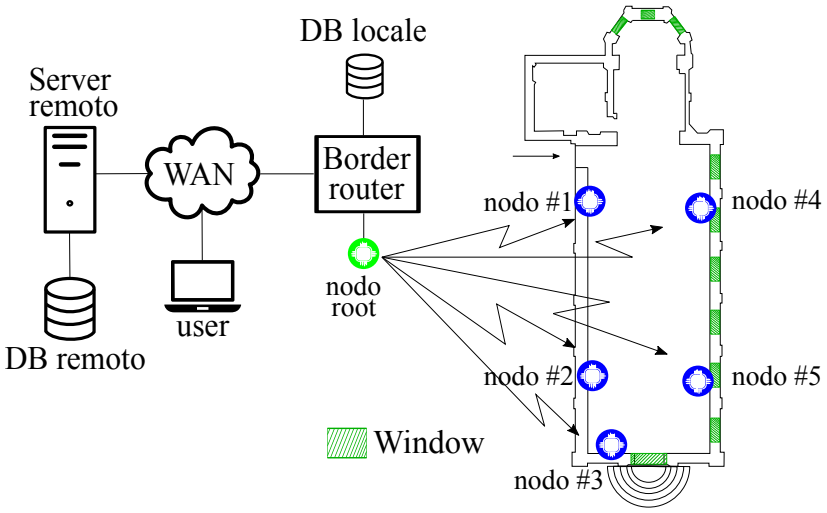

Figura 4.1: Architettura del sistema di monitoraggio installato presso la Cappella degli Scrovegni di Padova.

nodo *root* come descritto nel Cod. 3.7. Oltre ai nodi della rete, per completare il sistema di monitoraggio, è stato utilizzato un *border router* con le funzioni di gestore della WSN, di *database* locale, e di *server* HTTP per la visualizzazione dei dati, di controllo della rete, e di *router* per consentire la connettività WAN alla rete, il tutto realizzato con applicativi Python. Il *border router* che svolge le diverse funzioni di controllo della rete IoT è l'NPE X500 della TECHBASE[1]. Questo dispositivo è a tutti gli effetti un piccolo *computer* industriale, (*quadcore* 1.2 GHz, 1 GB RAM) con sistema operativo *Raspbian GNU/Linux 7 (wheezy)*, che fornisce un elevato numero di interfacce *Input/Output* (I/O) come:

- interfaccia digitale;
- interfaccia analogica;
- porte seriali RS-232/RS-485;
- interfaccia Ethernet;
- porta USB;
- interfaccia CAN;
- lettore SIM *card*;
- interfaccia 3G;

[1] *http://www.techbase.eu/en/products/npe-x500.html.*

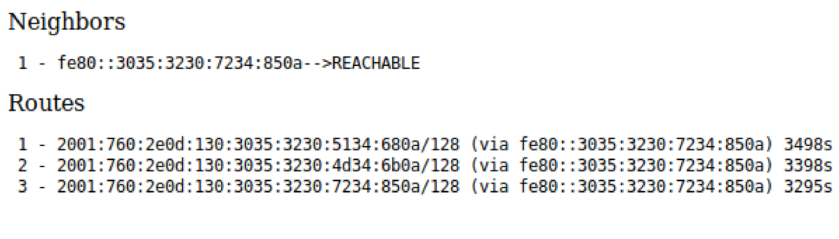

Figura 4.2: *Screenshot* della *home page* del nodo *root* della rete 6LoWPAN.

In particolare la porta seriale RS-232 viene utilizzata per collegare il *border router* al nodo *root* della rete 6LoWPAN (Cap. 3.3.3). La connessione viene instaurata tramite il *tool tunslip6* che permette di creare un'interfaccia virtuale sull'*host* per incapsulare il traffico IPv6 utilizzando il protocollo *Serial Line Internet Protocol* (SLIP) sul collegamento seriale. In questo modo si viene a creare un *tunnel* (tun) tra l'*host* e il nodo *root* dove vengono trasmessi pacchetti IP. Questa interfaccia viene attivata sul sistema operativo Linux con il seguente comando:

```
1 sudo ./tunslip6 -s /dev/ttySC0 2001:760:2e0d:130::1/64 &
```

Codice 4.1: Esempio del comando di avvio del *tool* tunslip6.

Con l'opzione (-s) viene indicata qual'è la porta seriale dell'*host* utilizzata per la creazione del *tunnel*, e a seguire, viene indicato l'indirizzo IPv6 da associare a questa nuova interfaccia di rete virtuale. Il nodo *root* che era in attesa del prefisso di rete IPv6, come mostrato nel primo *snippet* del Cap. 3.7, lo riceve e lo utilizza per creare la rete 6LoWPAN. Una volta che il nodo *root* ha a disposizione un indirizzo IPv6 è possibile visualizzare la sua *home page* di *default*, tramite un qualsiasi *browser web*, al fine di visualizzare l'elenco dei nodi che si sono connessi a questa rete 6LoWPAN. La Fig. 4.2 mostra lo *screenshot* della *home page* del nodo *root*, dove sono elencati i *neighbors* del nodo *root*, in questo caso un solo nodo identificato dall'indirizzo IPv6 locale fe80::3035:3230:7234:850a, e l'elenco gli indirizzi IPv6 di tutti che appartengono alla rete.

4.1.2 Gestione del sistema

La gestione dell'avvio del *tunslip6*, e delle altre funzionalità del *border router*, viene gestita da uno *script bash* di *init* che si occupa dell'avvio di tutti i moduli necessari al funzionamento del sistema di monitoraggio. Ciò è necessario dato che il sistema di monitoraggio deve essere in grado di avviarsi autonomamente, ad esempio nel caso di interruzioni di corrente o riavvii del sistema operativo.

```
### BEGIN INIT INFO
# Provides:          wiserootAS
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Wiseroot Autostart
# Description:       description
### END INIT INFO
```

Codice 4.2: Info dello *script* di init.

Sopra sono riportate le informazioni dello *script* di avvio "/etc/init.d/wiserootAS.sh" che si occupa della gestione del sistema di monitoraggio IoT collegato al *border router*. Le righe 3 e 4 dello *snippet* specificano quali devono essere i servizi attivi nel sistema operativo per poter avviare lo *script wiserootAS*. In questo modo il processo di *init* del sistema attente che tutti i servizi siano stati attivati al *boot* prima di eseguire questo *script*, e per *default* sarà avviato nei *run-levels* 2 3 4 5 e se necessario arrestato nei *run-levels* 0-1-6. All'interno dei sistemi Linux, l'avvio dei processi della macchina viene controllato dal System V, che si basa sull'init classico. Il *file* di configurazione di init che è inittab, controlla quali sono i processi da avviare e in quale ordine, e ciò avviene eseguendo i *links* simbolici presenti nel percorsi "/etc/rcN.d" dove N è il *runlevel* che va da 0 a 6. Questi *links* rimandano agli *script* che si occupano dell'avvio e arresto dei processi e che sono salvati in "/etc/init.d/" in conformità al *Filesystem Hierarchy Standard*. Oggi il System V è stato sostituito dal nuovo systemd che introduce la modalità di attivazione dei servizi nota come *socket activaton*, e ovviamente è in grado di supportare il classico System V init mantenendo la struttura dei *runlevel* e dei *links* simbolici in "/etc/rcN.d".

Nel seguente *snippet* viene riportata la funzione di *start()* dello *script* "/etc/init.d/wiserootAS":

```
start(){
        echo "Start wiseroot AS"

        cd /home/pi/Desktop/Wisense/
        echo "Mounting PenDrive"
        while [ $(ls /dev | grep -c 'sda') -eq 0 ]; do
                echo "no pendrive plugged in"
                sleep 5
        done
        mount -o umask=0022,gid=65534,uid=111 /dev/sda1 /media/
    SANDISK
        echo "PenDrive mounted"
        sleep 2

        echo "Starting WebSocket"
        python webServer/websocket/websocket.py &
        sleep 7
```

```
17
18          echo "WSAS"
19          python WiSeRootAutoStart.py &
20          sleep 10
21  }
```

Codice 4.3: Funzione di *start* dello *script* di avvio del processo wiserootAS.

Il ciclo in riga 6 attende che sia presente una penna USB prima di continuare l'esecuzione del codice, e quando la condizione è verificata, in riga 10 il *file system* della penna USB viene montato nel percorso "/media/SANDISK". Questa penna USB è utilizzata per contenere il *database* locale che memorizza le misurazioni effettuate dalla rete IoT. In riga 16 viene avviato il *websocket* per la visualizzazione sulla piattaforma *web* dei nuovi dati acquisiti dal sistema di monitoraggio in *real-time*, e infine, in riga 20 viene eseguito lo *script* Python che si occupa dell'avvio e gestione del sistema di monitoraggio.

4.1.3 La piattaforma *web*

La piattaforma *web* che si occupa della visualizzazione dei dati raccolti dalla rete IoT e della gestione della rete stessa è stata chiamata WebPyApp. Il servizio *web* WebPyApp è stato realizzato utilizzando il *framework* web.py [83] di Python. Utilizzando il *framework* web.py è triviale realizzare un'applicazione *web* in grado di restituire risorse HTML a chi ne fa richiesta. Per prima cosa occorre istruire il *server* su come instradare le richieste che riceve dagli utenti remoti, e ciò viene realizzato attraverso il sistema di *routing* degli URL. Il *framework* mette la variabile *urls* che deve essere popolata nel seguente modo:

```
1  import web # framework webpy
2  ...
3  urls=('/', 'simpleForm', '/command', 'command', '/login', 'login'
       , ...)
4  ...
5  ...
6  class login:
7      def GET(self):
8          ...
9      def POST(self):
10         ...
```

Codice 4.4: Esempio della gestione degli URL e delle richieste GET e POST con la libreria webpy.

La variabile *urls* (riga 3) è una tupla che contiene in sequenza l'URL e la classe Python che si occuperà della gestione della richiesta effettuata, *i.e.* l'URL '<ip_address>/login' sarà gestito dalla classe *login*. All'interno della classe *login* (riga 6) sono presenti due metodi, il primo si occupa della gestione delle richieste HTTP GET (riga 7), mentre il secondo della gestione delle richieste HTTP POST (riga 9). In questo modo

ogni richiesta HTTP che viene effettuata al *server web* identificato dall'indirizzo IP <ip_address> sarà inoltrata ad una specifica classe Python in accordo allo schema definito dalla variabile *urls*.

La libreria utilizzata mette a disposizione un linguaggio di *templating* per passare i dati dalle classi che gestiscono le richieste HTTP alle diverse risorse HTML che saranno inviate come risposta. Il *file* HTML che implementa il *template* della pagina *web* deve iniziare con sua prima riga con la seguente sintassi "$def with (<argomenti>)" come mostrato nel seguente *snippet*:

```
1 $def with (args)
2 <html>
3 ...
4 $if (args):
5     ... do somethings ...
6 ...
7 </html>
```

Codice 4.5: *Template* HTML per la libreria webpy.

All'interno del codice HTML può essere inserito il carattere di *escape* "$" per consentire alla libreria webpy di interpretare le istruzioni Python, come viene mostrato nella riga 4. Per passare i dati alla pagina HTML occorre preparare il *rendering* dei dati utilizzando il metodo *render* della classe *template*, come viene fatto nella riga 3 del seguente *snippet*.

```
1 import web
2 ...
3 dashboard=web.template.render('templates/dashboard/')
4 ...
5 class login:
6     def GET(self):
7         ...
8         return dashboard.dashboard(data)
```

Codice 4.6: *Rendering* del *template*.

L'argomento del metodo *render* è il percorso che contiene i *files* relativi ai *template* HTML, mentre a riga 8, vengono restituiti dal metodo GET i dati della richiesta ricevuta al *template* HTML specificato. La sintassi della funzione di ritorno ha la forma *percorso.nome_file_html(dati)*. Utilizzando il *rendering* dei *template* e degli *script* in JavaScript vengono trasmesse le pagine *web* agli utenti che ne fanno richiesta.

Il *JavaScript* viene anche utilizzato per generare i grafici che mostrano le grandezze fisiche misurate. Ciò viene realizzato utilizzando la libreria *JavaScript* amCharts [84] che permette di disegnare facilmente grafici x/y. All'interno dei *template* HTML sono presenti delle " " che sono utilizzate come contenitori dei grafici, e vengono passate al costruttore del grafico tramite il loro identificativo (id), come mostrato nell'esempio riportato:

```
1  ...
2  var chartRGB_c = AmCharts.makeChart("chartdivRGB-c", {
3      type: "serial",
4      dataProvider: chartDataRGB_c,
5      ... altre opzioni ...
6  });
7  ...
8  var graphR = new AmCharts.AmGraph();
9  ...
10 <div id="chartdivRGB-c" style="width: 100%; height: 400px;">
11 </div>
12 ...
```

Codice 4.7: Costruzione del grafico con amCharts

In riga 2 viene creato il grafico all'interno della *div* identificata dall'*id* "chartdivRGB_c" (riga 10), e viene associato allo stesso grafico l'*array* (riga 4) che conterrà i valori numerici da visualizzare. In riga 3 viene impostato come tipo di rappresentazione la curva per la visualizzazione dei dati, e possono essere passate ulteriori opzioni che servono a definire lo stile e l'aspetto del grafico che verrà mostrato all'interno della *div*.

Occorre ora passare i dati presenti sul *database* MongoDB (Cap. 5.3) relativi alle misure effettuate dai nodi sensori all'interno della Cappella degli Scrovegni. Per caricare i dati viene effettuata una richiesta HTTP all'URL "/chartCalculated", dove nella POST è contenuto il nodo sensore di cui si vogliono visualizzare le misure, e l'intervallo temporale d'interesse. Nel seguente *snippet* viene mostrata l'esecuzione della richiesta HTTP in *JavaScript* e la ricezione della rispettiva risposta.

```
1 var xmlhttp = new XMLHttpRequest();
2 var to_post=node+";"+days+";a";
3 xmlhttp.open("POST", "/webPyApp/chartCalculated", false);
4 xmlhttp.send(to_post);
5 ...
6 var my_Data = JSON.parse(xmlhttp.responseText);
```

Codice 4.8: Invio della POST in *JavaScript*.

Invece la classe sottostante viene utilizzata per ricevere la POST precedente ed esegue la *query* al *database* per ottenere le informazioni richieste dall'utente. La classe Python che corrisponde all'*url* "/chartCalculated" è la seguente classe:

```
1 class chartCalculated:
2     def POST(self):
3         rtn="["
4         node=web.data().split(";")[0]
5         day=int(web.data().split(";")[1])
6         code=web.data().split(";")[2]
7         if code=="a":
8             if day==-1:
9                 data=client.overlux.mbdata.find({'ipv6':node}).
      sort('msgtime',1)
```

```
10                      else:
11                          now=datetime.now()
12                          firsttime = (now - timedelta(days=day)).
    strftime('%Y-%m-%d %H,%M,%S')
13                          data=client.overlux.mbdata.find({"$and"
    :[{'ipv6':node},{'msgtime':{"$gte":firsttime}}]}).sort('
    msgtime',1)
14                      for dato in data:
15                          rtn=rtn+'{"date":"'+str(dato['msgtime']).
    replace(",",":")+'","val":'+str(dato['realdata'])+'},'
16                      if len(rtn)==1:
17                          rtn=""
18                      else:
19                          rtn = rtn[:-1]+"]"
20          client.close()
21          return rtn
```

Codice 4.9: Classe Python che passa i dati al *template* HTML.

Come si vede dal codice riportato, in riga 9 o 13, viene eseguita la *query* al *database* a seconda dell'intervallo temporale scelto, in riga 9 vengono scaricate tutte le misure del nodo IPv6 selezionato, mentre in riga 13 vengono richieste le misure dell'intervallo specificato con "now-timedelta(days=day)" (riga 12). I dati vengono poi inseriti in un *array* JSON (riga 15) e inviati alla pagina *web* (riga 21). Infine in JavaScript verrà passato questo *array* alla variabile associata al *dataProvider* del grafico (riga 4, Cod. 4.7), e a questo punto i dati sono visualizzati all'interno della *div* specificata. In questo modo vengono generati tutti i grafici relativi alle grandezze fisiche misurate dai 5 nodi sensori installati presso la Cappella degli Scrovegni (Fig. 4.1). Infine bisogna rendere disponibile il servizio *web* avviando il *server* HTTP.

Ciò viene realizzato utilizzando un'interfaccia che consente la comunicazione tra l'applicazione *web* scritta Python e il *server* che gestisce le richieste HTTP, cioè il *web server* Apache. Questa interfaccia è implementata dal protocollo Web Server Gateway Interface (WSGI), che viene configurato sia nel *server* HTTP che nello *script* Python. L'Apache HTTP Server viene configurato attraverso *files* di configurazione testuale. Per ogni sistema operativo questi *files* possono essere memorizzati il percorsi differenti, e nel caso dell'applicazione realizzata il *file* di configurazione di Apache è al seguente percorso: "/etc/apache2/apache2.conf". Il *file* di configurazione "apache2.conf" può essere aperto con un *editor* a piacere, e al suo interno sono presenti le direttive, cioè le istruzioni che definiscono il comportamento del *server web*. A questo *file* vanno aggiunte le direttive per consentire all'applicazione *web* in Python di ricevere e processare le richieste HTTP. Per ottenere questo risultato è sufficiente aggiungere in coda al *file* "apache2.conf" le seguenti istruzioni:

```
1 LoadModule wsgi_module modules/mod_wsgi.so
2 DocumentRoot /home/pi/Desktop/Wisense/webServer/
3 WSGIScriptAlias /webPyApp /home/pi/Desktop/Wisense/webServer/
      testServer.py
```

```
4 Alias /webPyApp/templates /home/pi/Desktop/Wisense/webServer/
      templates/
5 Alias /webPyApp/static /home/pi/Desktop/Wisense/webServer/static/
6 AddType text/html .py
```

Codice 4.10: Direttive per la gestione di Apache HTTP Server.

Alla riga 1 viene caricato il modulo WSGI che si occuperà di gestire l'interfaccia tra il *server* e lo *script* Python, e a riga 2 viene indicato qual'è il percorso contenente l'applicativo Python. La direttiva *DocumentRoot* indica al demone[2] *httpd*, che implementa il *server* qual'è il percorso da dove saranno forniti i *files*. Ad esempio, una richiesta alla pagina "http://<ip_address/webPyApp/login" sarà inoltrata al percorso "/home/pi/Desktop/Wisense/webServer/login.html". Invece nella riga 3 si ha la direttiva "WSGIScriptAlias", cioè è un particolare Alias che punta a uno *script* WSGI. La direttiva Alias ha lo scopo di indicare altri percorsi dove sono presenti *files* necessari al di fuori della cartella specificata nella *DocumentRoot*, e viene utilizzata in questo contesto per specificare i percorsi dove sono contenuti i *templates* e i *files* statici, come i Cascading Style Sheets (CSS).

Infine si deve rendere lo *script* Python compatibile con l'interfaccia WSGI, e ciò viene reso possibile utilizzando il metodo *wsgifunc* della libreria webpy (riga 4):

```
1 ...
2 ...
3 app = web.application(urls, globals(), autoreload=False)
4 application=app.wsgifunc()
5 ...
6 if __name__ == "__main__":
7     app.run()
```

Codice 4.11: Attivazione della compatibilità WSGI dell'applicazione Python.

Utilizzando questa combinazione, Apache HTTP Server e applicazione Python, vengono realizzati il *server web* locale e remoto per la visualizzazione dei dati misurati e per il controllo della rete installata [39].

4.1.4 Connettività remota

Per accedere al *server* locale, cioè al *server* in esecuzione sul *border router* (Fig. 4.1), occorre fornire a quest'ultimo una connessione IP pubblica e conoscerne il relativo indirizzo. Per risolvere questo problema senza dover ricorrere ad una connessione *Ethernet* appoggiata sulla rete del sito culturale della Cappella degli Scrovegni, è stata installata a bordo del dispositivo NPE X500 [3] una SIM *card* per comunicazioni M2M.

[2] Sono quei programmi che vengono eseguiti in *background* e non sono sotto il controllo diretto dell'utente. In genere il nome di questi programmi termina con una "d", per differenziarli dagli altri programmi eseguiti dall'utente.

[3] *http://www.techbase.eu/en/products/npe-x500.html.*

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:83.224.154.106  P-t-P:10.64.64.64  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:2281446 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3718871 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:161083918 (153.6 MiB)  TX bytes:271697461 (259.1 MiB)
```

Figura 4.3: Interfaccia di rete *ppp0*.

```
1  GPRS_PIN=****
2  GPRS_REG_RETRY=10
3  GPRS_MODE=PPP
4  GSM_MODE=AUTO
5  GPRS_MUX=Y
6  GPRS_AUTOSTART=Y
7  DEFAULT_ROUTE=GPRS
8  GPRS_APN_NAME=m2mbis.vodafone.it
9  GPRS_RECONNECT=Y
10 GPRS_AUTO_DNS=Y
11 GPRS_DNS_1=8.8.8.8
12 GPRS_LOGIN=ppp
13 GPRS_PASSWORD=ppp
14 GPRS_DYNDNS=N
15 GATEWAY_ETH=
16 GSM_BAUD=230400
17 GSM_PORT=/dev/ttyUSB0
18 GPRS_DNS_3=8.8.4.4
```

Codice 4.12: Configurazione della connessione GPRS

Lo *snippet* precedente riporta il contenuto del *file /home/core/syscfg* di configurazione del modulo GPRS per la connettività remota. Nella riga 1 viene inserita la *password* della SIM, e nelle righe 2 e 3, il numero di tentativi di connessione attraverso il protocollo PPP[4]. In riga 8 viene specificato l'*Access Point Name* (APN) per la connessione, e in riga 11 e 18, sono indicati i *server Domain Name System* (DNS)[5] per la risoluzione degli indirizzi. Con il comando *sudo gprs connect* viene avviato il programma che instaura la connessione con l'APN, e al termine della negoziazione PPP, tra le interfacce di rete del sistema sarà presente la nuova interfaccia *ppp0* (Fig. 4.3) che servirà per l'accesso remoto al *border router*, per la visualizzazione delle grandezze misurate, e per inviare le misurazioni al *database* remoto. L'indirizzo IPv4 associato alla connessione GPRS dell'interfaccia *ppp0* è tuttavia sconosciuto agli utenti remoti, e senza questa informazione non è possibile accedere in alcun modo ai servizi forniti dal *border router*. E' stato realizzato uno *script* Python per acquisire l'indirizzo IP

[4] *Point-to-Point Protocol*: protocollo di livello *datalink* per la comunicazione diretta tra due terminali di rete.

[5] Sono stati utilizzati come *server* DNS, il primario (8.8.8.8) e il secondario (8.8.4.4) di Google.

visualizzato dal comando "ifconfig -a ppp0", e successivamente caricarlo all'interno del *database* remoto (Fig. 4.1) installato presso la Facoltà di Ingegneria, e viene eseguito ogni 5 minuti dal demone *crond* per aggiornare il *database* con l'indirizzo IP raggiungibile. Infine all'interno del *server* remoto che ospita il *database* è presente una pagina *web* che permette la visualizzazione degli indirizzi IP associati ad installazioni di reti IoT effettuate. Così facendo si ha a disposizione l'indirizzo IP dei *border router* installati.

4.2 Monitoraggio energetico

I nodi della rete IoT realizzata durante i tre anni di dottorato, oltre a consentire il monitoraggio di parametri ambientali come illustrato nella sezione precedente con l'installazione portata a termine con successo presso la Cappella degli Scrovegni di Padova, i nodi permettono anche di poter effettuare misure relative ai consumi energetici di un impianto elettrico. Il sistema di monitoraggio descritto nella sezione precedente (Cap. 4.1) è stato riadattato per consentire l'acquisizione di misure energetiche e per la visualizzazione di dei dati raccolti sulla piattaforma *web* del *border router*.

Tuttavia il monitoraggio energetico non deve essere interpretato come il solo atto di acquisire le misurazioni dei consumi degli apparati in funzione, ma invece di acquisire informazioni utili sui flussi energetici del sito per una pianificazione strategica di un progetto di efficientamento e risparmio energetico. In questa sezione saranno introdotte le normative vigenti nel settore del monitoraggio energetico, e verranno presentate le figure che prendono parte attivamente alle varie fasi necessarie per ottenere degli effettivi risparmi sui consumi.

4.2.1 Motivazioni

Il monitoraggio energetico è quel sistema interno ad un'azienda utile per conoscere i propri flussi energetici, e dove necessario, è indispensabile per pianificare i progetti di efficientamento dei dispositivi energivori. Questo complesso sistema di monitoraggio non è solo consigliato per essere consapevoli dei propri consumi, ma infatti, con il d.lgs. 102 del 4 luglio 2014 in riferimento all'attuazione della Direttiva Europea 2012/27/UE sull'efficienza energetica, obbliga[6] le grandi aziende e le aziende energivore a effettuare attività di monitoraggio energetico. Con il termine "grandi aziende", il d.lgs. 102, indica le aziende con più di 250 dipendenti o con un fatturato $> 50M$ €, mentre con "aziende energivore" si riferisce alle aziende con un consumo energetico medio maggiore di 2.4 GWh/annui.

[6] In caso di mancato monitoraggio energetico per le aziende obbligate, sono previste sanzioni fino a 40.000 €.

Le figure responsabili e certificate per controllare il sistema di monitoraggio energetico o abilitate per effettuare l'operazione di *auditing* energetico sono: l'Esperto in Gestione Energia (EGE), l'*Energy Service Company* (ESC) e l'*Auditor* Energetico. L'EGE una figura tecnica esperta, introdotta dalla UNI-CEI 11339, che ha le competenze per effettuare progettazioni d'impianti, di realizzazione di studi di fattibilità, diagnosi e *audits* energetici. L'ESC ha invece il compito di ricercare le risorse finanziare necessarie per la messa in opera delle soluzioni di efficientamento e risparmio energetico. Infine si hanno gli *Auditor* Energetici, definiti dalla normative UNI-CEI EN 16247-5, che hanno le competenze e certificazioni richieste per condurre *audits* energetici per le aziende.

4.2.2 *Audit* energetico

L'*audit* energetico è il *report* dettagliato che permetta al soggetto interessato di inquadrare i propri consumi energetici e di identificare gli eventuali interventi da realizzare per aumentare l'efficienza e ridurre i consumi. Per chiarire meglio lo scopo e le finalità si riporta la definizione del d.lgs del 4 Luglio 2014:

> “L'*audit* energetico o diagnosi energetica è la procedura sistematica finalizzata a ottenere un'adeguata conoscenza del profilo di consumo energetico di un edificio o gruppo di edifici, di una attività o impianto industriale o commerciale o di servizi pubblici o privati, a individuare e quantificare le opportunità di risparmio energetico sotto il profilo costi-benefici e a riferire in merito ai risultati.”
>
> ***d.lgs. del 4 Luglio 2014, n. 102.***

L'*audit* energetico non è limitato alla sola valutazione dei consumi energetici dei dispositivi elettrici, ma valuta anche l'impianto dell'illuminazione, la produzione di Acqua Calda Sanitaria (ACS), l'efficienza degli impianti di riscaldamento, dell'impianto di climatizzazione e del sistema di ventilazione meccanica (Fig. 4.4). Ovvero con l'*audit* energetico si vuole avere piena conoscenza dello stato di tutti gli impianti alimentati ad energia elettrica, al fine di individuare sprechi dei sottosistemi, o aggiornare macchine datate per ridurre i costi e i consumi mantenendo lo stesso livello di *comfort*, e di conseguenza ridurre l'impatto sull'ambiente. Ridurre i consumi del sito in esame significa aumentare l'efficienza energetica, ridurre i costi dell'energia e ridurre i le emissioni di CO_2 e di gas serra. La diagnosi energetica o *audit* energetico, prodotto da esperti nel settore dell'energia (EGE, ESC, *auditor*), diventa quindi lo strumento fondamentale per un mirato intervento di efficientamento.

Questo documento è una relazione che descrive i flussi energetici all'interno di un'attività[7] e l'Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo

[7] Si possono effettuare *audits* energetici per le attività legate ai settori primario, secondario e terziario, e anche tutti i privati che sono interessati a ridurre i propri consumi energetici.

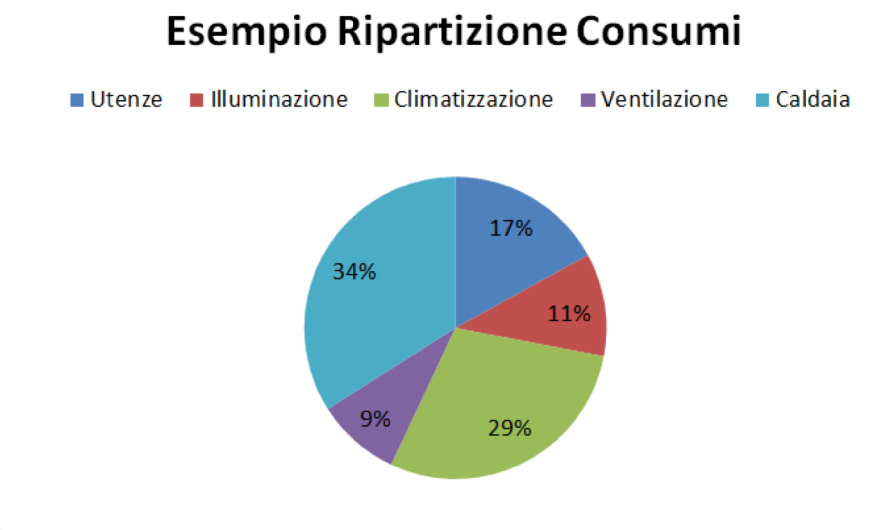

Figura 4.4: Esempio dei consumi considerati per un possibile *audit* energetico.

economico sostenibile (ENEA) fornisce le linee guida[8] [85] per l'analisi dell'attività in esame e il *template* [86] per la relazione finale. La linea guida dell'ENEA suggerisce che l'analisi deve coprire i seguenti aspetti energetici:

1. analisi dei punti d'ingresso dell'energia
2. analisi della ripartizione energetica all'interno del sito e suo impiego
3. analisi dei punti di utilizzo dell'energia
4. ipotesi di efficientamento dell'energia

L'*auditor* che ha il compito di produrre la diagnosi energetica deve essere informato sui processi produttivi, l'organizzazione del personale e le loro mansioni, i punti di accesso alle sorgenti elettriche, gli ingombri, i sistemi di riscaldamento/climatizzazione, e di tutti gli altri aspetti legati all'attività produttiva che sono legati ai consumi di risorse energetiche. Con queste informazioni, e dopo aver condotto un'ispezione del sito, l'*auditor* energetico può preparare la diagnosi riportando i consumi suddivisi tipo di utilizzo e per sorgente elettrica, il bilancio energetico, l'andamento temporale della domanda di energia, e parametri di efficienza energetica del sito. Con questi dati a disposizione il tecnico deve fornire delle indicazioni per aumentare le prestazioni e l'efficienza energetica sito.

[8] Queste linee guida non devono obbligatoriamente essere seguite, la figura che effettua l'*audit* energetico è libera di utilizzare propri strumenti di analisi e elaborazione dei dati raccolti, purchè vengano rispettate le norme dell'allegato 2 del d.lgs. 102/2014.

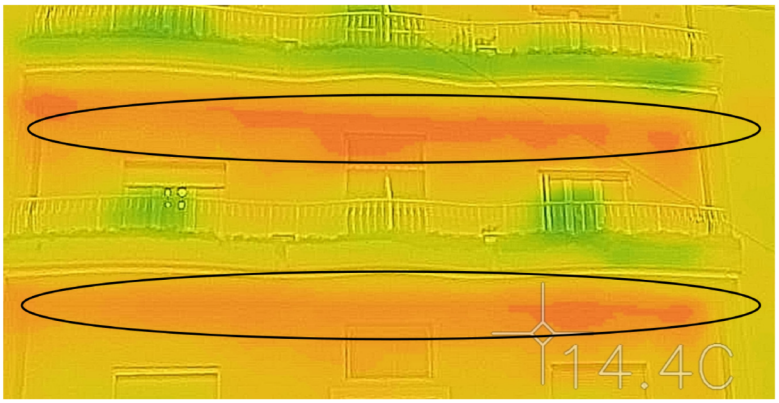

Figura 4.5: Analisi con termocamera di un edificio senza adeguato isolamento termico con l'esterno.

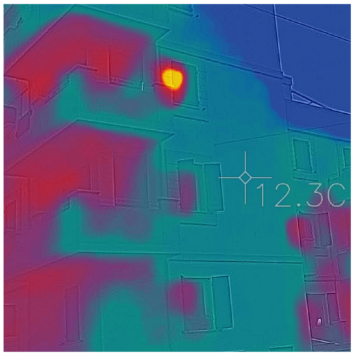

Figura 4.6: Edificio con buon isolamento termico.

Figura 4.7: Confronto della dispersione termica di due edifici. Sulla sinistra un edifico non isolato termicamente, sulla destra un edificio con cappotto.

Nella Fig. 4.5 viene mostrata l'immagine termica della facciata di un edificio realizzata con una termocamera FLIR ONE [87]. Con questo tipo di analisi gli *auditor* energetici posso valutare l'efficienza dell'isolamento di una struttura, in questo caso un edificio abitabile, per pianificare interventi di efficientamento e di riduzione dei consumi dell'impianto di riscaldamento e condizionamento. Come si vede dalle zone evidenziate nella Fig. 4.5, in questo edificio si hanno delle fughe di calore dai solai dei vari piani ($14.4°$C), che fanno aumentare i consumi e i costi del riscaldamento. Invece nelle prossime due figure vengono messi a confronto due edifici, uno con un buon isolamento termico (Fig. 4.6) fornito dal cappotto isolante dell'edificio ($12.3°$C), mentre in Fig.4.7 viene mostrato un edificio che disperde calore ($15.2°$C) affianco al precedente.

Prima di procedere con l'analisi del sistema di monitoraggio energetico basato su WSN applicate all'IoT realizzato per questo lavoro di tesi, saranno descritte breve-

mente alcune possibili soluzioni adottabili per l'efficientamento energetico.

4.2.3 Efficientamento consumi

Al termine della diagnosi energetica, l'*auditor* e l'azienda sono consapevoli delle aree su cui intervenire per aumentare l'efficienza e rendere l'azienda stessa più competitiva sul mercato.

Ci sono molti tipi di interventi che possono essere eseguiti per ridurre i consumi, e questi dipendono dal tipo di azienda, dal settore produttivo e dall'attuale stato degli impianti in uso. Una prima forma banale d'intervento che può essere condotta, è quella di cercare un piano tariffario più idoneo alle caratteristiche produttive dell'azienda analizzata, e ed è compito dell'*auditor* identificare se necessario quali sono le possibili soluzioni ottimali. Altre tipologie di interventi possono essere proposte dall'*auditor*, e tra queste si possono individuare le seguenti:

- isolamento termico
- sistemi di schermatura
- sostituzione degli impianti di climatizzazione e termoregolazione
- installazione di sistemi domotici per il controllo dei flussi energetici, *i.e. Building & Automation Control System* (BACS)
- installazione di sistemi di cogenerazione
- installazione di impianti fotovoltaici
- sostituzione delle sorgenti luminose con sorgenti a LED
- installazione di pompe di calore a energia geotermica o biomassa
- utilizzo di combustibili adeguati
- sistemi di recupero dell'energia dai processi industriali
- efficientamento dei trasformatori, e aumento del fattore di potenza dell'impianto elettrico
- *etc..*

4.2.4 Unità di *sensing*

L'adattamento del sistema di monitoraggio realizzato e descritto nel Cap. 4.1 è stato possibile grazie all'elevata portabilità del sistema di gestione e dei dispositivi realizzati. Infatti sui dispositivi IoT, realizzati dal gruppo di ricerca, si possono installare diverse tipologie di unità di *sensing* grazie alla presenza del *pin header* posizionato

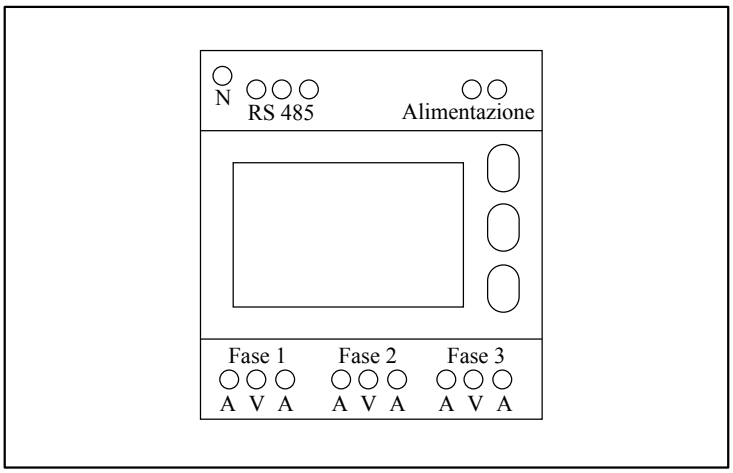

Figura 4.8: Rappresentazione del *power meter* utilizzato per l'acquisizione delle grandezze energetiche.

sulla *board* del dispositivo IoT (Fig. 3.19). In particolare in questa applicazione per il monitoraggio energetico è stata installata una scheda di espansione per consentire la comunicazione seriale con il protocollo modbus. Questo protocollo viene utilizzato dal nodo IoT per comunicare con l'unità di *sensig* esterna, cioè con un *power meter* sviluppato dalla Seneca per il monitoraggio energetico (Fig. 4.8). Il *power meter* utilizzato è il modello (Seneca S604E RS485 [88]) che consente l'acquisizione di diversi parametri energetici su reti monofase o trifase attraverso l'impiego di una delle seguenti possibili configurazioni per la connessione sulla rete elettrica:

- 4 cavi - 3 fasi
- 3 cavi - 2 fasi
- 1 cavo - 1 fase

Le modalità d'inserzione con bobina di Rogowski del *power meter* sulla linea elettrica sono schematizzate in Fig. 4.9 e, dove sono mostrate in sequenza le configurazioni a 3 fasi, 2 fasi e monofase. Ai fini della realizzazione del sistema di monitoraggio è stata utilizzata la seconda configurazione (3cavi, 2 fasi) (Fig. 4.11), che permette di misurare le seguenti grandezze elettriche:

- Tensione di linea 1-2 (V_{12})
- Tensione di linea 2-3 (V_{23})
- Tensione di linea 3-1 (V_{31})
- Tensione di sistema (V_{Sys})

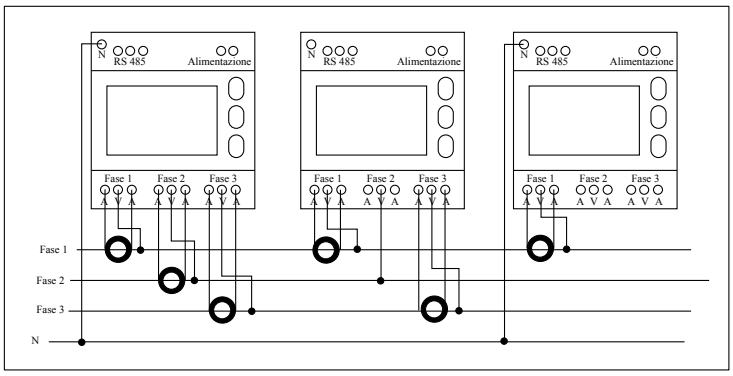

Figura 4.9: Modalità di inserzione del *power meter* Seneca S604E RS485 nelle sue tre configurazioni con la bobina di Rogowski.

- Corrente di fase 1 (A_1)
- Corrente di fase 2 (A_2)
- Corrente di fase 3 (A_3)
- Corrente di sistema (A_{Sys})
- Potenza attiva di sistema (P_{Sys})
- Potenza apparente di sistema (P_{aSys})
- Potenza reattiva di sitema (P_{rSys})
- Fattore di potenza (PF_{Sys})
- Tangente del fattore di potenza ($tanPF_{Sys}$)
- Distorsione armonica di corrente su fase 1 ($THDA1$)
- Distorsione armonica di corrente su fase 2 ($THDA2$)
- Distorsione armonica di corrente su fase 3 ($THDA3$)
- Frequenza (f)

Il dispositivo IoT collegato in seriale con il *power meter* utilizza una libreria di ContikiOS per la gestione della comunicazione con il protocollo modbus. I registri del *power meter* vengono letti in modo sequenziale e le informazioni acquisite vengono rese disponibili su risorse CoAP all'interno del dispositivo IoT, in questo modo, il *server* che si occupa della memorizzazione e visualizzazione dei dati può interrogare il *power meter* per ottenere i nuovi campioni energetici con delle GET CoAP. Due processi di Contiki OS (Cap. 3.5) si occupano della gestione del sistema di monitoraggio.

Figura 4.10: Macchina MTS per prove di trazione.

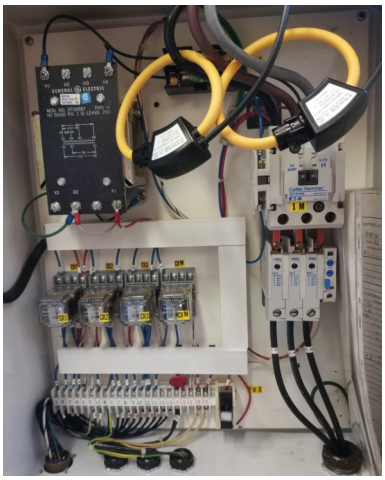

Figura 4.11: Collegamento delle due bobine di Rogowski al circuito di alimentazione della pompa idraulica della macchina MTS.

Un processo effettua le letture con il protocollo modbus sul *power meter* collegato alla linea elettrica, mentre un secondo processo si occupa della gestione delle richieste CoAP.

4.2.5 Gestione della misura

L'acquisizione dei valori energetici misurati dall'unità di *sensing* descritta nella sezione precedenta, è condotta da un processo (Cap. 3.5) del sistema operativo ContikiOS installato sul nodo IoT. Questo processo realizzato per questa specifica applicazione è il processo *mbus_read*, definito all'interno del sistema operativo nel modo seguente:

```
#include "mdatastruct.h"
#include "m_utilz.h"
...
...
PROCESS(mbus_read, "Modbus read");
...
process_start(&mbus_read, NULL);
...
```

Codice 4.13: Definizione e avvio del processo che gestisce la comunicazione modbus con il *power meter*.

In riga 4 e riga 6 viene definito e successivamente avviato il processo *mbus_read* che si occupa di gestire la comunicazione seriale con il *power meter* attraverso l'utilizzo del protocollo mobdus. Mentre in riga 1, viene inclusa nel sistema operativo una libreria che definisce una struttura utilizzata come contenitore per i valori letti sui registri del *power meter*. Questa struttura, *mbus_data_struc*, è definita nel seguente modo:

```
1  #ifndef MDATASTRUCT
2  #define MDATASTRUCT
3  #define SENECA_WORD 2
4  typedef struct M_MODBUS_DATA{
5      unsigned char v1[2*SENECA_WORD];
6      unsigned char v2[2*SENECA_WORD];
7      unsigned char v3[2*SENECA_WORD];
8      ...
9      ...
10     unsigned char THDa1[2*SENECA_WORD];
11     unsigned char THDa2[2*SENECA_WORD];
12     unsigned char THDa3[2*SENECA_WORD];
13     unsigned char THDaNeutro[2*SENECA_WORD];
14
15     char timestamp[6];
16     int config;
17 }M_MODBUS_DATA;
18
19 extern M_MODBUS_DATA mbus_data_struct;
20 #endif
```

Codice 4.14: Struttura che contiene i valori energetici misurati dal *power meter*.

Come si vede dallo *snippet* riportato, questa struttura ha lo scopo di contenere le possibili grandezze elettriche misurate[9] dal *power meter* della Seneca. Il processo *mbus_read* oltre ad interrogare i registri del *power meter* ha proprio il compito di scrivere i valori energetici acquisiti all'interno di questa *struct*. La stessa *struct* è anche condivisa con il *server* CoAP presente sul nodo sensore, in questo modo, quando un utente (o macchina) effettua una richiesta al nodo, vengono fornite le misure energetiche effettuate dal *power meter*. Per popolare questa *struttura* è stata realizzata una funzione apposita definita nell'*header file m_utilz.h* (Cod. 4.15), che viene utilizzata nel processo *mbus_read* (Cod. 4.13, riga 2). Questa funzione, nominata *set_mbus_data_*, ha il compito di inserire il valore letto dal registro i-esimo dell'unità di *sensing* nella corrispondente variabile della struttura *mbus_data_struct*.

```
1  #ifndef MUTILZ
2  #define MUTILZ
3
```

[9] Nel codice non sono state riportate tutte le grandezze elettriche incluse nella *struct*, l'elenco completo è disponibile nel *datasheet* del *power meter* [88].

```
4 void set_mbus_data_(unsigned int reg, unsigned char char0,
      unsigned char char1, unsigned char char2, unsigned char char3
      );
5
6 #endif
```

Codice 4.15: Funzione per la scrittura della *mbus_data_struct*.

Il corpo del processo *mbus_read*, riportato nello *snippet* Cod. 4.16, ha il compito di leggere i registri del dispositivo modbus con cui il nodo è connesso in seriale. In riga 8 viene impostato il *timer* di riavvio del processo, il quale ovviamente dipende dal tipo di applicazione, e nell'esempio riportato è impostato a 60 secondi. Nella riga 10 viene invece inizializzato il vettore *registers[]* che contiene l'elenco di tutti i registri che contengono i valori energetici misurati dal *meter* Seneca. Nelle righe 11, 12 e 13, vengono associati i registri alle rispettive configurazioni d'installazione del *power meter*. Ad esempio per il vettore *reg_idx_1[]* sono elencati gli indici associati alla configurazione monofase del dispositivo di monitoraggio.

```
1 PROCESS_THREAD(mbus_read, ev, data){
2
3   static struct etimer mbus_timer;
4   static struct etimer mbus_inter_reg;
5
6   PROCESS_BEGIN();
7   modbus_init();
8   etimer_set(&mbus_timer, 60*CLOCK_SECOND);
9   ...
10  static unsigned int registers[] = {0x1000, 0x1002, 0x1004, 0
      x1006, 0x1008, 0x100A , 0x100C, 0x100E, 0x1010, 0x1012, 0
      x1014, 0x1016, 0x1018, 0x101A, 0x101C, 0x101E, 0x1020, 0x1022
      , 0x1024, 0x1026, 0x1028, 0x102A, 0x102C, 0x102E, 0x1030, 0
      x1032, 0x1034, 0x1036, 0x1038, 0x103A, 0x103C, 0x103E, 0x1040
      , 0x1042, 0x1044, 0x1046, 0x1048, 0x104A, 0x104C, 0x104E, 0
      x1050, 0x1052, 0x1054, 0x1056, 0x1058};
11  static unsigned int reg_idx_1[] = {0, 7, 12, 16, 20, 24, 28,
      31, 35, 41};
12  static unsigned int reg_idx_2[] = {3, 4, 5, 6, 7, 8, 9, 11, 15,
       19, 23, 27, 34, 38, 39, 40, 41, 42, 43};
13  static unsigned int reg_idx_3[] = {0, 1, 2, 3, 4, 5, 6, 7, 8,
      9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
      24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
      39, 40, 41, 42, 43, 44};
14    ...
15    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&mbus_timer));
16    for (register_addr=0;register_addr<idx_len;register_addr++){
17      switch(mbus_data_struct.config){
18      case 1:
19        mb_rv = modbus_read_register(slave_addr, 4, registers[
    reg_idx_1[register_addr]], 2);
20        break;
21      case 2:
```

```
22          mb_rv = modbus_read_register(slave_addr, 4, registers[
    reg_idx_2[register_addr]], 2);
23          break;
24        case 3:
25          mb_rv = modbus_read_register(slave_addr, 4, registers[
    reg_idx_3[register_addr]], 2);
26          break;
27        default:
28          mb_rv = -1;
29          break;
30        }
31        ...
32        set_mbus_data_(registers[register_addr], m_buffer_mbdata
    [3], m_buffer_mbdata[4], m_buffer_mbdata[5], m_buffer_mbdata
    [6]);
33    ...
34    PROCESS_END();
35  }
```

Codice 4.16: Lettura dei registri del *power meter* con il processo *mbus_read*.

Alla riga 16 viene inizializzato il *for loop* che scorre nei registri della configurazione selezionata dallo *switch* di riga 17. Infine a riga 32, viene usata la funzione *set_mbus_data_* (Cod. 4.15) per salvare i valori appena acquisiti all'interno della *struct*. Così facendo, quando il nodo *smart* riceve una GET CoAP, il *server* è in grado di rispondere fornendo le informazione richieste dal *client*.

4.2.6 *Server* CoAP

Con una prima richiesta di *discovery* l'utente remoto ottiene la lista delle risorse CoAP disponibili sul dispositivo IoT, e in questa specifica applicazione sono identificate dai due URL *test/energy* e *test/power*, che contengono rispettivamente informazioni relative alla potenza e alle tensioni/correnti misurate. Tramite una GET CoAP a uno di questi URL vengono acquisite le misure di tensione e corrente o di potenza, effettuate dal *power meter*, e vengono poi scritte all'interno del *database* installato nel *border router* (Cap. 5.3).

Nello *snippet* riportato viene mostrato come è stata configurata la risorsa CoAP, sulla base di quanto descritto nel Cap. 3.5.1, che restituisce al *client* remoto le misure energetiche effettuate dal *power meter*.

```
1  #include "rest-engine.h"
2  #include "mdatastruct.h"
3  ...
4  RESOURCE(res_energy, "title=\"Energy\";rt=\"Text\"",
5      res_get_handler,NULL, NULL, NULL);
6
7  static void res_get_handler(void *request, void *response,
8      uint8_t *buffer, uint16_t preferred_size, int32_t *offset){
9
```

```
10      const char *len = NULL;
11      char msg[80]="";
12      switch(mbus_data_struct.config){
13        case 1:
14          sprintf(msg, "%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02
        x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x
        %02x%02x%02x%02x%02x%02x%02x%02x%02x%02x,%c%c%c%c%c%c",
15          mbus_data_struct.v1[0],
16          mbus_data_struct.v1[1],
17          mbus_data_struct.v1[2],
18          mbus_data_struct.v1[3],
19          ...
20          );
21  ...
22  ...
23      REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
24      REST.set_header_etag(response, (uint8_t *)&length, 1);
25      REST.set_response_payload(response, buffer, length);
26  }
```

Codice 4.17: Implementazione della risorsa CoAP identificata dall'URL *test/energy*.

Nelle prima riga viene inclusa la libreria necessaria per il funzionamento della risorsa CoAP, mentre nella seconda riga viene richiamata la libreria che definisce la struttura *struct* contenente i valori energetici misurati dal *power meter*. A riga 12, viene definito uno *switch* che prepara i dati di risposta alla GET ricevuta in funzione della configurazione[10] utilizzata nell'installazione del *power meter* (Cap. 4.2.4). Le tre possibili configurazioni utilizzate nella *mbus_data_struct* sono le seguenti:

1. *mbus_data_struct.config*$= 1 \iff$ monofase;
2. *mbus_data_struct.config*$= 2 \iff$ 3 cavi, 2 fasi;
3. *mbus_data_struct.config*$= 3 \iff$ 4 cavi, 3 fasi;

Con la riga 14 si copiano le informazioni specificate dalla configurazione selezionata nella *struct* all'interno della variabile *msg*, e quest'ultima sarà infine trasmessa come *payload* della CoAP *reply* (righe 23-25). Lo stessa procedura viene replicata per la risorsa CoAP identificata dall'URL *test/power*, che viene utilizzata per condividere con il *client* i valori di potenza misurati dal *power meter*.

4.2.7 *Webserver*

Nel *server* oltre al *database* è presente anche un'applicazione *web* scritta in Python, e gestita dal *web server* di Apache (Cap. 4.1.3), per la visualizzazione delle misure memorizzate all'interno del *database*. Nella Fig. 4.12 viene mostrata l'interfaccia della

[10] Le possibili configurazioni utilizzabili sono: 4 cavi - 3 fasi, 3 cavi - 2 fasi, 1 cavo - 1 fase, a cui corrispondo differenti quantità misurabili.

Figura 4.12: Interfaccia dell'applicazione *web* per la visualizzazione delle misure energetiche effettuate dal *power meter*. In evidenza: 1 - Selettore del nodo della rete IPv6; 2 - Selettore dell'intervallo temporale; 3 - Selettore delle fasi per tutti i grafici; 4 - Notifica di ricezione di un nuovo messaggio tramite il *websocket*; 5 - Selettori delle fasi da visualizzare nel grafico corrispondente.

home page dell'applicazione *web* durante l'acquisizione e visualizzazione di valori numerici casuali. Dall'interfaccia dell'applicazione si seleziona il nodo IoT (Fig. 4.12-1) del quale si vogliono visualizzare le misure effettuate nell'intervallo temporale indicato dai *radio buttons* (Fig. 4.12-2). Le nuove misure acquisite dal nodo IoT selezionato vengono visualizzate in *real-time* sull'applicazione *web* (Fig. 4.12-4) grazie all'utilizzo del *websocket* (Cap. 3.4.3) che riceve costanti aggiornamenti dal gestionale, il grafico viene aggiornato con i nuovi valori istantaneamente. E' inoltre possibile selezionare quali fasi visualizzare sui grafici, utilizzando le *checkbox* (Fig. 4.12-4), oppure si possono selezionare le fasi per ogni singolo grafico (Fig. 4.12-5).

4.2.8 Installazione

Questo sistema di monitoraggio energetico costituito dal *power meter*, nodo IPv6 e *border router*, è stato installato per verificarne il corretto funzionamento presso il Laboratorio di Tecnologia del DIISM dell'Università Politecnica delle Marche. Il *power meter* è stato collegato alla linea trifase che alimenta una pompa idraulica utilizzata per le prove di trazione dei materiali effettuate da una macchina *Material Test System* (MTS) (Fig. 4.10). Utilizzando la configurazione (2) (Fig. 4.9) del *power meter* vengono mostrate sull'applicazione *web* gli andamenti di tensione, corrente e potenza, cam-

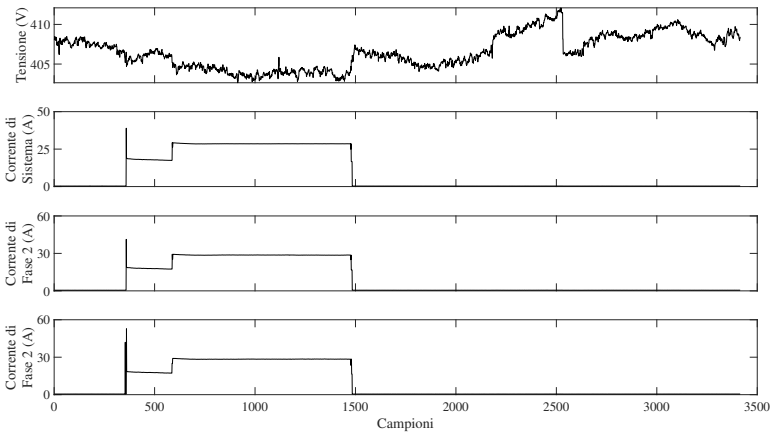

Figura 4.13: Andamento della tensione e delle correnti relative ai consumi energetici della pompa idraulica durante una prova di trazione con la macchina MTS del Laboratorio di Tecnologie.

pionati ogni 5 secondi. I dati presenti all'interno del *database* possono essere scaricati in formato *.csv* e analizzati con il *tool* di calcolo matematico MATLAB. Gli andamenti della tensione di sistema e delle correnti, misurate durante una prova di trazione, sono mostrati nella Fig. 4.13, mentre i corrispondenti andamenti delle potenze e del fattore di potenza sono mostrati nella Fig. 4.14.

Con l'installazione di questo sistema di monitoraggio energetico, basato sul paradigma IoT, è stato possibile valutare in *real-time* lo stato dei consumi energetici della pompa idraulica durante le prove di laboratorio. L'analisi degli andamenti dei consumi visualizzati sull'applicazione *web* durante i *test* di trazione hanno permesso anche di controllare il corretto stato di funzionamento della pompa idraulica. Questa rete IoT si configura come un utile strumento di misurazione per determinare lo stato dei processi industriale (in questo caso un *test* di trazione) nel contesto dell'*Industry* 4.0 [41].

4.3 Monitoraggio sismico

Con questa applicazione si vogliono monitorare gli eventi sismici che interessano la Torre della Facoltà di Ingegneria dell'Università Politecnica delle Marche. In questo progetto sviluppato in concerto con l'Istituto Nazionale di Geofisica e Vulcanologia (INGV), viene studiato il prototipo del sistema di monitoraggio sismico. Il prototipo è composto da un'unità di *sensing* e da un sistema *embedded*.

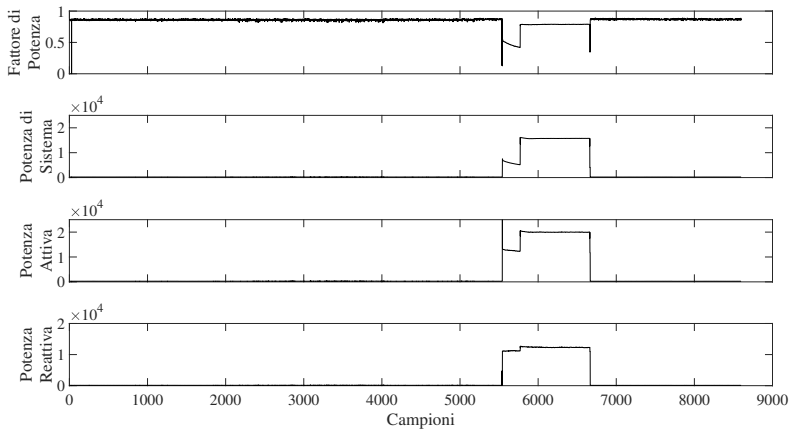

Figura 4.14: Andamento delle potenze (attiva, reattiva e apparente) e del fattore di potenza relativi ai consumi energetici della pompa idraulica durante una prova di trazione con la macchina MTS del Laboratorio di Tecnologie.

4.3.1 Unità di *sensing*

L'accelerometro ADXL355 costituisce l'unità di *sensing*, cioè è il sensore incaricato di rilevare le variazioni di accelerazione del terreno. Il sistema *embedded* è composto da una RaspberryPI 2 Model B che acquisisce i valori di accelerazione dal sensore ADXL355 connesso alla *board* tramite *bus* I^2C. I dati raccolti sono memorizzati localmente in formato MSEED e l'accesso ai dati viene fornito da un *ringserver* dedicato. In Fig. 4.15 viene mostrato il prototipo composto dalla RaspberryPi 2 Model B collegata, tramite il *bus* I^2C, all'accelerometro ADXL355 montato su una base di ottone di 500g.

La prima operazione da eseguire è l'abilitazione del *bus* I^2C della RaspberryPi, e ciò avviene accedendo alle funzionalità del *kernel* Linux con il comando "*sudo raspi-config*", e attivando l'interfaccia I^2C. Ora si può rilevare su quale indirizzo è collegato l'ADXL355 con il comando "*sudo i2cdetect -y {0|1}*". Va selezionato 0 oppure 1 a seconda del *bus* I^2C che è stato utilizzato per la connessione con il sensore (GPIO 27,28 $\rightarrow$ *bus* 0, GPIO 3,5 $\rightarrow$ *bus* 1).

Nella Fig. 4.16 viene mostrato il diagramma della libreria Python realizzata per questa applicazione di monitoraggio. La classe ADXL355 si occupa dell'acquisizione delle misure di accelerazione effettuate dal sensore connesso alla RaspberryPi tramite il *bus* I^2C. Con l'utilizzo di uno dei quattro metodi messi a disposizione si possono leggere i registri relativi a uno degli assi del sensore (x, y, z), oppure, di tutti i tre gli assi contemporaneamente. Le misurazioni ottenute vengono poi gestite da uno *script* Python che si occupa della loro scrittura in formato MSEED, sfruttando le API

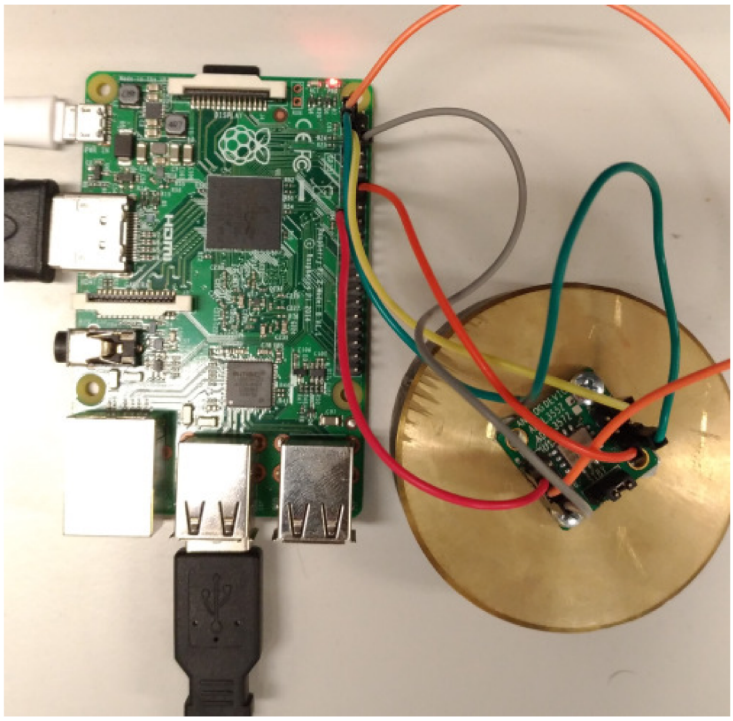

Figura 4.15: Prototipo del sistema di monitoraggio sismico costituito da Raspberry Pi Model B (sx) connessa con *bus* I^2C all'accelerometro ADXL355 (dx) su cilindro di ottone.

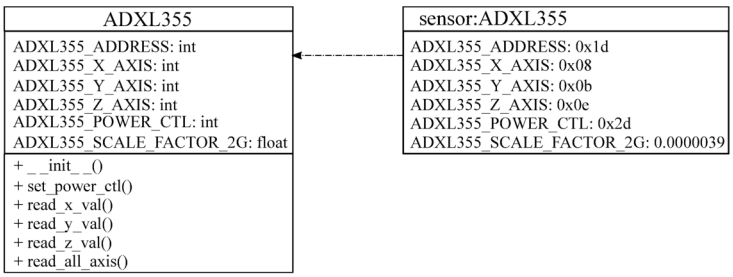

Figura 4.16: Diagramma della libreria Python realizzata per acquisire i valori di accelerazione dal sensore ADXL355

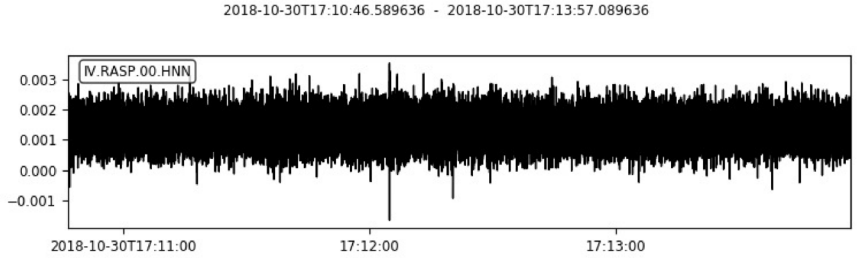

Figura 4.17: Visualizzazione con obspy dei valori di accelerazione misurati sull'asse X dell'ADXL355 e salvati in MSEED.

fornite dalla libreria obspy[11].

4.3.2 MiniSEED

Il MiniSEED è un formato per la memorizzazione di dati sismici e rispetto alla sua versione estesa (SEED) non contine l'*header* di controllo, ma soltanto dei metadati per l'identificazione del segnale sismico e una serie di dati temporale. Utilizzando questo formato vengono salvati *file* della dimensione tipica di 512-4096 Byte, che possono essere condivisi tramite un *ringserver* con il protocollo *seedlink*. I metadati contenuti nell'*header* del pacchetto MSEED servono per indicare la stazione di acquisizione degli eventi sismici e per indicare alcuni parametri del campionamento:

- *network* (nome della rete);
- *station* (nome della stazione di monitoraggio);
- *location* (indicazione della locazione);
- *channel* (canale x, y, o z);
- ntps (numero di campioni);
- *dataquality*;
- *sampling_rate* (frequenza di campionamento);
- *starttime* (istante del primo campione);

La libreria obspy permette di scrivere gli *stream* di dati acquisiti attraverso la libreria ADXL355 (Fig. 4.16) utilizzando opportune API. Per prima cosa va definito l'*header* del *file* MSEED che contiene i metadati necessari alla corretta interpretazione dello *stream*. Ciò viene fatto con l'utilizzo di un dizionario[12] in Python come

[11] Si tratta di un progetto *open-source* per la gestione di dati sismici.
[12] Si tratta di un tipo di dato in Python formato da coppie di chiave - valore.

mostrato nella riga 1 dello *snippet*. Nella riga 2 viene creato lo *stream* definendo una traccia che contiene i dati di accelerazione e i corrispondenti metadati. Infine alla riga 3 lo *stream* viene scritto sul *file* MSEED specificando il formato del *file* stesso, la codifica per i dati, e la lunghezza in Byte.

```
1    meta={'network':'IV', 'station':'RASP', 'location':'00', '
     channel':'HNN', 'npts': nPoints, 'sampling_rate':250, 'mseed'
     :{'dataquality':'D'}, 'starttime':starttime}
2    stream=Stream([Trace(data=data, header=meta)])
3    stream.write('pathToFile/nome.mseed', 'format'='MSEED', '
     encoding':'FLOAT_32', 'reclen'=512)
```

Codice 4.18: Scrittura di un *file* MSEED con la libreria Python obspy.

La Fig. 4.17 mostra il tracciato sismico relativo all'asse x del sensore ADXL355, memorizzato all'interno di un *file* MSEED e stampato grazie alle API della libreria obspy. Nell'angolo in alto a sinistra della figura sono mostrati alcuni dei metadati relativi al *file* MSEED mostrato, *network*: "IV", *station*: "RASP", *location*: "00", *channel*: "HNN". Queste *label* per i metadati sono state scelte in modo arbitrario per verificare il corretto funzionamento della procedura di acquisizione del valori e di scrittura in formato MSEED.

4.3.3 Accesso ai dati

L'ultimo componente utilizzato per il sistema di monitoraggio sismico è il *ringserver* che mette a disposizione degli utenti remoti gli *streams* di dati salvati nella RaspberryPi in formato MSEED. Il *ringserver* è un generico *server* ad anello che viene utilizzato per fornire dati di *streams* in tempo reale. Il codice sorgente del *server* può essere scaricato all'indirizzo "*https://github.com/iris-edu/ringserver/releases*", e una volta estratto dal *tarball* deve essere compilato con il comando "*make*". Occorre ora impostare almeno una configurazione minima, nel *file* ring.conf, per consentire al *ringserver* di operare correttamente:

```
1 RingDirectory ring
2 DataLinkPort 16000
3 SeedLinkPort 18000
4 ServerID "XX Seismic Network"
5 TransferLogDirectory tlog
6 TransferLogRX 0
7 MSeedScan /data/miniseed/
8 StateFile=scan.state
9 InitCurrentState=y
```

Codice 4.19: Esempio di configurazione del *ringserver*.

La riga 1 indica la cartella nella quale saranno memorizzate le informazioni relative allo stato del *buffer* del *server*. Nelle righe 2 e 3 sono specificate le porte TCP per il monitoraggio del *server* e per l'instaurazione delle connessioni SeedLink. La riga 4

assegna un nome al *ringserver* in uso, e le righe 5 e 6 indicano le cartelle dove vengono memorizzati *log* prodotti dal *server*. Invece nella riga 7 viene indicato il percorso dove sono salvati i *files* MSEED realizzati dallo *script* Python precedente, che contengono i dati sismici rilevati. Mentre nella riga 8 viene indicato un particolare *file* che ha la funzione di registrare i *files* MSEED già identificati dal *server*. Ciò evita che all'avvio il *ringserver* scansioni per intero la cartella indicata a riga 7 alla ricerca di nuovi MSEED. La riga 9 evita la scansione dei *files* se non è presente lo StateFile. Conclusa questa minima configurazione, si può avviare il *server* con il seguente comando:

```
1 sudo ./ringserver ring.conf &
```

Codice 4.20: Comando di avvio del *ringserver*.

I *clients* remoti possono collegarsi alla porta $SeedLinkPort = 16000$ dell'*host* che ospita il *ringserver* per visualizzare gli *streams* attraverso il protocollo SeedLink. Parte di questo lavoro è stato utilizzato per realizzare il progetto presentato alla conferenza internazionale ISCT 2019 tenuta ad Ancona [89].

Capitolo 5

Prestazioni

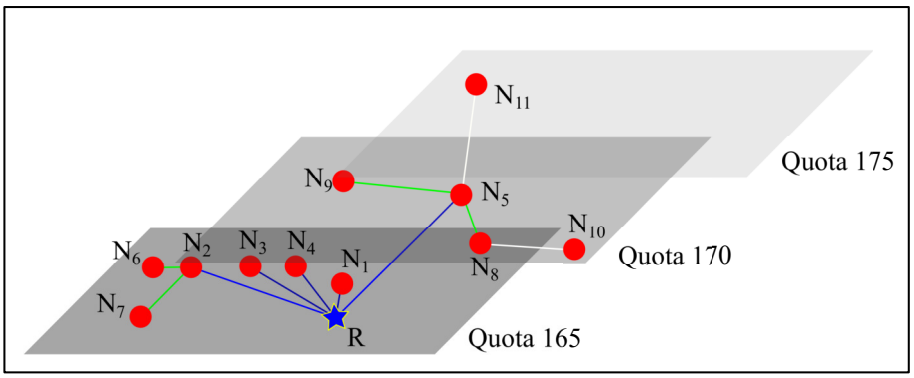

Figura 5.1: Distribuzione dei nodi IoT all'interno della Facoltà di Ingegneria. In blu i *link* a 1 *hop* di distanza dal nodo *root*, in verde a 2 *hop* e in bianco a 3 *hop*.

In questo capitolo verranno descritte le prestazioni ottenute dalla rete 6LoWPAN utilizzata nei diversi scenari applicativi, e le prestazioni ottenute dal *database* MongoDB utilizzato per memorizzare i dati raccolti dai nodi IoT.

5.1 Prestazioni di rete

Per valutare la latenza della rete IPv6 sono stati condotti dei *test* per misurare la *packet loss* e la *rate* a diversi livelli della pila ISO/OSI. Per poter effettuare queste misure, è stata utilizzata una rete IPv6 pubblica (2001:760:2e0d:130::/64) di dispositivi *wireless* installata all'interno del DII e del DIISM, composta da un nodo *root* e da 11 nodi *smart* [40]. I nodi di questa rete sono stati distribuiti su tre piani all'interno dei dipartimenti della facoltà di Ingegneria, in modo tale da poter valutare al meglio le prestazioni ottenibili sotto diverse condizioni di propagazione del segnale radio.

Nella Fig. 5.1 vengono mostrate le posizioni dei nodi *wireless* all'interno dei dipartimenti. Il nodo *root* (R) è contraddistinto da una stella blu, mentre tutti gli altri nodi

Tabella 5.1: Lista dei nodi 6LoWPAN distribuiti all'interno della Facoltà. I piani sono identificati dalla quota a livello del mare.

Nodo	Hop	Distanza (m)	Piano (m)
R	*root*	-	165
N_1	1	0.10	165
N_2	1	30.59	165
N_3	1	21.80	165
N_4	1	12.90	165
N_5	1	5.00	170
N_6	2	31.83	165
N_7	2	37.70	165
N_8	2	23.97	170
N_9	2	35.60	170
N_{10}	3	29.40	170
N_{11}	3	41.00	175

(N_i) da un cerchio rosso. I *links* di colore blu rappresentano nodi a una distanza di 1 *hop* dal nodo *root*, i *links* verdi di 2 *hop* e i *links bianchi* di 3 hop. Mentre nella seguente tabella (Tab. 5.1) sono riportati tutti i 12 nodi che appartengono a questa rete IPv6, e vengono caratterizzati dalla loro distanza in metri dal nodo *root*, dalla loro distanza in *hop* e dal loro piano all'interno della Facoltà di Ingegneria.

Per valutare le prestazioni di questa rete in termini di latenza, *packet loss* e *rate*, sono stati utilizzati il *tool* ping6 di Linux e uno *script* Python per il calcolo delle prestazioni a livello applicazione. Per valutare latenza e *packet loss* sono state trasmesse 1000 richieste ICMPv6 e 1000 richieste CoAP ad ognuno dei 11 nodi della rete 6LoWPAN, mentre, per valutare la *rate* sono state inviate 200 richieste ICMPv6 con *payload* crescente da 8 Byte a 128 Byte con passo di 4 Byte, cioè un totale di 6200 richieste per nodo.

Nella Fig. 5.2 sono mostrati i risultati della misura di latenza per ogni singolo nodo della rete in termini di *Round Trip Time* (RTT)[1]. Dalla figura si possono distinguere 3 gruppi di nodi che hanno circa lo stesso RTT, e confrontando i nodi di questi gruppi con la Tab. 5.1 si può verificare che questi coincidono con i nodi con la stessa distanza in *hop* dal nodo *root*. Questo risultato è ovviamente atteso, perché per i nodi ad una distanza in *hop* > 1 dal nodo *root* bisogna aggiungere il tempo necessario all'operazione di *forwarding* del messaggio.

Questa misura di latenza del rete IoT è stata ripetuta nuovamente aumentando la potenza in trasmissione dei nodi *wireless* di 10 dB, passando dai -30 dBm della prova precedente ai -20 dBm. I risultati ottenuti sono riportati in Tab. 5.3 e raffigurati in Fig. 5.3.

[1]Si tratta del tempo che intercorre tra l'invio del messaggio e la ricezione della risposta.

Tabella 5.2: Risultati delle misure di latenza e *packet loss* sulla rete 6LoWPAN.

Node	Packet loss (%)	RTT (ms)
N_1	0.5	94.5
N_2	26.1	99.9
N_3	47.3	105.8
N_4	0.9	94.9
N_5	18.8	95.3
N_6	23.5	176.4
N_7	92.2	186.3
N_8	73.7	175.8
N_9	44.4	180.9
N_{10}	91.1	259.7
N_{11}	86.5	254.6

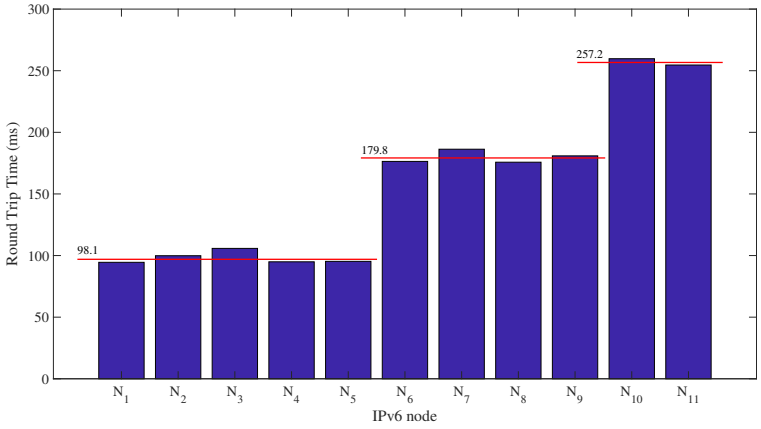

Figura 5.2: Misure di RTT per gli 11 nodi della rete 6LoWPAN.

Tabella 5.3: Risultati delle misure di latenza e *packet loss* e variazione di *hop* dei nodi sulla rete 6LoWPAN, trasmettendo con un potenza pari a -20 dBm.

Node	Hop	Packet loss (%)	RTT (ms)
N_1	1	0.2	94.7
N_2	1	22.9	99.3
N_3	1	29.6	98.5
N_4	1	1.0	94.4
N_5	1	18.1	96.6
N_6	1	20.6	100.3
N_7	1	19.3	95.0
N_8	2	36.4	176.7
N_9	2	47.0	175.3
N_{10}	2	61.8	190.5
N_{11}	2	38.3	185.8

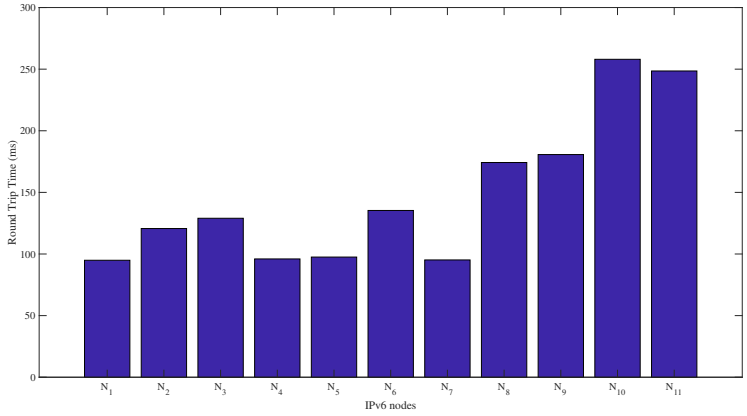

Figura 5.3: Misure di RTT per gli 11 nodi della rete 6LoWPAN con una potenza trasmissiva di -20 dBm.

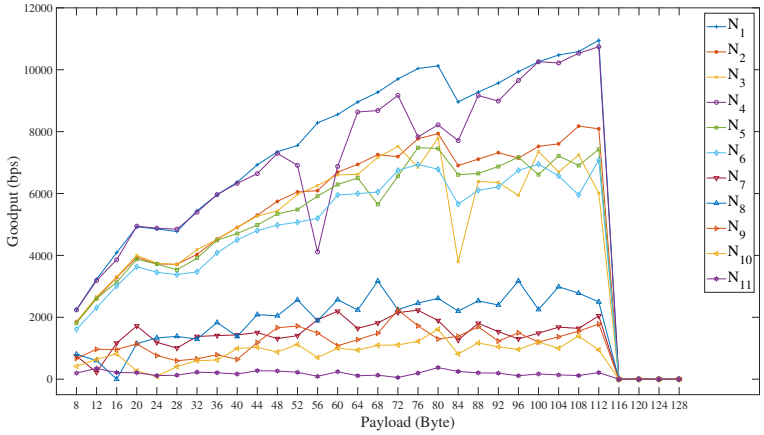

Figura 5.4: Misure di *goodput* per gli 11 nodi della rete 6LoWPAN.

Il risultato principale di questa seconda misura consiste nella variazione del numero di *hop* per raggiungere il nodo *root*. Ciò è dovuto all'aumento di potenza trasmissiva dei nodi della rete 6LoWPAN che ha causato una variazione delle rotte calcolate dalla OF0 del protocollo RPL (vedi Cap. 3.3.3) per l'instradamento dei pacchetti. Questa variazione implica una riduzione della *packet loss* e della latenza per i nodi che si sono avvicinati al nodo *root*. Per esempio, per quanto riguarda i nodi N_6 e N_7 si ha una sostanziale riduzione della *packet loss* e del RTT dei messaggi ICMPv6 trasmessi.

Il secondo tipo di misura effettuata sulla rete è stata la valutazione della *rate* in termini di *goodput*[2]. Per calcolare questo parametro è stato utilizzato il *tool* ping6 di Linux, per trasmettere richieste ICMPv6 ai nodi della rete 6LoWPAN, con l'opzione (-s) che permette di far variare la dimensione del *payload* dei messaggi inviati. Considerando che la dimensione K del *payload* varia da 8 Byte a 128 Byte con passo di 4, che il numero di messaggi inviati a ogni nodo è pari a $M = 200$, e il tempo da è pari a circa metà del RTT stimato dal *tool* ping6, risulta che:

$$goodput_K = \frac{1}{M} \sum_{i=1}^{M} \frac{8 * payload\_size_K}{\frac{time_i}{2}} \quad (bps) \tag{5.1}$$

Nella Fig. 5.4 viene mostrato il *goodput* per gli 11 nodi della rete, in cui il nodo N_1 raggiunge il valore massimo di circa 11 kbps. Invece il nodo N_{11} che è il più lontano dal nodo *root*, ha il *goodput* più basso che è di circa 200 bps. La valutazione del *goodput* è stata ripetuta andando ad aumentare la potenza trasmissiva di 10 dBm, e

[2] *Goodput Vs. Throughput.* Il primo termine si riferisce alla quantità di dati utili trasmessi per unità di tempo. Il secondo invece si riferisce alla quantità di dati trasmessi sul *link*, di conseguenza tiene conto degli *overhead* dei protocolli e delle eventuali ritrasmissioni.

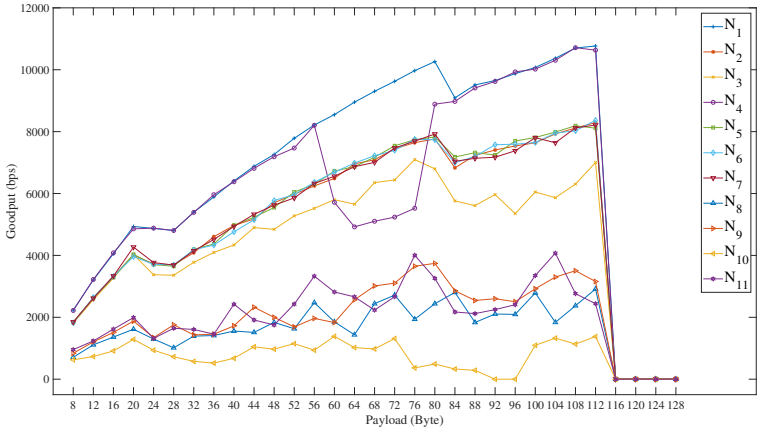

Figura 5.5: Misure di *goodput* per gli 11 nodi della rete 6LoWPAN con una potenza trasmissiva di -20 dBm.

i risultati sono mostrati in Fig. 5.5. Aumentando la potenza trasmissiva di 10 dB si ha una riduzione della *packet loss* (Tab. 5.2 e Tab. 5.3) e una modifica delle rotte che riduce il numero di *hop* dal nodo *root*, di conseguenza si ottiene una migliore capacità trasmissiva dei nodi. Infatti si può notare che il nodo N_{11} è passato da una *goodput* di 200 bps a una *goodput* di circa 2.5 kbps.

Dai risultati ottenuti è chiaro che l'aumento della potenza trasmissiva comporta un generale aumento delle prestazioni della rete, sia in termini di *packet loss* che di latenza e di *goodput*. Durante il corso del dottorato si sono cercate soluzioni alternative per migliorare le prestazioni della rete 6LoWPAN senza dover aumentare la potenza trasmissiva e di conseguenza i consumi energetici dei dispositivi *wireless*. Per questo motivo è stato sviluppato il prototipo del *tool Man-In-the-Middle Iot Computing* (MIMIC), che ha lo scopo di andare ad aumentare le risorse e le prestazioni dei nodi IoT utilizzando la virtualizzazione (vedi Cap. 5.2).

5.2 *Man-In-the-Middle Iot Computing*

Questo *tool* rappresenta un primo tentativo in assoluto [44] di integrare una ben nota tecnica di attacco informatico (*Man-In-The-Middle* (MIM) o MITM) ad un sistema di *Fog Computing* per l'IoT. L'attacco MIM sembra apparire per la prima volta nella comunità scientifica nel 1992 in [90], descritto come un attacco per intercettare lo scambio di chiavi in un canale non protetto. Più in generale il MIM, è un attaccante nascosto che passivamente ascolta il canale osservando tutti i messaggi scambiati tra gli utenti legittimi. Lo scopo MIM non è soltanto quello di osservare i messaggi, ma

anche di alterarli o rispondere a questi senza essere identificato. I danni che questo attaccante può causare all'interno di una rete di comunicazione sono molti. Si pensi ai diversi attacchi di *spoofing* che possono essere condotti ai diversi protocolli di comunicazione nelle reti IP, come l'ARP *spoofing*, il DNS *spoofing*, l'IP *spoofing*, *etc.* Il *tool* MIMIC si basa proprio sull'IP *spoofing*, cioè sulla capacità dell'attaccante di modificare l'*header* del pacchetto IP alterandone ad esempio il campo *source address*.

Questo innovativo sistema si basa sull'utilizzo del paradigma del *Fog Computing*, ovvero la capacità di portare le risorse virtuali fornite dal *Cloud Computing* vicino alla sorgente dei dati, cioè in questo caso la rete IoT. Nondimeno il *Fog Computing* può collaborare con le piattaforme *Cloud* al fine di filtrare i dati che vanno elaborati da piattaforme ad elevata capacità di calcolo, o che richiedono spazi di memorizzazione molto grandi. La necessità di utilizzare il *Fog Computing* è dovuta al fatto che il MIMIC deve avere a disposizione i dati raccolti dalla rete IoT per poter generare le risposte richieste dai *clients*. In questo modo il MIMIC è in grado di simulare (virtualmente) il nodo IoT e di rispondere al suo posto alle richieste. Inoltre il dispositivo *Fog* ha risorse *hardware* maggiori rispetto ai nodi *constrained* della WSN, di conseguenza si possono andare ad aggiungere servizi e *Quality of Service* (QoS) che sono previste per le reti 6LoWPAN.

Il vantaggio principale di questo *tool* rispetto ad altre soluzioni come quelle offerte da AWS IoT Core e Hub IoT di Microsoft, che utilizzano rispettivamente degli oggetti virtuali (*shadow* [91] e *twin* [92]) con un indirizzo specifico:

- https://endpoint/things/thingName/**shadow** (AWS)
- https://fully-qualified-iothubname.azure-devices.net/**twins**/id?api-version=2018-06-30 (Azure)

è che rimane del tutto trasparente dal punto di vista dell'utente e del sistema IoT che va ad assistere. Infatti il MIMIC non è indirizzabile, ovvero non ha un indirizzo IP che lo identifica all'interno della rete, e non è nemmeno identificabile da un *Uniform Resource Identifier* (URI). Di fatto risulta essere invisibile, proprio come si comporterebbe un attaccante MIM. Questa capacità di operare nascosto, consente al MIMIC di forgiare delle risposte come se fossero create dallo stesso nodo della rete, senza dover ricorrere a specifiche APIs o URI per interagire con esso.

Il principio di funzionamento del *tool* MIMIC è rappresentato in Fig. 5.6. Per prima cosa vengono caricati i dati raccolti dai nodi della rete IoT all'interno del dispositivo *Fog* (1), con diverse possibili modalità, cioè richieste inviate dal dispositivo *Fog* ai nodi, o messaggi di *upload* inviati autonomamente dai nodi. Più importante è la gestione delle richieste degli utenti da parte del MIMIC (2), (3), (4). Quando il MIMIC identifica un pacchetto IP (2) destinato ad uno dei nodi della rete IoT a cui è associato un nodo virtuale, il pacchetto viene rimosso dalla *chain* di *forwarding* del dispositivo *Fog* (3) e viene preso in carico dal MIMIC stesso. Questo *tool* non farà altro che inter-

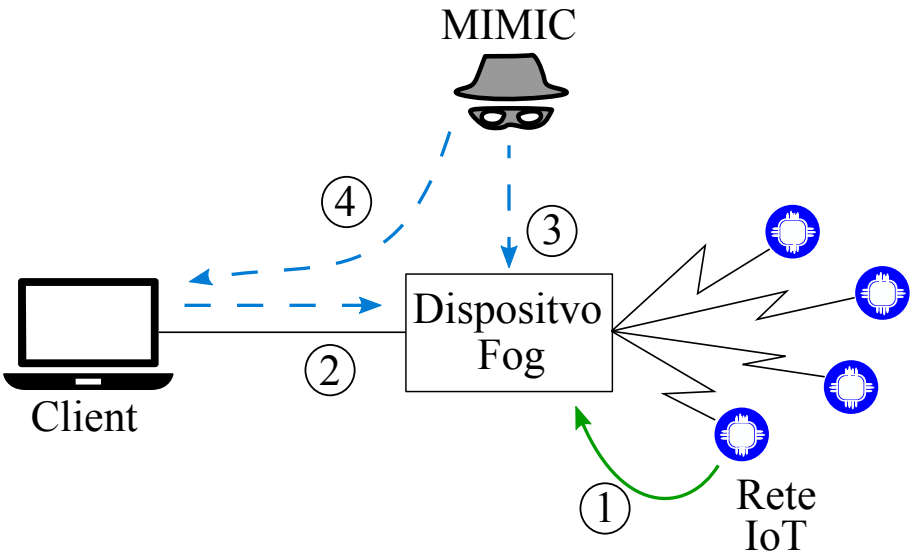

Figura 5.6: Schema di funzionamento del *tool* MIMIC.

pretare la richiesta ricevuta e fornire la risposta (4) recuperando i dati dal dispositivo *Fog* con cui collabora.

Così facendo il MIMIC va ad isolare la rete IoT dalla *Wide Area Network* (WAN), riducendo in questo modo le trasmissioni all'interno della WSN con un conseguente risparmio in termini energetici dei singoli nodi. Inoltre, la possibilità di utilizzare nodi virtuali permette di raggiungere prestazioni e di aumentare i servizi implementabili sui nodi *constrained*. Infatti in Fig. 5.7 viene mostrato un confronto tra il *goodput* di un nodo IoT e il *goodput* di un nodo virtuale. E' evidente dalla figura che le prestazioni del nodo virtuale sono nettamente migliori di quelle offerte dal nodo IoT. Ciò è dovuto allo *standard* 802.15.4 che utilizza una PSDU di 127 Byte, e di conseguenza quando il *payload* del pacchetto raggiunge il limite dello *standard* la *rate* cala a 0 bps. Invece per quanto riguarda il nodo virtuale, il suo *goodput* ha un andamento monotono crescente che raggiunge i 125 kbps in corrispondenza di un *payload* di 1.2 kB. Il nodo virtuale non è limitato dallo *standard* IEEE 802.15.4 e la comunicazione tra il dispositivo *Fog* e il MIMIC avviene in locale, e le risposte sono trasmesse direttamente in *frames ethernet* (IEEE 802.3) che permettono di ottenere *rate* maggiori del IEEE 802.15.4.

5.3 Prestazioni DB

Il sistema di monitoraggio realizzato nell'ambito del tema di ricerca ha coinvolto anche l'installazione di un *DataBase Management System* (DBMS) per la memorizzazione e gestione dei dati misurati dalle diverse tipologie di sensori utilizzati nei vari

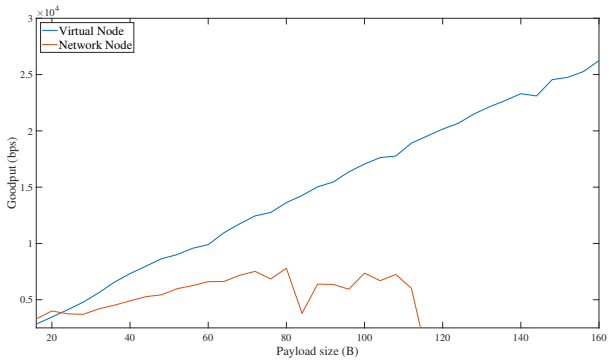

Figura 5.7: Confronto della *rate* tra un nodo IoT reale e il nodo virtuale del *tool* MIMIC.

scenari applicativi. Per scegliere al meglio quale delle diverse soluzioni di *storage* andare ad implementare all'interno del *border router* e del *server* remoto, che costituiscono parte dell'architettura del sistema di monitoraggio sviluppato, sono stati confrontati tre differenti DBMS [82], uno relazionale e due non-relazionali.

I *database* relazionali sono caratterizzati da una struttura rigida per la memorizzazione dei dati, cioè le tabelle mentre, i non-relazionali, sono distinti da uno schema più flessibile e dinamico come i grafi, i documenti o le coppie chiave-valore. Oltre alla struttura che definisce come vengono memorizzati i dati, la differenza risiede anche nel linguaggio utilizzato per accedere o caricare i dati nel DBMS. Questi due linguaggi sono lo *Structured Query Language* (SQL) per i relazionali e il *Non Structured Query Language* (NoSQL) per i non-relazionali, che hanno le proprie sintassi e APIs per accedere alle funzionalità dei rispettivi DBMS.

I *database* che sono stati selezionati come possibili candidati per il sistema di monitoraggio sono il *database* relazionale MariaDB, e i due non-relazionali MongoDB e CouchDB. Il MariaDB è una DBMS *open source* sviluppato dagli stessi creatori di MySQL, e infatti mantiene un'elevata compatibilità con quest'ultimo. Il *client* utilizzato per accedere al *server* di MariaDB è compatibile con MySQL, e allo stesso lo sono la porta IP e le API per la gestione del DBMS. Inoltre i *file* binari che contengono i dati memorizzati sono compatibili con MySQL.

MongoDB è un valido rappresentante della categoria dei DBMS non-relazionali, anch'esso caratterizzato dalla licenza *open-source* e con i dati memorizzati in una struttura detta *document*, che a sua volta è raccolta nelle cosidette *collections*. Il *document* è un oggetto JSON[3], cioè un formato a coppia chiave-valore per lo scambio di dati, facilmente interpretabile dall'uomo e facile da generare e analizzare dalle macchine. Il secondo *database* non-relazionale scelto per i test è CouchDB. Anche questa

[3] *Java Script Object Notation.*

Tabella 5.4: Caratteristiche degli *hosts* usati per i test sui DBMS.

Dispositivi	Architettura CPU	Clock	RAM
Laptop	x86_64	3100 MHz	8 GB
Macchina Virtuale	x86_64	2660 MHz	4 GB
Raspberry	armv6l	700 MHz	0.5 GB

Singolo *Client*	Da 1 a 15 *clients* concorrenti
scrittura di una misura 10000 volte	scrittura di una misura 10000 volte
lettura di una misura 10000 volte	lettura di una misura 10000 volte
100 misure scritte 1000 volte	100 misure scritte 1000 volte
blocco di 100 dati letti 1000 volte	blocco di 100 dati letti 1000

Tabella 5.5: Operazioni effettuate per valutare le prestazioni dei DBMS.

soluzione memorizza i dati sotto forma di documenti JSON, ma con la particolarità di poter effettuare le *queries* come richieste HTTP.

Oltre ai diversi tipi di DBMS sono stati considerati anche tre differenti tipologie di *host* per l'installazione del *database server*. Per questo motivo sono stati presi in considerazione un *laptop*, una macchina virtuale installata su un *server* dell'Università Politecnica delle Marche e una Raspberry model B per rappresentare un possibile *border router* della rete IoT. Le caratteristiche di questi dispositivi sono riportati in Tab. 5.4.

Le prestazioni di questi DBMS sono state valutate misurando il tempo impiegato per eseguire operazioni di lettura e di scrittura sui rispettivi *databases* e su le differenti classi di dispositivi *hardware*. Uno *script* in Python si occupa dell'esecuzione delle *queries* ai *databases* utilizzando i connettori (*client*) forniti dalle librerie Python, misurando il tempo che intercorre tra l'invio della *query* e dalla ricezione del messaggio di *ack*. Per ottenere una descrizione delle capacità dei *databases* nel contesto IoT, sono state effettuate le seguenti operazioni di lettura e scrittura eseguite da un singolo *client* o da un gruppo di *clients* concorrenti (Tab. 5.5). Le operazioni effettuate dai molteplici *clients* concorrenti servono per rappresentare l'inserimento dei dati nel *database* dai diversi nodi sensori della rete[4], e della richiesta di lettura da diversi possibili *host*.

Le seguenti figure mostrano i risultati ottenuti dalle prove descritte in Tab. 5.5. Questi risultati indicano che MariaDB offre le migliori prestazioni in scrittura (Figg. 5.8, 5.9, 5.10) rispetto ai due *database* non-relazionali. D'altra parte parte i due *databases* non-relazionali considerati offrono migliori prestazioni in termini di lettura

[4]Si consideri che nell'installazione presso la Cappella degli Scrovegni di Padova, il numero dei nodi sensori è pari a 5.

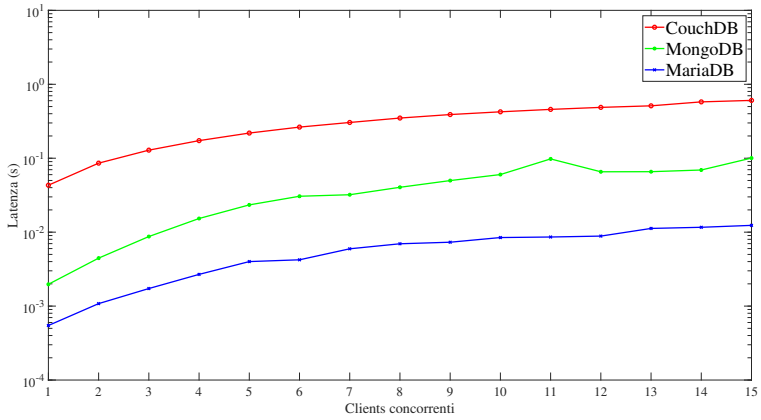

Figura 5.8: *Clients* concorrenti: media della latenza dell'operazione di scrittura di un singolo dato per 10000 volte, su un *laptop*.

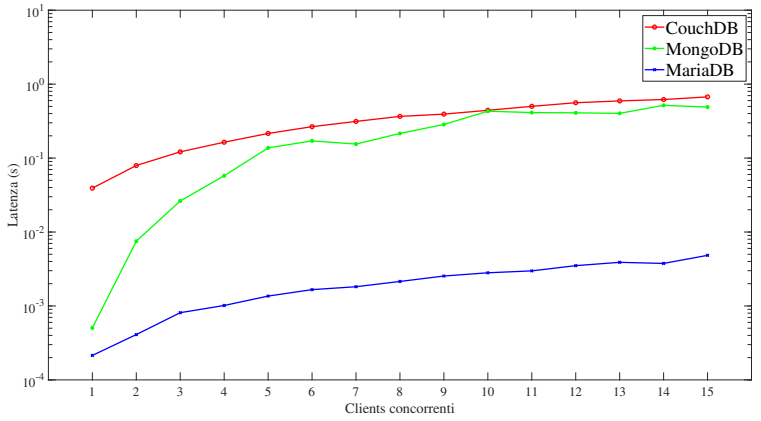

Figura 5.9: *Clients* concorrenti: media della latenza dell'operazione di scrittura di un singolo dato per 10000 volte, su una macchina virtuale.

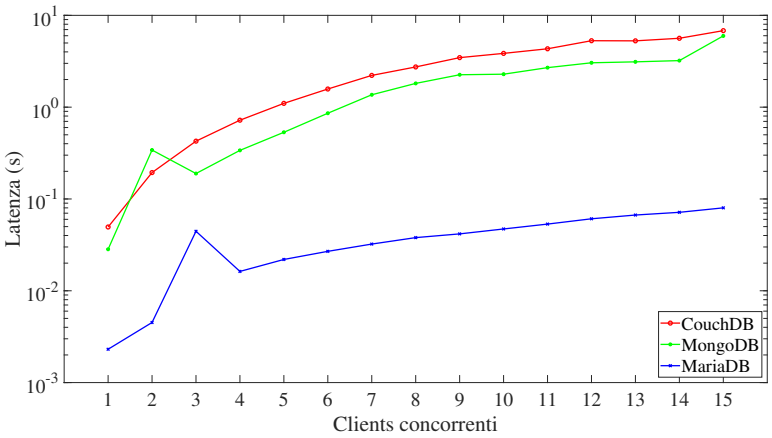

Figura 5.10: *Clients* concorrenti: media della latenza dell'operazione di scrittura di un singolo dato per 10000 volte, su una Raspberry.

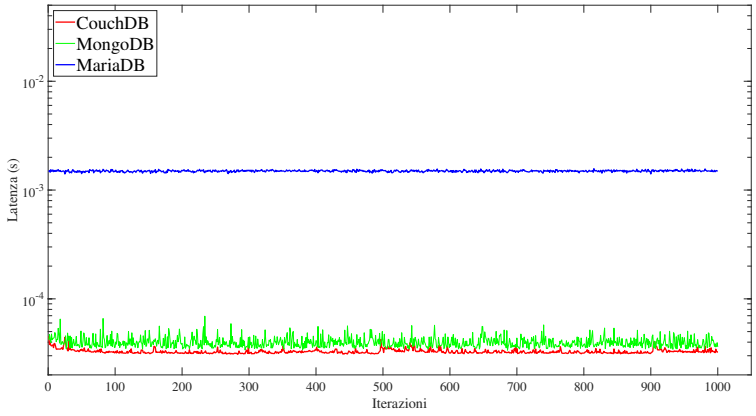

Figura 5.11: *Client* singolo: media della latenza dell'operazione di lettura di un blocco di dati per 1000 volte su un *laptop*.

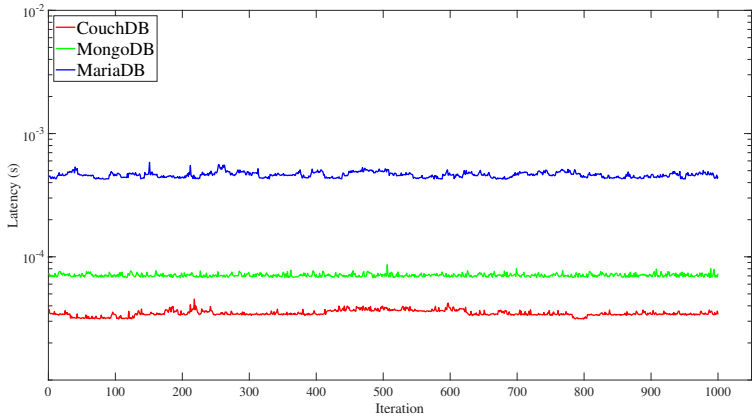

Figura 5.12: *Client* singolo: media della latenza dell'operazione di lettura di un blocco di dati per 1000 volte su una macchina virtuale.

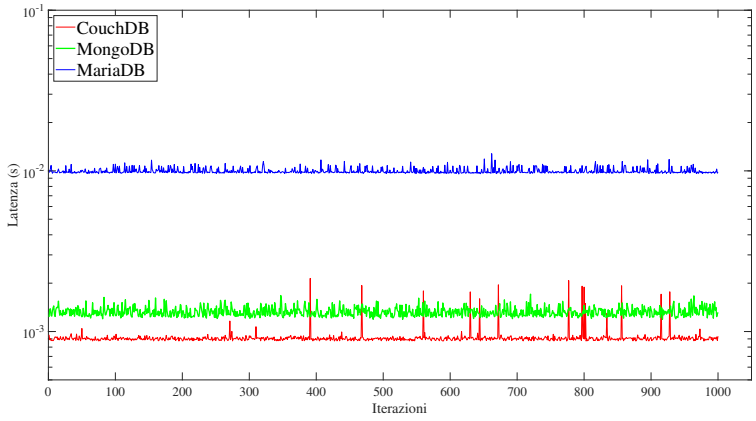

Figura 5.13: *Client* singolo: media della latenza dell'operazione di lettura di un blocco di dati per 1000 volte su una Raspberry.

di un blocco di dati memorizzati nel DB (Figg. 5.11, 5.12, 5.13). Nel contesto delle WSN si ha la necessità di memorizzare i dati raccolti dai diversi nodi di rete, che nelle prove effettuate sono rappresentati dai *clients* concorrenti, e dalla necessità di scaricare i dati raccolti per la loro elaborazione. Tuttavia non è stato possibile trovare un DBMS che soddisfa contemporaneamente questi due requisiti. Quindi per realizzare il sistema gestionale utilizzato per le diverse applicazioni di monitoraggio sviluppate durante il periodo di dottorato e descritte nei Capp. 4.1, 4.2 e 4.2, è stata scelta la soluzione non-relazionale offerta da MongoDB. Questa scelta dipende fortemente, oltre che dai risultati ottenuti, dalla elevata flessibilità garantita dallo schema di memorizzazione in documenti JSON. Infatti le WSN risultano essere dinamiche e la natura dei dati può cambiare durante il tempo di vita dell'applicazione, e le soluzioni non-relazionali si adattano molto bene a questa natura fortemente dinamica ed eterogenea dei sistemi IoT.

Capitolo 6

Machine Learning

In questa sezione verranno mostrati i risultati ottenuti dall'elaborazione dei dati di illuminamento memorizzati durante i tre anni di dottorato dal sistema installato presso la Cappella degli Scrovegni di Padova (Cap. 4.1). Lo scopo di questa applicazione è di verificare la possibilità di poter predire la misura di un nodo sensore a partire dalle misure degli altri nodi presenti nell'ambiente. La bontà di questo risultato dimostrerà la correlazione che sussiste tra i nodi della rete IoT, e sulla base della valori di correlazione tra le misure effettuate si possono programmare periodi di spegnimento dei nodi per ridurre i consumi energetici e così ottimizzare le prestazioni della WSN installata.

6.0.1 *Testbed*

I cinque nodi del sistema di monitoraggio inviano al *server* (locale e remoto) un campioni ogni 5 minuti, e da Settembre 2017 sono stati memorizzati più di 800000 messaggi all'interno del *database*. Questi dati raccolti possono essere analizzati dai algoritmi di *Machine Learning* (ML) per estrarne informazioni utili per la gestione del sito della Cappella degli Scrovegni.

Un aspetto importante, secondo la mia opinione, dell'applicazione di algoritmi per l'apprendimento automatico è la preparazione e rappresentazione dei dati, piuttosto che la scelta di uno specifico algoritmo. I dati rappresentati non correttamente possono causare errate interpretazioni da parte degli algoritmi di addestramento, restituendo risultati fuorvianti. Per questo motivo è stata dedicata molta attenzione alla preparazione dei dati prima della scelta e dell'implementazione degli algoritmi di ML.

6.0.2 Rappresentazione dei dati

I dati misurati dalla rete IoT sono memorizzati all'interno del database sotto forma di documenti JSON, e contengono numerose informazioni relative allo stato dell'ambiente all'interno del sito della Cappella degli Scrovegni. Ad esempio un documento memorizzato all'interno del *database* ha la seguente forma:

```
{
  "_id" : ObjectId("5a361a733d3425566ab4de75"),
  "omsgtime" : "2017-12-17 08,19,01",
  "omsgid" : 38693,
  "msgid" : 38693,
  "mbdata" : "[22,20,15,57,69,3988,24.10,29.69]",
  "msgtype" : "modbustot",
  "ipv6" : "aaaa::3035:3230:5234:840a",
  "realdata" : "
    [67.15606,3897.10144837,22.0,20.0,15.0,21.1,29.69]",
  "msgtime" : "2017-12-17 08,19,01"
}
```

Codice 6.1: Formato del documento JSON contenente i dati della Cappella degli Scrovegni.

Di questo documento JSON sono utili al fine dell'applicazione dell'algoritmo di apprendimento le due coppie chiave/valore, *real-data* e *msgtime*, che rappresentano rispettivamente il vettore dei campioni misurati dal nodo sensore e l'istante di campionamento. Il vettore dei campioni ([ill, CCT, R, G, B, T, H]) contiene informazioni relative a tutte le misure effettuate dal nodo sensore i-esimo, mentre al fine di studiare la distribuzione di luce all'interno dell'ambiente vengono considerati i soli campioni di illuminamento (ill). A partire da tutti i documenti presenti nel *database* viene create un *dataframe*[1] che contiene i valori di illuminamento (ill) e il rispettivo istante di campionamento ("msgtime").

Il *dataframe* ottenuto viene successivamente diviso per mesi, ottenendo così 25 nuovi *dataframes* (D_k con $k = 1, 2, ..., 25$) che verranno dati in pasto all'algoritmo di apprendimento automatico uno alla volta. L'algoritmo nella sua prima iterazione ($i = 1$) separa le misure del nodo S_i dagli altri nel *dataframe* k-esimo, e l'algoritmo utilizza questo vettore risultante come *output*[2] del modello di addestramento con supervisione ($y = S_i$). In questo modo l'algoritmo per $i = 1$ cerca proprio di predire le successive misure del nodo S_1, e nelle successive iterazioni $i \in (2, 3, 4, 5)$, proverà a predire le misure degli altri nodi addestrando nuovamente i parametri del modello. Considerando il caso $i = 1$ si avrà una matrice di *input*[3] X con la seguente forma:

$$X^{(1)}_{m\times 5} = \begin{bmatrix} 1 & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ 1 & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & x_{m,2} & x_{m,3} & x_{m,4} & x_{m,5} \end{bmatrix} \tag{6.1}$$

[1] Il *dataframe* è una particolare struttura dati bidimensionale (tabella) utilizzata dalla libreria Pandas di Python (https://pandas.pydata.org/). Con questa struttura si possono applicare semplicemente gli algoritmi di ML ai dati.
[2] Il vettore y è anche noto come *target*
[3] Il matrice X è anche nota come matrice delle *features*

E si ottiene il corrispondete vettore di *output* Y:

$$Y^{(1)}_{m\times 1} = [y_1 y_2 y_3 \dots y_m]^T \tag{6.2}$$

6.0.3 Modello di apprendimento

La matrice X e il vettore Y vengono successivamente divisi in due, il primo utilizzato per addestrare i parametri θ (*dataset* di *train*, 70%), il secondo per valutare i parametri ottenuti (*dataset* di *test*, 30%). Utilizzando la matrice di *input* e il vettore di *output* si possono così addestrare i parametri θ che saranno utilizzate per calcolare la previsione $\overline{y}$ con la seguente funzione lineare:

$$\overline{y} = \theta_0 \cdot 1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 = x^T\theta = \sum_{j=0}^{4} x_i\theta_i \tag{6.3}$$

Considerando la prima iterazione ($i = 1$) il dataframe k-esimo è stato separato nei due elementi $X_{44134\times 5}$ e $Y_{44134\times 1}$, i quali sono stati nuovamente divisi nei *dataset* di *train* ($X_{30893\times 5}$, $Y_{30893\times 1}$) e di *test* ($X_{13241\times 5}$, $Y_{13241\times 1}$). Al termine della prima iterazione ($i = 1$) sono stati effettuate le previsioni per il nodo sensore S_1, cioè l'algoritmo ha calcolato i parametri per il modello a regressione lineare (*Linear Regression* (LR)) e ha poi calcolato il vettore delle predizioni $\overline{y}$. Il risultato di questa previsione è riportato in Fig. 6.1, dove è stato ottenuto un punteggio pari a:

$$R^2(S_1) = \frac{\sum y_{test} - \overline{y}}{\sum y_{test} - \mathbb{E}[\overline{y}]} = 0.988 \tag{6.4}$$

Nella figure vengono mostrati con la curva nera il k-esimo *dataframe*, con i cerchi blu i valori del *dataset* di *test* ($Y_{13241\times 1}$), e infine con le circonferenze rosse si rappresentano i valori predetti $\overline{y}$ (Eq. 6.3). Questo risultato indica che è stato possibile predire le misure del nodo S_1 a partire dalle misure degli altri nodi (S_2, S_3, S_4, S_5) con un'elevata affidabilità ($R^2 = 0.988$, con $0 \leq R^2 \leq 1$).

6.0.4 Correlazione

Visti i risultati ottenuti è stata valutata la correlazione che sussiste tra le misure fatte dai 5 nodi della rete IoT nel k-esimo *dataframe* considerato. La Fig. 6.2 mostra proprio i risultati della correlazione tra le misure dei nodi e come era previsto sono presenti, oltre che nella diagonale principale ($Corr = 1$), altri valori di correlazione molto elevati ($Corr > 0.9$) che hanno reso possibile calcolare con successo le previsioni delle misure per il nodo S_1. Dato che il nodo S_1 è fortemente correlato con i valori misurati dai nodi S_3 ed S_5, la prova precedente è stata ripetuta, considerando una riduzione del *dataframe* k-esimo per eliminare parte della correlazione escludendo la componente relativa a S_3. Il risultato di questa operazione viene mostrato in

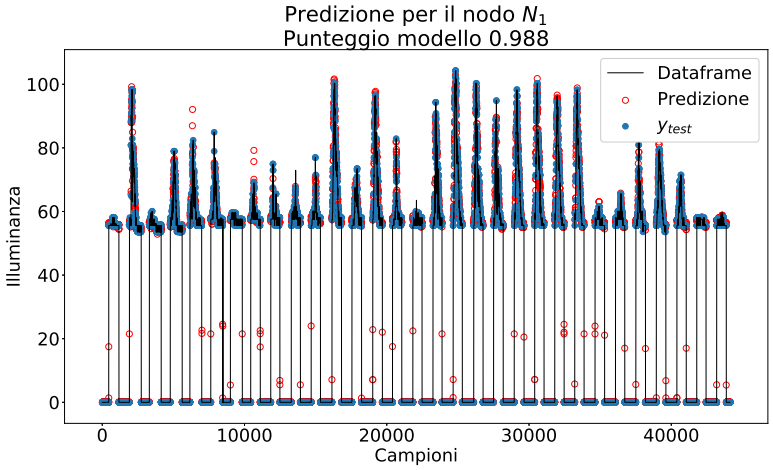

Figura 6.1: Predizione dei valori di illuminamento per il sensore S_1.

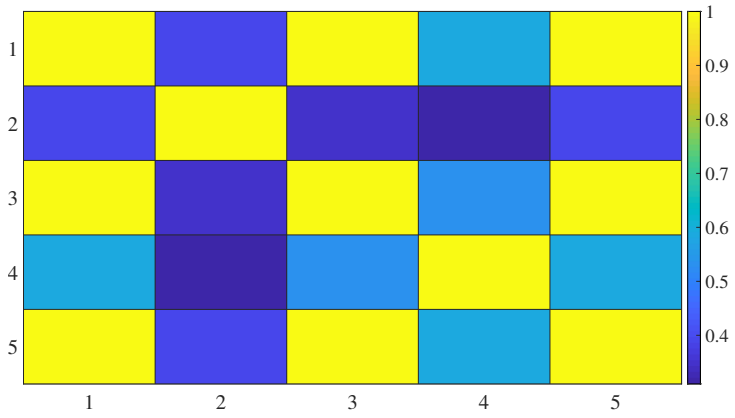

Figura 6.2: Rappresentazione della matrice di correlazione del k-esimo *dataframe* relativo alle misure d'illuminazione presso la Cappella degli Scrovegni di Padova.

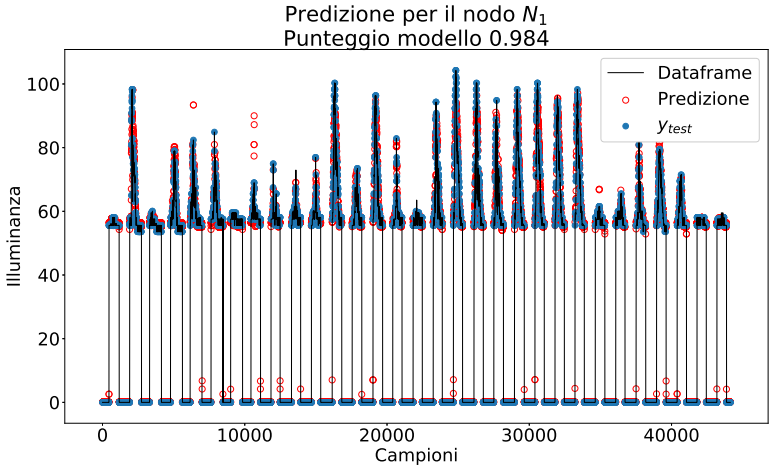

Figura 6.3: Predizione dei valori di illuminamento per il sensore S_1 riducendo la dimensionalità del *dataset* con l'eliminazione della componente di S_3.

Fig. 6.3, dove al costo della riduzione di dimensionalità della matrice di *input* $X^{(1)}$ si è perso solo lo 0.004 sul punteggio del modello LR utilizzato per la previsione delle misure del nodo S_1. Si è pertanto valutato di ridurre ulteriormente la dimensionalità di $X^{(1)}$ rimuovendo anche la componente relativa al nodo sensore S_5 ottenendo le previsioni mostrate in Fig. 6.4. E' triviale che rimuovendo le due componenti con la maggiore correlazione con S_1 ($Corr > 0.9$) si ha un peggioramento delle prestazioni del modello LR, infatti in questo caso si è ottenuto un punteggio pari a solo 0.776. Questo studio ha messo in evidenza le relazioni che sussistono tra le misurazioni dei nodi installati presso il sito della Cappella degli Scrovegni. La possibilità di predire le misurazioni di un nodo, negli esempi riportati di S_1, indica la presenza di una elevata correlazione tra le misure effettuate dai diversi nodi, e da ciò ne consegue che alcuni nodi non stanno contribuendo in modo significativo alla conoscenza della distribuzione di luce all'interno dell'ambiente considerato.

Nell'esempio riportato, in cui si vogliono predire le misurazioni del nodo N_1, e uno alla volta vengono eliminati i nodi che portano informazioni ridondate, cioè quei nodi che non contribuiscono in modo significativo alla conoscenza della distribuzione di luce all'interno dell'ambiente sotto osservazione. Con questa analisi si può pensare di andare a spegnere i nodi N_1, N_3 e N_5 una volta noti i loro coefficienti di peso θ_i.

In conclusione con questa analisi si possono individuare i nodi della rete che non stanno portando informazioni significative al sistema gestionale. Nell'ottica delle reti IoT e dei sistemi di monitoraggio in generale, e in particolare per le reti con nodi alimentati a batteria, sapere quali dispositivi non forniscono informazioni utili è importante perché possono essere disattivati al fine di risparmiare energia.

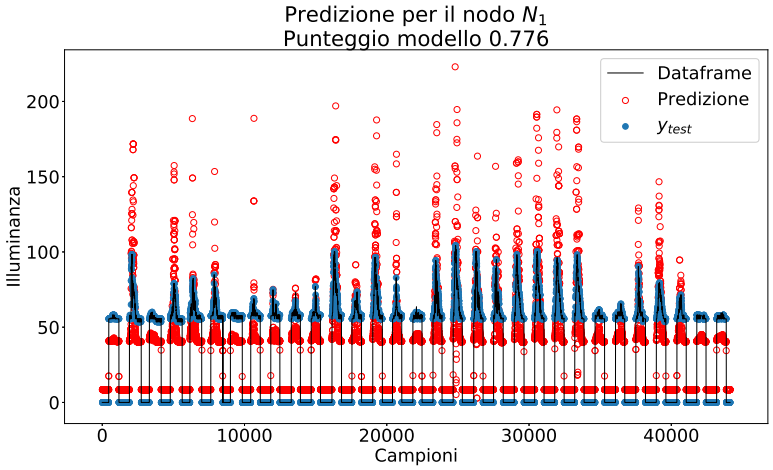

Figura 6.4: Predizione dei valori di illuminamento per il sensore S_1 riducendo la dimensionalità del *dataset* con l'eliminazione della componente di S_3 e della componente di S_5.

Capitolo 7

Conclusioni

In questo lavoro di tesi, che è stato portato avanti duranti i tre anni del dottorato di ricerca, sono state affrontate le tematiche delle WSN per la realizzazione di sistemi di monitoraggio nell'ambito dell'IoT. Per la realizzazione dei progetti inerenti al tema di ricerca, sono stati studiati i protocolli di comunicazione e i sistemi necessari per la progettazione e lo sviluppo di una rete *mesh* senza fili, come presentato nel Cap. 3. All'interno dello stesso capitolo, sono stati illustrati anche i protocolli applicativi per l'interazione dell'utente con il sistema di monitoraggio. I protocolli analizzati, in parte strettamente legati al mondo delle reti di sensori e in parte più legati al mondo dei servizi *web*, sono stati utilizzati per la realizzazione della rete di sensori e del *border router* installati con successo presso il sito della Cappella degli Scrovegni di Padova. Il sistema di monitoraggio dell'illuminazione e degli altri parametri ambientali è stato presentato in Cap. 4.1.

Sono state analizzate le prestazioni di questo sistema, in particolare sono state analizzate le prestazioni della rete (Cap. 5.1) e del DB utilizzato (Cap. 5.3) e allo stesso tempo, sono state studiate soluzioni innovative per migliorare l'efficienza dello stesso, come illustrato nei capitoli Cap. 5.2 e Cap. 6. La progettazione, i risultati e le prestazioni ottenuti dal sistema di monitoraggio dell'illuminazione sopracitato, sono stati riporti in 4 lavori scientifici pubblicati su rivista [39] e presentati a congressi internazionali [40, 44, 82]. Inoltre, i dati generati dalla rete di sensori hanno permesso l'analisi del comportamento della rete attraverso l'utilizzo di algoritmi di ML, i cui risultati saranno al più presto presentati su una pubblicazione scientifica.

Il sistema di monitoraggio realizzato è stato successivamente adattato per il monitoraggio energetico, grazie alla sua elevata portabilità, sia dei nodi sensori della rete IoT che del sistema gestionale installato nel *border router*. Questa seconda versione del sistema è stata utilizzata per il monitoraggio dei consumi energetici di una pompa idraulica che alimenta una macchina MTS per prove di trazione situata presso il Laboratorio di Tecnologia del DIISM dell'Università Politecnica delle Marche. Durante le prove di trazione, è stato misurato il consumo energetico della pompa idraulica, e i risultati ottenuti sono riportati in un lavoro scientifico e presentati in un congresso internazionale [41].

In conclusione sono stati realizzati con successo due sistemi di monitoraggio IoT, entrambi basati sulla stessa architettura WSN ma con differenti unità di *sensing*, e questi sistemi di monitoraggio sono stati installati con successo e ne è stato analizzato il loro corretto funzionamento.

Bibliografia

[1] M. Kusek, "Internet of things: Today and tomorrow," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018, pp. 0335–0338.

[2] "https://www.intel.it/content/www/it/it/internet-of-things/infographics/guide-to-iot-new.html."

[3] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-khah, and P. Siano, "Iot-based smart cities: A survey," in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, June 2016, pp. 1–6.

[4] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester, "City of things: An integrated and multi-technology testbed for iot smart city experiments," in *2016 IEEE International Smart Cities Conference (ISC2)*. IEEE, 2016, pp. 1–8.

[5] A. Gaur, B. Scotney, G. Parr, and S. McClean, "Smart city architecture and its applications based on iot," *Procedia computer science*, vol. 52, pp. 1089–1094, 2015.

[6] Y. U. Devi and M. S. S. Rukmini, "Iot in connected vehicles: Challenges and issues — a review," in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, Oct 2016, pp. 1864–1867.

[7] P. Ramaswamy, "Iot smart parking system for reducing green house gas emission," in *2016 International Conference on Recent Trends in Information Technology (ICRTIT)*, April 2016, pp. 1–6.

[8] J. Pacheco, S. Satam, S. Hariri, C. Grijalva, and H. Berkenbrock, "Iot security development framework for building trustworthy smart car services," in *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, Sep. 2016, pp. 237–242.

[9] W. Chin, W. Li, and H. Chen, "Energy big data security threats in iot-based smart grid communications," *IEEE Communications Magazine*, vol. 55, no. 10, pp. 70–75, Oct 2017.

[10] C. Pop, T. Cioara, M. Antal, I. Anghel, I. Salomie, and M. Bertoncini, "Blockchain based decentralized management of demand response programs in smart energy grids," *Sensors*, vol. 18, no. 1, p. 162, 2018.

[11] J. Pan, R. Jain, S. Paul, T. Vu, A. Saifullah, and M. Sha, "An internet of things framework for smart energy in buildings: Designs, prototype, and experiments," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 527–537, Dec 2015.

[12] M. Chen, J. Yang, X. Zhu, X. Wang, M. Liu, and J. Song, "Smart home 2.0: Innovative smart home system powered by botanical iot and emotion detection," *Mobile Networks and Applications*, vol. 22, no. 6, pp. 1159–1169, 2017.

[13] Y. Kim, Y. Park, and J. Choi, "A study on the adoption of iot smart home service: using value-based adoption model," *Total Quality Management & Business Excellence*, vol. 28, no. 9-10, pp. 1149–1165, 2017.

[14] S. Feng, P. Setoodeh, and S. Haykin, "Smart home: Cognitive interactive people-centric internet of things," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 34–39, February 2017.

[15] S. Tyagi, A. Agarwal, and P. Maheshwari, "A conceptual framework for iot-based healthcare system using cloud computing," in *2016 6th International Conference-Cloud System and Big Data Engineering (Confluence)*. IEEE, 2016, pp. 503–507.

[16] A. Ukil, S. Bandyoapdhyay, C. Puri, and A. Pal, "Iot healthcare analytics: The importance of anomaly detection," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, March 2016, pp. 994–997.

[17] P. A. Laplante and N. Laplante, "The internet of things in healthcare: Potential applications and challenges," *IT Professional*, vol. 18, no. 3, pp. 2–4, May 2016.

[18] M. S. Mekala and P. Viswanathan, "A survey: Smart agriculture iot with cloud computing," in *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, Aug 2017, pp. 1–7.

[19] S. Jaiganesh, K. Gunaseelan, and V. Ellappan, "Iot agriculture to improve food and farming technology," in *2017 Conference on Emerging Devices and Smart Systems (ICEDSS)*, March 2017, pp. 260–266.

[20] C. Brewster, I. Roussaki, N. Kalatzis, K. Doolin, and K. Ellis, "Iot in agriculture: Designing a europe-wide large-scale pilot," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 26–33, Sep. 2017.

[21] L. Barreto, A. Amaral, and T. Pereira, "Industry 4.0 implications in logistics: an overview," *Procedia Manufacturing*, vol. 13, pp. 1245–1252, 2017.

[22] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: state of the art and future trends," *International Journal of Production Research*, vol. 56, no. 8, pp. 2941–2962, 2018.

[23] F. Shrouf, J. Ordieres, and G. Miragliotta, "Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm," in *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, Dec 2014, pp. 697–701.

[24] A. Anzanpour, A.-M. Rahmani, P. Liljeberg, and H. Tenhunen, "Internet of things enabled in-home health monitoring system using early warning score," in *Proceedings of the 5th EAI International Conference on Wireless Mobile Communication and Healthcare*. ICST (Institute for Computer Sciences, Social-Informatics and ..., 2015, pp. 174–177.

[25] Alphonsa A. and Ravi G., "Earthquake early warning system by iot using wireless sensor networks," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, March 2016, pp. 1201–1205.

[26] S. Poslad, S. E. Middleton, F. Chaves, R. Tao, O. Necmioglu, and U. Bügel, "A semantic iot early warning system for natural environment crisis management," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 2, pp. 246–257, June 2015.

[27] S. H. Shah and I. Yaqoob, "A survey: Internet of things (iot) technologies, applications and challenges," in *2016 IEEE Smart Energy Grid Engineering (SEGE)*, Aug 2016, pp. 381–385.

[28] J. A. Manrique, J. S. Rueda-Rueda, and J. M. Portocarrero, "Contrasting internet of things and wireless sensor network from a conceptual overview," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2016, pp. 252–257.

[29] L. Mainetti, L. Patrono, and A. Vilei, "Evolution of wireless sensor networks towards the internet of things: A survey," in *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, Sep. 2011, pp. 1–6.

[30] R. Piyare and S. R. Lee, "Towards internet of things (iots): Integration of wireless sensor network to cloud services for data collection and sharing," *arXiv preprint arXiv:1310.2095*, 2013.

[31] T. Snyder and G. Byrd, "The internet of everything." *IEEE Computer*, vol. 50, no. 6, pp. 8–9, 2017.

[32] "https://www.cisco.com/c/dam/en_us/about/business-insights/docs/ioe-economy-faq.pdf."

[33] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, "A review on internet of things (iot), internet of everything (ioe) and internet of nano things (iont)," in *2015 Internet Technologies and Applications (ITA)*, Sep. 2015, pp. 219–224.

[34] "https://blogs.cisco.com/digital/how-the-internet-of-everything-will-change-the-worldfor-the-better-infographic."

[35] S. Schmid, G. Corbellini, S. Mangold, and T. R. Gross, "Led-to-led visible light communication networks," in *Proceedings of the fourteenth ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2013, pp. 1–10.

[36] ——, "Continuous synchronization for led-to-led visible light communication networks," in *2014 3rd International Workshop in Optical Wireless Communications (IWOW)*. IEEE, 2014, pp. 45–49.

[37] O. Ergul, E. Dinc, and O. B. Akan, "Communicate to illuminate: State-of-the-art and research challenges for visible light communications," *Physical Communication*, vol. 17, pp. 72–85, 2015.

[38] L. Incipini, A. Belli, L. Palma, M. Ballicchia, and P. Pierleoni, "Sensing light with leds: Performance evaluation for iot applications," *Journal of Imaging*, vol. 3, no. 4, p. 50, 2017.

[39] P. Pierleoni, A. Belli, L. Palma, S. Valenti, S. Raggiunto, L. Incipini, and P. Ceregioli, "The scrovegni chapel moves into the future: An innovative internet of things solution brings new light to giotto's masterpiece," *IEEE Sensors Journal*, vol. 18, no. 18, pp. 7681–7696, Sep. 2018.

[40] L. Incipini, L. Palma, A. Belli, S. Raggiunto, and P. Pierleoni, "Performance evaluation of a full ipv6-based internet of things wireless sensor network," in *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, June 2019, pp. 333–338.

[41] L. Incipini, T. Mancia, M. El Mehtedi, and P. Pierleoni, "Iot network for industrial machine energy monitoring," in *2019 AEIT International Annual Conference (AEIT)*, Sep. 2019, pp. 1–6.

[42] F. F. J. Lasso, K. Clarke, and A. Nirmalathas, "A software-defined networking framework for iot based on 6lowpan," in *2018 Wireless Telecommunications Symposium (WTS)*. IEEE, 2018, pp. 1–7.

[43] A. Azzara and L. Mottola, "Virtual resources for the internet of things," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 245–250.

[44] L. Incipini, A. Belli, L. Palma, R. Concetti, and P. Pierleoni, "Mimic: a cybersecurity threat turns into a fog computing agent for iot systems," in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2019, pp. 469–474.

[45] F. Van den Abeele, J. Hoebeke, G. K. Teklemariam, I. Moerman, and P. Demeester, "Sensor function virtualization to support distributed intelligence in the internet of things," *Wireless Personal Communications*, vol. 81, no. 4, pp. 1415–1436, 2015.

[46] J. S. Van der Veen, B. Van der Waaij, and R. J. Meijer, "Sensor data storage performance: Sql or nosql, physical or virtual," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on.* IEEE, 2012, pp. 431–438.

[47] G. Bragg, K. Martinez, P. Basford, and J. Hart, "868mhz 6lowpan with contikimac for an internet of things environmental sensor network," in *SAI Computing Conference (SAI), 2016.* IEEE, 2016, pp. 1273–1277.

[48] A. Fabre, K. Martinez, G. Bragg, P. Basford, J. Hart, S. Bader, and O. Bragg, "Deploying a 6lowpan, coap, low power, wireless sensor network," 2016.

[49] M. Wei, S. H. Hong, and M. Alam, "An iot-based energy-management platform for industrial facilities," *Applied Energy*, vol. 164, pp. 607–619, 2016.

[50] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," in *Computer Games I.* Springer, 1988, pp. 366–400.

[51] ——, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 44, no. 1.2, pp. 206–226, 2000.

[52] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, pp. 1–5, 2019.

[53] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in internet of things," *IEEE Internet of Things Journal*, 2019.

[54] W. Li, T. Logenthiran, V.-T. Phan, and W. L. Woo, "Implemented iot-based self-learning home management system (shms) for singapore," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2212–2219, 2018.

[55] C. Zhang, Y. Liu, F. Wu, W. Fan, J. Tang, and H. Liu, "Multi-dimensional joint prediction model for iot sensor data search," *IEEE Access*, vol. 7, pp. 90 863–90 873, 2019.

[56] "Ieee standard for low-rate wireless networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, April 2016.

[57] S. Rajagopal, R. D. Roberts, and S.-K. Lim, "Ieee 802.15. 7 visible light communication: modulation schemes and dimming support," *IEEE Communications Magazine*, vol. 50, no. 3, pp. 72–82, 2012.

[58] M. Kowalczyk and J. Siuzdak, "Influence of reverse bias on the leds properties used as photo-detectors in vlc systems," in *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2015*, vol. 9662. International Society for Optics and Photonics, 2015, p. 966205.

[59] F. De Santis, M. Ferrara, and H.-C. Neitzert, "Optical in situ characterization of isotactic polypropylene crystallization using an led array in avalanche-photoreceiver mode," *IEEE transactions on instrumentation and measurement*, vol. 55, no. 1, pp. 123–127, 2006.

[60] J. M. Luna-Rivera, C. Suarez-Rodriguez, V. Guerra, R. Perez-Jimenez, J. Rabadan-Borges, and J. Rufo-Torres, "Low-complexity colour-shift keying-based visible light communications system," *IET Optoelectronics*, vol. 9, no. 5, pp. 191–198, 2015.

[61] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Rfc 4944: Transmission of ipv6 packets over ieee 802.15. 4 networks," *Request for Comments*, 2007.

[62] J. Hui and P. Thubert, "Rfc 6282: Compression format for ipv6 datagrams over ieee 802.15. 4-based networks," *IETF: Fremont, CA, USA*, 2011.

[63] Z. Shelby, "Chakrabarti, s., nordmark, e., and c. bormann," neighbor discovery optimization for ipv6 over low-power wireless personal area networks (6lowpans)," RFC 6775, DOI 10.17487/RFC6775, November 2012,< http://www. rfc-editor. org . . . , Tech. Rep., 2012.

[64] A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "Rpl: Ipv6 routing protocol for low-power and lossy networks," in *RFC 6550*, 2012.

[65] P. Thubert, "Objective function zero for the routing protocol for low-power and lossy networks (rpl)," 2012.

[66] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.

[67] C. Bormann and Z. Shelby, "Blockwise transfers in coap draft-ietf-core-block-14," *Avaiable online: http://tools. ietf. org/html/draft-ietf-core-block-14 (accessed on 3 June 2014)*, 2015.

[68] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "The constrained application protocol (coap)(rfc 7252), 2014," 2016.

[69] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures.* University of California, Irvine Doctoral dissertation, 2000, vol. 7.

[70] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Rfc 7252: The constrained application protocol (coap)," *Internet Engineering Task Force*, 2014.

[71] K. Hartke, "Rfc 7641: Observing resources in the constrained application protocol (coap)," *Internet Engineering Task Force*, 2015.

[72] E. Gamma, *Design patterns: elements of reusable object-oriented software.* Pearson Education India, 1995.

[73] M. Kovatsch, "Demo abstract: Human-coap interaction with copper," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on.* IEEE, 2011, pp. 1–2.

[74] I. Fette and A. Melnikov, "Rfc 6455-the websocket protocol.(2011)," *URL https://tools. ietf. org/html/rfc6455*, 2016.

[75] S. Loreto, P. Saint-Andre, S. Salsano, and G. Wilkins, "Rfc 6202-known issues and best practices for the use of long polling and streaming in bidirectional http," *Internet Engineering Task Force*, pp. 1–19, 2011.

[76] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *29th annual IEEE international conference on local computer networks.* IEEE, 2004, pp. 455–462.

[77] A. Dunkels, "Towards tcp/ip for wireless sensor networks," Ph.D. dissertation, 2005.

[78] A. Dunkels, F. Österlind, and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proceedings of the 5th international conference on Embedded networked sensor systems.* ACM, 2007, pp. 335–349.

[79] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proceedings of the 4th international conference on Embedded networked sensor systems.* Acm, 2006, pp. 29–42.

[80] M. Kovatsch, S. Duquennoy, and A. Dunkels, "Erbium (er) rest engine and coap implementation for contiki," 2014.

[81] L. Palma, L. Pernini, A. Belli, S. Valenti, L. Maurizi, and P. Pierleoni, "Ipv6 wsn solution for integration and interoperation between smart home and aal systems," in *Sensors Applications Symposium (SAS), 2016 IEEE.* IEEE, 2016, pp. 1–5.

[82] L. Incipini, A. Belli, L. Palma, R. Concetti, and P. Pierleoni, "Databases performance evaluation for iot systems: the scrovegni chapel use case," in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2019, pp. 463–468.

[83] "Framework web.py." [Online]. Available: http://webpy.org/

[84] "Javascript charts & maps." [Online]. Available: https://www.amcharts.com/

[85] "Linea guida diagnosi energetica (enea)." [Online]. Available: http://www.agenziaefficienzaenergetica.it/per-le-imprese/documenti-1/LineeGuidaDiagnosi10214ENEADUEESPSESE.pdf

[86] "*Template* diagnosi (enea)." [Online]. Available: http://www.agenziaefficienzaenergetica.it/per-le-imprese/documenti-1/diagnosi-energetica/2019-02-15-rev0-template-rapporto-di-diagnosi

[87] "Termocamera flir one." [Online]. Available: https://www.flir.it/support/products/flir-one-gen-3#Specifications

[88] "Seneca power meter s604e." [Online]. Available: https://www.seneca.it/en/linee-di-prodotto/seneca-40/analizzatori-di-rete-serie-s604/s604e/

[89] P. Pierleoni, M. Conti, A. Belli, L. Palma, L. Incipini, L. Sabbatini, S. Valenti, M. Mercuri, and R. Concetti, "Iot solution based on mqtt protocol for real-time building monitoring," in *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, June 2019, pp. 57–62.

[90] S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," in *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE, 1992, pp. 72–84.

[91] "Aws shadow device," https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html.

[92] "Azure twin device," https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-device-twins.

[93] M. Baldi, F. Chiaraluce, L. Incipini, and M. Ruffini, "Code-based physical layer secret key generation in passive optical networks," *Ad Hoc Networks*, vol. 89, pp. 1–8, 2019.

[94] P. Pierleoni, A. Belli, A. Gentili, L. Incipini, L. Palma, S. Valenti, and S. Raggiunto, "A ehealth system for atrial fibrillation monitoring," in *Italian Forum of Ambient Assisted Living*. Springer, 2018, pp. 229–241.

[95] S. Schmid, D. Schwyn, K. Akşit, G. Corbellini, T. R. Gross, and S. Mangold, "From sound to sight: Using audio processing to enable visible light communication," in *2014 IEEE Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 518–523.

[96] P. Pierleoni, A. Belli, L. Palma, L. Incipini, S. Raggiunto, M. Mercuri, R. Concetti, and L. Sabbatini, "A cross-protocol proxy for sensor networks based on coap," in *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, June 2019, pp. 251–255.