Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria Informatica, Gestionale ed Automazione

---

Studio, sviluppo e test sperimentale di architetture di rete per l'IoT e loro integrazione con sistemi Cloud per finalità di EEW

Tesi di Dottorato di:
Roberto Concetti

Tutor:
Prof.ssa Paola Pierleoni

Coordinatore del Curriculum:
Prof. Franco Chiaraluce

XIX ciclo - nuova serie

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria Informatica, Gestionale ed Automazione

---

Studio, sviluppo e test sperimentale di architetture di rete per l'IoT e loro integrazione con sistemi Cloud per finalità di EEW

Tesi di Dottorato di:
Roberto Concetti

Tutor:
Prof.ssa Paola Pierleoni

Coordinatore del Curriculum:
Prof. Franco Chiaraluce

XIX 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

Ringraziamenti

A conclusione di questo percorso di Dottorato sento il dovere ed il piacere di ringraziare numerose persone. Questo viaggio è stato entusiasmante, arricchente, sia dal punto di vista professionale, sia dal punto di vista umano.

In primis desidero ringraziare la Prof.ssa Paola Pierleoni, punto di riferimento e fonte inesauribile di consigli. La ringrazio per il suo sostegno, per la fiducia riposta, per la sua professionalità e per la pazienza. Desidero ringraziare il dott. Alberto Belli ed il dott. Lorenzo Palma del gruppo di ricerca, sono stati veri e propri esempi.

Un sentito ringraziamento all'INGV sezione di Ancona per l'aiuto fornito a tutti i livelli, dal supporto nella configurazione degli applicativi utilizzati ad i consigli nelle selezioni dei dati e per il continuo confronto.

Un doveroso ringraziamento ad i colleghi dottorandi dell'Open Space, in particolar modo a Sara Raggiunto ed i nostri quotidiani viaggi, a Michele Paoletti per la condivisione della tematica di ricerca e per le sue esilaranti passioni, a Paolo Santini per l'accoglienza di ogni mattina ed i nostri vari momenti di confronto. Un sentito grazie a Lorenzo Incipini, con il quale ho avuto il piacere di collaborare per la creazione di alcuni script ed a Marco Mercuri per l'aiuto nei collegamenti del sensore e per le sue splendide parole.

Infine voglio ringraziare la mia famiglia, e la mia compagna Tatiana, per l'appoggio ed il sostegno che mi danno in tutte le mie scelte, anche le più folli.

La ricerca è stata in grado di aprirmi orizzonti nuovi ed affascinanti. Questo mondo è stato il mio ossigeno in questi tre anni. Con la speranza che possa aver lasciato in tutte le persone che ho incontrato anche solo un decimo di ciò che loro hanno lasciato in me.

Grazie a tutti Voi.

Ancona, Novembre 2020

Roberto Concetti

Sommario

Un sistema di Earthquake Early Warning (EEW) deve obbligatoriamente confrontarsi con la componente tempo. Escludendo la possibilità di predire un evento sismico, l'unica azione, ad oggi possibile, è l'invio di un'allerta verso target più distanti dal punto di nucleazione dello stesso prima dell'arrivo delle onde più distruttive. A tal fine, possiamo sfruttare le diverse velocità di propagazione delle onde sismiche nel terreno rispetto a quella dei mezzi trasmissivi delle informazioni ed avvalerci dell'ausilio di moderne tecnologie.
Questo lavoro di tesi presenta, quindi, lo studio, lo sviluppo ed il test sperimentale di architetture di rete per l'Internet of Things (IoT) e la loro integrazione con sistemi Cloud per finalità di allerta rapida in caso di terremoto. Dopo aver analizzato i ritardi aggiunti da ogni componente di un sistema di EEW, si è agito su una parte dell'architettura di sistema con lo scopo di una diramazione dell'allerta in tempi più contenuti rispetto agli attuali. L'utilizzo di differenti protocolli trasmissivi rispetto agli standard internazionali, di differenti tecniche di individuazione e di localizzazione, ha dimostrato come sia possibile ridurre il tempo totale di rilevamento, e di conseguenza di allerta, di circa 3 secondi. La modularità dell'architettura proposta ne permette l'integrazione nelle attuali reti sismiche ed è applicabile anche in altri scenari, come ad esempio quello del monitoraggio strutturale, o nel caso di allerta rapida in seguito ad una varietà di eventi naturali come quelli correlati al dissesto idrogeologico.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

L'attività di ricerca presentata in questa tesi ha riguardato i sistemi di Early Warning in caso di terremoto (EEW). In tale direzione sono state effettuate approfondite ricerche bibliografiche relative ai sistemi attualmente in uso a livello mondiale per quanto riguarda la strumentazione, le architetture, gli algoritmi di rilevazione ed i sistemi di comunicazione. Particolare attenzione è stata rivolta a sistemi che utilizzano il paradigma dell'Internet of Things (IoT). I dispositivi IoT infatti, grazie alle loro caratteristiche, possono essere integrati all'interno di reti sismiche esistenti, al fine di consentire un monitoraggio real-time più capillare sul territorio e contribuire ad una rilevazione più rapida di un evento sismico. Al tempo stesso, bisogna tenere in considerazione anche di alcune loro limitazioni, come capacità computazionali o di storage. Tali limiti hanno condotto a studiare le più efficienti integrazioni tra IoT e Cloud Computing. Tra le piattaforme EEW attualmente in servizio in campo internazionale, solamente due sfruttano le potenzialità di una reale integrazione tra l'IoT ed il Cloud Computing. Nondimeno, queste non utilizzano tutti i servizi che una piattaforma Cloud può offrire poiché non sfruttano appieno alcuni vantaggi offerti quali scalabilità, disponibilità, sicurezza ed affidabilità.

Le attività si sono sviluppate secondo diverse direttrici. È stato inizialmente effettuato uno studio sull'utilizzo di protocolli del mondo IoT per la pacchettizzazione e la trasmissione real-time dei tracciati acquisiti dalle stazioni sismiche. Sono state condotte simulazioni su eventi sismici utilizzando il protocollo MQTT ed una differente pacchettizzazione rispetto allo standard di riferimento in campo sismico. Tale studio, presentato nel terzo capitolo, è stato condotto sia confrontando le latenze dei protocolli sulla base dell'invio di medesimi pacchetti, sia mediante confronto sulla rilevazione delle fasi P degli eventi utilizzando una pacchettizzazione in JSON ed una trasmissione in MQTT.

Appurato come l'utilizzo di un differente tipo di pacchettizzazione ed un differente protocollo di trasmissione potesse portare dei vantaggi in termini di latenze, il lavoro è proseguito con lo sviluppo di un dispositivo IoT. Nel Capitolo 4 vengono illustrate le scelte implementative sia a livello hardware sia a livello software ed analizzate le performance del prototipo low-cost realizzato

in confronto ad un dispositivo precedentemente testato dal gruppo di ricerca durante la sequenza sismica nell'Italia Centrale del 2016.

Nel quinto Capitolo viene presentata una panoramica delle piattoforme Cloud attualmente leader del mercato, mediante un confronto a livello architetturale e prestazionale, per quanto riguarda tempi di servizio e latenze su diversi carichi simulati ed inviati tramite il protocollo MQTT. Lo scopo di questa analisi è stato quello di verificare la possibilità di utilizzo di una delle piattaforme per l'attività oggetto della ricerca del dottorato.

Le precedenti attività hanno condotto infine allo sviluppo di una intera piattaforma Cloud-IoT per l'EEW, presentata nel sesto Capitolo. Il dispositivo proposto, punto iniziale della soluzione, integra opzionalmente i componenti necessari ad una installazione non invasiva all'interno di reti sismiche esistenti. Inoltre, è in grado di rilevare a bordo un possibile evento sismico, inviando successivamente ed in tempo reale i campioni verso un'applicazione Cloud per ulteriori analisi. Tale applicazione ha il compito di dichiarare l'evento, localizzarlo ed inviare notifiche verso endpoint predefiniti in caso di potenziali eventi dannosi. All'interno del Capitolo è inoltre presentata una differente applicazione dell'architettura in riferimento al monitoraggio strutturale. Nel settimo ed ultimo Capitolo sono illustrate le conclusioni e i possibili sviluppi futuri del lavoro.

Capitolo 2

Internet of Things, Cloud Computing, sistemi di EEW

Perdite di vite umane, danni ad abitazioni ed a strutture pubbliche, collassi economici sono solamente alcune delle conseguenze catastrofiche dei terremoti, che appaiono ancora oggi come disastri naturali impossibili da prevedere. Negli ultimi anni, tuttavia, grazie all'ausilio di moderne tecnologie come ad esempio nel campo delle Telecomunicazioni, delle Reti di Sensori, del Cloud Computing, è stato possibile sviluppare tecniche sempre più efficaci per un monitoraggio sismico in tempo reale. Una rete sismica moderna è in grado, infatti, di elaborare automaticamente i segnali registrati e può, in determinate condizioni, fornire stime rapide ed affidabili dei parametri di un evento sismico, come istante di origine, localizzazione e magnitudo, prima che le onde sismiche più distruttive arrivino in un determinato punto. Questo tipo di procedure sono conosciute come Earthquake Early Warning (EEW). [2, 3].

L'idea alla base di un sistema per l'EEW, proposta da Cooper [4] nel 1868, è la maggiore velocità con cui l'informazione può viaggiare attraverso un segnale elettromagnetico rispetto alla velocità di propagazione delle onde sismiche. Pertanto, se un sistema posto in prossimità di una zona sismogena rileva un evento, potrebbe essere possibile inviare un allarme ad uno o più target con alcuni secondi o decine di secondi (in funzione della distanza ipocentrale) di anticipo rispetto all'arrivo delle onde di ampiezza più rilevante. In tutti i sistemi, perciò, una maggiore concentrazione di stazioni nell'area di generazione di terremoti per l'acquisizione in tempo reale delle forme d'onda potrebbe portare a benefici sia in termini di correttezza dei risultati sia di tempi di rilevamento. Fino a pochi anni fa, la realizzazione di una rete sismologica con unità sismiche particolarmente dense sarebbe stata onerosa sia dal punto di vista economico, a causa degli alti costi della strumentazione sismica, generalmente composta da sensori, datalogger, che dal punto di vista temporale in relazione ai tempi di trasferimento dell'informazione attraverso le consuete reti di telecomunicazione. Tuttavia, i recenti sviluppi nell'Internet of Things (IoT) hanno aperto la strada a scenari promettenti anche per le applicazioni di gestione dei disastri

[5]. I dispositivi IoT, grazie alle loro caratteristiche, possono essere integrati all'interno di reti sismiche già esistenti, al fine di consentire un monitoraggio real-time sul territorio più capillare e contribuire ad una rilevazione più rapida di un evento sismico. Inoltre, gli attuali miglioramenti tecnologici in termini di sensibilità e range dinamico nel settore dei sensori MEMS-based, hanno permesso l'utilizzo di tali dispositivi anche in campo sismico.

Al tempo stesso, quando si parlata di paradigma IoT bisogna tenere in considerazione anche alcune limitazioni. I dispositivi, infatti, solitamente hanno potere computazionale limitato, unitamente a dimensioni ridotte in termini di storage. Limiti che hanno portato la comunità scientifica allo studio di possibili integrazioni del mondo IoT con il Cloud Computing [6]. Il Cloud ha virtualmente infinite capacità elaborative e di storage e si basa sulla condivisione e massimizzazione delle risorse. Al Cloud solitamente vengono demandati i compiti più onerosi a livello computazionale che i dispositivi edge non sono in grado di effettuare.

In questo Capitolo vengono presentati il paradigma IoT, il mondo del Cloud Computing e la loro possibile integrazione, con particolare attenzione alle architetture di riferimento. Successivamente vengono analizzati i sistemi di EEW a livello mondiale e la soluzione sviluppata in Italia meridionale dall'Università "Federico II" di Napoli. Sono stati quindi selezionati ed analizzati i sistemi EEW che utilizzano il paradigma IoT. Viene poi presentato lo standard di riferimento per quanto concerne i protocolli trasmissivi per reti sismiche ed infine viene illustrata la soluzione proposta.

2.1 Internet of Things

Il termine IoT viene utilizzato per la prima durante un presentazione presso Procter&Gamble nel 1999 da Kevin Ashton [7]. Con tale neologismo egli ha indicato non solo gli oggetti, ma anche persone, luoghi, presenti nel mondo reale con possibilità di connessione ad Internet. L'idea alla base di questo concetto è la dilagante diffusione intorno a noi di una varietà di cose o oggetti -come i tag Radio-Frequency IDentification (RFID), sensori, attuatori, telefoni cellulari, ecc.- che, attraverso schemi di indirizzamento unici, sono in grado di interagire gli uni con gli altri, collaborando per raggiungere obiettivi comuni [8]. Con IoT si indicano, quindi, un insieme di tecnologie che permettono di collegare a Internet qualunque tipo di apparato. Lo scopo di questo tipo di soluzioni è quello di monitorare, controllare e trasferire informazioni in tempo reale relativamente a parametri misurabili quali temperatura, localizzazione, pressione, rumore, luce, umidità per poi svolgere azioni conseguenti. Inoltre, gli oggetti stessi possono essere in grado di prendere decisioni autonome in funzione dell'ambiente e della sua evoluzione. Pertanto, l'IoT comprende tutti quegli

oggetti cosiddetti “intelligenti” tra loro interconnessi in modo da scambiare le informazioni possedute, raccolte e/o elaborate. Si apre quindi in maniera dirompente la possibilità di creare nuovi servizi ed applicazioni innovative in una notevole varietà di scenari come, a titolo esemplificativo, ma non esaustivo:

- Smart city: monitoraggio e gestione degli elementi di una città (ad esempio mezzi per il trasporto pubblico, illuminazione pubblica, traffico veicolare, parcheggi) e dell’ambiente circostante [9, 10].
- Smart home: applicazioni per la gestione in automatico e/o da remoto degli impianti e degli oggetti connessi all’interno delle abitazioni [11, 12].
- Smart agricolture: migliorare la qualità dei prodotti, ridurre l’impatto ambientale mediante monitoraggio di parametri micro-climatici a supporto dell’agricoltura [13, 14].
- e-Health: monitoraggio di parametri fisiologici umani, assistenza alla popolazione più anziana, tracciamento dei dottori e dei pazienti all’interno di un ospedale [15, 16].
- Industrial: connessione e monitoraggio dei macchinari, degli operatori, manutenzione predittiva dei macchinari [17, 18].
- Smart cars: comunicazione di informazione in tempo reale al guidatore mediante connessione dei veicoli e/o tra i veicoli, unitamente all’infrastruttura circostante [19].
- Smart Environment: controllo della qualità dell’aria, della temperatura, del livello dei fiumi, rilevamento di alluvioni o incendi boschivi [20]
- Structural Monitoring: monitoraggio, valutazione dello stato di sicurezza di edifici [21, 22].
- Disaster management: sistemi di rivelamento rapidi di calamità naturali, localizzazione di potenziali zone maggiori di danno e di vittime [5].

Naturalmente le applicazioni possono essere innumerevoli, l’elenco rappresenta solamente alcuni dei possibili scenari applicativi.

2.2 Cloud Computing

Il Cloud Computing consiste nella distribuzione on-demand delle risorse IT tramite Internet con una tariffazione basata sul consumo. Il Cloud Computing è iniziato come un nuovo, più efficiente modo di fare "hosting gestito". Così come l’ambiente informatico è diventato sempre più standardizzato, molte imprese e

società di ricerca hanno da tempo capito che affittare i loro ambienti informatici poteva essere economicamente più conveniente, dal momento che l'acquisto, l'installazione e l'esecuzione di hardware sono costosi e difficili. Piuttosto che acquistare, possedere e mantenere i data center e i server fisici, tramite il Cloud Computing è possibile quindi accedere a servizi tecnologici, quali capacità di calcolo, storage e database, sulla base delle proprie necessità, affidandosi a un fornitore di servizi Cloud.

Tra le definizioni di Cloud Computing, quella indicata dal National Institute of Standards and Technology (NIST) fornisce una descrizione ampia delle caratteristiche principali:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [23]

La disponibilità di risorse on demand diviene quindi un punto focale. L'infrastruttura sottostante (calcolo, rete e spazio di archiviazione) può espandersi e contrarsi in base alle richieste delle applicazioni in esecuzione e secondo un modello per cui l'utente paga solo per le risorse IT realmente consumate.

Il NIST completa la definizione comprendendo le caratteristiche essenziali, i modelli di servizio ed i modelli di deployment. Le caratteristiche sono le seguenti:

- **On demand self-service**: un consumatore può acquisire unilateralmente e automaticamente le necessarie capacità di calcolo, come tempo macchina e memoria, senza richiedere interazione umana con i fornitori di servizi e secondo un modello pay-as-you-go.

- **Broad Access Network**: le capacità sono disponibili in rete e accessibili attraverso meccanismi standard che promuovono l'uso attraverso piattaforme eterogenee, come client *thin* o *thick* (come ad esempio telefoni mobili, tablet, laptops e workstations). I client *thin* si limitano alla visualizzazione come normalmente fanno i browser, mentre i client *thick* sono dotati di intelligenza locale che lavora direttamente sul sistema operativo.

- **Resource polling**: le risorse di calcolo del fornitore sono messe in comune per servire molteplici consumatori utilizzando un modello condiviso (multi-tenant), con le diverse risorse fisiche e virtuali assegnate e riassegnate dinamicamente in base alla domanda. Dato il senso di indipendenza

dalla locazione fisica, l'utente generalmente non ha controllo o conoscenza dell'esatta ubicazione delle risorse fornite, ma può essere in grado di specificare la posizione ad un livello superiore di astrazione (ad esempio, paese, stato o data center). Esempi di risorse includono memoria, elaborazione e larghezza di banda della rete.

- **Rapid elasticity**: le risorse possono essere acquisite e rilasciate elasticamente, in alcuni casi anche automaticamente, per scalare rapidamente verso l'esterno e l'interno in relazione alla domanda. Al consumatore le risorse disponibili spesso appaiono illimitate e disponibili in qualsiasi quantità ed in qualsiasi momento.

- **Measured service**: i sistemi Cloud controllano automaticamente e ottimizzano l'uso delle risorse, facendo leva sulla capacità di misurazione ad un livello di astrazione appropriato per il tipo di servizio (ad esempio memoria, elaborazione, larghezza di banda e utenti attivi). L'utilizzo delle risorse può essere monitorato, controllato e segnalato, fornendo trasparenza sia per il fornitore che per l'utilizzatore del servizio.

Definire quello che comprende il Cloud Computing non è semplice, data la molteplicità delle sue componenti. Risulta al tempo stesso più semplice suddividere le tecnologie e le componenti del Cloud Computing secondo tre modelli di servizio, come definito dal NIST, e noto come modello SPI (Software, Platform, Infrastructure). Tali modelli possono essere rappresentati in forma piramidale (Fig.2.1), nella quale più ci si avvicina verso il basso maggiore è l'autonomia di sviluppo e personalizzazione di cui può usufruire al consumatore.

- **Infrastucture as a Service (IaaS)**: in questo modello il consumatore ha la facoltà di acquisire elaborazione, memoria, rete e altre risorse fondamentali di calcolo, inclusi sistemi operativi e applicazioni. Non c'è la possibilità di controllo sulla struttura Cloud sottostante, ma è possibile gestire sistemi operativi, memoria, applicazioni. L'utente ha quindi a disposizione le potenzialità di un macchina fisica senza dovessi preoccupare dell'hardware o della continuità del servizio, in quanto compito del fornitore dei servizi.

- **Platform as a Service (Paas)**: il consumatore in questo modello ha la possibilità di distribuire sull'infrastruttura cloud applicazioni create in proprio oppure acquisite da terzi, utilizzando linguaggi di programmazione, librerie, servizi e strumenti supportati dal fornitore dei servizi. In questo caso l'utente non ha il controllo e non gestisce l'infrastruttura composta da server, rete, memoria e sistemi operativi sottostanti.

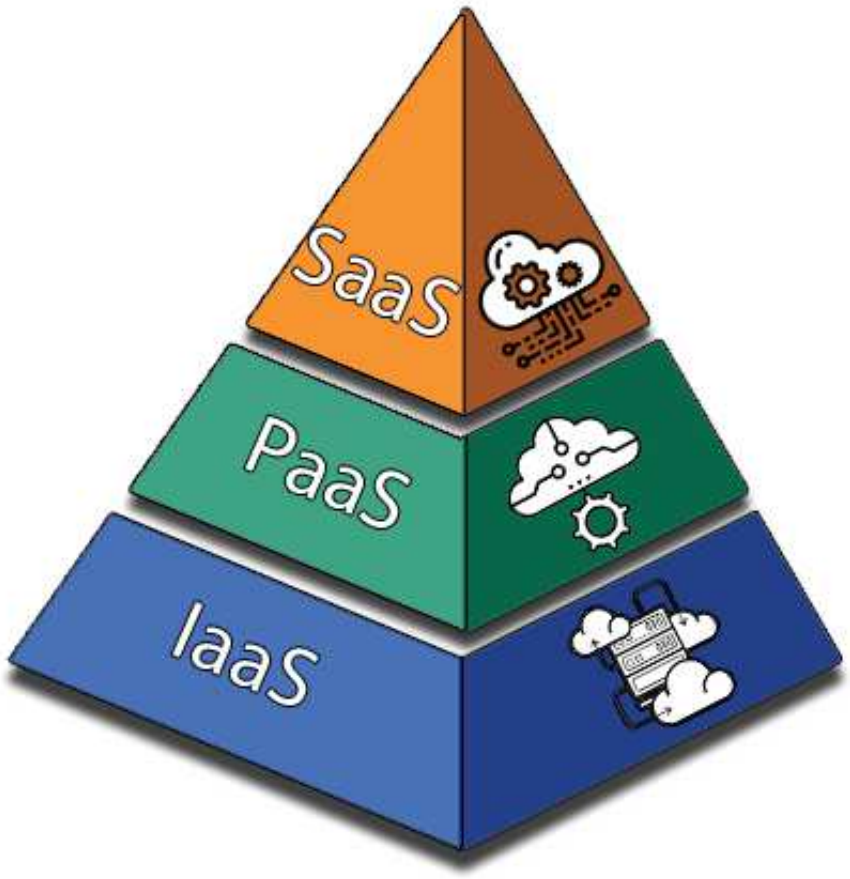

Figura 2.1: Cloud Computing - Modelli di Servizio.

- **Software as a Service (SaaS)**: in questo modello il consumatore ha la possibilità di usufruire delle applicazioni che il fornitore di servizi mette a disposizione su una infrastruttura Cloud. Anche in questo caso, l'utente non gestisce né controlla in alcun modo l'infrastruttura sottostante, se non alcuni parametri di configurazione delle applicazioni.

Infine, il NIST completa la definizione con i possibili modelli di distribuzione. La distribuzione delle risorse a livello locale, mediante gli strumenti di virtualizzazione e gestione delle risorse, viene definita **Cloud privato**. La distribuzione locale non è caratterizzata dai numerosi vantaggi tipici del Cloud Computing, ma a volte risulta la soluzione ideale per la sua capacità di offrire risorse dedicate. Nella maggior parte dei casi, questo modello di implementazione equivale all'infrastruttura IT esistente dove però vengono utilizzate tecnologie di virtualizzazione e gestione delle applicazioni per incrementare l'utilizzo delle risorse.

Quando si parla di **Cloud pubblico** si intende che l'infrastruttura Cloud è fornita per un uso aperto a qualsiasi consumatore. Può essere posseduta, diretta e gestita da un'azienda, da un'organizzazione accademica o governativa oppure da una combinazione delle precedenti. Il modello denominato **Cloud ibrido** consente di connettere l'infrastruttura e le applicazioni tra risorse basate sul Cloud e le risorse esistenti che non vi presenti. Il metodo più comune di implementazione ibrida prevede il collegamento tra il Cloud pubblico e un'infrastruttura locale esistente, al fine di estendere e potenziare l'infrastruttura di un'organizzazione mediante il collegamento delle risorse.

2.3 IoT e Cloud Computing: una possibile integrazione

L'IoT, come precedentemente illustrato, è quindi un paradigma basato su Internet che include diverse tecnologie interconnesse per lo scambio di informazioni tra dispositivi, generalmente piccole "cose" del mondo reale, che possono essere identificate e monitorate tramite Internet. Le applicazioni IoT devono tenere conto di diversi fattori a seconda del contesto dell'applicazione. I dati prodotti devono essere elaborati, interpretati, archiviati e ogni scelta di implementazione è importante per il successo di un'applicazione, come ad esempio la scelta dei migliori Data Base Management Systems (DBMS) per memorizzare i dati rilevati [24]. I dispositivi IoT sono ampiamente distribuiti e di solito hanno capacità di archiviazione ed elaborazione limitate, con problemi di affidabilità, prestazioni, sicurezza e privacy. Questi limiti hanno portato ad una integrazione del mondo IoT con il Cloud Computing [25], che ha una capacità virtualmente illimitata in termini di storage e potenza di calcolo, e si basa sulla condivisione delle risorse e sulla loro massimizzazione. Il Cloud Computing abbiamo visto essere un modello che consente l'accesso a un insieme di risorse condivise e configurabili (es. reti, server, strutture di archiviazione, applicazioni) offerte come servizi. Queste risorse possono essere rapidamente richieste, gestite e utilizzate in un modello con pagamento in base al consumo, in modo che l'utente paghi l'importo dell'uso effettivo di una risorsa. Il Cloud Computing è inoltre indipendente dalla posizione, consentendo l'accesso dell'utente ai servizi Cloud da qualsiasi posizione e con qualsiasi dispositivo tramite la connessione Internet.

In letteratura diversi studi [26, 6] hanno dimostrato l'effettiva complementarità di IoT e Cloud Computing in termini di comunicazione, archiviazione e calcolo [6]. La necessità di una integrazione tra il mondo Cloud ed il paradigma IoT è presentata in [27], dove gli autori affermano che il compito del Cloud è quello di uno strato intermedio tra le applicazioni e le *things*. Lo studio in [28]

mostra come il cosidetto Cloud-Assisted Remote Sensing (CARS) consenta la raccolta dei dati di sensori distribuiti, la condivisione di risorse, l'accesso agli stessi dati remoti in tempo reale, il provisioning e la scalabilità delle risorse. Una architettura *CloudThings* viene proposta in [29] dove viene esaminato uno scenario di smart home con utilizzo congiunto delle tecnologie IoT e Cloud. Liu et al. [30] propongono un framework IoT incentrato sui dati che sfrutta i vantaggi del Cloud pubblico di Azure per la realizzazione di un servizio di gestione IoT centralizzato. Appare quindi evidente come IoT e Cloud Computing possano convergere, come descritto nel 2013 da Gubbi et al. [31], dove viene presentata una implementazione Cloud per la gestione di dispositivi IoT utilizzando Aneka, che si basa sull'interazione di Cloud pubblici e privati.

Nei successivi paragrafi verranno presentate le architetture tipiche di una applicazione IoT e viene proposta una possibile architettura per l'integrazione dei due paradigmi.

2.3.1 Architetture IoT

Il numero sempre crescente di oggetti e dispositivi eterogenei connessi ed interconnessi rende necessaria un'architettura flessibile a strati. Anche se non esiste un modello di riferimento, i modelli di base in letteratura [32] sono un'architettura a 3 o 5 strati [33]. L'architettura a 3 livelli in Fig. 2.2(a), partendo dal livello più basso, consiste in:

- **Perception layer**: è il livello fisico, nel quale sono inclusi sensori ed attuatori per il rilevamento e la raccolta di informazioni sull'ambiente e per l'esecuzione di operazioni.

- **Network layer**: i dati generati dal Perception layer, vengono inviati medianti questo livello, che è quindi responsabile della connessione tra gli Smart Objects, dispositivi di rete, server, etc. Include tecnologie come RFID, 3G, GSM, UMTS, WiFi, Bluetooth Low Energy, ZigBee.

- **Application layer**: è il livello responsabile della fornitura di servizi specifici dell'applicazione all'utente.

Nell'architettura a 5 livelli (Fig.2.2(b)) viene introdotto un livello di astrazione maggiore mediante un **Middleware layer** tra il livello di Network ed Application ed un **Business layer** al livello più alto. Il Middleware elabora i dati ricevuti, prende decisioni e fornisce i servizi richiesti all'Application layer. Il livello di Business, infine, gestisce le attività e i servizi complessivi del sistema IoT e supporta i processi decisionali basati sui dati.

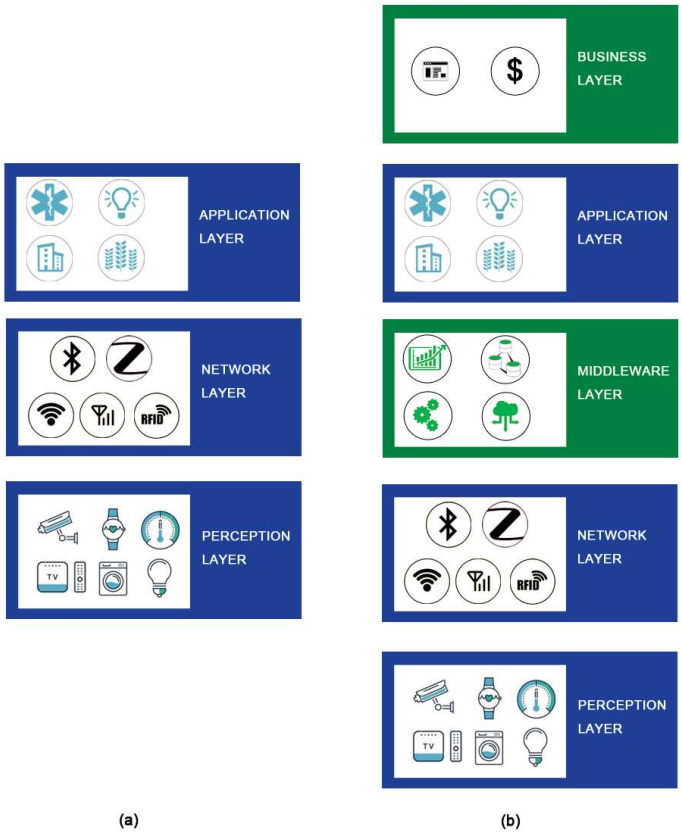

Figura 2.2: Architetture a 3 (a) e 5 (b) livelli per l'IoT.

2.3.2 Architetture Cloud-IoT

Il Cloud Standards Customer Council [34] ha creato una architettura standard di riferimento per sistemi IoT su larga scala che utilizzano il Cloud Computing. La "Cloud Customer Architecture for IoT (CCAIoT)" comprende una soluzione end-to-end che va dagli utenti dei dispositivi IoT alle imprese che gestiscono i dispositivi ed i dati da questi generati. I componenti principali dell'architettura, rappresentata in Fig. 2.3, abbracciano cinque domini: *User Layer*, *Proximity Network*, *Public Network*, *Provider Cloud*, ed *Enterprise Network*.
Lo *User Layer* contiene gli utenti IoT e le loro applicazioni per gli utenti finali. È indipendente da qualsiasi dominio di rete specifico. Il dominio *Proximity Network* ha al suo interno le capacità di rete per la connessione dei dispositivi,

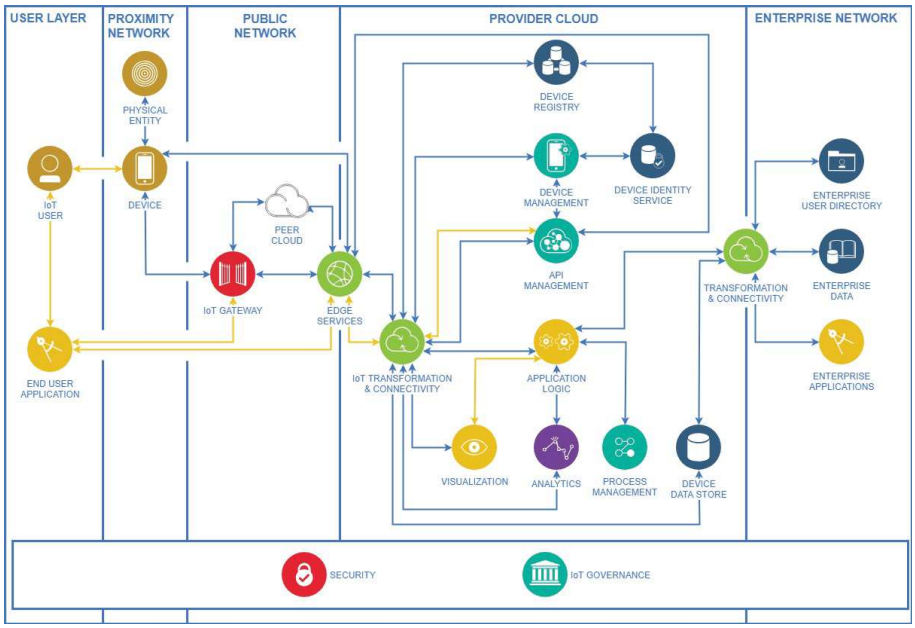

Figura 2.3: Elementi di una soluzione Cloud-IoT. *Fonte*: Cloud Standards Customer Council, *Cloud Customer Architecture for IoT, 2016.*

i quali vengono inclusi in questo dominio. Il *Public Network* interconnette i dispositivi di diverse *Proximity Network* attraverso una rete geografica, tipicamente Internet. Contiene anche servizi di Edge, che consentono il flusso sicuro dei dati da Internet verso il *Provider Cloud*. Il *Provider Cloud* acquisisce i dati dai dispositivi o da altre origini dati e fornisce applicazioni IoT e servizi associati (archiviazione, analisi, visualizzazione). Include componenti per la gestione dei dispositivi (provisioning, amministrazione remota, aggiornamento software, controllo remoto). Le informazioni generate dal *Provider Cloud* vengono utilizzate dall'utente e dalle applicazioni aziendali nel dominio *Enterprise Network*. I sottosistemi di sicurezza e governance dell'IoT abbracciano tutti gli elementi dell'architettura. Il sistema di sicurezza deve considerare la gestione dell'identità e degli accessi (IAM), la protezione dei dati ed il monitoraggio della sicurezza in tutta la sua interezza.

2.4 Sistemi di EEW

Come è stato introdotto, i sistemi di EEW si basano sul semplice principio della differente velocità di propagazione di un segnale elettromagnetico rispetto alla velocità di propagazione delle onde sismiche. Utilizzando tale principio, è possibile l'invio di potenziali allarmi dell'evento sismico in atto in punti target

prima dell'arrivo delle onde stesse. Il tempo diviene quindi un parametro chiave in tali sistemi: maggiore è il tempo tra il rilevamento di un terremoto, la stima dei suoi parametri ed il momento in cui lo scuotimento viene avvertito da un utente, maggiori possono essere le possibilità di intraprendere contromisure. Ad esempio, a livello industriale è possibile automatizzare il blocco di impianti pericolosi, rallentare i treni sulle linee ad alta velocità, mentre a livello domestico gli ascensori possono essere portati al primo piano utile. Possono essere inviati messaggi anche alla popolazione che può intraprendere le primissime misure di sicurezza, il cosiddetto "drop, cover and hold on". Indubbiamente è una corsa contro il tempo, quindi un EEW deve garantire velocità di esecuzione, affidabilità e correttezza dei risultati, trovando il giusto compromesso tra queste caratteristiche.
Così come il principio di funzionamento può esser definito semplice, la sua effettiva implementazione risulta maggiormente complessa. Dal come rilevare effettivamente un terremoto a come localizzarlo, dalla stima dei suoi parametri alla loro accuratezza, alle modalità di trasmissione in tempo reale sia dei campioni rilevati dalle stazioni che delle potenziali allerte, sono solamente alcune delle domande che la comunità internazionale si pone. In questa sezione illustreremo i concetti di riferimento, analizzeremo i sistemi di EEW attualmente in utilizzo o sviluppo a livello mondiale, concentrandoci, infine, sul caso italiano.

2.4.1 Concetti di riferimento

Quando un terremoto si genera in un'area detta di nucleazione, vengono a generarsi diverse tipe di onde con differenti caratteristiche sia a livello di velocità di propagazione che di potenziale distruttivo. Tali onde sismiche vengono suddivise in onde di volume ed onde di superficie. Le onde di volume si dividono in:

- **onde primarie (o onde P)**: dal latino *prima unda*, queste onde sono le più veloci, quindi le prime a raggiungere una stazione di rilevamento. Sono onde compressionali o longitudinali e viaggiano lungo la cosiddetta direzione di propagazione dell'onda. La loro velocità è compresa tra i 4 e gli 8 km/s ed il loro contenuto generalmente non è distruttivo;

- **onde secondarie (o onde S)**: dal latino *secunda unda*, queste onde viaggiano perpendicolarmente lungo la direzione di propagazione ad una velocità circa il 60% minore rispetto alle onde P. Le onde S possono essere avere un contenuto distruttivo anche a diversi chilometri di distanza dalla zona di nucleazione.

Le onde di superficie possono essere raggruppate in onde di Love ed onde di Rayleigh e sono le onde maggiormente distruttive, ma che viaggiano ad una velocità inferiore rispetto alle onde di volume. In Fig. 2.4 viene riportato un sismogramma registrato presso la stazione denominata MMUR dell'Istituto Nazionale di Geofisica e Vulcanologia (INGV), durante l'evento del 26-10-2016 di magnitudo (M_w) 5.9 con epicentro a Visso (MC, Italia), nel quale sono evidenziati i tempi di arrivo delle onde P ed S. Grazie alle diverse caratteristiche di queste onde, è possibile utilizzare il ritardo tra i tempi di arrivo per avviare l'analisi.

I sistemi di EEW possono essere raggruppati secondo l'approccio utilizzato op-

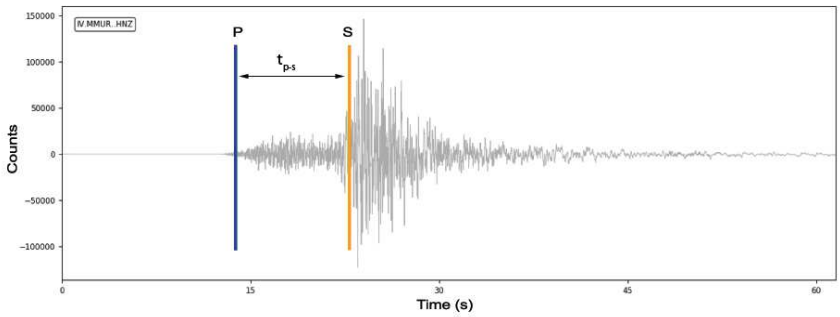

Figura 2.4: Esempio di una forma d'onda sismica registrata dalla stazione MMUR durante un terremoto. Le linee verticali blu ed arancione indicano i tempi di arrivo rispettivamente delle onde P e delle onde S. Tp-s indica il tempo tra i due arrivi. L'asse dei tempi è in s relativi al 26-10-2016T19:18:04 UTC.

pure secondo la distrubizione dell'allerta. Nel primo caso, i differenti algoritmi utilizzati producono una distinzione tra:

- **sistemi on-site**: lo scopo di questi sistemi è quello di rilevare le onde P e prevedere quello che sarà il massimo scuotimento atteso nel luogo stesso dove il sistema di rilevamento è posizionato. Questo è reso possibile dall'utilizzo di leggi di scala tra, ad esempio, il picco iniziale di ampiezza dello spostamento registrato e la severità dello scuotimento del suolo atteso;

- **sistemi regionali**: in questi sistemi la rete di rilevazione è posizionata in una regione prossima ad aree di nucleazione ed i target dell'allerta sono distanti rispetto ad essa. Le stazioni registrano i dati che vengono inviati in tempo reale ad un data center, il quale si occupa di determinare i parametri dell'evento e prevedere l'intensità dello scuotimento al suolo verso uno o più punti attraverso l'uso di leggi di attenuazione.

Mentre i sistemi on-site originano stime approssimative per quanto riguarda la localizzazione e la magnitudo, ma forniscono uno stima più veritiera della ampiezza massima attesa nella stessa posizione, i sistemi regionali, sfruttando le informazioni ricevute da più stazioni, riescono a fornire una localizzazione ed una stima della magnitudo più accurate, a fronte, però, di un tempo di risposta maggiore, dovuto proprio all'intervallo necessario all'acquisizione dei dati.
Per quanto riguarda la distribuzione dell'allerta i sistemi possono essere suddivisi in *Public distribution*, *Limited distribution* e *Real-time testing*. Mentre nei *Public distribution* l'allerta viene inviata in broadcast attraverso diversi canali verso tutto il pubblico, nei *Limited distribution* gli allarmi vengono diramati solamente a determinati gruppi di utenti (organismi pubblici per la gestione dell'emergenza, edifici strategici, ecc.). Infine, in diverse regioni a livello mondiale, dei sistemi sono in fase di costruzione e/o testing, ma le allerte non vengono inoltrate a nessuna tipologia di utenti diversa dalla comunità di sviluppo.

2.4.2 Sistemi di EEW nel mondo

In un recente lavoro di Allen et al. [1] viene presentata una approfondita analisi sui sistemi di EEW a livello mondiale secondo la suddivisione basata sulla distribuzione dell'allerta precedentemente descritta, mentre in [35] vengono presentati e descritti i sistemi di EEW in ambito Europeo (Fig.2.5). Il

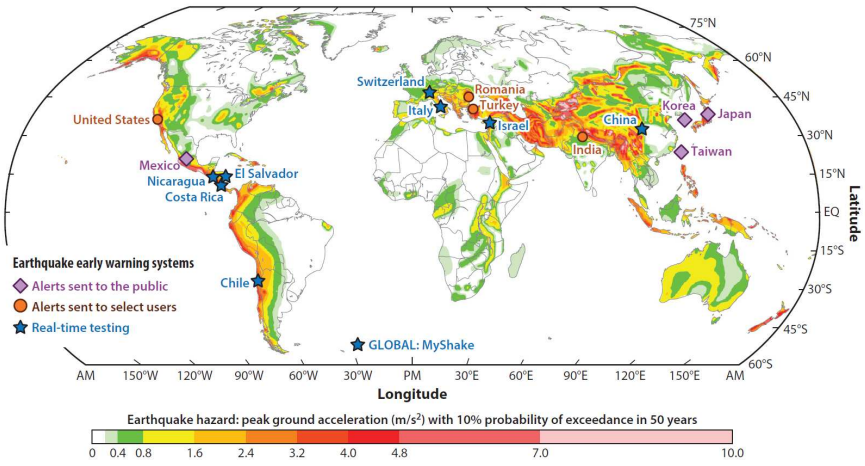

Figura 2.5: Lo stato dei sistemi di EEW nel mondo. Vengono mostrate i sistemi che emanano una allerta pubblica (viola), ad un numero selezionato di utenti (arancione) ed i sistemi attualmente in sviluppo o in fase di teste (blu). Sullo sfondo viene presentato il rischio sismico inteso come accelerazione massima al suolo con probabilità del 10% di superamento in 50 anni. *Fonte*:[1].

primo sistema a livello mondiale è stato sviluppato in Messico [36], dove è operativo dal 1991. Questo sistema utilizza il principio della valutazione della più probabile magnitudo di un terremoto rilevato da singole stazioni. Nel momento in cui 2 stazioni triggerano un evento, viene diramata una allerta verso le città dove lo squotimento atteso è superiore rispetto ad una soglia. Gli avvisi vengono emessi attraverso migliaia di ricevitori radio dedicati, distribuiti nelle scuole e negli uffici governativi. A Città del Messico, ad esempio, un allarme pubblico suona attraverso 12.000 sirene dislocate in tutta la città che possono essere udite dalla maggior parte dei residenti. La distanza della faglia dai centri abitati consente al sistema di diramare un'allerta anche fino a 60 s prima dell'arrivo delle onde distruttive. Per l'evento di M_w 7.3 del 14 settembre 1995, l'allarme è arrivato a Città del Messico con 72 s di anticipo rispetto all'arrivo delle onde S, consentendo l'evacuazione di edifici pubblici come scuole e lo stop delle linee metropolitane.
In Giappone, il sistema utilizza una combinazione dell'approccio on-site e regionale [37]. Quando una stazione rileva un picco di accelerazione che supera i 100 cm/s^2, viene triggerato un evento ed inizia la caratterizzazione dei parametri di sorgente. I tempi delle stazioni che hanno rilevato un evento vengono confrontati con le stazioni che ancora non lo hanno triggerato, in modo tale da correggere la stima della localizzazione. L'allerta viene diramata qualora l'evento superi una certa intensità attraverso canali dedicati ed anche attraverso app su smartphone, TV e radio.
La valutazione di un algoritmo denominato ElarmS [38] è stata condotta in Corea del Sud a partire dal 2008 [39]. Attualmente il sistema è in utilizzo e dirama allarmi alla popolazione mediante messaggi SMS di tipo broadcast per terremoti superiori a magnitudo 4. Anche in Taiwan viene utilizzato un sistema basato sull'analisi dei primi secondi delle onde P per determinare una stima della magnitudo dell'evento [40]. La generazione dell'allerta avviene in circa 15 s e viene inviato un messaggio a tutti i telefoni cellulari localizzati ad una distanza minima di 50 km dall'epicentro. Nella stessa regione è in via di sviluppo un sistema differente che verrà illustrato nel paragrafo 2.4.4.

Per quanto riguarda i sistemi che invece ancora non inviano un'allerta a tutta la popolazione, ma ad un numero ristretto di utenti, si segnala il sistema ShakeAlert [41] in uso sulla costa Ovest degli Stati Uniti d'America. Si tratta di un sistema distribuito formato da diversi componenti interconnessi. Questa architettura di sistema consente lo sviluppo indipendente di singoli componenti, dalle origini dati agli algoritmi, dall'associazione di eventi alla generazione ed invio di allarmi. Attualmente è stata distribuita un'app su smartphone solamente in alcune regioni. Una distribuzione dell'allerta limitata viene effettuata anche nei sistemi in uso nel nord dell'India [42] ed in Romania [43]. La Cina ha da poco iniziato la fase implementativa di un sistema a livello nazionale con

l'impiego di quasi 15000 sensori ed ingenti investimenti a livello economico [44].

In fase di test, infine, si collocano i sistemi del Cile, Israele, Svizzera ed il sistema sviluppato per l'Italia Meridionale, che viene analizzato nella prossima sezione.

2.4.3 Caso italiano: PRESTo

In Italia è in sperimentazione un sistema di EEW denominato PRESTo (PRobabilistic and Evolutionary early warning SysTem) [45]. Il sistema attualmente è operativo in tempo reale sulla rete sismica dell'Irpinia (ISNet [46]) ed è in fase di test sulla KIGAM network in Corea del Sud, sulla RoNet in Romania, sulla KOERI newtork nella regione della Marmara ed, infine, in un progetto che vede impegnate le regioni a nord-est dell'Italia, la Slovenia e l'Austria sulla CE3N network. In questa tesi si fa riferimento teorico alla sua applicazione sulla ISNet, ma grazie alla modularità e semplicità di configurazione del sistema, è stata possibile la sua applicazione pratica anche all'interno della Rete Sismica Nazionale (RSN) per quanto riguarda le stazioni controllate dall'INGV sezione di Ancona.

La ISNet è composta da 30 stazioni che in tempo reale trasmettono i segnali registrati a dei sistemi di controllo dedicati. Le stazioni sono organizzate in subnets, ognuna delle quali è composta da un massimo di 6 o 7 stazioni connesse a dei Local Control Centers (LCC), connessi a loro volta a un Network Control Center (NCC). Il sistema a livello architetturale è basato su 5 moduli.

- Acquisizione dati: in ogni LCC è presente un server SeiscomP [47]. Il NCC si collega via protocollo SeedLink (che verrà trattato nella sezione 2.5.1) ai server ed acquisisce le tre componenti accelerometriche registrate dalle stazioni. Ogni secondo che un nuovo pacchetto di dati viene ricevuto, il sistema assegna un fattore di qualità ad ogni stazione, ignorando nelle elaborazioni successive tutte le stazioni che sono considerate di qualità non sufficiente.

- Picking della fase P ed associazione: per ogni pacchetto ricevuto, il sistema effettua il picking della fase P sulla componente verticale dell'accelerazione mediante un algoritmo denominato FilterPicker [48]. Ogni pick viene memorizzato e se ne controlla la coerenza temporale con altri pick precedentemenenti memorizzati. Nel momento in cui il numero di pick coerenti supera un numero configurabile di stazioni, si procede alla fase successiva.

- Localizzazione: per tale fase, PRESTo utilizza una tecnica di localizzazione evolutiva e real-time denominata RTLoc [49]. Tale implementazione

permette di stimare la posizione ipocentrale di un evento in base alle stazioni che hanno effettuato il triggering della fase P e quelle che ancora non effettuato tale operazione. Vengono utilizzate griglie 3D precalcolate per ogni stazione contenenti i modelli di velocità ed i travel-time da ogni punto della griglia alla stazione stessa, sia per quanto riguarda le onde P che per le onde S.

- Stima della magnitudo: le componenti accelerometriche dall'istante di rilevamento della fase P, vengono filtrate e viene ricavato il rispettivo spostamento del suolo mediante doppia integrazione. Successivamente, mediante un algoritmo denominato RTMag [50], inizia la stima della densità di probabilità del valore di magnitudo.

- Stima dell'accelerazione di picco verso uno o più punti target: quando si hanno a disposizione le stime dei parametri di sorgente dell'evento è possibile, mediante l'utilizzo di leggi di attenuazione note, stimare il valore di più probabile dell'accelerazione di picco ad una certa distanza (i punti target sono anch'essi configurabili) dall'epicentro.

Il sistema può essere utilizzato sia in modalità real-time che in modalità simulazione. Nella prima modalità il sistema si interfaccia direttamente con le rete sismica sottostante come descritto in precendeza, mentre in modalità simulazione è possibile inserire nel sistema dei dati in formato SAC [51] precedentemente acquisiti o generati da una rete sismica. In questo tipo di modalità, PRESTo converte i files in stream SeedLink della durata di 1 secondo ed è possibile simulare una latenza di rete mediante configurazione.

La Fig. 2.6 riporta l'interfaccia grafica dell'applicativo PRESTo, dove nella parte sinistra vengono visualizzate in real-time le forme d'onda delle stazioni configurate. Nella parte destra, la finestra viene suddivisa in due sezioni: in quella superiore è presente una mappa della zona con triangoli verdi o rossi in corrispondenza delle stazioni attive o meno. In modalità simulazione vengono visualizzate in semi trasparenza le stazioni che non sono state considerate dall'utente, per le quali cioè non sono presenti file con registrazioni. Nella parte inferiore, viene riservata una sezione che verrà utilizzata in caso di evento sismico. Nel momento in cui ciò si verifica, nei sismogrammi vengono evidenziate linee verticali rosse in corrispondenze delle fasi P rilevate, mentre nella parte in basso a destra inizia la stima evolutiva di localizzazione e magnitudo. In contemporanea nella mappa viene riportato l'epicentro ed il lead time verso uno o più punti target. Durante il suo funzionamento, PRESTo genera un rapporto sia dell'effettivo funzionamento delle componenti sia dei calcoli effettuati in un file di log. Tale file contiene l'indicazione temporale di generazione in ogni riga ed è utile sia per il debug dell'applicazione che per fini statistici in merito alle prestazioni del sistema.

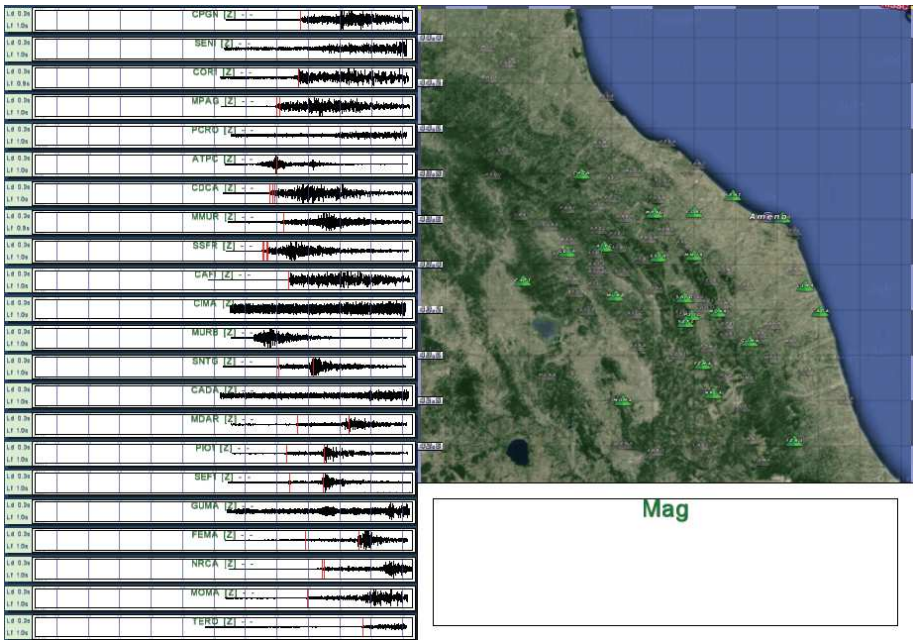

Figura 2.6: Interfaccia grafica di PRESTo durante la simulazione di un evento sismico.

In collaborazione con l'INGV Sezione di Ancona sono state sviluppate le configurazioni richieste dal sistema per le stazioni riportate in Fig. 2.7. Sono stati redatti quindi i modelli di velocità per ogni stazione, calcolati i coefficienti per le leggi di regressione ed i parametri di configurazione per il picking della fase P. Ad oggi il sistema è in piena fase di test e vista la notevole attività microsismica della catena appenninica dell'Italia centrale è possibile settare i parametri con una verifica sul campo.

2.4.4 Sistemi di EEW basati sul paradigma IoT

Negli ultimi anni la comunità sismologica si è concentrata sullo studio di dispositivi a basso costo al fine di costruire una rete di monitoraggio sismico più fitta. Un nuovo tipo di accelerometri basati su MEMS è stato introdotto nelle applicazioni sismiche [52]. Esempi molto significativi di MEMS e sensori a basso costo utilizzati per creare reti sismiche dense sono la Community Seismic Network (CSN) [53], la Quake-Catcher Network (QCN) [54], e il P-alert system in Taiwan [55]. Il CNS è costituito da accelerometri MEMS a tre componenti a basso costo in grado di registrare accelerazioni fino al doppio del livello di gravità. I sensori sono ospitati da volontari e la telemetria è fornita tramite Internet. Il prodotto principale del CSN è una mappa dello scuotimento del

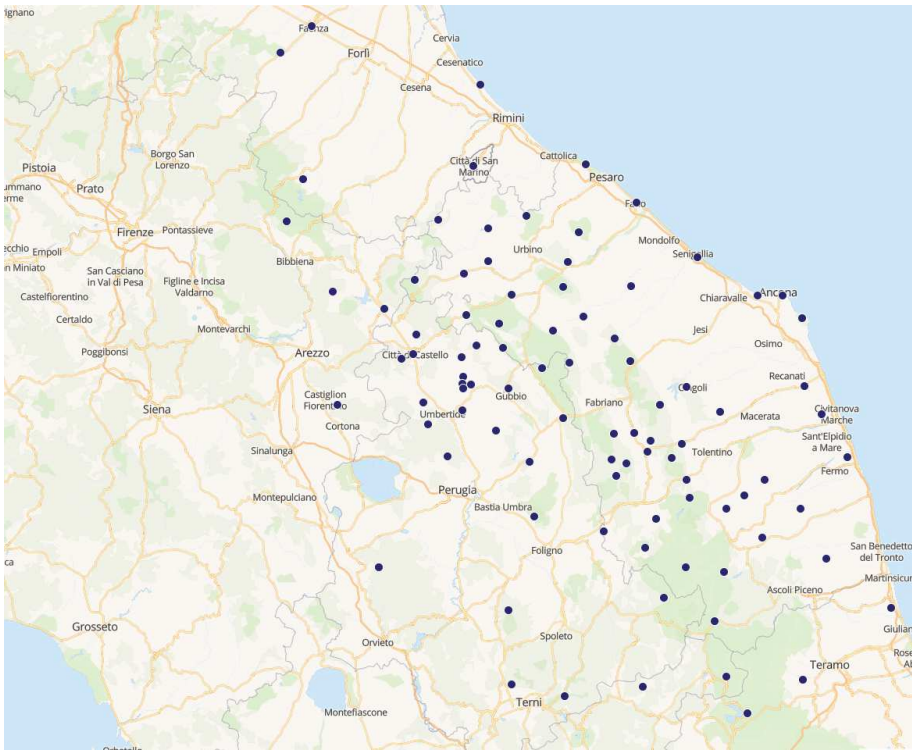

Figura 2.7: Una mappa dell'Italia centrale dove i punti blu indicano la localizzazione delle stazioni utilizzate nella configurazione dell'applicativo.

suolo che può essere erogata entro pochi secondi dallo scuotimento maggiore. Lo stesso concetto di rete sismica che implementa il calcolo distribuito/volontario è alla base del QCN. A differenza del CSN, in questo progetto il dispositivo ospitato invia un trigger al server QCN solo quando si verifica un rilevamento significativo, in modo tale da evitare un eccessivo carico al server.
Il sistema P-alert consiste in una serie di dispositivi di allarme denominati Palert che utilizzano accelerometri MEMS per EEW di tipo on-site. Ogni dispositivo ha anche capacità di rete per costruire un sistema EEW regionale.

Grazie alla tecnologia smartphone, al paradigma IoT e al modello crowdsourcing, nell'ultimo decennio si è assistito allo sviluppo di piattaforme non ufficiali parallelamente ai sistemi EEW gestiti da enti governativi a livello nazionale. Il Self-organizing Seismic Early Warning Information Network (SOSEWIN) [56] è costituito da componenti a basso costo che intraprendono un'elaborazione dei dati on-site, effettuano un'analisi preliminare, procedono all'archiviazione e comunicazione dei dati. SOSEWIN è una rete wireless mesh ad-hoc decentralizzata e auto-organizzata in cui ogni nodo sensore è in grado di comunicare

nel modo più veloce con il nodo principale più vicino. L'architettura è progettata in modo da adattarsi al numero di nodi sensori installati, ma si basa su protocollo di routing ad-hoc.
Un altro tipo di approccio è basato sull'utilizzo dei sensori con cui sono equipaggiati i moderni smartphone. Il progetto MyShake [57] e l'Earthquake Network Project [58] utilizzano un'app da scaricare ed installare su smartphone che raccoglie i campioni accelerometrici e li invia ad un server centrale che si occupa della rilevazione dell'evento sismico. L'app Earthquake Network Project inizia a raccogliere dati solo quando lo smartphone è in carica e non in uso mentre l'app MyShake è costantemente attiva ed è in grado di distinguere tra movimenti dovuti alla vita quotidiana rispetto a possibili tremori dovuti a un terremoto grazie ad algoritmi di intelligenza artificiale. Nell'Earthquake Network Project, grazie ad un algoritmo statistico, il server decide in tempo reale se si sta verificando un terremoto ed in questo caso inviare un avviso agli utenti attorno all'epicentro. Nell'app MyShake, quando l'algoritmo rileva un possibile terremoto, viene inviato un trigger al centro di elaborazione dove un algoritmo di rilevamento conferma che è in corso un terremoto ed inizia a determinare la posizione, l'ora di origine e la magnitudo. L'utilizzo degli smartphone viene proposto anche in [59], dove gli autori illustrano un'architettura a 3 livelli composta da una rete di sensori utilizzata per il rilevamento di un evento sismico, che viene quindi confermato da un server intermedio il quale invia le notifiche all'ultimo livello mediante il protocollo MQTT. L'uso di un dispositivo autonomo dotato di un sensore di accelerazione a basso costo e di minime risorse di calcolo viene presentato in [60] e [61].

Infine, citiamo due progetti che utilizzano il Cloud come strato intermedio. Il progetto SeismoCloud [62] e il progetto Earthcloud [63]. La maggior parte dei sismometri della rete SeismoCloud è composta da smartphone iOS e Android dotati dell'app SeismoCloud che monitora le vibrazioni solo quando il telefono è su una superficie piana, ma è possibile anche utilizzare un dispositivo a basso costo connesso al Cloud. Ad oggi, a causa della limitata diffusione, l'applicazione non è in grado di inviare alcun tipo di notifica. Il progetto Earthcloud si basa su un dispositivo IoT a basso costo dotato di geofoni. I dati generati vengono incapsulati in messaggi MQTT inviati ad Amazon Web Service (AWS) IoT Core. Viene emesso un avviso quando i dispositivi da una stessa posizione superano una soglia impostata.

2.5 Protocolli trasmissivi per reti sismiche

In un sistema di EEW, studiare i ritardi che ogni componente aggiunge riveste notevole importanza. Partendo dal lavoro presentato in [64], si evince come i contributi più importanti in termini di tempo vengono introdotti dalla

data latency, intesa come somma di tempi di pacchettizzazione e di invio dei campioni rilevati dal dispositivo.
Grazie ai continui progressi nel campo delle telecomunicazioni, la trasmissione in tempo reale dei dati rilevati dalle stazioni è divenuta sempre più comune. Ad oggi non esiste un protocollo concordato a livello internazionale per la trasmissione real-time ed i produttori di apparecchiature spesso utilizzano i propri formati. Tuttavia, il numero di protocolli è limitato ed i sistemi più comuni sono SeedLink, SCREAM, NAQS, CD1.x, Antelope, EarthWorm, NRTS e LISS [65]. Il primo protocollo sviluppato è stato proprio il LISS, il quale consiste nell'invio di pacchetti dati su di un socket. Il ricevitore può aprire il socket corrispondente e ricevere in tempo reale i dati. Il LISS ha lo svantaggio che, se i dati vengono persi, non c'è alcun modo di richiederli a causa dell trasmissione unidirezionale. Problema che è stato risolto grazie alla implementazione del protocollo SeedLink attraverso il quale è possibile richiedere anche dati precedenti a livello temporale rispetto agli attuali. Nella prossima sezione viene analizzato tale protocollo, essendo il più utilizzato e soprattutto essendo impiegato nel sistema PRESTo.

2.5.1 Il protocollo SeedLink

Il protocollo SeedLink [66] è lo standard de-facto per la trasmissione in tempo reale delle tracce sismiche ed è basato su TCP/IP. La connessione viene inizializzata dal client il quale, durante la fase di handshake, si sottoscrive a specifici stream di dati. La caratteristica fondamentale del protocollo è costituita dalla robustezza: deve infatti tollerare client che possono disconnettersi e riconnettersi in maniera aleatoria e provvedere affinché non vi sia perdita di dati. Le eventuali trasmissioni perse possono essere recuperate purché esse siano ancora presenti nei buffer del server. Dopo l'apertura della connessione e la seguente fase di handshake, il server inizia l'invio dei dati che consistono in 8 byte di un header seguiti da 512 byte di record miniSEED.
I pacchetti inviati sono identificati da un numero di sequenza, univoco per ciascun pacchetto del flusso. Questo accorgimento fa sì che, in caso di discontinuità dell'infrastruttura di rete, sia possibile riottenere i pacchetti a partire dal momento in cui si era interrotto il flusso, eliminando così la perdita di pacchetti dovuta all'interruzione temporanea della connessione. La capacità di riprendere un flusso dati è legata alla quantità di dati che nel tempo il dispositivo SeedLink può contenere all'interno del suo buffer. Esistono, inoltre, pacchetti accessori generati dai server Seedlink e utilizzati per la comunicazione di dettagli da parte del server ai vari client e per implementare lo scambio dei pacchetti di tipo "keep alive". Questi pacchetti accessori sono in formato XML e sono incorporati in comuni record miniSEED (all'interno di comment records).

Il protocollo, inoltre, consente due diverse modalità di trasmissione dati denominate *Unistation* e *Multistation*. La prima modalità funziona trasmettendo un singolo flusso di dati (da una singola stazione) attraverso una connessione di rete. In questo modo non è necessario che il client specifichi l'origine del flusso dati, poiché è implicito nell'indirizzo Internet e nella porta. Nel caso *Multistation* il protocollo funziona trasmettendo un flusso dati multiplexato di più stazioni attraverso l'utilizzo di una sola connessione di rete.

Il miniSEED è un sottoinsieme dello Standard for the Exchange of Earthquake Data (SEED), che è uno standard realizzato allo scopo di permettere lo scambio di dati sismici grezzi tra la comunità scientifica o le istituzioni mediante dispositivi elettronici e reti a commutazione di pacchetto. Il formato SEED è diviso in due parti, una contenente i dati delle serie temporali (miniSEED) e una contenente informazioni come gli identificatori di rete e delle stazioni e le risposte dello strumento, chiamate dataless SEED. Il miniSEED contiene una sezione fissa dell'intestazione che fornisce informazioni sulla codifica dei dati, nonché informazioni di base sugli identificatori di rete e stazione, posizione, identificatore di canale, lunghezza dei dati della serie temporale. La parte rimanente contiene i dati effettivi delle serie temporali. I record miniSEED vengono trasmessi tramite SeedLink in pacchetti da 512 byte, quindi è necessario attendere il completo riempimento di ogni pacchetto prima della sua trasmissione. Effettua, inoltre, una compressione del dato dipendente dalla velocità con cui il segnale varia, per cui il numero di campioni all'interno di un pacchero è variabile e può coprire un intervallo temporale di diversa durata, non necessariamente un multiplo esatto di un secondo.

Si è detto che dopo che la connessione TCP/IP è stata stabilita, il SeedLink attende una prima fase di handshaking. Durante questa fase il client invia i comandi al server SeedLink, inclusa la selezione del flusso secondo il Seed Station Naming Convention (SSNC). Questa convenzione assegna codici appropriati per identificare un flusso attraverso la sequenza di codice di rete, codice stazione, codice posizione, codice canale. Il Codice di rete è assegnato dalla International Federation of Digital Seismic Networks (FDSN) e consiste in un'abbreviazione di una o due lettere che identificano la rete (ad esempio, "IV" è la Rete Sismica Nazionale Italiana, "IX" la Rete Sismica Irpinia). Il codice stazione si riferisce a una posizione fisica in cui si trovano gli strumenti ed è rappresentato da un massimo di cinque lettere. Il codice posizione è composto da due lettere o cifre che aiutano a distinguere canali o strumenti con nomi simili che si trovano nella stessa stazione. Infine, il Channel Code rappresenta il singolo flusso di dati ed è composto da tre lettere che indicano il codice della banda (es. "H" per High Broad Band), lo strumento (es. "N" per accelerometri) e l'orientamento (Verticale, Nord-Sud, Est-Ovest).

Uno degli svantaggi più evidenti del protocollo riguarda proprio il fatto che

i dati vengano inviati in pacchetti miniSeed da 512 byte. Nei datalogger pertanto possono essere scelte due vie alternative, cioè attendere il riempimento completo del pacchetto oppure attendere un intervallo temporale ed inviare il pacchetto inserendo contenuto nullo fino alla sua dimensione massima. Se da un lato la prima modalità consente di sfruttare a pieno la banda disponibile, dall'altro la seconda modalità risulta essere più appropriata in un contesto di Early Warning, mantenendo i ritardi dovuti alla pacchettizzazione sull'ordine del secondo. Quale che sia la scelta implementativa nei datalogger, risulta sempre necessario attendere del tempo prima dell'invio effettivo dei campioni.

2.6 La soluzione proposta

Un sistema di monitoraggio sismico può essere studiato a livello architetturale come una applicazione IoT integrata con il mondo Cloud. Come mostrato in Fig.2.8 i dispositivi nel *Perception Layer* inviano i propri dati al Cloud nel *Middleware Layer*; questo livello è responsabile del processamento dei dati ricevuti e dell'invio delle informazioni ricavate verso l'*Application Layer*, il quale espone i servizi specifici per gli utenti. Pertanto l'idea è quella di studiare tecnologie utilizzate nel mondo IoT e nel mondo Cloud e la loro possibile applicazione nella pacchettizzazione, trasmissione e processamento di dati sismici in una applicazione EEW.

Quando si parla di tecnologie si fa riferimento a tutti i livelli dello stack architetturale proposto, dall'utilizzo di protocolli trasmissivi utilizzati nel paradigma IoT, all'impiego di sensori a basso costo, dalla possibile implementazione di diverse tecniche di pacchettizzazione, alla loro trasmissione verso il Cloud, alla scelta della piattaforma Cloud più consona all'applicazione oggetto di studio. Allo stesso tempo, analizzare le performance della soluzione proposta riveste fondamentale importanza. A tal scopo, si è utilizzato il sistema italiano PRESTo come standard di riferimento. Grazie alla collaborazione con l'INGV Sezione di Ancona è stato possibile configurare l'applicativo in modo tale da ricevere i dati da alcune stazioni della RSN. Allo stesso tempo, è stato possibile simulare eventi sismici reali sulla rete, comparando quindi i risultati ottenuti da PRESTo con ciò che avremmo potuto ottenere con una stessa conformazione delle rete, ma con dispositivi differenti e con differenti tecniche implementative, sia per quanto concerne la trasmissione dei dati, che per lo stack tecnologico utilizzato.

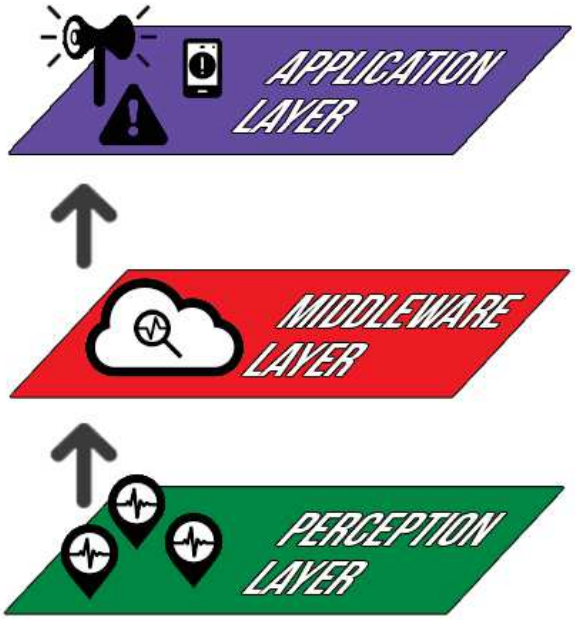

Figura 2.8: Generica architettura a 3 livelli per una applicazione Cloud-IoT finalizzata all'EEW.

Capitolo 3

Utilizzo del protocollo MQTT per sistemi di EEW

3.1 Introduzione

Se si utilizzano una o più stazioni sismografiche, non solo per rilevamenti sismici, ma anche per servizi di Early Warning, è evidente che si renda necessario un sistema di trasmissione a bassa latenza. Considerando che i dispositivi sono sul campo, e che le risorse trasmissive non sono illimitate, particolare importanza è rivolta di conseguenza allo studio di sistemi di telemetria ed invio dati che siano non solo efficienti, ma anche veloci.
Nel caso dell'invio di dati miniSEED tramite protocollo SeedLink, la latenza è definita dal numero di campioni per pacchetto. Un numero fisso di campioni per pacchetto garantisce una latenza fissa. Il miniSEED implementa nativamente diverse algoritmi di compressione (es. STEIM1-2) e si potrebbe pensare che il processamento dell'informazione e la compressione precedente all'invio determinino necessariamente un aumento della latenza. Andando ad agire sulla dimensione del pacchetto e sulla frequenza di invio dello stesso è tuttavia possibile agire anche sulla latenza al fine di modificarla e rendere tale valore arbitrariamente piccolo. Ma nel caso in cui, come nel protocollo SeedLink, i pacchetti da trasmettere debbano essere composti da un numero fisso di byte, la compressione fa aumentare la latenza; ne segue che decidendo il numero di elementi che devono essere in un pacchetto è possibile agire sulla latenza in maniera arbitraria. Nel sistema PRESTo, ad esempio, a livello progettuale è stata effettuata la scelta di inserire nel pacchetto miniSEED 1 secondo di campioni e riempire con contenuto nullo la rimanente parte sino ai 512 byte previsti dal protocollo.

Uno dei protocolli più utilizzati nell'IoT è il protocollo MQTT. È un protocollo basato su un pattern publish/subscribe. Il ruolo principale è ricoperto dal broker di messaggi, che ha un ruolo intermediario tra la pubblicazione dei messaggi dei client e la ricezione dei messaggi dei subscriber. Diversi studi in letteratura hanno dimostrato i migliori risultati di MQTT in termini di latenza

end-to-end e consumo di larghezza di banda riferiti ad altri protocolli utilizzati nelle applicazioni IoT come HTTP [67], AMQP [68] o COAP [69]. Grazie alla struttura semplice, MQTT può essere facilmente implementato in dispositivi con processori a bassa potenza e basse prestazioni e si sta rapidamente diffondendo.

Lo scopo di questo capitolo è dimostrare che l'implementazione del protocollo MQTT può ridurre significativamente sia il ritardo di pacchettizzazione e compressione introdotto dal datalogger, sia il ritardo di trasmissione del sistema di comunicazione. A tal fine, sono stati confrontati i risultati ottenuti sul picking dell'onda P con PRESTo e con la nostra soluzione, valutati su un dataset di terremoti italiani. L'idea non è quella di sostituire il protocollo standard SeedLink, ma di studiare la possibilità di implementazione del protocollo MQTT parallelamente agli standard attuali implementati nei datalogger.

3.1.1 Il protocollo MQTT

Il protocollo MQTT è uno standard ISO nato nel 1999 studiato per finalità di telemetria: è estremamente leggero e affidabile, anche nel caso di reti con connessioni non perfettamente stabili (delay and distruption tolerant). Essendo pensato principalmente per la comunicazione M2M, grazie alle proprietà di leggerezza, affidabilità e stabilità, riesce a garantire anche una bassa latenza. Il protocollo prevede una sorgente (detta *Publisher*) che pubblica messaggi su di una coda specifica denominata *Topic* ospitata su di un server detto *Broker*. Un client, denominato *Subscriber*, può sottoscriversi ad un determinato Topic in modo tale da ricevere dal Broker aggiornamenti riguardo le nuove pubblicazioni avvenute nel Topic per cui è abbonato. I Topic sono semplici stringhe gerarchiche e sono definiti senza nessun formato specifico. Ogni client è in grado di pubblicare e ricevere messaggi allo stesso tempo, ma, per come è strutturata la rete, due client non possono scambiarsi messaggi tra loro se non passando per il Broker. Potendo più utenti prelevare il messaggio salvato sul Broker MQTT, la comunicazione è definita "one to many" (un singolo broker può inoltrare lo stesso messaggio a più client). Il meccanismo publish/subscribe previsto dall'MQTT coinvolge quindi tre attori fondamentali:

- **Publisher**: client che produce i dati e li invia al Broker.
- **Subscriber**: client che si iscrivono ad un determinato Topic di loro interesse e ricevono notifiche ogni qual volta un nuovo messaggio nel Topic desiderato è disponibile.
- **Broker**: filtra i dati ricevuti sulla base del Topic e li distribuisce ai vari Subscriber.

I ruoli di Publisher e Subscriber sono puramente logici; lo stesso dispositivo può fungere da Publisher per un Topic e contemporaneamente Subscriber per un altro. I messaggi scambiati tramite MQTT possono essere di qualsiasi formato, sebbene nell'ambito IoT grande rilevanza viene data al formato JSON (Java Script Object).

L'MQTT prevede la consegna dei messaggi in base al livello di Quality of Service (QoS) assegnatogli nella fase di subscription. Ogni client viene trattato in maniera diversa e indipendente in base al livello di QoS scelto per il messaggio. Le QoS previste dal protocollo sono le seguenti:

- **At most once (QoS 0)**: il messaggio viene consegnato in relazione alle capacità della rete. Non viene inviata nessuna notifica di ricezione dal ricevitore e il trasmettitore invia il messaggio una sola volta, pertanto non è prevista garanzia di effettiva ricezione del messaggio.
- **At least once (QoS 1)**: tale livello di QoS garantisce che il messaggio arrivi almeno una volta al ricevitore, il quale notifica l'avvenuta ricezione mediante l'invio di un ACK al broker.
- **Exactly once (QoS 2)**: in questo livello non è contemplata la perdita del messaggio e la duplicazione dei pacchetti. Mediante un meccanismo a quattro vie il messaggio verrà consegnato una ed una sola volta.

3.2 Parametri di studio

La soluzione proposta è quella di effettuare una pacchettizzazione utilizzando il formato JSON ed inviare i pacchetti mediante il protocollo MQTT. Nel caso in esame, per ridurre l'arbitrarietà del client publisher (il dispositivo sismico), è stato scelto di utilizzare i topic secondo il SSNC descritto in precendenza, pertanto con la forma:

`Network/Station/Location/Channel`

Per quanto riguarda il messaggio, la struttura del payload sarà la seguente:

```
{
"timestamp" :  <Integer>,
"values" :  [Array],
"encoding" :  <String>,
"sps" :  <Integer>
}
```

dove nel campo *timestamp* viene inserito il valore in millisecondi del timestamp del primo valore contenuto nell'array *values*. Questo array può essere

di lunghezza arbitraria, può cioè contenere un numero configurabile di campioni. I dati possono essere codificati secondo diversi formati (ad esempio INT16, INT32, FLOAT32, FLOAT64, STEIM1, STEIM2) [70]; questa informazione è contenuta nel campo *enconding*. Il campo *sps* contiene il numero di campioni per secondo, da cui è possibile quindi ottenere la frequenza di campionamento. Tramite l'utilizzo di questo schema è possibile scegliere il numero di campioni da inserire all'interno dell'array, fino ad arrivare alla teorica trasmissione sample-by-sample.

L'obiettivo è stato duplice: da un lato comparare i tempi di latenza dovuti all'utilizzo del protocollo SeedLink e del protocollo MQTT, dall'altro verificare eventuali guadagni in termini di tempo dovuti all'implementazione della soluzione proposta rispetto alle tradizionali soluzioni miniSEED/SeedLink.

3.2.1 La trasmissione real-time delle tracce acquisite

Al fine di confrontare le latenze dei due protocolli, sono stati trasmessi identici pacchetti miniSEED tramite SeedLink e come payolad di un messaggio MQTT. Il sistema PRESTo, come descritto precedentemente, può essere utilizzato in modalità simulazione mediante l'utilizzo di file di tipo SAC contenenti le registrazioni delle stazioni in esame. Il sistema si occupa della traduzione in pacchetti miniSEED di 1 secondo ed è possibile simulare eventuali latenze o ritardi di rete mediante la configurazione di un parametro. Nonostante la presenza di questa importante feature, una simulazione più realistica prevede l'utilizzo di uno o più server SeedLink reali.

Come mostrato in Fig. 3.1, partendo da tracce sismiche, sono state create finestre temporali mobili di un secondo, al fine di simulare quindi il comportamento real-time di un datalogger collegato al sistema PRESTo. Ogni secondo il pacchetto viene inviato ad un SeedLink server ed a un broker MQTT che risiedono in una stessa macchina. Il dispositivo simulato e l'architettura descritta sono quindi perfettamente conformi ad un sistema composto da datalogger e acquisitore di una seismic network generica, aggiungendone la funzionalità di server MQTT. Sono stati generati i due client SeedLink ed MQTT, mendiante l'utilizzo della libreria Python Obspy [71] e della libreria Paho [72], all'interno di un'altra macchina e tramite un analizzatore di rete sono stati registrati i tempi di ricezione del pacchetto. La differenza tra l'istante di ricezione e l'istante di invio di un medesimo pacchetto permette di confrontare le latenze dei due protocolli, in quanto le condizioni di analisi sono le medesime (larghezza di banda, traffico di rete, tempo dedicato alla pacchettizzazione, etc.). Particolare attenzione è stata dedicata alla sincronizzazione della macchina che simula il dispositivo sismico ed della macchina dove risiedono i client SeedLink ed MQTT tramite l'utilizzo del protocollo NTP [73].

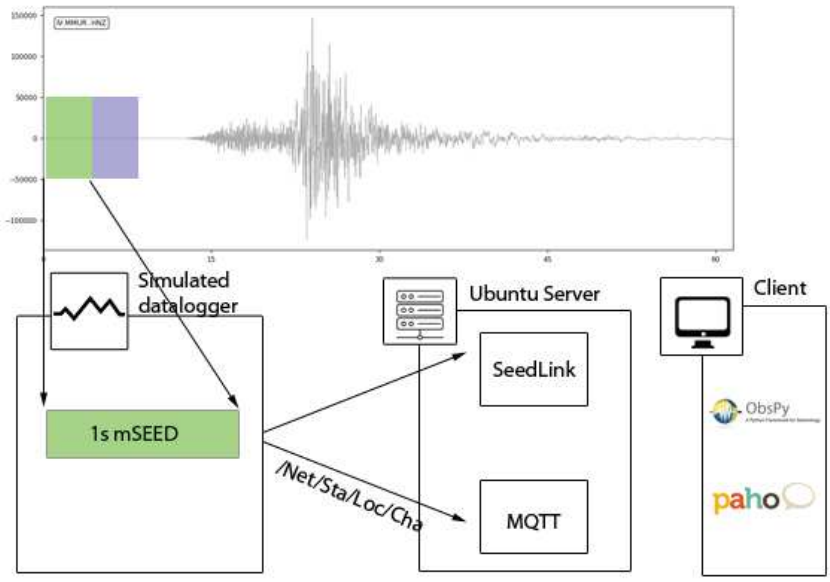

Figura 3.1: Schema della metodologia per la valutazione dei tempi di trasmissione delle forme d'onda mediante il protocollo SeedLink ed il protocollo MQTT.

3.2.2 Il picking della fase P

Per effettuare il secondo confronto è stato utilizzato come standard di riferimento il log in output del software PRESTo. Per non incorrere in errori di valutazione dovuti a differenti scelte implementative, è stata condotta una analisi preliminare. Il primo interesse in questa fase è valutare che i tempi di rilevazione di una fase P in un'onda sismica sottoposta a PRESTo siano gli stessi che vengono rilevati da una soluzione sviluppata internamente. Mantenendo inalterate le modalità di pacchettizzazione ed invio, è stato quindi sviluppato un client SeedLink tramite l'utilizzo della libreria Obspy, che supporta anche estensioni per il real-time signal processing. Per la rilevazione della fase P di ogni traccia è stata utilizzata l'implementazione Python del FilterPicker [74], settando i parametri di input come calcolati dall'INGV-An per il test di PRESTo, come riportato in Table 3.1.

Sono state eseguite, quindi, delle simulazioni su tracce accelerometriche, nelle stesse modalità riportate per il confronto delle latenze dei protocolli, tralasciando l'invio in MQTT al broker. È stato quindi creato il client SeedLink tramite Obspy nella stessa macchina dove risiede PRESTo. Il client creato fornisce in output l'istante di rilevazione della fase P e l'istante della fase stessa. Confron-

Tabella 3.1: Parametri utilizzati per il FilterPicker

Parameter	**Value**
Filter Window (T_{filter})	1.0 s
Long Term Window (T_{long})	5.0 s
Threshold 1 (S_1)	15
Threshold 2 (S_2)	15
tUpEvent (T_{up})	0.1 s

tando tale output con il log creato dal software di riferimento ci si aspettano risultati pressoché identici, sia in termini di tempo di rilevazione che di calcolo real-time sulla traccia. La simulazione è stata ripetuta su ogni traccia per dieci volte, ottenendo per ogni simulazione risultati coincidenti, confermando quindi la correttezza della soluzione utilizzando le librerie citate. È stato possibile, di conseguenza, creare tramite le stesse librerie il client MQTT che compie le stesse operazioni del client SeedLink, processando però il payload JSON anzichè il pacchetto miniSEED.

3.2.3 Metriche di comparazione

Al fine di studiare le differenze temporali nell'utilizzo delle due diverse modalità, andiamo a definire le metriche da comparare. Facendo riferimento alla Fig.3.2, per quanto riguarda il datalogger, definiamo come

$$T_{pack,i} = t_{ep,i} - t_{sp,i} \tag{3.1}$$

dove $T_{pack,i}$ è il tempo di pacchettizzazione dell'i-esimo pacchetto, ottenuto da $t_{sp,i}$ e $t_{ep,i}$, cioè l'istante del primo ed ultimo campione nel pacchetto stesso. $T_{pack,i}$ dipende dal sampling rate e dal numero di campioni contenuti in ogni pacchetto. Dal punto di vista del client definiamo $t_{r,i}$ come il tempo di arrivo del pacchetto i al client. Pertanto il *data latency* $T_{l,i}$ del pacchetto i-esimo è definito come la differenza temporale tra l'istante dell'ultimo campione nel pacchetto ed l'istante di ricezione dello stesso:

$$T_{l,i} = t_{r,i} - t_{ep,i} \tag{3.2}$$

Questa quantità contiene anche il tempo necessario al datalogger per l'invocazione della funzione di pacchettizzazione.
Il *picker time* T_{pick} è definito come il tempo necessario al rilevamento della fase P una volta che il pacchetto è arrivato al client che si occupa del processing ed è ottenuto mediante differenza tra il timestamp di triggering t_{trig} e $t_{r,i}$:

$$T_{pick} = t_{trig} - t_{r,i} \tag{3.3}$$

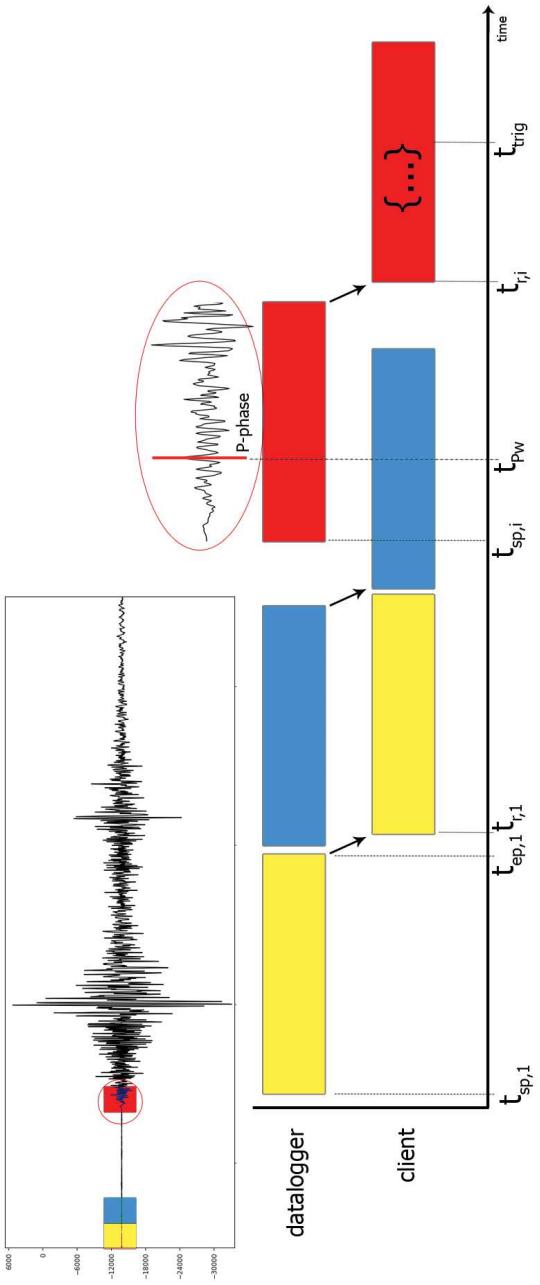

Figura 3.2: Riepilogo delle metriche considerate. Il datalogger riempie ogni pacchetto della traccia e lo invia al client. Durante la pacchettizzazione del pacchetto indicato in colore rosso ha inizio un evento sismico con conseguente ritardo della sua rilevazione sul client.

Infine, è possibile definire le metriche di performance: la prima è il tempo totale T_{tot}, cioè la somma dei tempi di pacchettizzazione, latenza e picking, ossia la differenza tra l'istante di triggering e l'istante temporale del primo campione nel pacchetto:

$$T_{tot} = T_{pack,i} + T_{l,i} + T_{pick} \tag{3.4a}$$

cioè:

$$T_{tot} = t_{trig} - t_{sp,i} \tag{3.4b}$$

La seconda metrica di performance è il tempo di rilevamento, indicato come la differenza il risultato dell'algoritmo di triggering (cioè la fase P stessa, t_{Pw}) e t_{trig} presentato in precedenza:

$$T_{det} = t_{trig} - t_{Pw} \tag{3.5}$$

Abbiamo generato uno script che legge il contenuto di una cartella contenente le tracce accelerometriche di eventi sismici reali. Per ogni traccia vengono attivati due processi in contemporanea che si occupano della pacchettizzazione e della trasmissione nelle due modalità. Il processo che si occupa della modalità tradizionale, divide la traccia in esame in finestre di un secondo e riempie un pacchetto miniSEED per ogni finestra, inviandolo ad un server SeedLink interno. Il processo che si occupa invece della seconda modalità, invia un JSON contenente un campione alla volta via MQTT al passo di campionamento. Anche in questo caso il broker MQTT risiede internamente alla macchina che simula i dispositivi.

3.3 Analisi delle prestazioni

I client sono stati simulati in una macchina con le seguenti caratteristiche: Intel Xeon X5650 (x2) CPU, 12 MB cache, 2.66 GHz, 16 GB RAM con Ubuntu 18.04.1 LTS. Le caratteristiche della macchina sono molto performanti, questo perchè vengono simulati molti dispositivi in contemporanea. Al fine di verificare la fattibilità pratica della soluzione in casi reali, dove gli acquisitori hanno solitamente limitate capacità di elaborazione, abbiamo simulato un solo dispositivo tramite l'utilizzo di una Raspberry Pi 3 Model B con CPU Quad Core 1.2 GHz Broadcom BCM2837 64 bit ed 1 GB RAM, installando al suo interno lo script per la lettura di una sola traccia accelerometrica e relative chiamate ai due processi descritti in precedenza, un SeedLink server ed un broker MQTT. Per quanto riguarda il server SeedLink abbiamo installato e configurato il *ringserver* [75] distribuito da *Incorporated Research Institutions for Seismology* (IRIS) [76]. Per quanto riguarda il broker, abbiamo utilizzato Mosquitto [77], un message broker leggero ed open source che implementa il

protocollo MQTT. I risultati ottenuti sul Raspberry evidenziano un utilizzo della CPU mai andato oltre il 40% durante le simulazioni, confermando pertanto la possibilità di implementazione anche su dispositivi meno performanti. Sono stati sviluppati ed installati in una macchina sotto una rete differente i client che si occupano della ricezione real-time delle tracce trasmesse tramite i due protocolli e del triggering della fase P. Entrambi i client sono stati sviluppati mediante l'utilizzo della libreria Python Obspy e sono stati sincronizzati con il simulatore tramite protocollo NTP. Ad entrambi i client viene passato un file di configurazione contenente le stazioni simulate (e relativi parametri secondo il SSNC). Ogni client crea tanti processi quante le stazioni contenute nel file ed ogni processo effettua il P-phase picking su 10 s di traccia mediante il FilterPicker. Per quanto riguarda il client SeedLink è stato utilizzata la classe `EasySeedLinkClient` in `obspy.clients.seedlink.easyseedlink`, mentre per il client MQTT abbiamo utilizzato la libreria Paho-MQTT.

3.3.1 Dati utilizzati

Gli eventi sismici su cui effettuare l'analisi sono stati selezionati dall'INGV Strong Motion Database (ISMD) [78], che fornisce le forme d'onde ed i relativi metadati in quasi real-time per i terremoti di $M_w \geq 3.0$ occorsi nel territorio italiano e localizzati dall'INGV. Per questa analisi sono stati selezionati tutti gli eventi avvenuti dal 01-01-2016 al 27-05-2020 ad una latitudine compresa tra 42.02 e 43.82 ed una longitudine compresa tra 12.04 e 13.4. Nella mappa alla sinistra della Fig. 3.3 viene riportata con dei cerchi la distribuzione geografica dei 200 eventi selezionati, con la rispettiva magnitudo evidenziata proporzionalmente alla dimensione del diametro. In basso a destra è riportata la distribuzione della magnitudo degli eventi considerati.

Per ogni evento sono state selezionate le forme d'onda delle stazioni riportate in Fig. 2.7, se presenti, tramite l'utilizzo del package `massDownloader` della libreria Obspy, che contiene funzionalità per effettuare query ed integrare i dati da un numero di web services della FDSN simultaneamente. L'interrogazione è stata ristretta al web service provider dell'INGV sulla rete sismica IV selezionando successivamente 6 minuti per ogni traccia su ogni canale (3 minuti precedenti e successivi la dichiarazione dell'evento), ottenendo un totale di 13818 tracce miniSEED.

Prima dell'effettivo start della simulazione, uno script si occupa di preprocessare le tracce. Le operazioni effettuate sono le seguenti:

1. ricavare il minimo start-time del set delle tracce;

2. calcolo della differenza temporale tra il minimo start-time del dataset e lo start-time di ogni traccia;

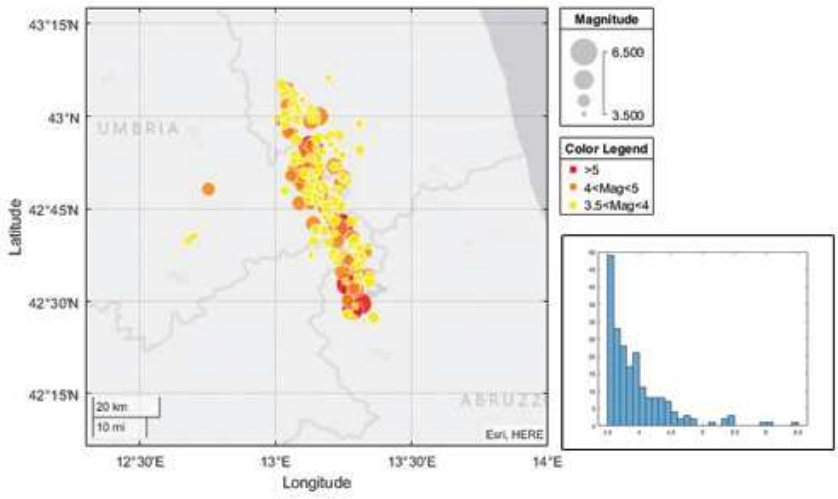

Figura 3.3: Distribuzione degli eventi sismici considerati in questo lavoro. Le diverse dimensioni ed i colori dei cerchi sono in relazione alla magnitudo. In basso a destra: distribuzione della magnitudo degli eventi elaborati.

3. per ogni traccia ricavare valor medio e deviazione standard dei primi 400 campioni (momento di quiete);

4. aggiungere rumore bianco sulla base dei valori calcolati precedentemente all'inizio della traccia fino al valore di minimo start-time (tanti valori quanti calcolati al punto 2).

Una volta compiute queste operazioni, tutte le tracce hanno lo stesso istante di inizio. Per simulare effettivamente il comportamento real-time abbiamo inoltre modificato l'istante di inizio di ogni traccia con l'effettivo timestamp di inizio simulazione. In questo modo è come se l'evento sismico fosse in corso di svolgimento durante la simulazione stessa.

Il protocollo MQTT supporta, come illustrato in precedenza, 3 livelli di QoS. Le simulazioni sono state effettuate con una QoS pari ad 1 (*at least once*), al fine di riprodurre il comportamento del protocollo SeedLink. Tutte le misurazioni sono state condotte al termine della fase di connessione e sottoscrizione del protocollo MQTT e successivamente alla fase di apertura della connessione e di handshake del protocollo SeedLink.

3.3.2 Confronto tra MQTT e SeedLink

Lo scopo di questa misurazione è quello di verificare le latenze dei due protocolli in esame, intesa come la differenza temporale tra l'istante di ricezione e l'istante di invio di un medesimo pacchetto miniSEED, inviato tramite SeedLink o come payload di un messaggio MQTT. Per quanto riguarda il client MQTT, per istante di ricezione si intende l'istante in cui il payload del messaggio viene estratto e può essere processato come pacchetto miniSEED, così da non incorrere in errori dovuti all'utilizzo di differenti tecniche di processing di ogni pacchetto ricevuto. In Tabella 3.2 sono riportate le statistiche dei risultati ottenuti dai due procolli. Il protocollo MQTT presenta i risultati migliori in termini di latenza, attestandosi su un tempo di consegna medio di 33.17 *ms* con una deviazione standard pari a 17.03 *ms*, mentre il protocollo SeedLink risulta avere un ritardo medio pari a 696.13 *ms* con deviazione standard di 439.90 *ms*. Per completezza di informazioni abbiamo riportato anche le misurazioni di mediana e 90th percentile.

Tabella 3.2: Statistiche ottenute dalle simulazioni per le latenze dei protocolli MQTT e SeedLink. I tempi sono in ms.

Protocol	**Mean**	**St.Dev.**	**Median**	**90th perc.**
MQTT	33.13	17.03	30.79	33.37
SeedLink	696.13	439.90	592.83	1346.64

Nel grafico in Fig.3.4 sono riportate le distribuzioni di probabilità ottenute dalle simulazioni, dove è stato suddiviso l'asse delle ascisse in bin di 50 *ms*. Gli istogrammi sono normalizzati con le frequenze relative sul totale dei campioni analizzati, cioè 13818 tracce moltiplicato per i 200 campioni contenuti in ogni secondo di traccia.

Dal grafico risulta evidente come le distribuzioni siano molto differenti. La distribuzione riguardante il protocollo MQTT presenta una asimettria positiva, pertanto è indicato calcolare anche il range interquantile che risulta essere di 1.074 *ms*. Tale valore confrontato con il 90th percentile ed il valore di mediana in Tabella 3.2 ci porta ad affermare che le prestazioni del protocollo sono molto stabili durante tutte le simulazioni, con sporadici ouliers. Analizzando la distribuzione per quanto riguarda il protocollo SeedLink si evidenzia la approssimazione ad una funzione multimodale con un primo valore di moda nel range 250-300 *ms*, un secondo nel range 600-650 *ms* ed un terzo valore nel range 1150-1220 *ms*. Cercando quindi un fit con una distribuzione con tali caratteristiche, abbiamo ricavato tre valori di media e deviazione standard: per l'approssimazione alla prima distribuzione un valor medio di 280 *ms* con deviazione standard pari a 78 *ms*, per la seconda valor medio di 597 *ms* e deviazione standard di 89 *ms*, mentre per la terza un valore di 1187±235 *ms*.

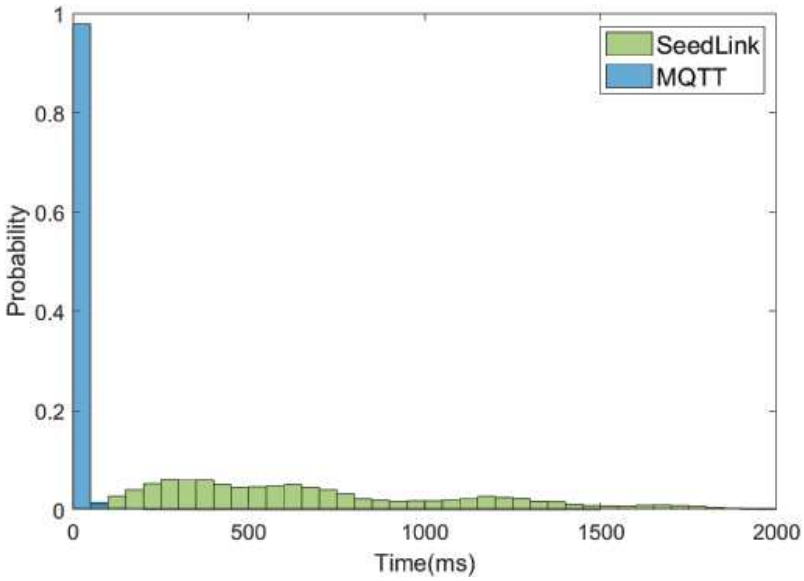

Figura 3.4: Distribuzione di probabilità dei T_l per i protocolli MQTT e SeedLink. Gli istogrammi sono stati normalizzati.

Le simulazioni quindi hanno evidenziato un comportamento non uniforme del protocollo SeedLink, con una grande variabilità di risultati visto il range di valori ottenuti. Al fine di confermare che tali valori non venissero influenzati da altri fattori, abbiamo installato il client SeedLink nella stessa macchina che simula i dispositivi, elimando in questo modo qualsiasi latenza dovuta alla rete. Abbiamo ottenuto distribuzioni identiche rispetto a quelle ottenute dalle simulazioni precedenti, spostate verso sinistra (spostamento dovuto proprio al mancato invio dei pacchetti in rete), confermando quindi la relativamente alta instabilità del protocollo in confronto al protocollo MQTT, nonché tempi di latenza medi notevolmente più alti. Pertanto, inviando medesimi pacchetti miniSEED, il protocollo MQTT risulta essere più performante in termini di latenza e più sicuro in termini di stabilità, candidandosi quindi ad una possibile compresenza con il protocollo SeedLink, e ad un utilizzo in applicazioni di EEW dove tempo ed affidabilità sono le key performance.

3.3.3 Pacchettizzazione, trasmissione e picking della fase P

In questa sezione vengono illustrati i risultati ottenuti per quanto riguarda il tempo di detection della P-Phase ed il tempo totale di rilevazione.
Per tempo di detection della P-Phase si intende la differenza temporale tra l'istante di rilevamento della P-Phase sul processing hub e l'effettivo istante in

cui l'onda P si è generata. Tale differenza temporale tiene conto in maniera non uniforme dei tempi di pacchettizzazione, questo perché l'onda P può essere presente in un posizione casuale all'interno del pacchetto che il datalogger riempie prima del suo invio. Il tempo totale di rilevazione invece è stato definito come la differenza temporale tra l'istante in cui viene rilevata l'onda P dall'algoritmo di triggering sul client e l'istante di inizio del pacchetto contenente l'onda stessa.
Per quanto riguarda il SeedLink, il pacchetto miniSEED è stato riempito con un 1 s di campioni, fissando sempre la frequenza di campionamento a 200 Hz, mentre tramite il protocollo MQTT vengono inviati pacchetti JSON al passo di campionamento. L'algoritmo di triggering utilizzato e discusso in precedenza, viene richiamato lato client ogni volta che un pacchetto viene ricevuto, su una finestra temporale di 10 s; questa operazione risulta tuttavia essere troppo onerosa in termini computazionali per quanto riguarda il caso MQTT, in quanto verrebbe richiamata 200 volte al secondo per ogni traccia. Pertanto si è scelto di richiamare tale funzione ogni 50 campioni ricevuti (250 *ms*). Tale valore è risultato essere il giusto trade-off tra la correttezza dei risultati ottenuti dalla funzione ed il suo tempo di esecuzione. In questo modo il $t_{sp,i}$, nel solo caso MQTT, viene fissato a 250 *ms* prima rispetto al suo reale start time. In

Tabella 3.3: Statistiche ottenute dalle simulazioni per i tempi di rilevamento. I tempi sono in ms.

Protocol	**Mean**	**St.Dev.**	**Median**	**90th perc.**
MQTT	751.93	195.69	685.32	1236.74
SeedLink	2057.52	639.28	1958.57	2684.61

Tabella 3.3 sono riportate le statistiche per quanto riguarda i detection time. Confrontando i valor medi ottenuti si osserva come l'utilizzo della soluzione proposta renda possibile la rilevazione della fase P con un anticipo di circa 1300 *ms* rispetto al tradizionale utilizzo di pacchettizzazione miniSEED e protocollo SeedLink. Tale valore risulta essere simile anche confrontando i valori di mediana e di 90th percentile. Analizzando inoltre le deviazione standard è possibile notare come i valori ottenuti dall'utilizzo del protocollo SeedLink siano in un intervallo molto più ampio rispetto a quelli ottenuti dal protocollo MQTT, confermando come la soluzione proposta contribuisca a contenere anche l'incertezza sui tempi di rilevazione. Come nel caso delle distribuzioni descritte nel precedente paragrafo, per quanto riguarda l'utilizzo delle modalità tradizionali si nota un comportamento ancora non uniforme, confermando come la latenza contribuisca in maniera fondamentale. Maggiore uniformità si evince dalla distribuzione dei valori ottenuti dalle simulazioni effettuate con l'utilizzo del protocollo MQTT. Tale comportamento è proprio quello che ci si attendeva

dopo aver analizzato le latenze del protocollo, in quanto la distribuzione presentava sì una asimettria positiva, ma un range interquantile nell'ordine di 1 ms.

Tabella 3.4: Statistiche ottenute dalle simulazioni per il tempo totale di rilevamento. I tempi sono in ms.

Protocol	**Mean**	**St.Dev.**	**Median**	**90th perc.**
MQTT	910.69	395.06	840.34	1491.73
SeedLink	2558.94	828.48	2388.80	3311.35

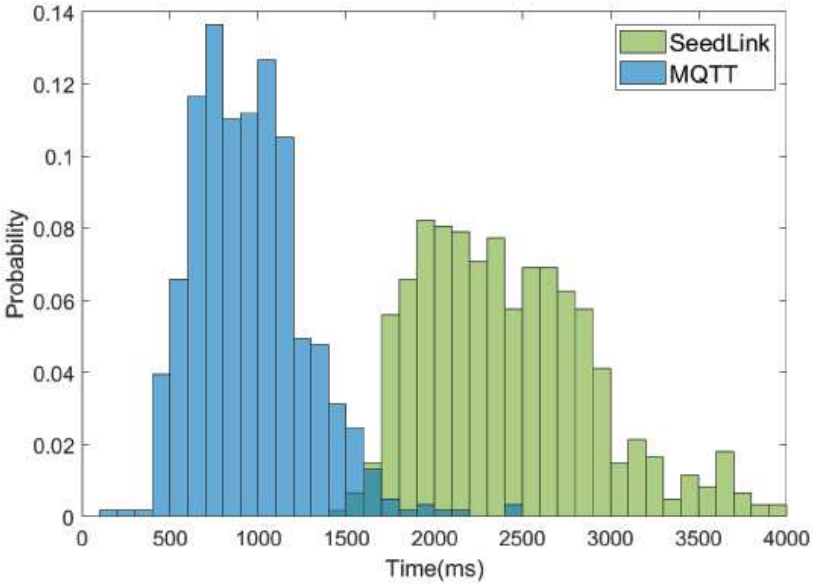

Figura 3.5: Distribuzione di probabilità di T_{tot} per i protocolli MQTT e SeedLink. Gli istogrammi sono stati normalizzati. I grafici sono stati troncati per una maggiore chiarezza.

Nella Tabella 3.4 vengono presentate le statistiche per quanto riguarda il tempo totale di rilevazione. Tale metrica introduce anche il tempo di pacchettizzazione. Anche in questo caso, confrontando i valori medi ed i valori di mediane ottenuti, la soluzione proposta presenta un miglioramento delle prestazioni di circa 1600 *ms*. I valori di deviazione standard suggeriscono nuovamente un comportamento più uniforme. Lo spostamento medio di circa 260 *ms* del risultato ottenuto nella soluzione proposta è in linea con quanto affermato sulle chiamate dell'algoritmo di triggering, così come per quanto riguarda il protocollo SeedLink il risultato ottenuto è in linea con quanto analizzato in precedenza

riguardo triggering e latenza. In Fig. 3.5 sono riportate le distribuzioni di probabilità: la forma simile a quanto ottenuto dalla precedente analisi per quanto riguarda il protocollo MQTT continua a suggerire come la soluzione proposta sia maggiormente stabile ed influenzata in modo pressochè costante dalla pacchettizzazione, mentre la differenza delle forme delle distribuzioni ottenute per il SeedLink inducono a pensare come la pacchettizzazione e la latenza contribuiscano pesantemente nei tempi totali di rilevazione.

3.3.4 Applicazione su un reale evento sismico

Tra agosto 2016 e gennaio 2017 una lunga sequenza sismica ha colpito l'Italia Centrale, causando ingenti danni economici, vittime e costringendo migliaia di persone ad abbandonare le loro abitazioni risultate inagibili [79]. Dopo gli eventi del 24 agosto 2016 di Amatrice di M_w 6.0 e del 26 ottobre 2016 di M_w 5.9 di Visso, il più grande evento è stato registrato il 30 ottobre 2016 che ha raggiunto una M_w pari a 6.5 con epicentro a Norcia. Gli autori di [80] hanno analizzato le prestazioni di PRESTo e di un sistema di EEW standalone denominato on-Site-Alert-leVEl (SAVE) v.1.0 [81] in riferimento allo sciame sismico citato, selezionando nove eventi con M_w superiore a 5.0. Nella loro analisi hanno un utilizzato i sofware in modalità *playback* considerando solo il ritardo dovuto alla pacchettizzazione e non i ritardi di trasmissione. Pertanto, in questo paragrafo verrà selezionato a scopo esemplificativo un solo evento (il M_w 6.5 del 30 ottobre 2016) e verranno effettuate simulazioni tenendo conto anche delle latenze risultate dall'analisi del paragrafo 3.3.2, in modo tale da avere un confronto più veritiero con la soluzione proposta.

A tal fine, siamo partiti dalle stesse configurazioni utilizzate per le precedenti simulazioni, usando quindi le stesse stazioni ed i medesemi parametri utilizzati in precedenza. Sebbene i risultati descritti in precedenza abbiano fornito distribuzioni per il SeedLink non uniformemente distribuite, l'unica possibilità consentita da PRESTo è quella di aggiungere dei ritardi generati casualmente da una distribuzione Gaussiana con media e deviazione standard conosciute. Abbiamo settato tali parametri con i valori ricavati nelle simulazioni precedenti, potendo così ottenere risultati in cui viene tenuto conto dei ritardi di trasmissione. Abbiamo quindi confrontato tali risultati con quanto ottenuto dalla simulazione del medesimo evento con pacchettizzazione al passo di campionamento ed in invio di messaggi con payload in JSON tramite MQTT. La nostra analisi, come detto, si ferma al P-Phase picking, mentre PRESTo fornisce stime di localizzazione, di magnitudo e di tempo di allerta verso uno o più target. Essendo settando come parametro di binding un numero di stazioni pari a 5, è possibile tuttavia confrontare i valori ricavati dall'algoritmo di triggering per queste stazioni, lasciando inalterati i ritardi dovuti alle elaborazioni suc-

cessive per la dichiarazione dell'evento e dell'allerta, per stabilire il vantaggio dovuto all'utilizzo della soluzione proposta.

Tabella 3.5: Risultati della simulazione: istanti delle onde P ed istanti di triggering.

		t_{trig}	
Station	t_{Pw}	**PRESTo**	**MQTT**
MDAR	06:40:24.73	06:40:26.13	06:40:24.87
SSM1	06:40:25.44	06:40:26.64	06:40:25.69
GAG1	06:40:25.75	06:40:27.83	06:40:25.90
TRE1	06:40:27.67	06:40:28.77	06:40:27.80
FOSV	06:40:27.85	06:40:30.37	06:40:28.05

Per le simulazioni abbiamo lasciato le tracce con i riferimenti temporali originali, senza quindi modificare gli start time come fatto in precedenza. In Tabella 3.5 sono riportati i tempi delle fasi P rilevate sulle prime cinque stazioni, e gli istanti temporali in cui l'algoritmo di triggering ha fornito il risultato, sia nella simulazione con PRESTo che con la soluzione MQTT. Questi risultati confermano sia la correttezza dell'implementazione del FilterPicker, in quanto l'elaborazione in entrambi i casi fornisce lo stesso t_{Pw}, sia quanto la soluzione proposta riesca ad ottenere tale risultato in anticipo rispetto a PRESTo. Confrontando infatti i t_{trig} ottenuti si ha che la fase P viene rilevata tra i 950 ed i 2320 *ms* prima rispetto a PRESTo, con un valore medio di 1486 *ms*, coerentemente con i risultati ottenuti in precedenza. Continuando con l'analisi del log di PRESTo, dopo aver fatto il binding e dopo una prima stima di localizzazione e magnitudo, il software dichiara per la prima volta l'evento alle 06:40:33.43, cioè circa 3 s dopo la rilevazione alla quinta stazione. Pertanto è possibile affermare che tale dichiarazione, nel caso di utilizzo della soluzione proposta, può avvenire alle 06:40:31.11, cioè in anticipo di 2320 *ms*. Se si assume una velocità costante di propagazione delle onde S di 3.3 km/s [80], tale anticipo consente di ridurre la cosiddetta blind zone, la zona cioè entro la quale non è possibile lanciare alcun tipo di allarme, di circa 7.7 km. Nel caso in esame il target per l'allarme è posto presso stazione AMEN, situata a 92 km dall'epicentro: nell'esecuzione di PRESTo, il primo allarme avrebbe dato 16.21 s di pre-allerta al target prima dell'arrivo delle onde S, mentre l'utilizzo della pacchettizzazione e del protocollo suggerito avrebbe aumentato questo tempo sino a circa 18.5 s.

3.4 Risultati

In un EEWS maggiore è il tempo a disposizione prima che un evento sismico colpisca un punto target, maggiori e più efficaci potranno essere le azioni che

possono essere intraprese. Studiare quindi i ritardi che ogni componente di un sistema aggiunge riveste notevole importanza. Partendo dal lavoro presentato in [64] si evince come i contributi più importanti in termini di tempo vengono introdotti dalla data latency, intesa come somma di tempi di pacchettizzazione e di invio dei campioni rilevati dal dispositivo. Potendo paragonare un sistema di monitoraggio sismico ad un'applicazione tipica dell'IoT a livello architetturale, ci siamo chiesti se l'adozione del protocollo MQTT, uno dei protocolli più utilizzati in questo ambito, potesse esser utile ai fini di un minore impatto sul tempo di rilevazione di un'onda sismica e, più in generale, sull'invio di un segnale di allerta. A tale scopo, abbiamo utilizzato come sistema di riferimento per condurre le simuliazioni il sistema PRESTo che si basa, così come la maggior parte delle soluzioni attualmente in uso a livello mondiale, sullo standard internazionale di riferimento, cioè il protocollo SeedLink. Tale protocollo introduce un ritardo a monte che di conseguenza influenza tutte le successive fasi per la rilevazione. La nostra analisi confronta i tempi sino al riconoscimento della fase P, che è lo step fondamentale attraverso il quale si può procedere per successive stime di localizzazione e magnitudo. Partendo da un dataset di più di 13000 tracce accelerometriche di reali eventi sismici, abbiamo in un primo momento analizzato le latenze dei due protocolli, inviando lo stesso pacchetto miniSEED sia tramite SeedLink che come payload di un messaggio MQTT, evidenziando come l'utilizzo del protocollo proposto ottenga vantaggi sia in termini di latenza che di variabilità del ritardo. Successivamente abbiamo inviato pacchetti JSON al passo di campionamento tramite MQTT e confrontato i risultati del P-phase picking mediante lo stesso algoritmo utilizzato nel sistema di riferimento. Grazie a questi accorgimenti è possibile anticipare la rilevazione di circa 1.3 s; risultati confermati anche nella simulazione completa effettuata per l'evento di M_w 6.5 del 30-10-2016, ottenendo un guadagno totale verso il punto target di 2.3 s.

Capitolo 4

Sviluppo di un dispositivo IoT per sistemi di EEW

Dopo aver analizzato le prestazioni di uno dei protocolli più diffusi nel mondo IoT ed aver concluso che tale utilizzo comporta dei vantaggi sia in termini di latenza, intesa come tempo di trasmissione di medesimi pacchetti miniSEED, che in termini di rilevamento della fase P, l'attenzione è stata rivolta allo studio ed allo sviluppo di un dispositivo IoT per l'acquisizione e l'invio dei segnali sismici.

Una stazione sismica solitamente è composta da uno o più sensori per il monitoraggio dei movimenti del suolo, collegati ad un data logger i cui compiti sono l'acquisizione dei segnali, la loro eventule archiviazione e la loro trasmissione in real-time a centri di calcolo. Attualmente, l'INGV utilizza una rete mista di sensori costituita da velocimetri e accelerometri per il monitoraggio sismico. Mentre i velocimetri hanno la capacità di registrare movimenti anche più deboli, sia a livello locale (velocimetri a breve periodo) che a livello telesismico (a banda molto larga), gli accelerometri sono integrati nella rete sismica e vengono utilizzati prevalentemente per registrare eventi di magnitudo più elevate. Per quanto riguarda il data logger, l'INGV ha sviluppato un sistema di acquisizione dati denominato GAIA (oggi alla sua seconda versione GAIA2), cioè un sistema modulare costituito da due schede principali, la prima delle quali è una scheda di acquisizione a 24 bit e la seconda incorpora il sistema operativo Linux per consentire la registrazione locale dei dati e la loro trasmissione in tempo reale. Tali stazioni sono estremamente onerose a livello economico, così come la loro installazione e configurazione che richiedono notevoli competenze ee effort in termini di tempo.

In questo panorama il paradigma IoT può venire in aiuto aprendo strade fino ad oggi non percorribili. I recenti sviluppi tecnologici nel campo dei sistemi Micro Elecro Mechanical Sytems (MEMS) hanno permesso il loro utilizzo in svariati settori, dal monitoraggio ambientale ad applicazioni per smartphone, dal settore automobilistico all'healtcare. Un'unità accelerometrica MEMS a basso costo ed alte prestazioni può divenire elemento centrale di uno Smart

Object. La riduzione dei costi di implementazione rispetto alle soluzioni tradizionali, potrebbe portare ad una diffusione maggiore dei dispositivi producendo una rete più densa con possibili guadagni nei tempi di rilevamento.

L'obiettivo del presente capitolo è quindi quello di studiare la possibilità di impiego di un accelerometro MEMS-based collegato ad un sistema di acquisizione che sia in grado di compiere le stesse operazione dei sistemi attualmente in dotazione dall'INGV, con funzionalità quindi di acquisizione, archiviazione e trasmissione. Si studierà, inoltre, una ulteriore possibilità di riduzione dei tempi di rilevamento mediante l'implementazione del P-phase picking di un evento sismico direttamente a bordo del dispositivo stesso.

4.1 MEMS in sismologia

Nell'ultimo decennio, gli accelerometri di tipo MEMS sono stati sempre più studiati anche per applicazioni sismiche ad integrazione delle consueti reti di monitoraggio, visto il loro costo estremamente ridotto, l'esiguo ingombro e, grazie ai miglioramenti ottenuti in termini di range dinamico e sensibilità, la loro capacità di registrare eventi di moderata magnitudo. Come illustrato nel paragrafo 2.4.4, reti interamente costituite da sensori MEMS sono in fase di sviluppo, come la Quake Catcher Network o la Community Seismic Network o ancora il P-alert system in Taiwan. In Italia, l'INGV sta attualmente sperimentando, presso il Laboratorio di Palermo, l'implementazione di stazioni di monitoraggio basate su tecnologia MEMS, il che ha permesso la realizzazione di reti sismiche in aree urbane (come ad esempio Catania, Acireale, Messina, Ragusa, Noto e Siracusa) con finalità di Protezione Civile [82]. In un recente studio [83] viene fornita una completa revisione degli accelerometri capacitivi MEMS, dai principi di funzionamento fisico alle procedure di produzione, specificando come, nonostante i requisiti esatti dipendano dall'applicazione specifica, devono essere considerati i seguenti parametri:

- la densità spettrale del rumore dovrebbe avere una risposta piatta ed essere nel range 10^{-5} - 10^{-7} $m \cdot s^{-2}/\sqrt{Hz}$;
- la sensibilità, definita come il rapporto tra l'ingresso fisico e l'uscita elettrica, dovrebbe essere nell'ordine di $10^2 mV \cdot m^{-1} \cdot s^2$;
- l'intervallo di ampiezza rilevabile dovrebbe essere tra $\pm 2 \cdot 10^0$ e $\pm 2 \cdot 10^1 m \cdot s^{-2}$ e, ad ogni modo, dipende dagli obiettivi applicativi;
- la larghezza di banda dovrebbe sovrapporsi, anche parzialmente, nell'intervallo tra 10^{-2} e 10^2 Hz;
- la risoluzione, definita come la minima accelerazione rilevabile, dovrebbe essere dell'ordine di 10^{-2} - 10^{-3} $m \cdot s^{-2}$.

Il self-noise influisce inevitabilmente su ogni sensore e sulla qualità delle sue misurazioni. Le tecniche per la caratterizzazione del self-noise nei sensori utilizzati in sismologia, ed in generale in tutti i sensori inerziali, sono ben consolidate e standardizzate. In particolare, il self-noise è il risultato della combinazione di diversi tipi di errori, che possono essere modellati analiticamente o matematicamente mediante analisi nel dominio delle frequenze o del tempo [84, 85]. In un recente lavoro [60] vengono comparati quattro differenti sensori accelerometrici MEMS sia per quanto riguarda parametri quali la sensibilità, la risoluzione ed il prezzo sia mediante il livello di self-noise ottenuto mediante analisi proprio delle densità spettrali di potenza sulle tre componenti. Basandoci proprio su questo lavoro, la scelta per quanto concerne l'unità di sensing è ricaduta sull'ADXL355Z.

4.2 Il prototipo sviluppato

Il dispositivo sviluppato è comparabile ad una stazione sismica attualmente in uso sulla RSN, con il vantaggio aggiuntivo di essere completamente open source ed a basso costo. La stazione si avvia in modo autonomo una volta collegata ad una fonte di alimentazione e controlla durante lo startup la presenza di un file di configurazione, in caso contrario presenta una interfaccia per la sua compilazione. Il file comprende i seguenti campi:

- *deviceID*, identificativo del dispositivo;
- *position*, due valori di tipo float che identificano latitudine e longitudine;
- *sps*, il numero di campioni per secondo, ovvero la frequenza di campionamento;
- *fpParams*, parametri per il P-phase picking;
- *winPick*, dimensione in campioni della finestra temporale su cui eseguire ricorsivamente l'algoritmo di rilevamento della fase P;
- *isSeedLink*, un flag che indica se il dispotivo deve avere le funzionalità anche di SeedLink Server.

La stazione, inoltre, è in grado di trasmettere i dati in real-time sia mediante connessione WiFi che con cavo Ethernet ed è sincronizzata via NTP.

4.2.1 Schema di progetto

Il prototipo è stato realizzato secondo lo schema di progetto in Fig. 4.1

Il dispositivo è composto da una unità di sensing collegato ad un Single Board Computer che ha il compito di acquisire i campioni rilevati dal sensore

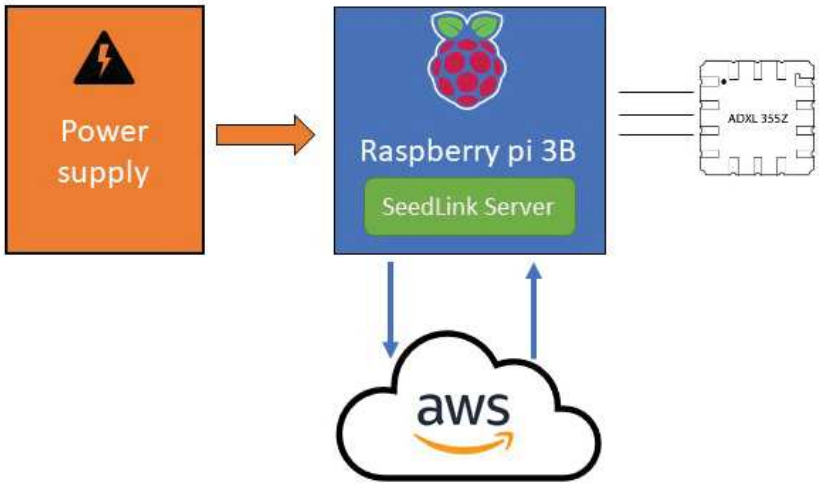

Figura 4.1: Schema di progetto del prototipo realizzato.

ed inviarli al Cloud AWS in tempo reale. Il SBC, se dotato di indirizzo IP raggiungibile dall'esterno, può fungere allo stesso tempo da server SeedLink.

Le caratteristiche principale del sensore ADXL355Z sono:

- 0 g offset vs. temperature: 0.15 mg/°C maximum;
- low power consumption: 660 µW in modalità misurazione e 69.3 µW in modalità standby;
- output da ±2 g a ±8 g full scale range (FSR);
- ultra low noise density: 25 $\mu g/\sqrt{Hz}$;
- 20-bit analog-to-digital converter (ADC);
- sensibilità massima di 3.9mg/LSB @ ±2g;
- fino a 4 kHz Output Data Rate (ODR);
- presenza a bordo di un filtro low-pass/high-pass programmabile;
- sensore di temperatura integrato;
- temperature range: da -40°C a 125°C.

Il dispositivo è in grado di lavorare su un'ampia gamma di range settabili via software (±2.048 g, ±4.096 g, and ±8.192 g) e predispone nativamente di una soluzione per il proprio Self Testing elettromeccanico. La conversione analogico digitale dei valori di accelerazione acquisiti viene effettuata da tre differenti ADC ad alta risoluzione che utilizzano una tensione di 1.8 V come livello di riferimento. Pertanto, il funzionamento dell'ADC del dispositivo non dipenderà dalle fluttuazioni della tensione di alimentazione. Il dispositivo esegue inoltre delle operazioni di filtraggio sia prima della conversione analogico-digitale che dopo l'ADC. Attraverso il registro "Filter Settings" è possibile agire sul posizionamento dei poli dei filtri. Di default i filtri sono settati in modo da garantire DC-off (HPF) che consiste in un filtro con il compito di bloccare la componente continua, un filtro passa basso a 100 Hz ed un terzo filtro in grado di garantire un output data rate di 4 kHz. Compito del filtro passa basso (anti aliasing) è anche quello di ridurre la banda di rumore ed i limiti in larghezza di banda.

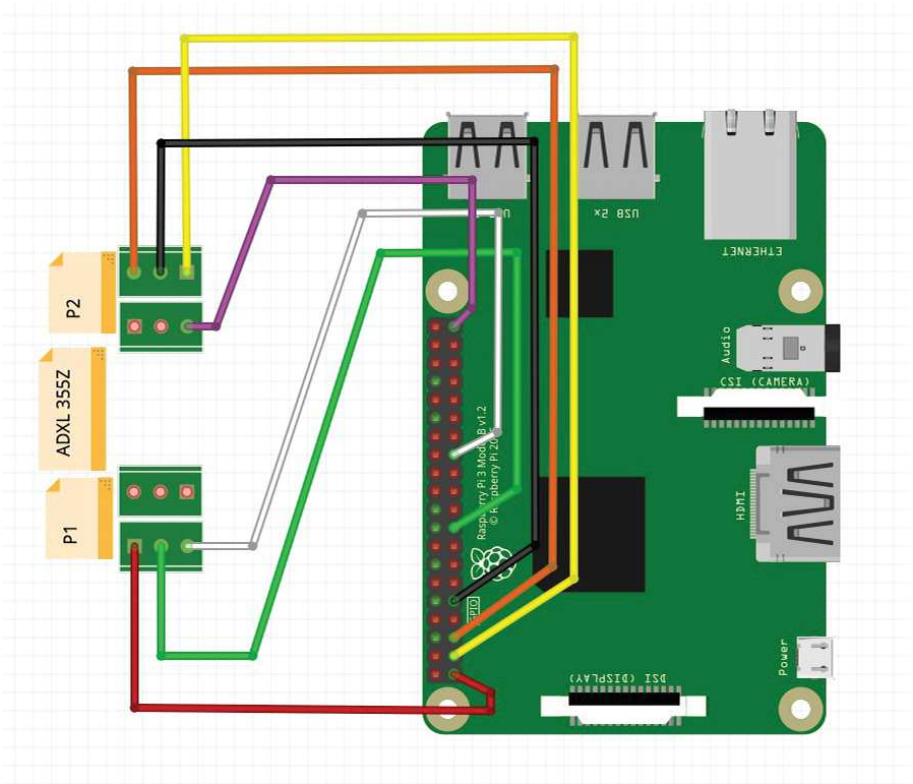

Figura 4.2: Diagramma di collegamento tra Raspeberry pi 3B ed ADXL355Z.

Per quanto riguarda il SBC, la scelta è stata quella di utilizzare un Raspberry Pi 3 Model B, dotato di CPU Quad Core 1.2 GHz Broadcom BCM2837

64 bit CPU con 1 GB RAM. Il SBC è collegato alla rete Internet mediante cavo Ethernet o via connessione wireless. Il diagramma di collegamento tra Raspberry e ADXL355Z viene riportato in Fig. 4.2. La prima operazione che è stata eseguita è l'abilitazione del bus I^2C della RaspberryPi, mediante modifica delle funzionalità del kernel Linux con il comando "sudo raspiconfig". Mediante il comando "sudo i2cdetect -y 0|1" è possibile rilevare su quale indirizzo è collegato l'ADXL355. Una volta fatto ciò è necessario procedere con le configurazioni di rete sulla opportuna interfaccia. Qualora il dispositivo funga anche da SeedLink server è stato anche in questo caso installato e configurato il *ringserver* distribuito da IRIS. Viene installato anche il software `sl-archive`, il quale crea una connessione al server SeedLink, richiede i data streams ed archivia i pacchetti ricevuti in una directory o file.

4.2.2 Acquisizione dati

Il programma principale si occupa della ricezione dei campioni accelerometrici dal sensore, della loro associazione temporale e dell'avvio di 3 thread principali: il primo ha il compito di creare i pacchetti miniSEED, il secondo è per la rilevazione della fase P, mentre al terzo sono demandate le funzionalità di interfacciamento con l'AWS IoT Core.

Per quanto concerne il programma principale, la connessione con il sensore accelerometrico e l'acquisizione dei campioni è resa possibile dall'utilizzo di una libreria Python, il cui diagramma è mostrato in Fig. 4.3. Il programma acquisisce i campioni alla massima frequenza di campionamento possibile, scegliendo, quindi, il campione che a livello temporale si avvicina maggiormente al passo di campionamento, ovvero al valore sps^{-1}. Lo schema a blocchi del

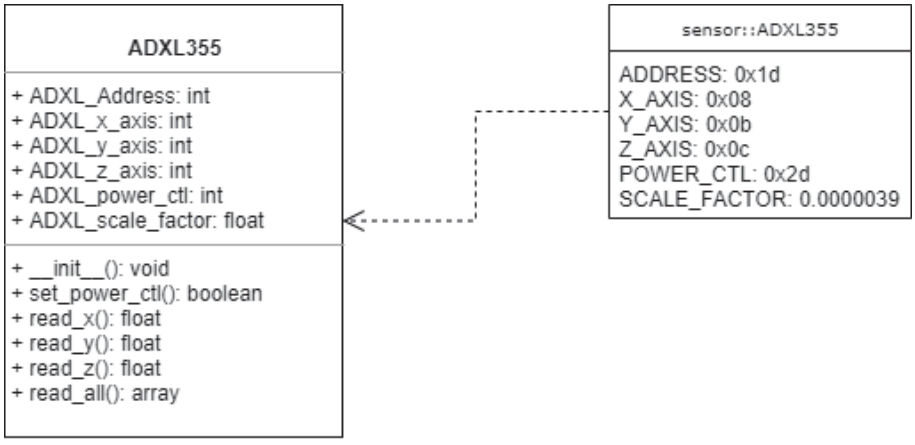

Figura 4.3: Diagramma delle classi per l'acquisizione dei valori di accelerazione dall'ADXL355.

processo viene mostrato in Fig.4.4. Ogni campione viene inserito in due code differenti. La prima coda viene riempita fino al valore *sps* e successivamente passata alla funzione di pacchettizzazione miniSEED. Se nel file di configurazione il flag *isSeedLink* è settato a *true*, il file miniSEED creato viene scritto all'interno di un *ringbuffer* per la trasmissione real-time via SeedLink. La scelta implementativa è stata quella di riempire i pacchetti miniSEED con 1 s di acquisizioni in modo tale da avere tempi di pacchettizzazione non variabili nel tempo. Pertanto viene creato un pacchetto miniSEED per secondo: in questo modo le directory del ringbuffer verrebbero a riempirsi di moltissimi file da 512 byte. Per evitare ciò, viene attivato un secondo processo che si occupa della cancellazione dei file antecedenti il minuto ed, in contemporanea, l'`sl-archive` si occupa della loro archiviazione in un unico stream miniSEED giornaliero.

La seconda coda viene passata come argomento al thread che si occupa del rilevamento della fase P, effettuata mediante l'algoritmo FilterPicker (lo stesso utilizzato da PRESTo). I parametri per il corretto funzionamento dell'algoritmo vengono indicati all'interno del campo *fpParams* del file di configurazione. L'algoritmo in questo modo verrebbe richiamato al passo di campionamento, rendendo onerosa a livello computazionale la sua esecuzione. Si è trovato il corretto trade-off fissando il richiamo della funzione ogni 250 *ms*, ma tale valore può essere modificato agendo sul parametro *winPick* del file di configurazione. Infine, per quanto riguarda il terzo thread, è stato installato direttamente nel dispositivo il servizio di AWS denominato IoT Greengrass. Si tratta di un servizio che consente l'estensione del Cloud AWS nel dispositivo stesso. Attraverso Greengrass è possibile connettersi in modo sicuro ai servizi AWS e utilizzare l'IoT-Core in modo affidabile grazie alla sua integrazione nativa con la piattaforma Cloud. All'avvio del dispositivo, viene inviata una richiesta di registrazione al cloud AWS che includerà il dispositivo in una white list.

4.2.3 Trasmissione real-time

A prescindere dal funzionamento del dispositivo come SeedLink server o meno, il paccetto miniSEED creato viene inviato al Cloud AWS via MQTT, in virtù dei risultati ottenuti ed illustrati nel paragrafo 3.3.2.

Nel momento in cui il FilterPicker rileva una possibile fase P dalla traccia accelerometrica, viene creato un JSON che contiene l'identificativo del dispositivo, l'istante di rilevamento e l'istante della fase P rilevata. Tale oggetto viene incapsulato come payload di un messaggio MQTT che viene inviato al broker in un topic nella forma `/<deviceID>/picks`. A partire da questo istante vengono inviati anche i campioni rilevati per i successivi 30 s all'interno di un altro topic avente la forma `/<deviceID>/picks`. In quest'ultimo caso i campioni vengono inviati al passo di campionamento all'interno di un nuovo oggetto JSON.

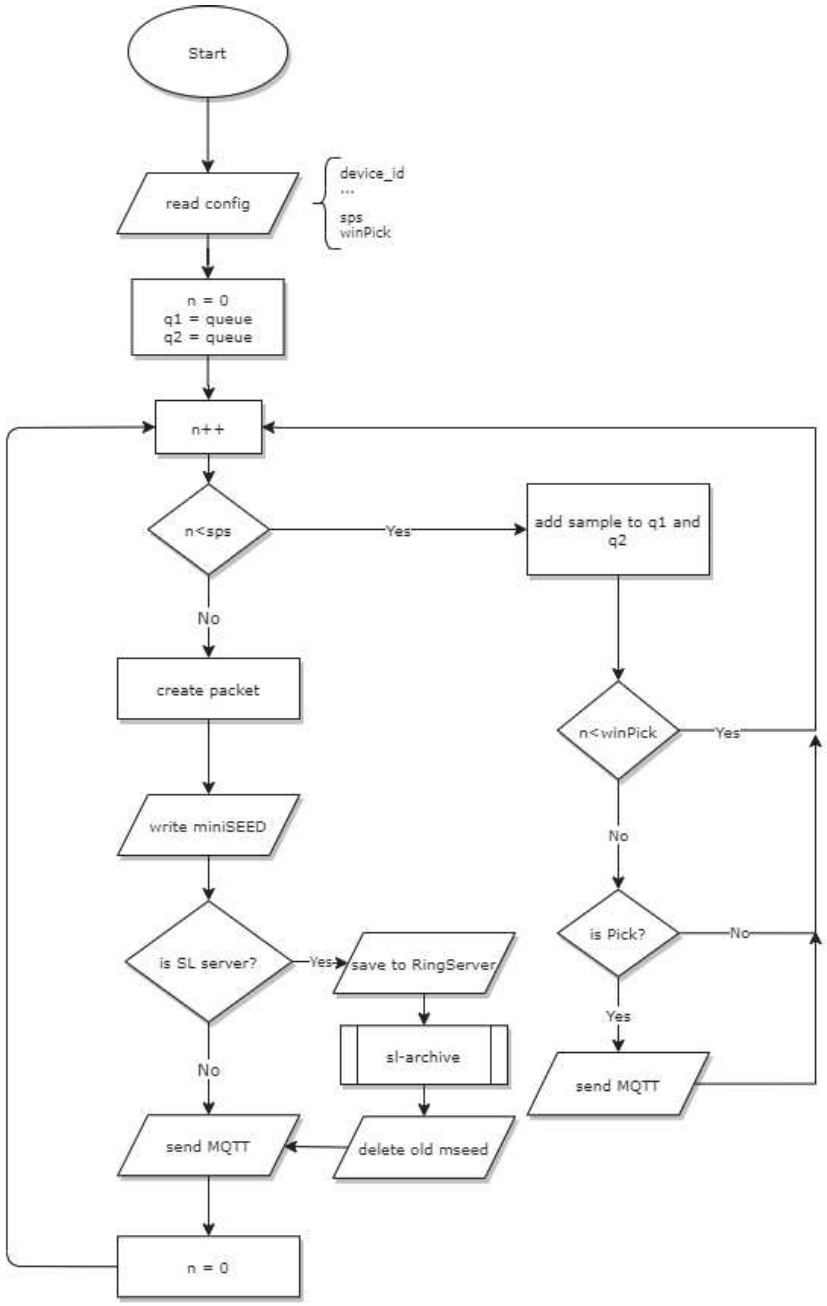

Figura 4.4: Schema a blocchi dei thread di acquisizione e rilevamento della fase P.

Al fine di testare il funzionamento del SeedLink server interno e la trasmissione real-time delle tracce accelerometriche acquisite, è stato utilizzato Sei-

sgram2K, un software sviluppato in Java che consente la visualizzazione delle tracce sismiche sia in modalità real-time che leggendo file dei formati comunemente utilizzati per i dati sismici (SAC, Full SEED, miniSEED, Titan ecc.). In Fig. 4.5 viene mostrato uno screenshot dell'applicazione collegata in real-time al server SeedLink interno al dispositivo sviluppato. Interessante notare come l'applicazione fornisca anche, sulla sinistra, i valori denominati "Ld" e "Lf". Tali valori identificano la "data latency", intesa come differenza temporale tra l'istante di ricezione dell'ultimo pacchetto e l'istante dell'ultimo campione all'interno del pacchetto, e la "feed latency", intesa come differenza temporale tra l'istante attuale e l'istante dell'ultimo pacchetto ricevuto. Tali valori, grazie alla nostra implementazione, sono sempre inferiori al secondo. A titolo comparativo, viene mostrato anche il segnale real-time di una stazione di riferimento collegata al datalogger GAIA, ubicata accanto al nostro dispositivo e connessa nella stesse modalità rispetto al nostro, dove i valori di data latency sono superiori ai 3 s.

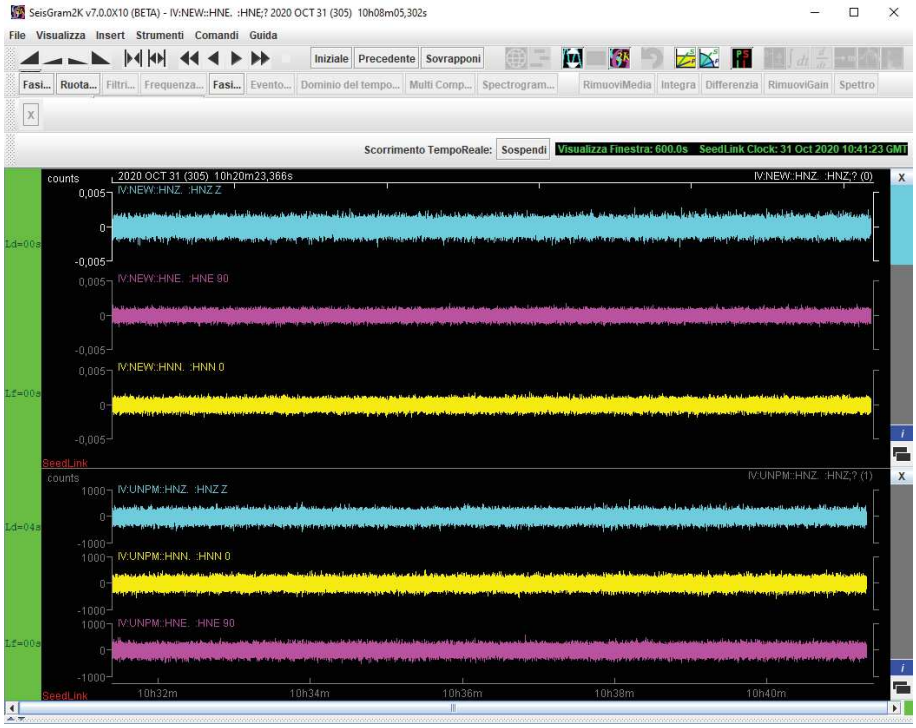

Figura 4.5: Screenshot dell'applicativo Seisgram2K: nella parte superiore della finestra sono mostrate le tre componenti accelerometriche acquisite dal dispositivo sviluppato, mentre nella parte inferiore le acquisizioni di una stazione di riferimento.

4.3 Analisi delle prestazioni

4.3.1 Confronto tra le unità accelerometriche

Il dispostivo IoT creato è attualmente installato alla base della Torre della Facoltà di Ingegneria dell'Università Politecnica delle Marche, in Ancona dal 10-04-2019. Il dispositivo è connesso ad Internet mediante cavo Ethernet collegato direttamente agli switches del Dipartimento di Ingegneria dell'Informazione al fine di garantire una latenza minima. Il dispositivo è ospitato in un armadio isolato e riparato. Accanto al nostro dispositivo è stato installato un altro dispositivo sviluppato in un precedente lavoro del nostro gruppo di ricerca [86]. Quest'ultimo è composto da tre accelerometri MEMS VS1002 (Safran Colibrys) collegati all'acquisitore INGV denominato GAIA2. Tale dispositivo è stato testato durante la sequenza sismica dell'Italia Centrale del 2016-2017 e messo a confronto con la stazione della RSN denominata GUMA, dimostrandosi in grado di rilevare terremoti con magnitudo locale superiore a 2.5 a distanze relativamente brevi ($< 40km$) con un buon rapporto segnale/rumore ($S/N \geq 3$) per le fasi P ed S. Abbiamo considerato, pertanto, il sistema Colibrys+GAIA2 come riferimento per validare il dispositivo sviluppato. Per far ciò abbiamo confrontato le densità spettrali di potenza in coincidenti finestre temporali sulle componeneti orizzontali. Sono state valutate 24 h di acquisizioni durante una giornata senza alcuna attività antropica. Al fine di ottenere valori comparabili tra le due unità accelerometriche, le tracce sono state processate mediante:

- rimozione dell'offset sulle componenti di entrambi i dispositivi;
- utilizzo di un filtro passa-banda Butterworth a 4 poli con la frequenza di taglio inferiore di 0,2 Hz e la frequenza di taglio superiore di 20 Hz (intervallo di frequenza di interesse).

Nel caso della stazione di riferimento i valori sono in count, pertanto si è provveduto anche alla moltiplicazione per il corrispondente valore di sensitività per ottenere l'unità corretta (accelerazione di gravità).

Nel grafico in Fig.4.6 abbiamo riportato le PSD ottenute. La curva ottenuta per quanto riguarda il sistema sviluppato è riportata in linea nera, mentre in viola il sistema di riferimento precedentemente testato. Nel range delle frequenze 0.3-10 Hz la PSD del nostro sistema si posiziona tra i -44 ed i -48 dB $(ms^{-2})^2 Hz^{-1})$, mentre per il sistema Colibrys+GAIA2 la risposta nello stesso range è pressoché costante intorno ai -63 dB. Il risultato è in linea con le carattestiche dei dispositivi stessi, con un livello medio di self-noise del nostro dispositivo superiore di circa 18 dB rispetto al sistema di riferimento.

Successivamente, abbiamo selezionato gli eventi sismici riportati in Tabella 4.1, registrati presso la stazione sismica denominata GUMA. La selezione è

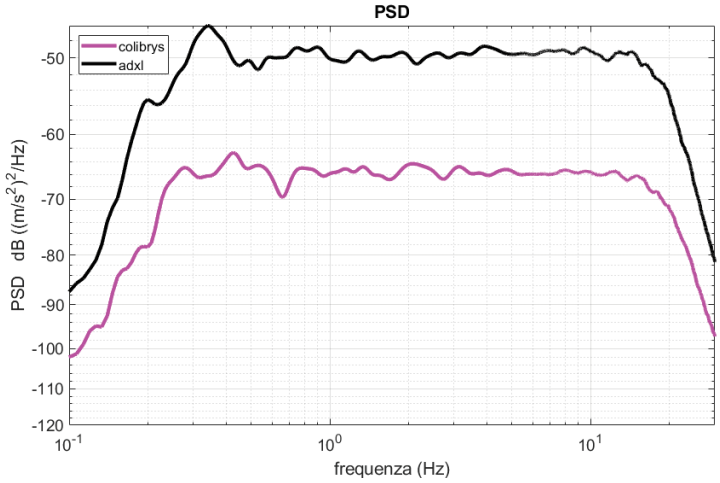

Figura 4.6: Grafico delle densità spettrali di potenza del self-noise generato dai due dispositivi. In colore nero il dispositivo sviluppato, in colore viola il dispositivo di riferimento.

Tabella 4.1: Eventi sismici selezionati per il confronto.

Date time (UTC)	**Lat(N)**	**Long(E)**	**Mag**	**Dist (km)**
2016-09-03 10:18:51	42.866	13.215	4.5	24
2016-10-26 17:10:36	42.879	13.129	5.4	26
2016-10-26 19:18:05	42.915	13.128	5.9	23
2016-10-27 03:19:27	42.844	13.150	4.1	28
2016-11-03 00:35:01	43.029	13.049	4.7	23
2016-12-11 12:54:52	42.900	13.113	4.3	25
2017-04-27 21:16:58	42.957	13.044	4	26
2018-04-10 03:46:15	43.070	13.053	3.5	23

stata effettuata limitando gli eventi con una distanza ipocentrale minore di 30 km rispetto alla stazione di riferimento e con magnitudo superiore a 3.5 e scegliendo, nel caso fossero presenti più eventi di stessa magnitudo, quello avvenuto a distanza minore. Anche in questo caso è stata calcolata la PSD dopo aver effettuato la correzione per ottenere i valori secondo l'accelerazione di gravità.

In Fig. 4.7 sono riportate le PSD degli eventi selezionati e dei due dispositivi nel range delle frequenze di interesse in ambito simico, nel quale cioè si trova il contenuto energetico maggiore. Tali risultati, confrontati con quanto ottenuto nel nostro precedente lavoro [86], confermano la capacità del sistema di riferimento di rilevare terremoti superiori a 3.5 di magnitudo (linee rosse). Infatti, nel range considerato l'ampiezza della PSD del sisma è superiore ri-

spetto a quella del dispositivo. Per quanto riguarda il dispositivo sviluppato, il contenuto energetico dello scuotimento inizia ad emergere dal rumore di fondo del dispositivo per l'evento di magnitudo 4.1 (linea blu a puntini). Risultato ancora più evidente per l'evento di magnitudo 4.3 (linea blu continua), ma risulta difficile la rilevazione di eventi ≤ 4.0. Le linee verdi rappresentano eventi con magnitudo superiore a 4.5 e sono considerevolmente al di sopra del valore di self-noise per il range di frequenze di interesse.

Considerati tali risultati, è possibile affermare che il dispositivo sviluppato è in grado di rilevare eventi di magnitudo superiore a 4.1 localizzati a meno di 30 km di distanza. Un sistema di EEW dovrebbe inoltrare allerte solo nel momento in cui il valore stimato di magnitudo possa comportare danni a persone, abitazioni ed infrastrutture, pertanto il valore ottenuto è in linea con lo scenario applicativo.

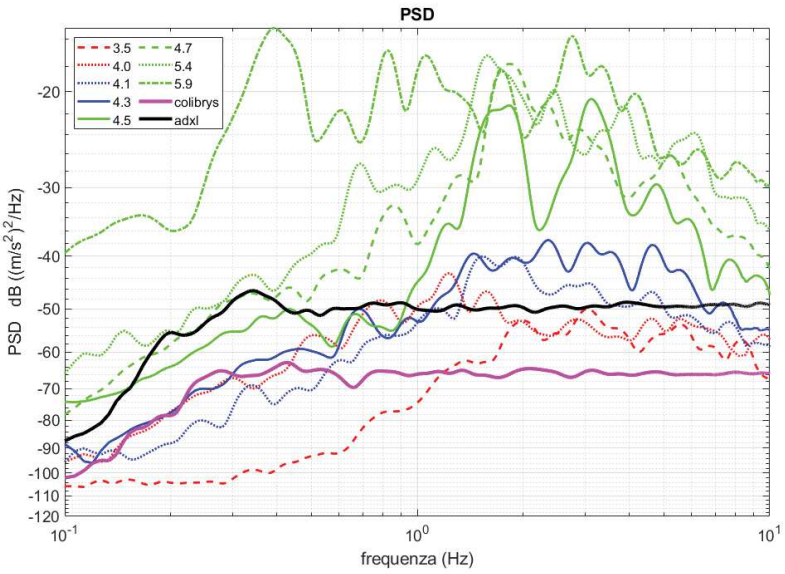

Figura 4.7: Grafico delle densità spettrali di potenza del self-noise dei dispositivi (linee nere e viola) e di una selezione di eventi di diverse magnitudo. Le linee in rosso indicano gli eventi che il dispositivo non riesce a rilevare, le linee blu gli eventi che escono dal self-noise solo per alcuni range di frequenze, in linee verdi gli eventi che è possibile rilevare.

4.3.2 Simulazione su un evento sismico

Per condurre la simulazione abbiamo selezionato dal database ISMD l'evento sismico del 26 Ottobre 2016 avvenuto alle ore 17:10:36 UTC, localizzato alle

coordinate 42.879 di latitudine e 13.129 di longitudine, di magnitudo pari a 5.4. Abbiamo configurato PRESTo generando le griglie di travel-time delle stazioni mediante il software NonLinLoc [87] e generando i file di configurazione necessari, cioè la descrizione della rete ed i parametri relativi al picking automatico. Tali parametri sono stati utilizzati anche nei nostri file di configurazione. Allo start della simulazione vengono modificati i tempi di inizio delle tracce sottosposte con il timestamp attuale, in modo tale da riprodurre l'evento come se fosse in atto al momento della simulazione stessa. A questo punto un thread simula le stazioni, creando un pacchetto miniSEED di 1 s ed inviandolo al server SeedLind a cui PRESTo è collegato. Allo stesso tempo, ogni campione viene inserito in una coda per l'analisi on board real-time per la rilevazione della fase P. Nel momento della rilevazione, viene creato il messaggio ed inviato via MQTT al broker. Dato che i risultati dell'algoritmo di rilevazione sono i medesimi, abbiamo misurato il tempo intercorso tra l'effettivo istante della fase P in una traccia e l'istante in cui viene rilevata da PRESTo e dal nostro dispositivo.

Tabella 4.2: Risultati della simulazione: tempi di triggering per PRESTo e la soluzione proposta. I tempi sono in s.

Station	**Lat(N)**	**Long(E)**	**PRESTo**	**Our device**
FEMA	42.9621	13.0497	2.13	0.102
GUMA	43.0627	13.3335	1.84	0.086
SEF1	43.1468	12.9476	2.04	0.105
MDAR	43.1927	13.1427	2.41	0.197
GAG1	43.238	13.0674	2.33	0.139

La prima fase rilevata è sulla traccia accelerometrica della stazione FEMA. PRESTo riesce a rilevare la fase dopo 2.13 s, mentre il nostro sistema dopo 312 ms. Questa differenza temporale è dovuta principalmente alla differente modalità di pacchettizzazione, infatti il nostro dispositivo può effettuare una analisi direttamente on-board, mentre PRESTo deve attendere il completo riempimento del pacchetto miniSEED ed il suo invio al server SeedLink. Simili risultati sono stati ottenuti su tutte le tracce esaminate, come riportato in Tabella 4.2, dove vengono indicate anche le coordinate delle stazioni.

Il dipositivo sviluppato, pertanto, è perfettamente conforme a quanto normalmente viene utilizzato all'interno di una stazione sismica tradizionale. La possibilità di installazione di un server SeedLink interno al dispositivo lo rende perfettamente integrabile all'interno della rete INGV. Inoltre, la possibilità di rilevamento della fase P a bordo consente notevoli risparmi in termini di tempo in quanto non si rende necessario il completamento del pacchetto miniSEED prima di effettuare l'analisi. È stato inoltre dimostrato come il dispositivo sia in grado di rilevare eventi superiori ad una magnitudo di 4.1 ad una distanza

di circa 25 km, confermando quindi la sua possibilità di utilizzo per sistemi di allerta rapida.

Capitolo 5

La scelta della piattaforma Cloud

5.1 Introduzione

Nel paragrafo 2.3 si è illustrato come negli ultimi anni si sia assistito ad uno studio sempre maggiore delle tecnologie di Cloud Compunting applicate al mondo dell'IoT. Sul mercato globale iniziano a essere sviluppate le prime piattaforme IoT integrate con servizi di Cloud Computing, parallelamente a studi sulle piattaforme native IoT-Cloud. Lo studio condotto in [88] presenta un sondaggio che identifica diversi domini di servizio di cui le piattaforme IoT-Cloud dovrebbero occuparsi. In [89] gli autori propongono un framework per la valutazione delle piattaforme IoT considerando i servizi messi a disposizione per le potenziali esigenze nello sviluppo di applicazioni IoT, concludendo che nessuna delle piattaforme odierne offra un supporto completo. Recentemente, nel lavoro [90] è stata anche introdotta una tassonomia delle proprietà delle applicazioni IoT ed è stata effettuata una indagine su 23 casi d'uso di IoT-Cloud mediante una classificazione dettagliata.

Come riportato in [91], l'incremento dell'utilizzo delle tecnologie basate su Cloud, conduce ad una inevitabile analisi delle prestazioni dei servizi erogati. Possono essere eseguiti diversi tipi di test per la valutazione delle performance: in [92] ad esempio vengono proposti micro-benchmark per la valutazione di diversi sistemi Cloud IaaS, mentre iniziano ad essere sviluppati anche dei tool come CloudHarmony [93] o RIoTBench [94] che consentono di confrontare provider Cloud ed i loro servizi, come compute engine, storage, DNS, CDN, ma non sono presenti studi o tool che confrontino le performance di piattaforme selezionate sulla base dei servizi IoT messi a disposizione.

Ad oggi, esistono centinaia di piattaforme IoT rese disponibili da una vasta gamma di fornitori. Alcune piattaforme sono altamente specializzate, altre si concentrano sulla fornitura di un sottoinsieme di funzioni che potrebbero essere richieste per un sistema IoT. Secondo il "IoT Developer Survey 2018" [95] redatto dall'Eclipse IoT Working Group, in collaborazione con il progetto Agile-IoT H2020, IEEE e Open Mobile Alliance (ora OMA SpecWorks), le principali piattaforme end-to-end Cloud-IoT che attualmente detengono la leadership sul

mercato globale sono Amazon Web Service (AWS), Microsoft Azure e la Google Cloud Platform. Pertanto, nei successivi paragrafi verrà condotta una analisi che si concentrerà sul confronto tra queste piattaforme Cloud per quanto riguarda i servizi che mettono a disposizione per l'IoT e per quanto riguarda le loro prestazioni. Per poter condurre una completa analisi prestazionale comparativa delle tre piattaforme selezionate, è necessario costruire uno scenario tipico di una applicazione IoT. Al tempo stesso, la vastità dei campi applicativi (ad esempio smart-home, smart-traffic, industrial applications, e-health) e conseguente diversità dei servizi coinvolti, renderebbe lo studio limitato al settore considerato. Ciò nonostante, le piattaforme selezionate si basano su una comune architettura: i dispositivi IoT, dopo aver effettuato una connessione sicura con il Cloud, possono iniziare ad inviare i dati dei propri sensori, così come ricevere comandi per aggiornare il proprio stato ad esempio, così come applicazioni connesse al Cloud possono allo stesso tempo ricevere i dati pubblicati. Le piattaforme si basano tutte su un meccanismo di publish/subscribe, pertanto la scelta è stata quella di condurre una analisi di performance proprio su tale servizio.

5.2 Piattaforme Cloud per l'IoT

Secondo la architettura illustrata nel paragrafo 2.3.2, le piattaforme Cloud-IoT forniscono un middleware per la connessione e la gestione dei dispositivi hardware e per i dati da essi raccolti. Le soluzioni IoT nel Cloud richiedono una comunicazione bidirezionale sicura tra i dispositivi ed il back-end della soluzione. Una applicazione IoT coinvolge molti dispositivi IoT eterogenei, con sensori che producono ed inviano dati o eventi attraverso la rete, che possono essere utilizzati per generare insights (ad esempio elaborazione e analisi dei dati). Nel resto di questo paragrafo vengono esaminati il livello *platform* ed il livello *enterprise* della architettura Cloud-IoT e vengono mappati componenti e funzionalità delle piattaforme cloud selezionate in ogni livello.

I concetti chiave considerati per l'analisi sono i seguenti:

- *Device management.* La gestione dei dispositivi hardware IoT è una delle funzioni principali di una piattaforma IoT. Include il provisioning di nuovi dispositivi, il monitoraggio e la manutenzione dei dispositivi operativi. Le piattaforme IoT devono fornire funzionalità per la gestione della registrazione dei dispositivi, della configurazione, degli aggiornamenti over-the-air e della gestione delle risorse, inclusa la possibilità di elencare e cercare i dispositivi collegati e di interrogare e gestire i metadati dei dispositivi stessi.

- *Data communication protocols.* Per consentire la gestione remota dei dispositivi e la trasmissione dei dati, è essenziale una comunicazione sicura ed affidabile tra dispositivi IoT, gateway ed app e servizi basati su Cloud. In letteratura [96] questi tipi di comunicazione per l'ambiente IoT sono generalmente indicati come "Device-to-Device (D2D)", "Device-to-Application (D2A)", "Device-to-Gateway (D2G)" e "Device-to-Cloud (D2C)". Alcuni dati possono essere archiviati ed elaborati localmente (ad es. field gateway nelle Wireless Sensors Network), così come alcuni dei dati raccolti dai sensori ed altri dispositivi IoT devono essere consegnati ai servizi Cloud per ulteriori elaborazioni. Le piattaforme IoT incorporano servizi di broker di messaggi per consentire a dispositivi e gateway di inviare e ricevere messaggi con bassa latenza e su larga scala. I servizi di broker di messaggi utilizzano protocolli di comunicazione standard come il già discusso MQTT, il CoAP [97] o l'XMPP [98] e supportano i web-socket.

- *Regole.* Una volta che i dati vengono importati nel back-end IoT, il flusso di elaborazione dei dati può variare, a seconda degli scenari e delle applicazioni. Indipendentemente da ciò, le piattaforme IoT possono offrire un motore di regole con condizioni utilizzate per attivare delle azioni.

- *Data storage.* I dati generati dai dispositivi IoT devono essere archiviati per essere accessibili tramite ulteriori elaborazioni. I dati vengono solitamente suddivisi in archivi dati *hot* e *cold* [99]. È necessario accedere agli archivi dati *hot* con alta frequenza e bassa latenza, mentre agli archivi dati *cold* si accede raramente, in genere con un'elevata latenza e con costi di archiviazione inferiori, pur mantenendo questi ultimi.

- *Integrazione.* Le piattaforme IoT forniscono integrazione con altre piattaforme, dispositivi, servizi Web, strumenti e applicazioni tramite SDK e API.

- *Sicurezza.* Deve essere inclusa la protezione dei dispositivi e della comunicazione di rete oltre all'implementazione della sicurezza a livello di applicazione e Cloud.

- *Costi.* In un ambiente Cloud tutti i servizi sono offerti in un modello pay-as-you-go, in modo tale che gli utenti paghino solo il reale utilizzo di un servizio o risorsa. In questa analisi viene considerato solo il costo della soluzione IoT, senza considerare altri servizi connessi, come lo storage o l'analisi dei dati.

L'importanza relativa di ciascuna di queste funzionalità dipende dal caso d'uso. Ad esempio, si consideri un sistema di automazione domestica su piccola

scala. La gestione dei dispositivi non è una preoccupazione primaria dato il numero limitato di dispositivi da gestire. Nel caso d'uso di automazione domestica è meno importante che la piattaforma IoT fornisca capacità di archiviazione e analisi dei dati. In questo caso, la piattaforma IoT deve supportare protocolli di comunicazione dati veloci e affidabili per garantire che gli eventi rilevati dai sensori non vengano persi e che si possa rispondere quasi in tempo reale. Altro caso può essere ad esempio un'applicazione IoT industriale, che coinvolge migliaia di sensori che monitorano le apparecchiature di una fabbrica al fine di rilevare guasti ai macchinari e prevedere quando eseguire la manutenzione. Le capacità di analisi e gestione dei dispositivi della piattaforma IoT sarebbero ovviamente molto più importanti in questo specifico caso. Trovare la giusta combinazione tra le funzionalità di cui sopra ed il caso d'uso è la base per progettare una soluzione IoT basata su Cloud.

5.2.1 Amazon Web Services

L'architettura di AWS IoT-Core è rappresentata in Fig. 5.1. I dispositivi segnalano il proprio stato pubblicando messaggi su argomenti MQTT, che hanno un nome gerarchico per identificare il dispositivo stesso. Il messaggio viene inviato al Message Broker, che lo inoltra a tutti i client iscritti a tale argomento. Ogni dispositivo ha un oggetto shadow (una rappresentazione virtuale del dispositivo) che viene utilizzato per archiviare e recuperare le informazioni sullo stato. Lo shadow altro non è che un documento JSON diviso in una parte relativa all'ultimo stato segnalato dal dispositivo ed in una relativa al suo stato desiderato. Un'applicazione può inviare una richiesta con lo stato corrente del dispositivo o con una modifica del suo stato. Quando il messaggio arriva al broker, il motore delle regole fornisce l'elaborazione del messaggio e l'integrazione con altri servizi AWS.

- *Device management.* AWS IoT Device Management è il servizio della piattaforma AWS IoT che consente l'organizzazione, il monitoraggio e la gestione dei dispositivi IoT. Consente ai dispositivi di registrarsi in blocco e di permette la loro organizzazione in gruppi, con la possibilità anche di definizione dei criteri di accesso. AWS IoT fornisce un registro per gestire i dispositivi, che vengono archiviati come dati JSON. L'interazione con il registro è possibile con la console AWS IoT o mendiante la AWS Command Line Interface. Inoltre, la piattaforma fornisce Device SDK per Android, iOS, Java, JavaScript, C ++, Python ed Embedded C che includono librerie open source.

- *Data communication protocols.* La comunicazione da e verso AWS IoT Core è consentita attraverso un servizio di broker di messaggi di pubblicazione/sottoscrizione. Il broker di messaggi supporta MQTT per la

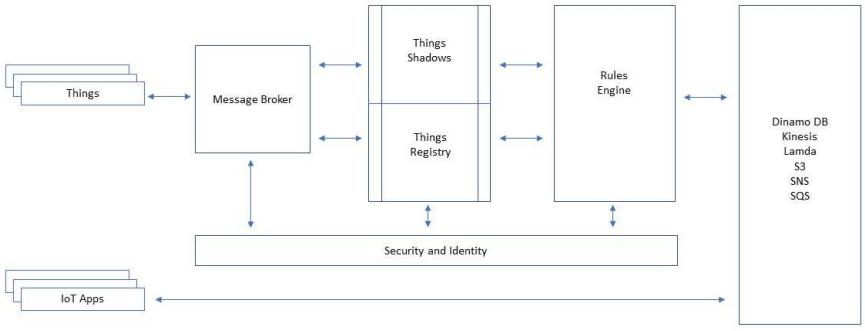

Figura 5.1: Schema a blocchi dell'architettura del servizio AWS IoT Core e sua integrazione con altri servizi AWS.

pubblicazione e la sottoscrizione ed HTTPS solo per la pubblicazione, sia tramite IPv4 che IPv6. L'implementazione del broker di messaggi si basa su MQTT v.3.1.1, ma non supporta QoS 2 e non consente la connessione di due o più client con lo stesso identificativo contemporaneamente. Tutti gli argomenti che iniziano con $ sono argomenti riservati, utilizzati per operazioni shadow del dispositivo (ad esempio, ottenere o aggiornare lo stato). Il broker supporta le connessioni con il protocollo HTTP tramite l'utilizzo dell'API REST.

- *Regole.* La piattaforma utilizza le regole per interagire con altri servizi AWS. Le regole sono composte da un trigger scritto con una sintassi simile a SQL a cui vengono collegate ed attivate una o più azioni.

- *Data storage.* I dati importati in AWS IoT Core devono essere archiviati. AWS IoT Core offre la connessione diretta con Amazon DynamoDB (database NoSQL) e AWS S3 (Simple Storage Service), uno storage scalabile nel Cloud AWS.

- *Integrazione.* Per la creazione dei dispositivi e per l'interazione con essi, AWS IoT fornisce un'interfaccia a riga di comando (CLI), oppure le API AWS IoT per creare applicazioni utilizzando richieste HTTP o HTTPS o ancora gli SDK del dispositivo. AWS offre servizi per la raccolta e l'elaborazione dei dati: Amazon Kinesis Data Stream per l'elaborazione in tempo reale dei dati in streaming, AWS Lambda per eseguire codice serverless, Amazon Simple Notification Service per inviare o ricevere notifiche e Amazon Simple Queue Service per archiviare i dati in una coda.

- *Sicurezza.* I dispositivi devono disporre di credenziali per accedere al broker di messaggi e tutto il traffico deve essere crittografato da Transport Layer Security (TLS). La piattaforma supporta i certificati X.509 di autenticazione (generalmente utilizzati dai dispositivi AWS IoT), unitamente a utenti, gruppi e ruoli di Identity Access Management (IAM), identità federate utilizzate da applicazioni Web e desktop e identità di Amazon Cognito (generalmente utilizzate da applicazioni mobili), che consentono l'utilizzo di altri provider di identity.

- *Costi.* AWS fattura separatamente i servizi di connettività, messaggistica, utilizzo dello shadow del dispositivo (archiviazione dello stato del dispositivo), utilizzo del registro (archiviazione dei metadati del dispositivo) e utilizzo del motore delle regole (trasformazione e instradamento dei messaggi), tutti in base alla regione selezionata.

5.2.2 Microsoft Azure

Microsoft Azure per l'IoT offre due percorsi: una soluzione PaaS denominata acceleratore di soluzioni Azure IoT e una soluzione SaaS denominata Azure IoT Central. Entrambe le soluzioni utilizzano l'Hub IoT di Azure come Cloud gateway per accettare in modo sicuro i dati e per fornire funzionalità di gestione dei dispositivi. L'Hub è integrato in modo nativo con altri servizi Cloud di Azure e consente una comunicazione bidirezionale sicura tra dispositivi ed applicazioni. L'architettura Azure per IoT è rappresentata nella Fig. 5.2, secondo l'architettura Cloud-IoT a 3 livelli descritta in precedenza. Il messaggio inviato da un dispositivo autenticato arriva all'Hub, che dispone di funzionalità integrate di instradamento dei messaggi per il suo invio ad uno o più endpoint di altri servizi. I dispositivi hanno una rappresentazione virtuale nel Cloud chiamata dispositivo gemello, archiviata come documento JSON che contiene le proprietà desiderate (impostate da un'applicazione e lette dal dispositivo) e le proprietà segnalate (impostate dal dispositivo e lette da un'applicazione).

- *Device management.* Il servizio di provisioning dei dispositivi dell'hub IoT di Microsoft Azure è un servizio che consente il provisioning just-in-time dei dispositivi a un hub IoT, senza richiedere l'intervento umano. I dispositivi contattano l'endpoint del servizio di provisioning passando le informazioni di identificazione. Il servizio associa e registra il dispositivo con un hub IoT e popola lo stato del dispositivo gemello desiderato. Inoltre, Azure fornisce device-SDK che possono essere usati su dispositivi o gateway per semplificare la connettività all'hub IoT. Gli SDK sono disponibili per .NET, C, Java, Node.js, Python e iOS.

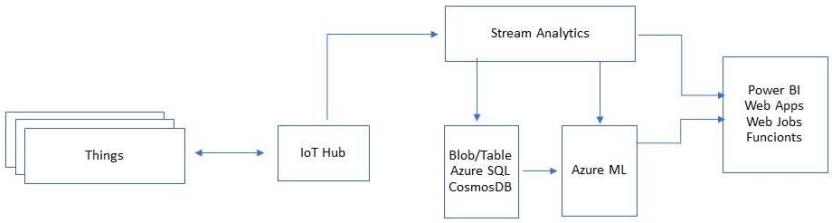

Figura 5.2: Schema a blocchi dell'architettura del servizio Azure Hub IoT e sua integrazione con altri servizi Azure.

- *Data communication protocols.* Oltre all'Hub IoT, la piattaforma offre il servizio Hub Eventi al fine di garantire la comunicazione con il Cloud. Entrambi i servizi sono progettati per date ingestion su vasta scala, ma l'hub IoT include funzionalità più specifiche per il contesto IoT, come la comunicazione bidirezionale da e verso il Cloud e la gestione dell'identity a livello di dispositivo. L'hub IoT di Azure fornisce supporto per AMQP 1.0 con supporto WebSocket facoltativo, MQTT 3.1.1 e protocolli HTTP 1.1 nativi su TLS. La QoS 2 del protocollo MQTT è supportata, ma non consigliata a causa dell'elevato impatto sulla latenza e disponibilità dell'intero sistema. La piattaforma offre anche l'IoT Protocol Gateway per abilitare l'utilizzo di altri tipi di protocollo per l'hub IoT.

- *Regole.* L'hub IoT di Azure espone le sue funzionalità tramite l'uso di endpoint. Per instradare i messaggi dal dispositivo a questi endpoint, Azure usa regole scritte in una sintassi simile a SQL che valutano le intestazioni ed il corpo dei messaggi.

- *Data storage.* Per quanto riguarda l'archiviazione dei dati, secondo la sezione precedente, Azure offre servizi per l'archiviazione *hot* e *cold.* Nel percorso hot, i dati devono essere disponibili con una latenza inferiore rispetto ai dati in un cold storage. I servizi di Azure per l'archiviazione "a caldo" sono Azure Cosmos DB come database NoSQL e Azure SQL DB come DBMS SQL relazionale. I servizi per l'archiviazione "a freddo" sono l'archiviazione BLOB di Azure (un database di archiviazione file) e Azure Data Lake, un archivio dati distribuito. Azure offre anche Time Series Insight che fornisce servizi di analisi, archiviazione e aggregazione.

- *Integrazione.* L'hub IoT è connesso in modo nativo con altri servizi di Azure come il Servizio app di Azure, una piattaforma gestita per creare app Web e mobili, l'Hub di notifica per inviare notifiche push ed il servizio PowerBI per creare dashboard.

- *Sicurezza.* Azure considera tre aree principali per quanto riguarda la sicurezza: a livello di dispositivo, di connessione e sicurezza del Cloud. Azure Hub Identity Registry fornisce l'archiviazione sicura delle identità dei dispositivi e di ogni chiave di sicurezza; tutte le connessioni devono essere avviate dal dispositivo all'Hub, non viceversa, e devono utilizzare l'autenticazione TLS con certificato X.509; Azure Active Directory infine viene usato per l'autenticazione dell'utente e l'autorizzazione per l'accesso al Cloud.

- *Costi.* La gestione dei costi di Azure si basa su due livelli di servizio. In ogni livello troviamo tre sottolivelli. Ogni livello ha un limite giornaliero di messaggi dopo il quale si verificherà una limitazione. Il consumo di unità dell'Hub IoT viene misurato su base giornaliera e la fatturazione viene generata su base mensile. I clienti vengono fatturati in base al numero di unità dell'Hub IoT che sono state consumate durante il mese.

5.2.3 Google Cloud Platform

La soluzione di Google per l'IoT è denominata Google IoT Core. I componenti principali sono il gestore dispositivi ed il bridge di protocollo. Il device manager ha il ruolo di registrare i dispositivi con il servizio, mentre i due bridge protocols (HTTP/MQTT) sono utilizzati dai dispositivi per la connessione e l'invio dati verso il Cloud. Secondo l'architettura della Google Cloud Platform rappresentata in Fig. 5.3, i dispositivi inviano dati al Google IoT Core, il quale è direttamente connesso a Google Cloud Pub/Sub; si tratta di un middleware orientato ai messaggi per il Cloud. I messaggi vengono quindi consegnati a un servizio di pipeline, Google Cloud Data Flow, che elabora i dati e li invia ad altri servizi Cloud, a seconda del caso d'uso del progetto IoT. I dispositivi sono rappresentati da un ID e identificati da un nome completo di risorsa. È possibile definire metadati personalizzati per un dispositivo, uno stato che viene inviato al Cloud ed una configurazione, che viene inviata dal Cloud al dispositivo. A differenza delle piattaforme precedenti, nella soluzione Google le informazioni possono essere un blob di dati arbitrario definito dall'utente.

- *Device management.* IoT Core Device Manager fornisce il servizio per la gestione dei dispositivi. Comprende i processi di registrazione, autenticazione e autorizzazione. Con il gestore dispositivi è possibile creare e configurare registri e dispositivi al loro interno. Il registro del dispositivo

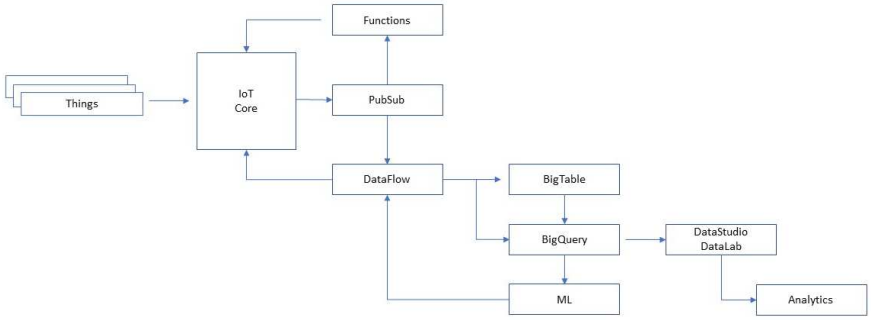

Figura 5.3: Schema a blocchi dell'architettura del servizio IoT Core della Google Cloud Platform e sua integrazione con altri servizi.

è configurato con uno o più argomenti Cloud Pub/Sub in cui vengono pubblicati gli eventi di telemetria per tutti i dispositivi di quel registro. Un dispositivo è definito con metadati, invia messaggi di telemetria e riceve configurazioni.

- *Data communication protocols.* La piattaforma supporta MQTT e HTTP per la gestione dei dispositivi e delle comunicazioni. Con l'uso di MQTT, i dispositivi inviano richieste di pubblicazione ad un argomento specifico, mentre utilizzando HTTP, i dispositivi non mantengono una connessione alla piattaforma, ma inviano richieste e ricevono risposte. Per quanto riguarda la qualità del servizio del protocollo MQTT, il bridge MQTT supporta QoS 0 e 1.

- *Regole.* Per gestire i dati che arrivano al Cloud, la piattaforma utilizza il concetto di pipeline offerto da Google Cloud Data Flow: permette di trasformare, aggregare, arricchire e spostare i dati su altri servizi. È anche possibile operare su ogni evento pubblicato individualmente dal servizio Google Cloud Functions, che può essere utilizzato per filtrare dati non validi, attivare allarmi o richiamare altre API.

- *Data storage.* I dispositivi inviano diversi tipi di dati: dal loro stato (normalmente in modo strutturato), ai dati di telemetria ed a dati non strutturati (ad es. Flussi video). Nel caso di dati strutturati che identificano lo stato di un dispositivo, l'archiviazione avviene direttamente nel servizio fornito dall'IoT Core. I dati di telemetria arrivano con alta frequenza e devono essere disponibili con bassa latenza e modalità ad al-

te prestazioni: la piattaforma offre Cloud Datastore e Cloud BigQuery come database NoSQL e Cloud BigTable come data warehouse completamente gestito con interfaccia SQL. Il Cloud Storage viene utilizzato per archiviare i dati utilizzati di rado e per i dati non strutturati.

- *Integration.* La piattaforma fornisce il Google Cloud SDK, che contiene uno strumento a riga di comando chiamato *gcloud*. Le operazioni sono possibili anche dalla console o mediante librerie client API per C#, Java, NodeJS, GO, PHP, Python e Ruby. IoT Core è nativamente integrato con i servizi di analisi di big data e machine learning di Google come Cloud ML, Data Studio e DataLab.

- *Security.* L'IoT Core offre autenticazione con chiave pubblica/privata per i dispositivi utilizzando i JSON Web Tokens (JWT) e supporta gli algoritmi RSA o Elliptic Curve per la verifica delle firme. Per quanto riguarda la sicurezza delle comunicazioni, il protocollo TLS 1.2, con l'utilizzo di root certificate authorities, è richiesto per le connessioni MQTT. Infine il Google Cloud Identity and Access Management (IAM) consente di controllare, autenticare e autorizzare l'accesso all'API Cloud IoT Core.

- *Costs.* I prezzi di IoT Core sono calcolati sul volume di dati utilizzato in un mese, se supera i 250 MB, in tre diversi livelli e considerando una dimensione minima di 1024 byte per ogni messaggio (per messaggi inferiori a 1024 byte viene applicato il costo relativo a 1024 byte). Tutti gli altri servizi utilizzati in una soluzione vengono fatturati separatamente.

In Tabella 5.1 vengono riportate le carattistiche delle diverse piattaforme per una migliore lettura comparativa.

Tabella 5.1: Riepilogo delle caratteristiche delle piattaforme Cloud selezionate.

	AWS	**Azure**	**GCP**
Device management	IoT Device Management Console Command Line Interface	Device Provisioning Service Console Command Line Interface	IoT Core Device Manager Console Command Line Interface
Data communication protocols	MQTT 3.1.1 HTTP(S)	MQTT 3.1.1 HTTP(S) AMQP 1.0 (WebSocket support)	MQTT 3.1.1 HTTP(S)
Rules	Rules Engine (SQL-like rules)	Endpoints exposed for SQL-like rules	Google Cloud Data Flow Google Cloud Functions
Cold Storage	AWS S3	Azure Blob Storage Azure Data Lake	Cloud Storage
Hot Storage	Amazon DynamoDB	Azure CosmosDB Azure SQL DB	Cloud Datastore Cloud BigQuery Cloud BigTable
Integration	APIs, SDK Amazon Kinesis AWS Lambda AWS SNS AWS SQS	APIs, SDK Azure App Service Notifications Hub PowerBI	APIs, SDK Cloud ML Cloud Data Studio Cloud DataLab
Languages	Java JS C++ Python Embedded C	.NET C Java NodeJS Python	C# Java NodeJS GO PHP Python Ruby
Security	TLS X.509 certificate AWS IAM Federated Identities Amazon Cognito	Azure Hub Identity Registry TLS X.509 certificates Azure Active Directory	JWT TLS X.509 certificates Google Cloud IAM

5.3 Analisi delle prestazioni

L'obiettivo del nostro studio è quello di misurare le performance dei broker delle piattaforme: il broker message di AWS, l'hub IoT di Azure e l'MQTT Bridge della Google Cloud Platform. La metrica che in letteratura viene maggiormente presa in considerazione è il tempo end-to-end, inteso cioè come il tempo che impiega un messaggio dal client publisher ad arrivare al client sottoscrittore. Tale tempo di latenza però è influenzato da fattori, quali il tempo di trasmissione, di propagazione, di processing e di queuing, condizioni che non sempre sono stabili durante l'esecuzione di test in diversi momenti. Ciò che pertanto andremo a considerare sarà il solo tempo di elaborazione del broker in diverse condizioni, che vengono illustrate nella costruzione degli scenari nel paragrafo 5.3.2.

5.3.1 Metriche di comparazione

Consideriamo un semplice caso con un solo client publisher P1 ed un solo client subscriber S1, entrambi client MQTT dello stesso broker MQTT in una piattaforma Cloud. Il flusso della simulazione viene mostrato in Fig.5.4: il client P1 pubblica un messaggio con QoS pari ad 1 nell'istante t_0 con timestamp quindi pari a t_0. All'istante t_1 il messaggio arriva al broker che risponde a P1 con un acknowledge (PUBACK). Questo PUBACK arriva all'istante t_4 al client publisher. Allo stesso tempo il broker inoltra il messaggio nell'istante t_2 al client subscriber S1 che lo riceve all'istante t_3. La metrica che consideriamo, il Broker Service Time, è dato da (5.1)

$$T_c = t_2 - t_1 \tag{5.1}$$

Tutte e tre le piattaforme forniscono agli utenti un sistema di log che può essere analizzato per ottenere questo valore. Il nostro scopo è ottenere T_c senza l'utilizzo dei log delle piattaforme. Sarebbe possibile implementare una regola nel Cloud che aggiunga nuovi campi al messaggio in arrivo con i timestamp t_1 e t_2, ma questa scelta non è praticabile perché questa operazione aggiungerebbe ritardi significativi e non facilmente quantificabili, incidendo negativamente sul risultato finale. Inoltre, non è possibile confrontare i tempi dei client con i tempi della piattaforma a meno che non si utilizzino algoritmi di sincronizzazione. Allo stesso tempo, per garantire efficienza e scalabilità, le piattaforme prevedono un endpoint di connessione, ma l'ubicazione effettiva dell'endpoint è ovviamente diversa nel tempo, rendendo la sincronizzazione molto costosa in termini di calcolo. Pertanto, abbiamo implementato i client simulati nella stessa macchina fisica, in modo da evitare qualsiasi problema di sincronizzazione tra host. Partendo dagli algoritmi del protocollo NTP, è possibile calcolare il

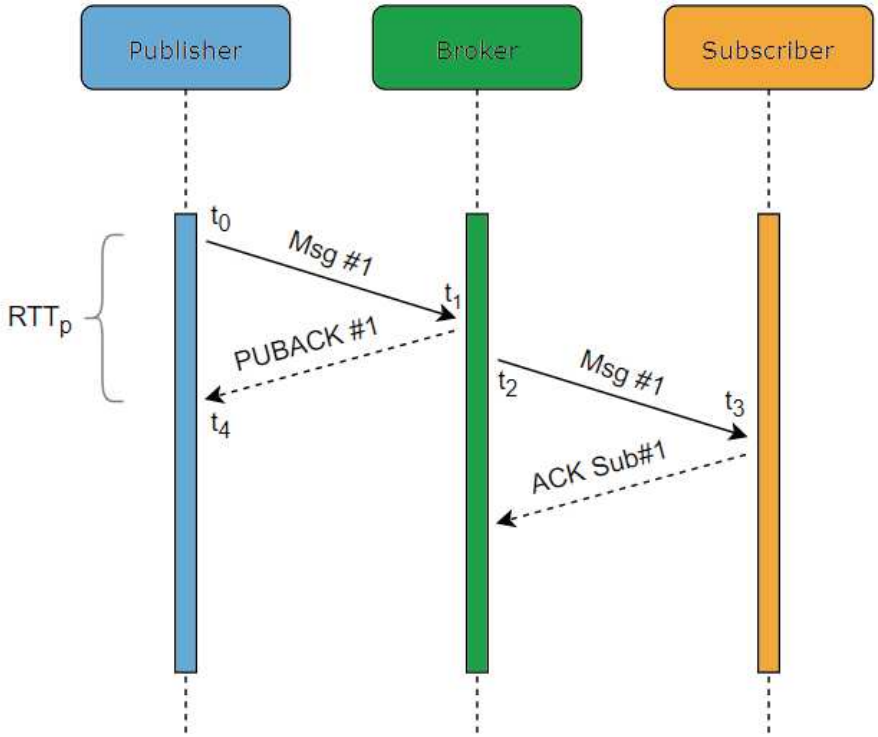

Figura 5.4: Schema di base del flusso dei messaggi per la conduzione dei test. Il client Publisher invia un messaggio al Broker che lo inoltra al client Subscriber.

Round Trip Delay (5.2)

$$RTD = (t_3 - t_0) - (t_2 - t_1) \tag{5.2}$$

Il Round Trip Time lato publisher è calcolato mediante la (5.3)

$$RTT_p = t_4 - t_0 \tag{5.3}$$

in cui t_0 è l'istante temporale in cui P1 pubblica il messaggio con QoS pari ad 1 e t_4 identifica l'istante in cui il PUBACK inviato dal broker è ricevuto da P1. Dal momento che i client sono simulati nella stessa macchina, è possibile ipotizzare che i One Way Delays siano simmetrici, pertanto il valore ricercato T_c è semplicemente ottenuto da (5.4):

$$T_c = t_2 - t_1 = t_3 - t_4 \tag{5.4}$$

Al fine di verificare che queste ipotesi non siano troppo semplificative, con conseguente inconsistenza dei risultati successivi, viene effettuata una validazione dei dati mediante simulazioni da confrontare con i log ottenuti da una piattaforma. Esaminiamo la piattaforma AWS che fornisce il servizio CloudWatch [100] per l'analisi dei log. Dopo aver creato una policy nell'IoT-Core che permette la comunicazione tra i due servizi, è possibile analizzare il log generato, un file JSON con il seguente formato:

```
  { "timestamp":  "2019-08-19 10:54:07.180",
"logLevel":  "INFO",
"traceId":  "xxxxxxxx-xxxx-xxxx-xxxx-xx",
"accountId":  "0123456789",
"status":  "Success",
"eventType":  "Publish-In",
"protocol":  "MQTT",
"topicName":  "/path/to/topic",
"clientId":  "cliendId",
"principalId":  "principalIdentification",
"sourceIp":  "xxx.xxx.xxx.xxx",
"sourcePort":  01234 }
```

Il campo *eventType* contiene il valore "Publish-in" quando un messaggio arriva verso il broker ed il valore "Publish-out" quando il broker invia il messaggio verso il client subscriber. Il valore che dobbiamo ricercare è la differenza tra i campi *timestamp* dei due file JSON. Al tal fine abbiamo sviluppato uno script in Python che simula un client publisher ed un client subscriber: il publisher pubblica un messaggio contenente un numero identificativo e scrive l'istante t_4 in un file di testo. Il subscriber compie la stessa operazione nel momento in cui riceve il messaggio scrivendo l'istante t_3.
Abbiamo condotto 100 test in 23 diffenti simulazioni e per ogni simulazione è stata calcolato il valor medio e la deviazione standard, sia dai file di log di Amazon che dai file di testo ottenuti dai test, che abbiamo analizzato per la verifica delle assunzioni effettuate. Partendo da un caso base in cui un client pubblica un messaggio per secondo (mps) in un topic con un solo sottoscrittore attivo, sono stati inviati 1000 messaggi ed è stata tenuta traccia dei risultati ottenuti. La frequenza di invio è stata incrementata da 1 a 1000 mps e successivamente il numero di client publisher e subscriber è stato innalzato a 5 ed a 10 (mantenendo sempre un rapporto 1:1), con il risultato di arrivare ad una massima frequenza di invio di 10000 mps. I valor medi e le deviazioni standard ottenute, come mostrato in Tabella 5.2, sono tra loro molto vicini per ogni caso ed in tutti i test le distribuzioni di probabilità seguono trend identici. Pertanto ci è stato possibile affermare che gli errori ottenuti dalle semplificazioni

adottate sono trascurabili per la valutazione della metrica in esame.

Tabella 5.2: Media e deviazione standard dei Cloud Service Time ottenuti dai log di AWS e dalle simulazioni. I tempi sono in ms.

Case Id	**Client(pub/sub)**	**Mps**	**T-Cloud (Log)**	**St.Dev. (Log)**	**T-Cloud (Sim.)**	**St.Dev. (Sim)**
1	1	1	26.1703	0.1974	26.1457	0.2445
2	1	10	29.042	3.5295	29.0924	3.6021
3	1	20	27.4792	1.2097	27.5122	1.2839
4	1	50	27.5687	4.2957	27.5172	4.3147
5	1	100	27.2277	6.1284	27.1758	6.0668
6	1	200	27.1459	5.7324	27.2542	5.7913
7	1	500	27.0698	5.0916	26.8597	5.138
8	1	1000	28.0938	4.4566	28.1551	4.6476
9	5	5	27.3336	2.1754	27.5341	2.0532
10	5	50	26.8552	3.4997	26.8982	3.6003
11	5	100	27.6465	3.5523	27.5893	3.3221
12	5	250	27.5393	3.1055	27.5828	3.0937
13	5	500	27.3057	3.1549	27.1894	3.2549
14	5	1000	27.001	3.1324	27.1843	3.2905
15	5	2500	26.7681	4.5647	26.6941	4.4938
16	10	10	25.7714	1.3541	25.7149	1.2945
17	10	100	25.7922	0.5585	25.8143	0.4947
18	10	200	25.847	5.0159	25.8023	5.0648
19	10	500	28.0939	1.4566	28.2154	1.5471
20	10	1000	27.9916	2.1064	27.8596	2.0754
21	10	2000	28.1483	2.7193	28.0868	2.6738
22	10	5000	28.5947	3.2648	28.4681	3.8844
23	10	10000	30.7249	4.8845	30.7684	4.9901

5.3.2 Scenari di riferimento

Al fine di analizzare le prestazioni in termini di tempo di elaborazione del Cloud Gateway delle piattaforme, sono stati considerati diversi scenari con carichi differenti. I carichi si riferiscono a diverse dimensioni, come il numero di client publisher, il numero di client subscriber, il numero di messaggi scambiati, la dimensione dei messaggi, la percentuale di messaggi pubblicati o consumati. Sono state inoltre fatte le seguenti ipotesi per la costruzione degli scenari:

- dimensione fissa del payload del messaggio pari a 150 byte;
- numero di argomenti pari al numero di client publisher (ovvero ogni client pubblica messaggi sul proprio argomento riservato);
- ogni client, sia publisher che subscriber, lavora creando la propria connessione al gateway Cloud. In questo modo le connessioni non vengono condivise da più client, in modo tale da simulare più fedelmente diversi dispositivi (o applicazioni) connessi.

Sono stati considerati 3 casi, aumentando i carichi per ogni caso, in modo tale da poter analizzare sia il tempo di servizio del broker per ciascuna piattaforma, sia l'affidabilità dei limiti imposti dalle piattaforme stesse.
Per tutte le piattaforme sono stati sottoscritti account gratuiti, che introducono alcune limitazioni, soprattutto per quanto riguarda la piattaforma Microsoft. Infatti, come riportato in [101], il livello gratuito per Azure IoT Hub consente la creazione di una singola unità, con un limite di 100 connessioni/sec accettate e un massimo di 8000 messaggi/giorno.

I casi considerati sono i seguenti:

- *Point to Point:* in questo scenario il numero di client publisher è uguale al numero di client subscriber, ognuno dei quali è in ascolto su di un singolo argomento. Il carico viene aumentato in due modi diversi: mantenendo fisso la frequenza dei messaggi inviati dai publisher a 10 mps ed aumentando il numero di client (tra 1 e 600), oppure mantenendo fisso il numero di client (100 client publisher e 100 subscriber) variando la frequenza di invio dei messaggi (tra 1 mps e 10 mps per client).
- *Fan in:* in questo scenario, più client pubblicano messaggi (ciascuno sul proprio argomento) ed un client subscriber è in ascolto, utilizzando la wildcard #, su tutti gli argomenti. I carichi vengono modificati agendo sul numero di client publisher collegati con una velocità di invio messaggi pari a 10 mps per client.
- *Fan out:* in questo scenario, chiamato anche "broadcast", un client publisher pubblica i messaggi su un argomento, su cui sono iscritti più subscriber. In questo scenario, i casi vengono simulati mantenendo la velocità

di invio del messaggio fissata a 10 mps ed aumentando il numero di client sottoscritti.

5.3.3 Risultati ottenuti

I client sono stati simulati come client statici in una macchina con le seguenti caratteristiche: Intel Xeon X5650 (x2) CPU, 12 MB cache, 2.66 GHz, 16 GB RAM con sistema operativo Ubuntu 18.04.1 LTS. I client sono stati implementati in GoLang, un linguaggio di programmazione sviluppato da Google, il cui approccio alla concorrenza è differente rispetto al classico utilizzo di thread e di memoria condivisa utilizzati nella maggior parte dei linguaggi di programmazione. La concorrenza per GoLang è una parte intrinseca del linguaggio di programmazione stesso ed è gestita dalle *Goroutine* e dai *Channels*. Le *Goroutine* altro non sono che funzioni o metodi che vengono eseguite in modo concorrente ad altre funzioni o metodi nello stesso spazio di indirizzi, in modo tale che l'accesso alla memoria condivisa sia per forza di cosa sincronizzato. Non vengono intesi come veri e propri thread, ma, come definito dalla documentazione stessa, dei "light-weight threads" gestiti interamente dal runtime di Go. I *Channel* invece sono delle *pipe* che hanno il compito di connettere le *Goroutine* concorrenti.

Abbiamo sviluppato un tool che prende in input dalla command line i seguenti parametri:

- l'endpoint del broker MQTT nella forma `scheme://endpoint-url:port`;
- lo scenario da simulare;
- il numero di client da simulare;
- il numero di messaggi;
- l'intervallo in *ms* tra i messaggi;
- la dimensione dei messaggi;
- il livello di QoS dei client publisher e subscriber;
- eventuali credenziali per la connessione sicura.

Nel caso della Google Cloud Platform è necessario indicare anche il JWT.

È stata utilizzata l'implementazione in GoLang della libreria Paho che permette anche le connessioni verso un broker MQTT via TCP, TLS o over WebSocket. Il tool va a creare tante *Goroutine* quante il numero di client contenuto nel parametro in input, a seconda del caso considerato (lo scenario "A" identifica il Point-to-Point, quindi lo stesso numero di client publisher e subscriber; lo scenario "B" il "Fan-in", quindi il parametro `client` identifica il numero di

client publisher; viceversa per lo scenario "C" che identifica il "Fan-out", dove il parametro `client` identifica il numero di client subscriber). Le routine attendono che ogni client abbia effettuato la connessione sicura, quindi si sia autenticato, con il broker. Al termine delle procedure di connessione inizia la pubblicazione dei messaggi dei client sui propri topic riservati. Allo stesso tempo i subscriber si mettono in ascolto su questi topic. I client publisher tengono traccia dell'istante di ricezione del PUBACK ricevuto dal broker (grazie alla QoS pari ad 1) in un array il cui indice è l'identificativo del messaggio contenuto nel payload. I subscriber, nella stessa modalità, tengono traccia dell'istante di ricezione.
Al termine dell'esecuzione delle *Goroutine*, gli array creati sono inviati alla routine di main che si occupa del salvataggio dei valori di ogni publisher ed ogni subscriber in un array bi-dimensionale. Le matrici ottenute sono passate ad una funzione che si occupa della memorizzazione in un database relazionale MySql unitamente ai parametri della simulazione effettuata. Le performance, intese anche come tempi di latenza, dei servizi Cloud possono variare anche durante una stessa giornata. Pertanto, sono state eseguite 42 differenti misurazioni per ogni simulazione (2 al giorno in diversi orari per 3 settimane). Al termine delle simulazioni, una funzione calcola i valor medi e la deviazione standard per la metrica considerata per ogni simulazione. Tali valori vengono memorizzati in un database al quale si connette una funzione MatLab che si occupa del plot dei risultati ottenuti.

- *Point-to-Point*: come descritto nel paragrafo 5.3.2, Azure impone delle limitazioni riguardanti il numero massimo di connessioni accettate al secondo all'interno del free-tier, il numero massimo di messaggi per giorno accettati, così come il numero massimo di messaggi dai dispositivi verso il Cloud pari a 100 per secondo. Pertanto la scelta è stata quella di separare i risultati ottenuti da questa piattaforma rispetto alle altre.

  In questa tipologia di scenario le piattaforme di Amazon e Google hanno un comportamento simile ed uniforme in termini di tempi di risposta del broker in relazione al numero di messaggi pubblicati per secondo. In Fig.5.5, nelle ascisse, è riportato il numero di messaggi per secondo, ottenuto incrementando la frequenza di invio messaggi con un solo client publisher connesso fino a 100 mps. Successivamente a tale valore la frequenze di invio è stata fissata a 10 mps ed è stato aumentato il numero di client publisher connessi tra un minimo di 1 ed un massimo di 600. La piattaforma di Google risponde più velocemente rispetto alle altre per tutti i valori di mps simulati, ad eccezione del range 150-750 mps dove è Amazon a fornire i tempi migliori.

  Azure, anche con condizioni di carico che possono essere comparate ai

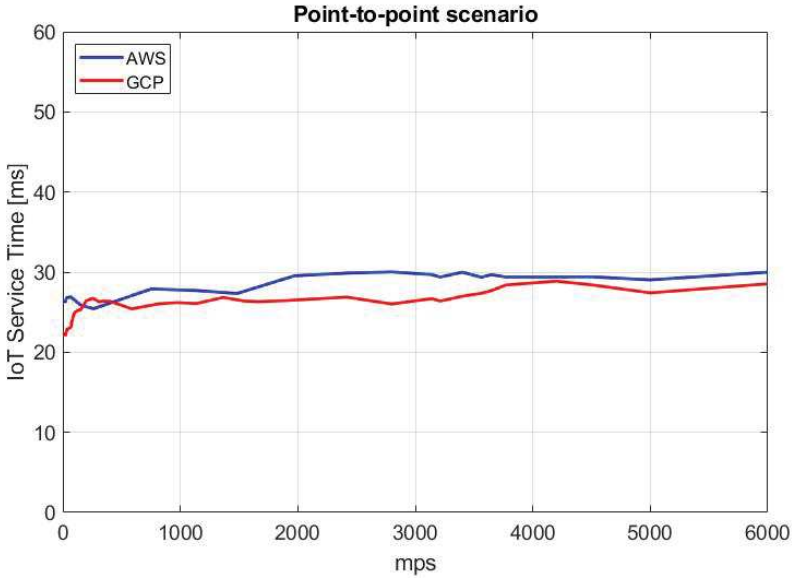

Figura 5.5: Scenario Point-to-Point: Cloud Service Times in relazione ai mps inviati dai client. Il carico viene aumentato agendo sul numero di client collegati (tra 1 e 600) e fissando 10 mps/client.

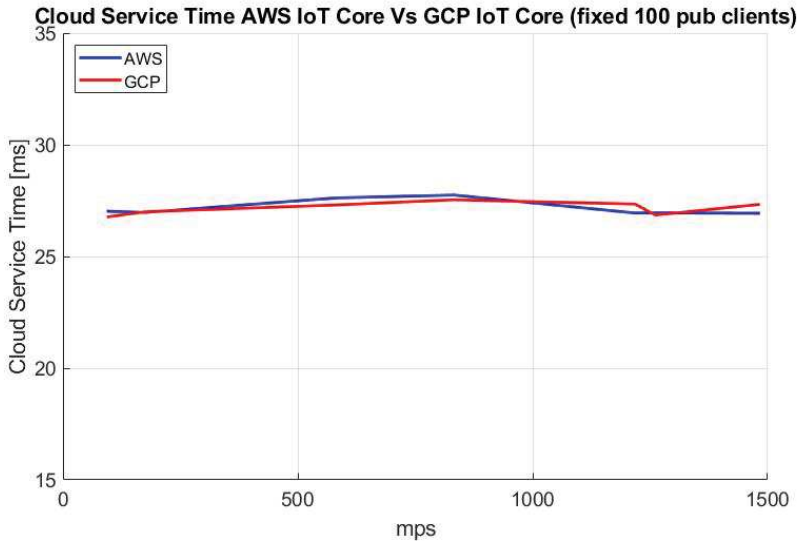

Figura 5.6: Scenario Point-to-Point: Cloud Service Times in relazione ai mps inviati dai client. Il carico viene aumentato agendo sui mps inviati (tra 1 e 15) e fissando il numero di client (100).

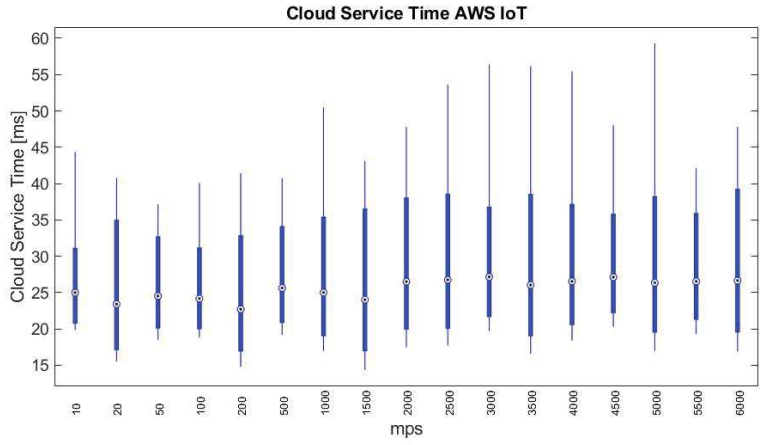

Figura 5.7: Boxplot dei risultati di AWS per lo scenario point-to-point.

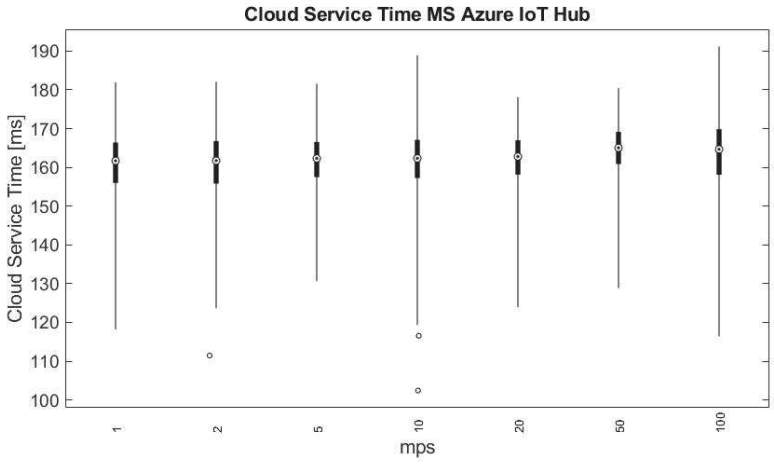

Figura 5.8: Boxplot dei risultati di Azure per lo scenario point-to-point.

carichi a cui sono state sottoposte le altre piattaforme, ottiene un tempo di servizio molto più alto rispetto ai competitor, attestandosi su $180ms \pm 20ms$. Le piattaforme non sembrano risentire particolarmente a livello prestazionale degli aumenti di carico.

I limiti dichiarati dalle documentazioni delle piattaforme sono rispettati, in particolare il numero massimo delle connessioni accettate per secondo e per account ed il numero massimo di messaggi per secondo accettati dai broker.

In Fig. 5.6 il carico verso il broker è stato aumentato solamente agen-

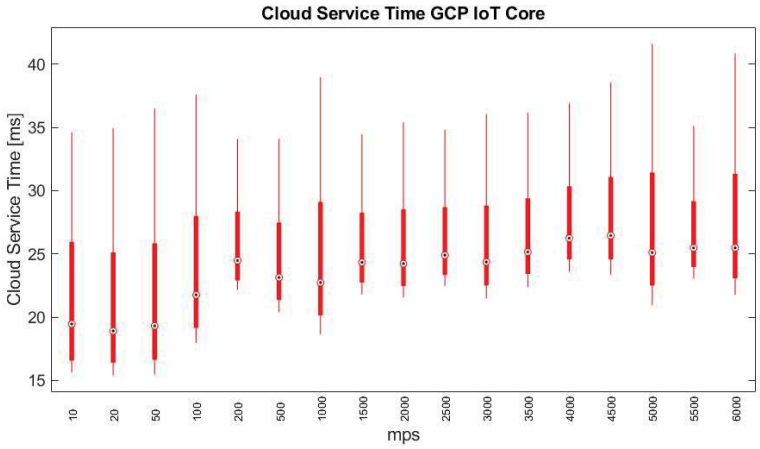

Figura 5.9: Boxplot dei risultati della Google Cloud Platform per lo scenario point-to-point.

do sulla frequenza di invio e fissando il numero di client connessi a 100. In questo caso i broker delle piattaforme hanno un comportamento più uniforme rispetto al caso precendete, non mostrando significativi cambiamenti in termini di tempo di risposta. Ad ogni modo, in questo caso, la piattaforma che riporta i migliori risultati è quella di Amazon, attestandosi su $27ms \pm 0.3ms$.
Al fine di fornire una descrizione delle distribuzioni delle simulazioni effettuate, nelle Figg. 5.7 - 5.8 - 5.9 vengono mostrati i box plot. Particolarmente interessante notare come i risultati di Azure (5.8) seguano distribuzioni più simmetriche rispetto alle altre piattaforme e come i valori siano maggiormente concentrati intorno alla mediana. Le distribuzioni ottenute dai risultati di AWS e di Google sono asimettriche positive, evidenziando quindi una maggiore dispersione per valori di carico più elevati. Infine, da notare la completa assenza di outliers per quanto riguarda AWS e Google, minimamente presenti invece in Azure.

- *Fan-in*: in questo tipo di scenario, in cui ogni client pubblica messaggi sul proprio topic ed un singolo subscriber è sottoscritto mediante wildcard a tutti i topic, è interessante valutare il *Message Loss*, inteso come il numero di messaggi che non sono stati inoltrati al subscriber rispetto al numero totale dei messaggi inviati. Da notare il fatto il che Azure e Microsoft non permettono la diretta sottoscrizione mediante wildcard, ma consentono il forward di tutti i messaggi verso altri servizi Cloud addizionali: Google, ad esempio, permette la registrazione di tutti i dispositivi

in un registro al quale è possibile associare un topic nel servizio PubSub [102]. Ogni client invia i propri messaggi di telemetria verso il proprio topic e la piattaforma automaticamente li inoltra al PubSub. Abbiamo quindi sviluppato un'altra applicazione collegata a questo servizio ed al topic del registro, simulando pertanto perfettamente lo scenario considerato. Azure ugualmente permette la nativa integrazione del proprio IoT Hub con uno o più endpoint ed anche in questo caso abbiamo creato una applicazione collegata a tali endpoint che riceve i messaggi inoltrati.

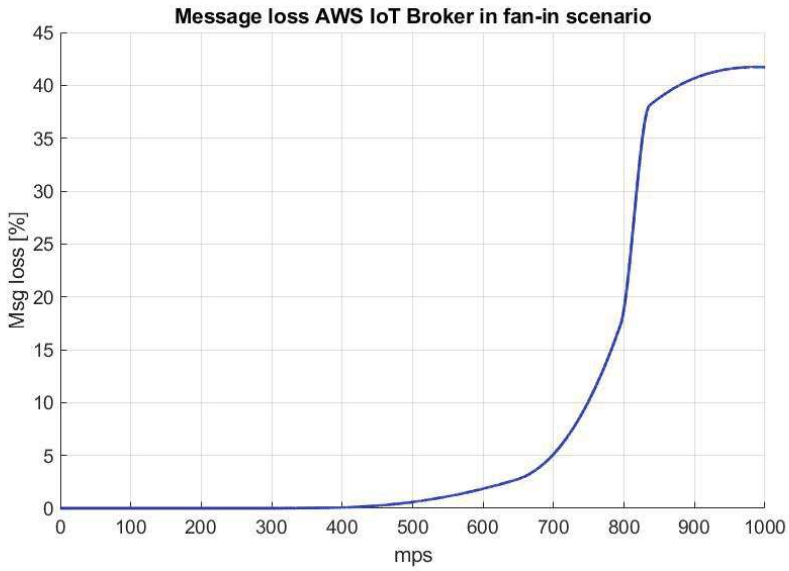

Figura 5.10: Fan-in scenario: Message loss di AWS in funzione dei mps inviati ed ottenuti mediante incremento del numero di client connessi con 10 mps/client.

Pertanto, i risultati riportati in Fig. 5.10 riguardano solamente il *Message Loss* relativamente alla piattaforma di Amazon, ottenuto mediante connessione diretta al broker MQTT e sottoscrizione a tutti i topic. Il broker è stato in grado di effettuare il forward di tutti i messaggi verso il subscriber fino al limite di 400 mps (ottenuto con 40 client connessi che inviano 10 mps). Importanti perdite iniziano ad avvenire con 70 client connessi con un andamento esponenziale fino a circa 810 mps, stabilizzandosi intorno al 42% per un carico di 1000 mps. Questi valori ottenuti possono risultare utili in molte applicazioni e possono permettere ai solution architect di studiare una ripartizione differente tra topic e subscriber.

Con la finalità di effettuare le stesse analisi per tutte e tre le piattaforme, abbiamo utilizzato un servizio aggiuntivo di Amazon chiamato SNS [103].

Tale servizio può essere connesso nativamente con l'IoT-Core mediante la creazione di una regola che effettua il forward di tutti i messagi verso il servizio. Lo stesso è stato fatto per Azure, è stata cioè creata una regola che effettua ugualmente il forward di tutti i messaggi verso il servizio di message bus [104]. I client subscriber, per tutte le piattaforme, sono in ascolto sui canali verso i quali vengono inoltrati i messaggi e conteggiano il numero effettivo di messaggi recapitati. In questo caso abbiamo assistito alla ricezione di tutti i messaggi da parte di tutte le piattaforme in qualsiasi condizione di carico considerata. Sono stati calcolati quindi

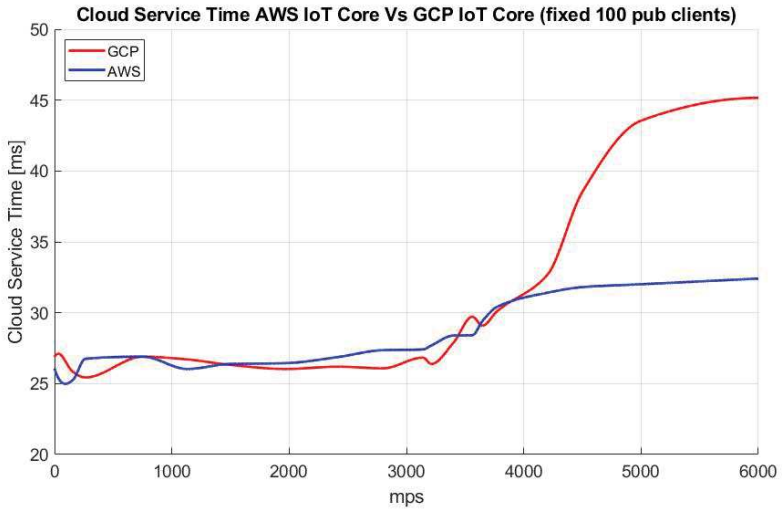

Figura 5.11: Fan-in scenario: Cloud service time vs mps inviati dai client (tra 1 e 600) con 10 mps/client.

i Cloud Service Time con le procedure illustrate in precedenza, ma in questo caso il tempo tiene conto anche dell'inoltro dei messaggi verso i rispettivi servizi successivi. Si ritiene che questi risultati ugualmente utili per gli specialisti di soluzioni Cloud-IoT per effettuare la scelta dell'architettura migliore in relazione ai bisogni delle applicazioni. Le simulazioni, riportate in Fig. 5.11, mostrano come i trend dei tempi siano molto simili a quanto ricavato dallo scenario precedente fino ai 3000 mps. Dopo questo valore, Amazon e Google mostrano un importante incremento nei tempi. Analizzando i log delle piattaforme, per cercare di capire il motivo di tale incremento, non si notano variazioni così importanti durante quei valori di carico; pertanto questi ritardi possono essere attribuiti al client subscriber che non riesce ad inviare in tempo i PUBACK richiesti dalla QoS pari ad 1, raggiungendo quindi ciò che nelle documentazioni è dichiarato come "Maximum inbound/outbound unacknowledged QoS 1 publish

requests". Il raggiungimento di tale valore obbliga il client subscriber ad attendere quei millisecondi che risultano nei grafici. Per quanto concerne Azure, i risutalti sono gli stessi ottenuti per lo scenario point-to-point di Fig. 5.8, mostrando quindi come la piattaforma non sia influenzata in alcun modo dal cambiamento di scenario.

- Fan-out: il carico per quanto concerne questo scenario è generato da un singolo client publisher che produce 10 mps verso un singolo topic. Un numero via via crescente di subscriber è iscritto a tale topic. Il Cloud Service Time è stato quindi analizzato in funzione del "fan-out factor", cioè, in questo caso, il numero dei client subscriber attivi. I risultati ottenuti sono mostrati in Fig. 5.12 dove è possibile osservare come la piattaforma di Google ottenga i migliori risultati rispetto sia ad AWS che ad Azure per valori di "fan-ou factor" superiori a 15. Prima di questo valore è AWS a ottenere i risultati migliori. L'Hub IoT di Azure inoltra 15 volte più lentamente i messaggi rispetto alle altre piattaforme, mostrando però (così come negli altri casi) un gap minore tra gli outlier. Infatti, considerando un fan-out factor pari a 1, il Cloud Service Time per Google, AWS e Azure è di 26.479 ms, 24.991 ms and 160.567 ms rispettivamente, mentre con 300 subscriber la differenza percentuale è del 26.7%, 68.1% and 7.1%.

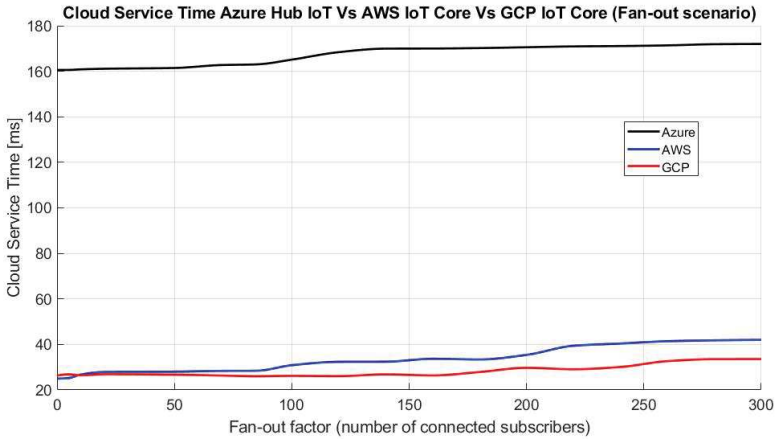

Figura 5.12: Fan-out scenario: Cloud service times vs fattore di fan-out (da 1 a 300 client subscriber).

5.3.4 Panoramica sui costi

In questo paragrafo, secondo le documentazioni fornite da ciascuna piattaforma, verranno analizzati i costi in funzione di diversi carichi di lavoro. Ogni

Tabella 5.3: Prezzi in $ dell'Azure IoT Hub.

EDITION TYPE	**MONTHLY COST/UNIT**	**MSG/DAY/UNIT**
Free	Free	8.000
S1	25	400.000
S2	250	6.000.000
S3	2.500	300.000.000
B1	10	400.000
B2	50	6.000.000
B3	500	300.000.000

piattaforma ha un differente approccio in termini di fatturazione:

1. Nella piattaforma di Amazon sono fatturati separatamenti i costi di connettività, messaggistica, le operazioni effettuate tramite i dispositivi ombra, i registri ed il motore di regole. I costi sono applicati in funzione della regione/zona in cui viene utilizzato il servizio ed i messaggi sono misurati mediante incrementi di 5 kB. Per la zona "EU-London" i prezzi sono i seguenti:
   - la connettività è misurata su base incrementale di un minuto e sul tempo totale di connessione verso l'AWS IoT Core: $ 0.096 per milione di minuti di connessione;
   - la messaggistica è misurata conteggiando il numero di messaggi scambiati tra i dispositivi e l'IoT Core: $ 1.20 quando il volume non raggiunge 1 miliardo di messaggi su base mensile, $ 0.96 per i successivi 4 miliardi, $0.84 per volumi superiori a 5 miliardi.
2. L'IoT Hub di Azure offre 2 tier, un "basic" ed uno "standard", con differenti tipi di caratteristiche supportate. Il tier standard offre tutte le caratteristiche per la comunicazione bi-direzionale tra dispositivi e Cloud, mentre il basic solo per la comunicazione dal dispositivo verso il Cloud. I prezzi sono riportati in Tabella 5.3.
3. I prezzi per quanto riguarda il Google IoT Core sono calcolati sulla base del volume di dati scambiati durante un mese, senza sovrapprezzi per quanto riguarda operazioni di creazione, lettura e modifica dei dispositivi. I messaggi con dimensione inferiore di 1024 bytes vengono considerati come 1024 bytes, mentre per messaggi di dimensione superiore viene conteggiata la effettiva dimensione. I prezzi sono riportati in Tabella 5.4.

Per condurre l'analisi abbiamo considerato uno scenario in cui i dispositivi inviano un messaggio di 1 kB per minuto. I dispositivi sono costantemente

Tabella 5.4: Prezzi in $ del Google IoT Core.

PRIZE	MONTHLY DATA VOLUME
0,00	<250 MB
0,0045 per MB	From 250 MB to 250 GB
0,0020 per MB	From 250 GB to 5 TB
0,00045 per MB	>5 TB

connessi alle piattaforme. In ascissa nel grafico di Fig. 5.13 viene riportato il numero dei dispositivi connessi, mentre in ordinata il costo mensile in dollari (il volume del traffico mensile è ottenuto semplicemente moltiplicando il numero di dispositivi per 1440 msg/giorno per 30 giorni per kB). È stata utilizzata una scala logaritmica per una visualizzazione migliore delle variazioni.

L'evoluzione dei costi è stata analizzata variando il numero dei dispositivi connessi tra 1 e 500000 (da 1440 a 720 milioni di messaggi al giorno, tra 1.40 MB e 687 GB al giorno). In Tabella 5.5 vengono mostrati i costi delle piattaforme in corrispondenza di alcuni range di dispositivi connessi, evidenziando per ogni range la piattaforma con i costi minori.

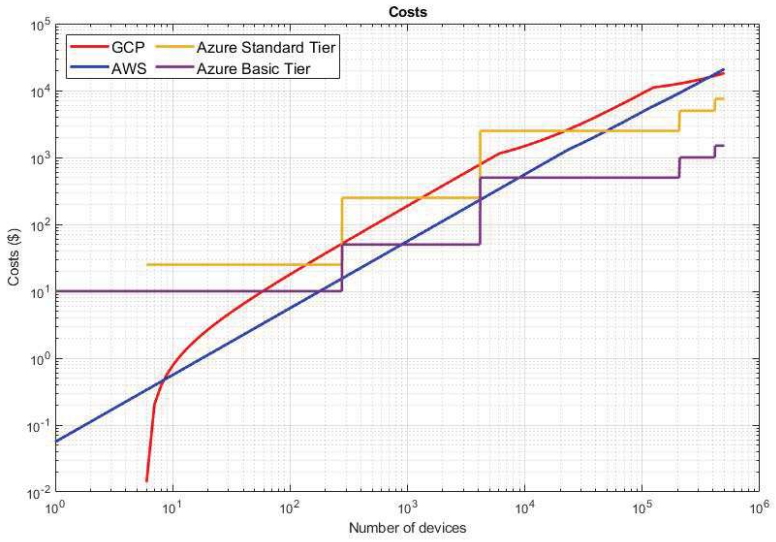

Figura 5.13: Grafico in scala logaritmica dei costi in funzione di un numero crescente di dispositivi connessi.

Tabella 5.5: Costi in $ delle piattaforme in riferimento al numero dei dispositivi connessi.

Devices	**Azure (Basic Edition)**	**Azure (Standard Edition)**	**AWS**	**GCP**
1-6	10	**Free**	<15	**Free**
7-70	10	25	**<3**	<10
70-250	10	25	**3-15**	10-45
250-1000	50	250	**15-56**	45-185
1000-4100	50	250	**56-230**	185-810
4100-10000	500	2500	**230-560**	810-1440
10000-50000	500	2500	**560-2500**	1440-4640
50000-100000	500	**2500**	2500-4800	4640-8640
100000-420000	500/1000	**2500/5000**	4800-17700	8640-16300
420000-500000	1500	**7500**	17700-21058	16300-17815

5.4 Risultati

In questo capitolo abbiamo analizzato le principali piattaforme Cloud ed i servizi che mettono a disposizione per l'IoT. Ogni piattaforma fornisce un punto di accesso al Cloud (un gateway) grazie al quale i dispositivi possono connettersi in maniera sicura verso altri servizi disponibili. Dopo l'autorizzazione alla connessione, i dispositivi possono inviare i propri dati di telemetria al Cloud mediante uno dei protocolli più utilizzati nel mondo IoT, l'MQTT. I test che sono stati condotti non hanno avuto il fine di raggiungere il massimo throughput, ma quello di valutare le performance del broker di messaggi sottoposto a diversi scenari e diversi carichi. I risultati hanno dimostrato come le piattaforme riescano a rispondere in maniera uniforme, seppur con tempi diversi, al tempo stesso garantendo prestazioni prevedibili ed in linea con quanto riportato nelle documentazioni.

In un sistema di EEW sono presenti più dispositivi che pubblicano le proprie rilevazioni su singoli topic ai quali è sottoscritto un unico client tramite wildcard, cioè l'applicazione che deve occuparsi dell'associazione delle tracce e successive analisi per la dichiarazione di un evento sismico. Tale scenario è pertanto comparabile con l'analizzato scenario di "Fan-in". Comparando le performance ottenute ed analizzando al contempo i costi coinvolti, la nostra scelta per l'implementazione del sistema è andata verso la piattaforma Cloud di Amazon.

Capitolo 6

La soluzione Cloud-IoT proposta

Le analisi svolte nei precedenti capitoli hanno permesso lo sviluppo ed il test sperimentale di una soluzione completa che sfrutta il paradigma IoT ed i servizi messi a disposizione dalla piattaforma di Amazon AWS. Partendo dall'architettura di una applicazione generica Cloud-IoT descritta nel paragrafo 2.3.2, siamo andati a collocare nei tre livelli quanto sperimentato in precendenza. In questo capitolo andremo ad illustrare il sistema suddividendo le funzionalità nei tier ed introducendo le tecnologie aggiuntive utilizzate. Inoltre, è stato sviluppato un differente algoritmo di localizzazione di cui seguirà una illustrazione e conseguente verifica sperimentale. Infine, verranno analizzate le performance ottenute su un evento sismico simulato all'interno della nostra soluzione in confronto a quanto ottenuto da PRESTo e concluderemo con una applicazione della stessa architettura con finalità di monitoraggio strutturale.

6.1 Architettura di riferimento

In Fig. 6.1 sono mostrati tutte le componenti della architettura a 3 livelli proposta: è formata da *Perception tier*, un *Platform tier* ed un *Enterprise tier*. Il Perception tier comprende il dispositivo sviluppato e descritto nel Capitolo 4, che acquisisce i dati da un sensore accelerometrico MEMS, li invia in un server SeedLink (che può opzionalmente essere interno), esegue il picking della fase P ed invia le tracce seguenti ad un broker MQTT nel Platform tier. I compiti di questo livello consistono nel monitoraggio dei dispositivi collegati, nel rendere disponibili le tracce acquisite in real-time mediante il protocollo SeedLink e nell'eseguire le operazioni di aggregazione sui dati ricevuti in modo tale da identificare e localizzare l'evento sismico. Nell'ultimo livello sono posizionate le applicazioni per la visualizzazione dello stato dei dispositivi e delle tracce in tempo reale mediante web app appositamente create, nonché le applicazioni per la notifica di malfunzionamenti ed allarmi.

Per garantire la scalabità, il sistema è basato su file di configurazione che risiedono nel Cloud. Il primo file è un JSON che contiene le informazioni sui dispositivi registrati, secondo il seguente formato:

```
   { stationIDvalue:
{"stationcode":  string(5),
"network":  string(3),
"location":  string(2),
"channel":  string(3),
"latitude":  float,
"longitude":  float,
"xUTM":  float,
"yUTM":  float,
"elevation":  float
}
}
```

Si tratta di un nested JSON composto da tanti elementi root quanto il numero dei dispositivi connessi. Per ogni dispositivo vengono registrati il nome, secondo lo standard SSNC e la posizione, sia mediante i tradizionale valori di latitudine e longitudine, sia mediante l'Universal Transverse Mercator (UTM). Un secondo file contiene le configurazioni dei programmi per l'analisi real-time, come il numero della stazioni dopo il quale è possibile dichiarare la localizzazione stimata, il valore di soglia del picco di accelerazione registrato affinché venga inviata una notifica di allerta e la posizione di uno o più punti target per il calcolo del tempo rimanente prima che le onde più distruttive arrivino. Nei prossimi paragrafi entriamo nel dettaglio di ogni livello, soprattutto per quanto riguarda le scelte implementative e le funzionalità.

6.1.1 Perception tier

In questo livello si trovano i dispositivi, i quali hanno tutte le funzionalità descritte in precedenza. Nel dettaglio le funzionalità principali sono:

- Pacchettizzazione miniSEED (frequenza di camionamento di default di 200 Hz, pacchetti da 1 s).
- Sincronizzazione via NTP.
- Invio pacchetti miniSEED via MQTT al broker nel Cloud.
- Archiviazione forme d'onda (opzionale).
- SeedLink server interno (ozionale).
- Rilevazione fase P on-board.
- Invio JSON trace via MQTT al passo di campionamento (successivamente alla rilevazione della fase P. Invio di 30 s).

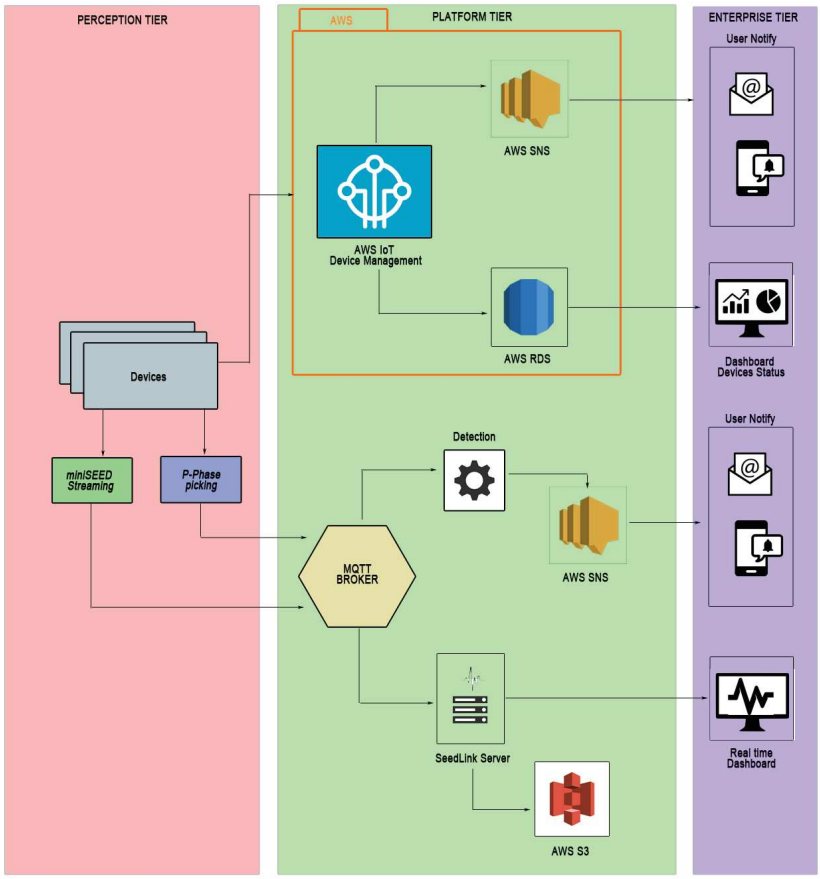

Figura 6.1: Componenti dell'architettura e 3 livelli.

6.1.2 Platform tier

Il Platform tier contiene al suo interno due livelli: uno con il compito di Device Managament ed uno per l'Application Logic. Per quanto riguarda il primo, abbiamo utilizzato la piattaforma AWS con il servizio "Device Managament" all'interno del servizio AWS IoT Core. Questo servizio consente la registrazione, l'organizzazione ed il monitoraggio dei dispositivi IoT. L'Application Logic è basata sul servizio AWS chiamato "Amazon Elastic Compute Cloud" (EC2). Si tratta di un web service che fornisce una capacità computazionale sicura e scalabile nel Cloud AWS. Un broker MQTT è stato installato all'interno della istanza AWS EC2, così come è stato sviluppato il software necessario per la dichiarazione di un evento sismico e successiva notifica. Inoltre, all'interno dell'istanza sono installati anche un SeedLink server per la trasmissione real-time

delle tracce e sono stati utilizzati altri servizi AWS per la memorizzazione delle forme d'onda.

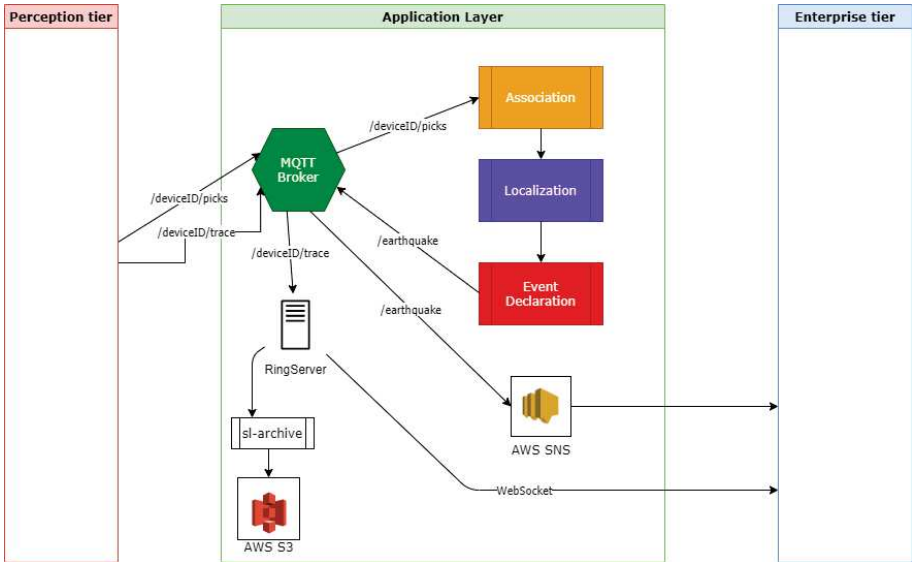

Figura 6.2: Schema a blocchi di componenti e funzionlità dell'Application Layer.

- Device Management: Il monitoraggio costante dei dispositivi connessi è di fondamentale importanza per le applicazioni IoT. In particolare, nell'EEW la conoscenza in ogni momento dell'effettivo funzionamento delle stazioni sismiche gioca un ruolo ancora più cruciale sia per quanto riguarda la dichiarazione di un evento sismico sia per la successiva stima dei parametri di origine e conseguente eventuale allerta. Al tempo stesso, visto l'obiettivo della creazione di una densa rete di stazioni, riuscire ad inviare aggiornamenti Over-the-Air ai dispositivi può contribuire a risparmi a livello temporale ed economico. Pertanto abbiamo scelto di utilizzare il servizio Device Management di AWS, vista la sua completa integrazione con AWS IoT Greengrass installato nei dispositivi e con ulteriori servizi AWS. Nel momento in cui un dispositivo si disconnette, il motore AWS invia un messaggio al servizio AWS SNS, un servizio web che coordina e gestisce la consegna o l'invio di messaggi a degli endpoint o a client di sottoscrizione. È stato creato un argomento dove viene spedito un messaggio JSON contenente le informazioni del dispositivo che ha segnalato un malfuzionamento. A tale argomento è iscritto un sottoscrittore che si occupa dell'inoltro del messaggio direttamente via email e/o SMS agli utenti specificati. Inoltre, al Device Management è collega-

to il servizio AWS Relational Database Service (RDS) per MySQL, dove vengono memorizzate le informazioni dei dispositivi.

- Application Logic: In Fig.6.2 è illustrato lo schema di funzionamento del livello. In una istanza AWS EC2 di tipo `t2.micro` è stato installato il sistema operativo Linux Ubuntu 18.04 - Bionic, unitamente al broker MQTT ed a un SeedLink Server. La scelta per quanto riguarda il broker è ricaduta su Mosquitto e, per quanto riguarda il SeedLink così come nel dispositivo stesso, è stato configurato il *ringserver*, unitamente al software `sl-archive`.

  Per la memorizzazione e trasmissione real-time delle tracce miniSEED, l'applicativo si sottoscrive al topic `/<deviceID>/trace`, legge il payload del messaggio inviato dal dispositivo, che contiene un secondo di traccia accelerometrica in miniSEED. Dopo la sottoscrizione, il funzionamento è il medesimo descritto nelle pagine precedenti per quanto riguarda il dispositivo (cfr. Paragrafo 4.2.3), con l'aggiunta della scrittura ulteriore all'interno di un bucket AWS Simple Storage Service (S3). Ogni traccia viene archiviata mediante `sl-archive` nel filesystem secondo il SeiscomP Data Structure (SDS), ovvero nella forma:

  ```
  Year/NET/STA/CHAN.TYPE/NET.STA.LOC.CHAN.TYPE.YEAR.DAY
  ```

  dove NET, STA, CHAN.TYPE e LOC identificano the Network Code, Station Code, Channel Code, Location Code secondo il SSNC. Al termine di ogni giornata uno script si occupa di replicare il contenuto delle cartelle in un bucket S3 di tipo Standard-Infrequent Access con funzionalità di storage a lungo termine e backup. Inoltre, è stato abilitato il *ringserver* anche per connessioni over WebSocket.
  Per quanto concerne l'analisi real-time per la dichiarazione dell'evento, è stato creato un modulo software che compie le seguenti operazioni:

  - Sottoscrizione tramite wildcard al topic /+/picks
  - Sottoscrizione tramite wildcard al topic /+/trace-picks
  - Associazione dei tempi di arrivo delle fasi P
  - Localizzazione epicentrale
  - Calcolo valori PGA sui successivi secondi della traccia
  - Dichiarazione dell'evento
  - Stima del tempo di arrivo delle onde S ad uno o più punti target

– Notifica dell'evento

Nel momento in cui viene ricevuto un messaggio contenente la fase P rilevata da un dispositivo, il software lo memorizza in una lista e ne determina la coerenza con la propagazione di un terremoto rispetto ad altri picchi memorizzati, utilizzando un criterio di coincidenza temporale. Considerando un velocità media di propagazione delle onde P di 6.5km/s ed una massima distanza tra 2 stazioni vicine di 40 km, se la differenza temporale tra due picchi consecutivi di due differenti stazioni è maggiore di 6 s, il picco non viene associato. Nel momento in cui n stazioni (valore configurabile, default=5) rilevano un picco ed il software li considera coerenti, inizia la fase di localizzazione.

Negli ultimi anni sono stati presentati ed adottati diversi algoritmi di localizzazione per sistemi di EEW [105, 106, 107, 49]. La nostra scelta è stata quella di utilizzare il metodo delle Iperboli (presentato in dettaglio in [108]), considerando però i Time Difference of Arrival (TDoA) [109]. La stima di una localizzazione mediante TDoA viene applicata in una vasta gamma di applicazioni, principalmente nelle reti di comunicazione wireless. La multilaterazione mediante TDoA si basa sul presupposto di una velocità di propagazione costante. Abbiamo quindi considerato una velocità media di propagazione delle onde P pari a 6.5 km/s. Inoltre, analizzando i dati relativi ai terremoti avvenuti nella catena appenninica dell'Italia centrale negli ultimi 10 anni e di magnitudo superiore a 3, la profondità media è di circa 10 km. Abbiamo quindi fissato queste due grandezze come valori di input per l'algoritmo di localizzazione. Questi valori sono configurabili, in modo che sia possibile testare il sistema su diverse impostazioni.
Nel momento in cui l'evento viene localizzato, il sistema calcola il valore di PGA da ogni traccia sui 3 s successivi al rilevamento della fase P. Se questi valori superano una soglia configurabile, l'evento viene dichiarato pubblicando un messaggio JSON contenente i dati raccolti sul topic `/earthquake`. Allo stesso tempo, viene calcolato il tempo di arrivo previsto delle onde S verso uno o più punti target. Il servizio AWS SNS è connesso all'argomento `/earthquake` per l'invio dell'avviso tramite notifiche push, e-mail e/o SMS.

6.1.3 Enterprise tier

In questo livello si trovano le interfacce per la visualizzazione dello stato dei dispositivi e delle tracce in tempo reale, unitamente agli endpoint per le notifiche sia di malfunzionamento dei dispositivi che di allerta in caso di terremoto.

Per quanto concerne le interfacce, è stata sviluppata una web-app basata su NodeJS, una piattafoma run time per l'esecuzione di app server-side in JavaScript. Grazie alle SDK fornite da AWS per JavaScript, è possibile interrogare lo stato dei dispositivi registrati nell'AWS IoT-Core. Affinché l'applicazione possa autenticarsi e connettersi con l'AWS IoT si è resa indispensabile la configurazione del servizio di identity di AWS denominato Cognito Identity Pool. Nella mappa vengono visualizzate le stazioni con colorazioni differenti a seconda del loro stato. Per la visualizzazione real-time delle tracce accelerometriche delle stazioni abbiamo sviluppato una applicazione JavaScript basata sulla libreria D3 e su SeisplotJS, una collection di routines JS per la richiesta, la manipolazione ed il plot di dati sismici, la quale ha anche delle estensioni per la visualizzazione real-time. In Fig.6.3 viene riportato uno screenshot della applicazione configurata con un set di stazioni accelerometriche della RSN ed il playback dell'evento del 24-08-2016 di magnitudo 6.0 avvenuto alle coordinate (latitudine e longitudine) 42.7, 13.23 e registrato dalla stazione FEMA.

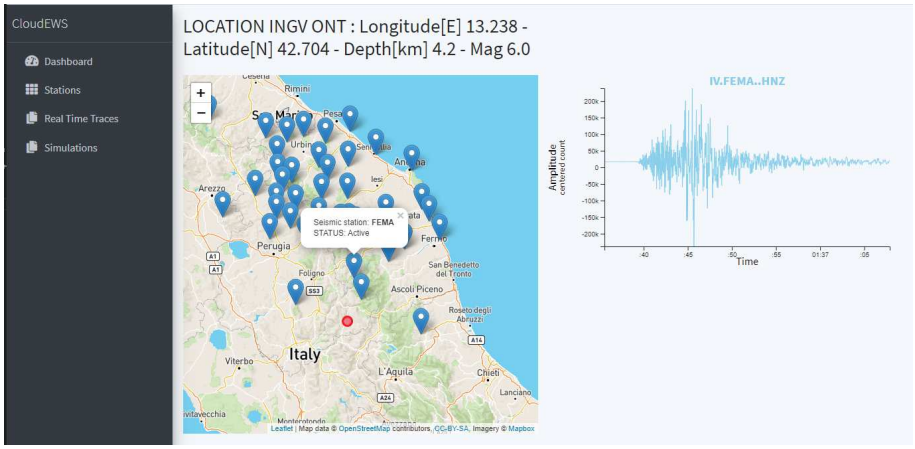

Figura 6.3: Screenshot dell'interfaccia grafica della web application sviluppata. Nella parte sinistra la mappa delle stazioni configurate, nella parte destra viene visualizzata la forma d'onda di una componente accelerometrica registrata durante un evento sismico da una stazione.

6.2 Analisi delle prestazioni

6.2.1 Trasmissione dei dati verso il Cloud

Abbiamo quindi testato le latenze verso il Cloud AWS. Amazon EC2 è ospitato in differenti localizzazione a livello mondiale, composte da "Regions", "Availability Zones" e "Local Zones". Abbiamo sviluppato una funzione che effettua

un ping all'endpoint pubblico di EC2 (ec2.*region*.amazonaws.com) e memorizza il Round Trip Time (RTT). Abbiamo eseguito 100 ping per ogni regione ogni ora per 7 giorni. Al fine di descrivere le distribuzioni di probabilità delle simulazioni effettuate, in Fig. 6.4 vengono mostrati i box-blot, dove sono riportate solamente le migliori 10 regioni in termini di latenza. La regione che ha risposto più velocemente è stata la `eu-south1`, con un tempo medio di risposta pari a $73.17ms \pm 27ms$ ed una mediana pari a 48 ms. Da notare come quasi tutte le regioni testate siano asimettriche positivamente, evidenziando quindi una maggiore dispersione per i valori più elevati.

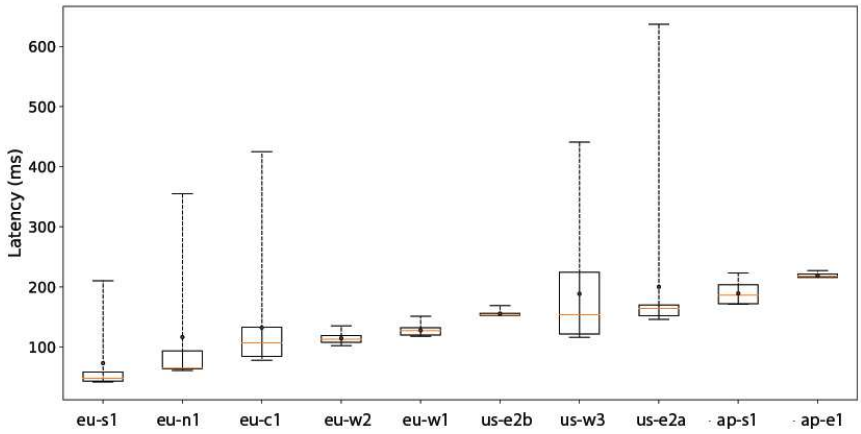

Figura 6.4: Boxplot dei RTT verso differenti istanze di AWS in diverse regioni.

6.2.2 Algoritmo di localizzazione

Le simulazioni per la valutazione dell'algoritmo di localizzazione sviluppato sono state effettuate in una macchina con le seguenti caratteristiche: Intel Xeon X5650 (x2) CPU, 12 MB cache, 2.66 GHz, 16 GB RAM con Ubuntu 18.04.1 LTS. L'algoritmo è stato sviluppato in Python e testato in modalità simulazione sulle tracce degli eventi selezionati con l'utilizzo della libreria Obspy.
Abbiamo sviluppato un programma che simula il comportamento real-time delle stazioni durante gli eventi selezionati. Le tracce vengono passate al programma in formato SAC o miniSEED ed analizzate al passo di campionamento. L'algoritmo di picking effettua l'analisi ogni 50 campioni e, nel momento in cui viene rilevata una possibile fase P, viene creato un messaggio MQTT contenente i dati della stazione ed i timestamp di rilevazione e della fase. Il programma che effettua la localizzazione è sottoscritto al topic ed inizia la stima dell'epicentro

come riceve il primo picco. Ad ogni picco successivo ricevuto, l'algoritmo corregge la stima utilizzando il metodo delle iperboli mediante l'utilizzo dei TDoA. I TDoA possono essere utilizzati per determinare la distanza tra l'epicentro e le stazioni. Ad esempio, se supponiamo che l'epicentro si trovi su un cerchio di raggio d metri dalla stazione in cui è stato ricevuto per primo il segnale, allora deve trovarsi anche su un cerchio di raggio $d + v * TDoA$ m dalla stazione dove il segnale è stato ricevuto per secondo, dove v è la velocità di propagazione. L'epicentro deve quindi collocarsi nei punti di intersezione dei due cerchi, fornendo due possibili posizioni. Iterando su un intervallo di valori di d e trovando ad ogni iterazione l'intersezione dei due cerchi, è possibile determinare un luogo di possibili posizioni dell'epicentro. Con n stazioni questo processo può essere ripetuto tra la stazione che per prima ha ricevuto il messaggio e ogni altra stazione per produrre $n-1$ luoghi di punti. Il nostro algoritmo quindi corregge ad ogni arrivo la stima della localizzazione, producendo il risultato finale quando il numero n di stazioni che hanno segnalato l'onda P è pari al valore letto dal file di configurazione.

Di seguito riportiamo i risultati dettagliati della simulazione sull'evento sismi-

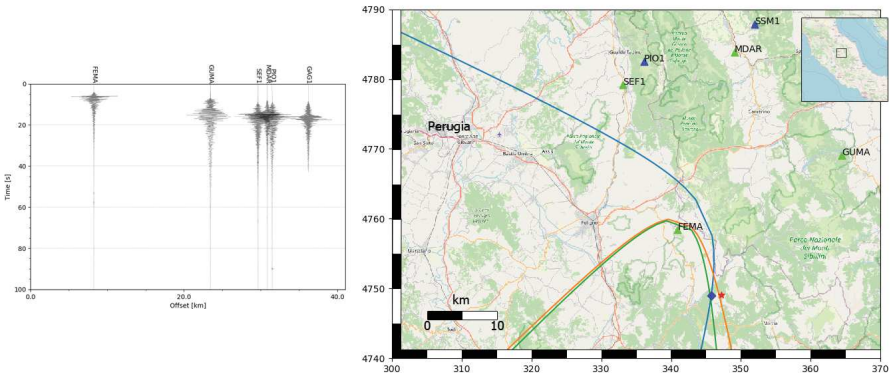

Figura 6.5: Risultato grafico dell'algoritmo di localizzazione proposto. A sinistra grafici delle tracce accelerometriche considerate ordinate per distanza epicentrale. A destra i triangoli verdi indicano le stazioni che hanno già registrato l'evento ed inviato la fase P, i triangoli blu le stazioni che ancora devono rilevare. Le iperbole indicano l'esecuzione dell'algoritmo fino a questo punto, la stessa rossa la reale posizione epicentrale, il diamante blu il risultato dell'algoritmo.

co del 2016-10-26 17:10:36UTC, localizzato alle coordinate 42.879 di latitudine e 13.129 di longitudine, di magnitudo pari a 5.4. In Fig.6.5 abbiamo riportato sia i sismogrammi delle prime 6 stazioni che hanno registrato l'evento, sia

l'esecuzione dell'algoritmo dopo il picking della fase P su 4 stazioni. Il grafico dei sismogrammi è ordinato in funzione della distanza dalla reale posizione epicentrale dell'evento sismico ed il tempo è riferito alla differenza temporale con le ore 17:10:34.00UTC. All'istante 3.45 s la stazione FEMA, distante 8.259 km dall'epicentro, esegue la rilevazione della fase P ed invia il messaggio via MQTT verso il broker. Il thread sottoscrittore stima quindi la prima localizzazione fissando come epicentro la posizione della stazione stessa. All'istante 7.11 s lo stesso procedimento viene effettuato dalla stazione GUMA. L'algoritmo inizia la stima della localizzazione collocando l'epicentro lungo l'iperbola ottenuta. Per ogni nuova stazione ricevuta, l'algoritmo corregge la stima della localizzazione fino a produrre in output il risultato nel momento in cui le stazioni da cui ha ricevuto la fase P sono uguali al valore letto nel file di configurazione. Nella mappa in Fig.6.5 vengono indicate con i triangoli verdi le stazioni che hanno inviato la propria fase P rilevata, mentre in blu quelle che ancora devono rilevarla. Con la stella rossa viene indicata la reale posizione epicentrale del sisma, mentre con il diamante blu la posizione calcolata. Per completezza vengono indicate anche le iperbole ottenute a questo punto dell'elaborazione.
Nel grafico in Fig.6.6 viene mostrata l'evoluzione temporale dell'errore in metri dopo che ogni stazione ha rilevato l'onda P. L'asse dei tempi viene indicato in riferimento all'istante della prima fase P arrivata. Da notare come già dopo due stazioni (istante t=3.66 s) l'algoritmo riesce a commettere un errore inferiore a 6 km. Errore che diminuisce all'aumentare delle informazioni ricevute dalle stazioni, sino ad arrivare a stabilizzarsi sotto ai 2 km dopo solamente 4 stazioni (t=5.99 s). È da sottolineare come in un sistema di EEW, esiste un evidente trade-off tra la correttezza dei risultati ed i tempi di risposta. Un errore come quello ottenuto dalla simulazione è da considerare accettabile [45, 64].

In Fig. 6.7(a) riportiamo il grafico delle distribuzioni di probabilità ottenute dalle simulazioni sul dataset completo, dividendo l'asse delle ascisse in bin di 5 km. Gli istogrammi sono stati normalizzati con le relative frequenze sul totale dei campioni analizzati pari a 200 eventi. La distribuzione ha un'asimmetria positiva, quindi calcoliamo anche il range interquantile che è pari a 8,836 km. Il valore medio è 9,6307 km, la mediana è 5,2851 km e il 90th percentile è 11,2834 km. Gli errori riportati dipendono dalla ipotesi semplificativa sulla profondità che è stata fissata a 10 km, infatti i maggiori scostamenti sono dovuti a profondità superiori ai 25 km. Nel grafico in Fig. 6.7(b) abbiamo riportato gli scostamenti in funzione della magnitudo dell'evento. Considerando che un EEW system dovrebbe diramare allarmi in caso di eventi potenzialmente di media/alta intensità, analizzando il dataset si nota come per eventi superiori a 4.5 il valor medio dell'errore scenda a 7.7953 km e quello della mediana a 3.9007 km.

Per quanto riguarda i tempi di esecuzione dell'algoritmo stesso, abbiamo no-

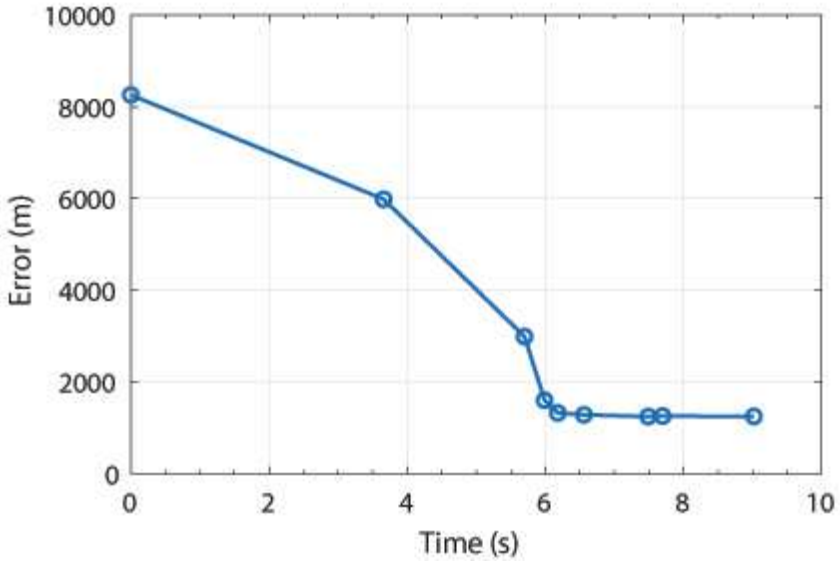

Figura 6.6: Grafico dell'evoluzione temporale dell'errore di localizzazione negli istanti in cui ogni stazione rileva l'evento.

tato come non subiscano peggioramenti dipendenti da un numero elevato di stazioni configurate, attestandosi mediamente su 195ms con deviazione standard pari a 43ms. In virtù di questi elementi, una maggiore concentrazione delle stazioni comporebbe una riduzione dei tempi di risposta, permettendo l'esecuzione dell'algoritmo in anticipo rispetto alla attuale dislocazione.

6.2.3 Simulazione di un evento sismico

In questa sezione confrontiamo le performance dell'intero sistema proposto con il sisteam di EEW PRESTo. Come è stato illustrato nel paragrafo 2.4.3, PRESTo può essere utilizzato sia in tempo reale che off-line. In tempo reale il sistema si interfaccia direttamente con la rete sismica sottostante, mentre nella seconda modalità opera su file di dati in formato SAC, precedentemente acquisiti o generati, relativi ad una rete sismica. Nella modalità di simulazione, PRESTo legge i file SAC e li converte in flussi SeedLink simulati con pacchetti di dati di 1 s. È possibile simulare latenze e guasti di rete mediante l'uso di intervalli e ritardi casuali configurabili. Nonostante la presenza di questa importante caratteristica, una simulazione ancora più vicina alla realtà consiste nell'utilizzare uno o più server SeedLink ai cui iniettare le registrazioni dell'evento sismico. Per condurre il nostro confronto abbiamo installato PRESTo sulla stessa istanza AWS EC2 in cui abbiamo implementato il nostro progetto

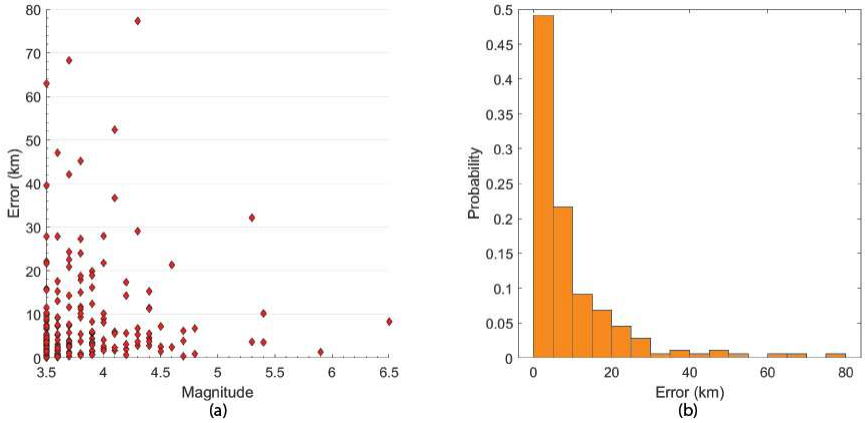

Figura 6.7: Statistiche dei risultati dell'algoritmo di localizzazione per il dataset considerato. In (a) l'errore ottenuto in funzione della magnitudo, in (b) distribuzione di probabilità dell'errore.

ed il server SeedLink nella stessa macchina in cui sono stati simulati i dispositivi e le stazioni. A differenza di PRESTo, il nostro sistema non effettua delle stime sulla magnitudo prima di inviare l'allerta, pertanto il nostro confronto si ferma al momento della localizzazione, assumendo di poter inviare inviare l'allerta subito dopo tale procedimento. Inoltre però abbiamo inserito anche una valutazione sui tempi di diramazione dell'allerta mediante l'utilizzo dei servizi AWS descritti in precedenza.
La simulazione è stata eseguita nelle stesse modalità descritte nel paragrafo 4.3.2 e sullo stesso evento sismico. I risultati, riportati in Tabella 6.1, sono leggermente differenti in virtù delle latenze aggiunte dal deploy dei servizi sull'istanza di AWS.

Tabella 6.1: Risultati della simulazione: tempi di triggering di PRESTo e del nostro sistema. I tempi sono in secondi.

Station	**Lat(N)**	**Long(E)**	**PRESTo**	**Our Solution**
FEMA	42.9621	13.0497	2.25	0.312
GUMA	43.0627	13.3335	1.97	0.286
SEF1	43.1468	12.9476	2.26	0.301
MDAR	43.1927	13.1427	2.63	0.410
GAG1	43.238	13.0674	2.49	0.332

L'output della localizzazione avviene dopo aver effettuato il binding delle fasi P provenienti da un numero configurabile di stazioni. Abbiamo fissato per entrambi i sistemi tale parametro a 5, pertanto, per il calcolo del tempo necessario agli algoritmi di localizzazione abbiamo fissato l'istante di rilevazione

dell'ultima fase come istante di inizio. PRESTo ha fornito la prima indicazione della localizzazione dopo 1.64 s, con un errore iniziale di 13.34 km, mentre il nostro sistema dopo 251 ms ha fornito un errore di 1.341 km. A questo punto PRESTo stima la magnitudo in meno di 100 ms e calcola il valore atteso della PGA verso un punto target. In questo caso il punto configurato è localizzato in Ancona, cioè ad 84.87 km rispetto all'epicentro. L'allarme verso il punto viene inviato quando rimangono 14.49 s prima dell'arrivo delle onde S al punto target, senza però essere inoltrato verso alcun altro tipo di dispositivo. Il nostro sistema, avendo anticipato la totale rilevazione dell'evento di 3.18 s rispetto a PRESTo, è in grado di diramare l'allarme quando rimangono ancora 17.67 s all'arrivo delle onde S al punto target e contribuendo inoltre a ridurre la cosiddetta blind zone di circa $11km$.
Per quanto riguardo la valutazione dei tempi di notifica verso gli endpoint configurati nel Cloud AWS, abbiamo analizzato il log fornito dalla piattaforma sul servizio AWS CloudWatch, registrando l'istante in cui l'algoritmo dichiara l'evento e l'istante in cui il servizio SNS invia la notifica verso l'endpoint. Controllando infatti il tempo intercorso tra l'operazione di *Publish-In* e quella di *RuleExecution* notiamo come il tempo medio è pari a 143 ms, mentre il tempo medio per la consegna vera e propria verso l'endpoint è pari mediamente a 207 ms.

6.3 Altre applicazioni: monitoraggio strutturale real-time

Tutte le strutture, siano esse ponti, oleodotti, gasdotti, gallerie, piattaforme di perforazione, pavimentazioni stradali, rotaie, ma anche navi, aerei, treni o abitazioni private, sono soggette ad una varietà di fattori interni ed esterni che possono causare usura o malfunzionamento. Le cause scatenanti possono essere, ad esempio, il deterioramento, un processo di costruzione non eseguito a regola d'arte, la mancanza di controlli di qualità o una situazione estrema, come un incidente o uno stress ambientale.

Negli ultimi anni si è assistito ad un crescente interesse per lo studio e la previsione delle modalità di guasto di tali strutture, equipaggiandole con numerosi sensori e raccogliendo dati sulla loro risposta a sollecitazioni e vibrazioni [110]. Un tale approccio è alla base del Monitoraggio Strutturale o "Structural Healt Monitoring" (SHM). L'SHM viene applicato per monitorare, valutare e prevedere lo stato di sicurezza ed integrità di un'ampia gamma di strutture (critiche o non), attraverso l'osservazione sperimentale del loro comportamento in servizio. Il monitoraggio del comportamento delle strutture è in grado di rilevare nel tempo eventuali anomalie, consentendo una più efficiente attuazione

degli interventi di manutenzione e riparazione, con conseguente riduzione dei costi di esercizio. In caso di eventi come i terromoti, la valutazione dello stato di sicurezza di un edificio può essere di vitale importanza, sia per conoscere le funzionalità di una struttura in condizioni normali, sia per l'adeguamento sismico, sia per una stima quanto più rapida possibile degli effetti che il sisma può aver provocato sulla stabilità della struttura stessa.

La domanda che ci siamo posti è stata, quindi, se fosse possibile utilizzare gli stessi dispositivi e la medesima architettura di sistema per una finalità differente rispetto all'EEW. In questo paragrafo viene presentata una soluzione Cloud-IoT per il monitoraggio continuo e in tempo reale degli edifici basata sul protocollo MQTT e su accelerometri MEMS. L'idea è quella di posizionare sulla struttura da monitorare un sensore smart, dotato di connettività Internet e, quindi, in grado di inviare in continuo i dati misurati da un accelerometro MEMS triassiale a basso costo. L'utente finale, previa autenticazione, può accedere da remoto alla piattaforma web espressamente sviluppata e personalizzata per la specifica applicazione. Qui può visualizzare in tempo reale i dati accelerometrici e le frequenze naturali delle vibrazioni della struttura monitorata, accedere ai dati storici e, se necessario, inviare comandi al nodo sensore.

Come struttura di prova abbiamo utilizzato la Torre della Facoltà di Ingegneria dell'Università Politecnica delle Marche, monitoriando la risposta dinamica strutturale dovuta a sollecitazioni ambientali grazie al sistema IoT sviluppato. Lo scopo del progetto è quello di consentire un monitoraggio a lungo termine e continuo degli edifici in condizioni di normale esercizio, al fine di poter valutare eventuali variazioni della frequenza di oscillazione della struttura conseguenti ad eventi straordinari (es. ristrutturazione dell'edificio, terremoti , esplosioni, ecc.).

6.3.1 Requisiti dell'applicazione

La soluzione di monitoraggio deve permettere la visualizzazione real-time in una applicazione web based dei dati raccolti dal sensore accelerometrico e la loro densità spettrale di potenza. Al tempo stesso, deve essere possibile inviare comandi di configurazione al sensore (ad esempio una differente frequenza di campionamento, l'inizio e la fine di una acquisizione finalizzata al monitoraggio strutturale, ecc.) tramite un pannello di controllo remoto, così come deve permettere la memorizzazione dei dati di telemetria al fine di permettere una analisi off-line dei dati stessi. Pertanto il sistema dovrà garantire i seguenti requisiti funzionali:

- R1. Acquisizione e trasmissione di dati accelerometrici. Il sistema deve essere in grado di leggere i dati dal sensore di accelerazione rendendoli immediatamente disponibili su Internet.

- R2. Archiviazione dei dati. Il sistema deve essere in grado di memorizzare i dati inviati dal nodo sensore in un database, al fine di consentire interrogazioni e analisi dei dati storici.
- R3. Visualizzazione in tempo reale. Il sistema deve fornire un'interfaccia di visualizzazione in tempo reale tramite un'applicazione web, accessibile da ogni dispositivo (browser, smartphone, tablet).
- R4. Analisi in tempo reale nel dominio della frequenza. Il sistema dovrebbe essere in grado di campionare i dati di accelerazione per l'analisi in tempo reale nel dominio della frequenza.
- R5. Registrazione di simulazioni. Per valutare le prestazioni e l'affidabilità dell'unità di rilevamento deve essere possibile eseguire simulazioni su acquisizioni in tempo reale e la loro memorizzazione per un confronto offline.
- R6. Invio di comandi all'unità di rilevamento. Dall'applicazione web deve essere possibile inviare comandi al nodo sensore per modificare alcune impostazioni (fondo scala, offset per ogni asse del sensore, abilitare il filtro passa basso interno , aumentare o diminuire la frequenza di campionamento dal sensore, start e stop di campionamento e/o invio, ecc.).
- R7. Gestione dei livelli di autenticazione degli utenti. Il sistema deve garantire un accesso sicuro all'applicazione web, consentendo un accesso diversificato in base a diversi ruoli (ad esempio user admin, sensor node manager, real-time display, real-time frequency analysis, simulation manager).

Il sistema inoltre deve garantire la continuità del servizio nel tempo, con conseguente capacità di risposta rapida in caso di failure. Altra caratteristica fondamentale è la scalabilità, in previsione di una grande quantità di dati da memorizzare ed elaborare e soprattutto in previsione di una maggiore diffusione di dispositivi.

6.3.2 Applicazione in tempo reale

Tenuto conto dei requisiti funzionali e non funzionali esposti, la soluzione proposta si basa sulla medesima architettura trattata in questo lavoro di tesi e suddivisa in tre livelli. I componenti dei tre livelli sono gli stessi descritti in precedenza, con delle peculiarità aggiuntive. Infatti, il dispositivo, nel momento in cui riceve il comando di start dall'Enterprise tier, inizia ad inviare al passo di campionamento i samples rilevati sulle tre componenti verso il broker

MQTT ospitato nel Platform tier in un topic nella forma `/<deviceID>/shm`. A tale topic è sottoscritto uno script che si occupa della memorizzazione in un formato prefissato, riportato in Tab. 6.2, e senza ulteriori variazioni anche in caso di dispositivi aggiunti. La dimensione delle acquisizioni di una intera giornata dipende dalla frequenza di lettura dal sensore (es. con 64 campioni al secondo, si avranno circa 150 MB/giorno/dispositivo). Non è necessario eseguire operazioni di aggregazione in tempo reale sui dati (es. medie a finestre variabili, minimi, massimi o altre operazioni statistiche), ma i dati devono essere archiviati *as is* al fine di consentire anche analisi off-line su base temporale. Le precedenti considerazioni ci hanno condotto ad optare per un database di tipo relazionale ospitato su AWS, cioè il servizio RDS per MySQL (cfr. Par. 6.1.2).

Tabella 6.2: Strutta della tabella per le acquisizioni.

Column Name	**Data Type**	**Storage Requirement**
primary_id	bigint	8 bytes
sensor_id	int	4 bytes
sensor_time	timestamp	4 bytes
x_value	float	4 bytes
y_value	float	4 bytes
z_value	float	4 bytes

Lo script si sottoscrive mediante wildcard ai topic, memorizza localmente i dati di un minuto ed esegue un inserimento di tipo bulk dei campioni di accelerazione e dei loro tempi UTC in millisecondi. Alla fine della giornata, i dati raccolti vengono spostati in un'entità settimanale, che consente sia il salvataggio di tutti i dati storici, sia una interrogazione più rapida dei dati più freschi. Inoltre, lo stesso database viene utilizzato per gestire i dati, il ruolo e le conseguenti abilitazioni dell'utente (visualizzazione in tempo reale, invio di comandi al sensore).

L'interfaccia grafica dell'applicazione è navigabile solo dopo l'autenticazione dell'utente ed il server web risponde con un diverso tipo di home page correlata alle abilitazioni. L'home page mostra una mappa con la posizione di tutti i sensori per i quali l'utente ha almeno il privilegio di visualizzazione in tempo reale. L'utente può vedere una pagina di dettaglio per ogni sensore su cui è autorizzato, dove è possibile eseguire azioni in base al livello di abilitazione, come mostrato in Fig. 6.8.

Le operazioni possono sono le seguenti:

- visualizzazione in tempo reale dei valori accelerometrici in funzione del tempo lungo i 3 assi, con possibilità di abilitare / disabilitare uno o più assi;

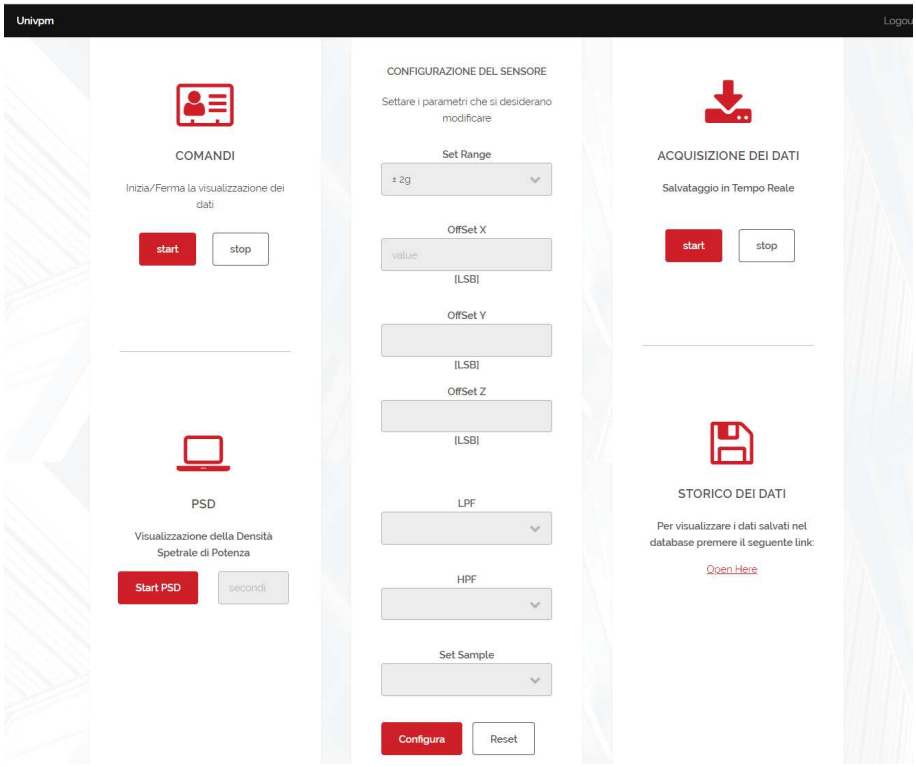

Figura 6.8: Screenshot della home della web application sviluppata per il monitoraggio strutturale.

- visualizzazione sul grafico dei dati storici (memorizzati nel database) relativi ad un intervallo di tempo selezionabile dall'utente;
- invio di comandi all'unità di rilevamento. Questa funzionalità invia un messaggio MQTT over WebSocket in un topic a il dispositivo è sottoscritto;
- esportazione CSV dei dati storici di un intervallo temporale selezionabile;
- cisualizzazione della PSD del segnale real-time (o di un segnale acquisito in precedenza) sui tre assi.

In un precedente lavoro del nostro gruppo di ricerca [111] sono state testate le performance del sensore ADXL355 in confronto ad un tipico sistema ad alte performance. I sensori erano stati installati all'ultimo piano della Torre della Facoltà di Ingegneria dell'Università Politecnica delle Marche ed i risultati avevano mostrato come il sensore low-cost fosse in grado di fornire dati per l'identificazione delle frequenze naturali dell'edificio. Proprio grazie a questi

risultati, abbiamo optato per lo stesso dispositivo che è stato installato e configurato nel contesto architetturale descritto. L'utente finale, se autorizzato, può monitorare la PSD che specifica proprio le frequenze naturali di oscillazione della struttura.

In Fig. 6.9 viene mostrato uno screenshot dell'applicazione web dove vengono riportate le densità spettrali di potenza dei valori di accelerazioni delle componenti orizzontali, ottenute durante una giornata non lavorativa.

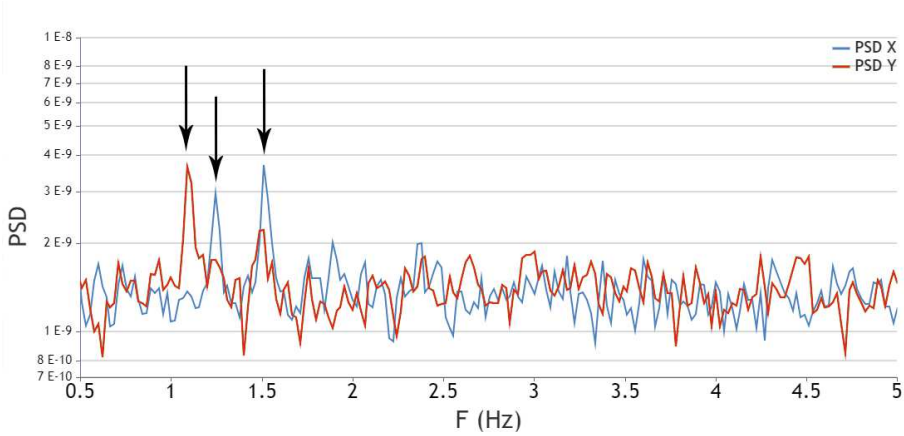

Figura 6.9: Screenshot della web application che mostra il grafico della densità spettrale di potenza delle componenenti orizzontali con indicazioni delle frequenze naturali di oscillazione.

Ad oggi il sistema è funzionante ed in operatività continua all'ultimo piano della Torre della Facoltà di Ingegneria. I dati sinora raccolti consentirebbero ulteriori analisi, ad esempio riguardo i rumori ambientali, oppure per sviluppare algoritmi di machine learning idonei alla gestione avanzata dello stato di salute dell'edificio monitorato. Inoltre, in futuro, potrebbe essere valutato l'uso di database NoSQL o Time Series Databases (TSDB), al fine di archiviare una quantità ancora maggiore di dati, compresi dati non strutturati, da un numero crescente di dispositivi. Tenendo conto della modularità architetturale della soluzione, l'utilizzo di un diverso data layer all'interno del Platform tier non comporterebbe modifiche strutturali all'intera applicazione, e, per questo motivo, sarebbe facilmente possibile scalare il sistema sviluppato, includendo un numero maggiore di edifici in contemporanea.

Capitolo 7

Conclusioni

In questo lavoro di tesi sono stati descritti gli step che hanno condotto allo sviluppo ed al test sperimentale di una architettura che sfrutti il paradigma IoT e le tecnologie di Cloud Computing per scopi di allerta rapida in caso di eventi sismici importanti. L'idea alla base del lavoro vede lo studio di un sistema di EEW, a livello architetturale, come un'applicazione IoT. Pertanto la domanda iniziale è stata se l'utilizzo di protocolli peculiari del mondo IoT potesse portare dei benifici rispetto agli standard in campo sismico per quanto concerne i tempi di trasmissione delle tracce ed il rilevamento delle prime fasi di un terremoto. I risultati dei test sperimentali hanno evidenziato un guadagno di circa 2 s, solo agendo sulla prima parte di una complessa architettura. Proprio tali risultati ci hanno spinto a cercare ulteriori guadagni sia in termini di tempo, sia in termini di affidabilità.

In un sistema di EEW di tipo regionale ogni componente introduce ritardi. Ad esempio, la distanza delle stazioni dall'epicentro e la loro densità fanno variare i tempi di rilevazione, associazione e localizzazione di un evento. Al tempo stesso, gli attuali dispositivi e tecnologie utilizzate in ambito sismico, avendo scopi differenti rispetto a quelli trattati in questo lavoro, hanno caratteristiche di precisione più elevate. Ne conseguono un costo ed una complessità implementativa e di installazione che non permettono una diffusione così capillare sul territorio, specialmente se non dettata da una specifica utilità in termini di guadagno in applicazioni di allerta rapida. Pertanto, la disponibilità di dispositivi a basso costo, con la capacità ad ogni modo di rilevare eventi di una certa importanza (eventi cioè per i quali una allerta verso la popolazione o operazioni di tipo automatizzato possano salvare vite umane), potrebbe consentirne una diffusione più ampia, con conseguenti guadagni sostanziali in termini di rilevamento e valutazione dell'evento stesso. I dispositivi dovrebbero avere semplicità di installazione ed è auspicabile la loro integrazione nella rete sismica attualmente in uso. In questa direzione, è stato creato un prototipo, composto da un accelerometro MEMS e da un Single Board Computer a basso costo, in grado di rilevare eventi di magnitudo superiore a 4 che integra nativamente anche le funzionalità di pacchettizzazione e trasmissione attualmente

in uso nella rete sismica. I test, anche in questo caso, hanno portato ad un ulteriore guadagno in termini di rilevamento delle prime fasi del terremoto, in quanto questa operazione viene effettuata direttamente a bordo del dispositivo.

Per quanto riguarda la citata affidabilità del sistema, soprattutto quando sono in atto eventi sismici di dimensioni importanti, non è improbabile assistere a danni che possono toccare punti fondamentali come ad esempio centri di calcolo. Pertanto, considerando anche i limiti a livello computazionale che un dispositivo IoT può avere, ci siamo spinti verso lo studio delle più efficienti integrazioni con il mondo Cloud. Ciò ha portato ad una analisi comparativa prestazionale dei servizi messi a disposizione da una selezione delle principali piattaforme Cloud sul mercato per quanto concerne il paradigma IoT. Tale analisi ha condotto al successivo studio: implementazione e test di una soluzione completa Cloud-IoT per EEW mediante l'utilizzo della piattaforma Cloud di Amazon. La soluzione completa, considerando anche l'utilizzo di un differente algoritmo di localizzazione rispetto agli standard di riferimento e tenute presenti anche le latenze introdotte dall'utilizzo del Cloud, ha mostrato risultati incoraggianti, con guadagni totali per i tempi di prima allerta di oltre 3 s. Riuscire ad avvicinare l'elaborazione che viene fatta nel Cloud verso i dispositivi stessi, potrebbe abbattere maggiormente questi tempi. A tal scopo, in futuro, possono essere studiati sistemi che utilizzino l'Edge/Fog Computing per la valutazione di possibili ulteriori miglioramenti.

La stretta collaborazione con l'INGV Sezione di Ancona ha permesso la conduzione dei test di confronto con il sistema di EEW PRESTo, che è stato installato e configurato sulla rete sismica dell'Italia Centrale. In futuro, in virtù della suddetta collaborazione, sarà possibile l'installazione direttamente sul campo sia dei prototipi sviluppati, sia della completa architettura, con un'implementazione parallela. Infatti, stante la modularità della soluzione proposta e la configurabilità del sistema PRESTo, potrebbe essere possibile un confronto attendibile, vista la elevata sismicità della zona. Inoltre, mediante le configurazioni possibili in entrambi i sistemi, possono essere testati i funzionamenti su eventi anche con magnitudo minori, che potranno fornire risultati valutabili anche per eventuali (e si spera non frequenti) eventi maggiori.

Il sistema proposto trova la sua innovazione proprio nell'utilizzo di protocolli di pacchettizzazione e di trasmissione, di rilevamento e localizzazione di eventi che possono essere utilizzati anche a prescindere dall'integrazione con servizi di Cloud Computing, che, ad ogni modo, si è visto possono dare ulteriori guadagni, ad esempio in termini di scalabilità, modularità ed affidabilità. Pertanto, il lavoro, non intende in alcun modo sostituirsi alle strumentazioni ed alle tecnologie utilizzate in ambito sismico, ma combinarsi con esse nella corsa contro il tempo su cui ogni sistema di EEW è basato.

Lista Pubblicazioni

Riviste Internazionali

P. Pierleoni, A. Belli, R. Concetti, L. Palma, F. Pinti, S. Raggiunto, L. Sabbatini, S. Valenti, A. Monteriù "Biological Age Estimation Using an eHealth System based on Wearable Sensors", in *Journal of Ambient Intelligence and Humanized Computing*, 2019, 1-12

P. Pierleoni, R. Concetti, A. Belli and L. Palma, "Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison," in *IEEE Access*, vol. 8, pp. 5455-5470, 2020, doi: 10.1109/ACCESS.2019.2961511

M. Paoletti, R. Concetti, P. Pierleoni, A. Belli, L. Palma "Seismic Noise and Site Response Analysis Using Accelerometer Sensors". *Sensors and Transducers*, 238(11), 2019, 8-15

Sottomesso: P. Pierleoni, R. Concetti, A. Belli and L. Palma, "Earthquake Early Warning Systems Optimization Through the MQTT Protocol", *IEEE Transaction on Network and Service Management*

Sottomesso: P. Pierleoni, R. Concetti, A. Belli and L. Palma, "A Cloud-IoT based system for Earthquake Early Warning", *Future Generation Computer Systems*, Elservier

Congresso Internazionale

M. Paoletti, R. Concetti, P. Pierleoni, A. Belli and L. Palma "Spectral analysis of seismic noise using HVSR technique" in *IFSA Frequency and Time Conference (IFTC' 2019)*

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 BuildingMonitoring" in *IEEE 23rd International Symposium on Consumer Technologies (ISCT 2019)*

L. Incipini, A. Belli, L. Palma, R. Concetti and P. Pierleoni, "MIMIC: a Cybersecurity Threat Turns into a Fog Computing Agent for IoT Systems", 2019 *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, 2019, pp. 469-474. doi: 10.23919/MIPRO.2019.8756651

L. Incipini, A. Belli, L. Palma, R. Concetti and P. Pierleoni, "Databases Performance Evaluation for IoT Systems: the Scrovegni Chapel Use Case," 2019 *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, 2019, pp. 463-468. doi: 10.23919/MIPRO.2019.8756813

Congresso Nazionale

P. Pierleoni, A. Belli, L. Palma, L. Incipini, S. Raggiunto, R. Concetti, L. Sabbatini "A Cross-Protocol Proxy for Sensor Networks Based on CoAP", in *10th Forum Italiano dell'Ambient Assisted Living (FORITAAL 2019)*, June 2019.

P. Pierleoni, A. Belli, R. Concetti, L. Palma, F. Pinti, S. Raggiunto, S. Valenti, A. Monteriù "A Non-invasive Method for Biological Age Estimation Using Frailty Phenotype Assessment" in *9th Forum Italiano dell'Ambient Assisted Living (FORITAAL 2018)*, July 2018

Capitolo di libro

P. Pierleoni, A. Belli, R. Concetti, L. Palma, F. Pinti, S. Raggiunto, S. Valenti, A. Monteriù "A Non-invasive Method for Biological Age Estimation Using Frailty Phenotype Assessment". In *Ambient Assisted Living*, pp. 81–94. Springer, Cham, 2018. DOI: 10.1007/978-3-030-05921-7_7

Bibliografia

[1] R. M. Allen and D. Melgar, "Earthquake early warning: Advances, scientific challenges, and societal needs," *Annual Review of Earth and Planetary Sciences*, vol. 47, pp. 361–388, 2019.

[2] Y. Nakamura, "Development of earthquake early-warning system for the shinkansen, some recent earthquake engineering research and practical in japan," *The Japanese national committee of the international association for earthquake engineering*, pp. 224–238, 1984.

[3] T. H. Heaton, "A model for a seismic computerized alert network," *Science*, vol. 228, no. 4702, pp. 987–990, 1985.

[4] J. Cooper, "Letter to editor, san francisco daily evening bulletin," *Nov*, vol. 3, p. 1868, 1868.

[5] P. P. Ray, M. Mukherjee, and L. Shu, "Internet of things for disaster management: State-of-the-art and prospects," *IEEE Access*, vol. 5, pp. 18 818–18 835, 2017.

[6] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future generation computer systems*, vol. 56, pp. 684–700, 2016.

[7] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[8] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[9] C. Santos, J. A. Jiménez, and F. Espinosa, "Effect of event-based sensing on iot node power efficiency. case study: Air quality monitoring in smart cities," *IEEE Access*, vol. 7, pp. 132 577–132 586, 2019.

[10] 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, 2018.

[11] H. Jiang, C. Cai, X. Ma, Y. Yang, and J. Liu, "Smart home based on wifi sensing: A survey," *IEEE Access*, vol. 6, pp. 13 317–13 325, 2018.

[12] M. Tao, J. Zuo, Z. Liu, A. Castiglione, and F. Palmieri, "Multi-layer cloud architectural model and ontology-based security service framework for iot-based smart homes," *Future Generation Computer Systems*, vol. 78, pp. 1040 – 1051, 2018.

[13] 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, 2017.

[14] N. Gondchawar, R. Kawitkar *et al.*, "Iot based smart agriculture," *International Journal of advanced research in Computer and Communication Engineering*, vol. 5, no. 6, pp. 838–842, 2016.

[15] P. Pierleoni, A. Belli, O. Bazgir, L. Maurizi, M. Paniccia, and L. Palma, "A smart inertial system for 24h monitoring and classification of tremor and freezing of gait in parkinson's disease," *IEEE Sensors Journal*, vol. 19, no. 23, pp. 11 612–11 623, 2019.

[16] P. Pierleoni, A. Belli, R. Concetti, L. Palma, F. Pinti, S. Raggiunto, S. Valenti, and A. Monteriù, "A non-invasive method for biological age estimation using frailty phenotype assessment," in *Italian Forum of Ambient Assisted Living*. Springer, 2018, pp. 81–94.

[17] E. Manavalan and K. Jayakrishna, "A review of internet of things (iot) embedded sustainable supply chain for industry 4.0 requirements," *Computers & Industrial Engineering*, vol. 127, pp. 925–953, 2019.

[18] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[19] R. Faria, L. Brito, K. Baras, and J. Silva, "Smart mobility: A survey," in *2017 International Conference on Internet of Things for the Global Community (IoTGC)*. IEEE, 2017, pp. 1–8.

[20] L. M. Oliveira and J. J. Rodrigues, "Wireless sensor networks: A survey on environmental monitoring." *JCM*, vol. 6, no. 2, pp. 143–151, 2011.

[21] C. A. Tokognon, B. Gao, G. Y. Tian, and Y. Yan, "Structural health monitoring framework based on internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 3, pp. 619–635, 2017.

[22] 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)*. IEEE, 2019, pp. 57–62.

[23] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[24] 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)*. IEEE, 2019, pp. 463–468.

[25] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[26] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "On the integration of cloud computing and internet of things," in *Future internet of things and cloud (FiCloud), 2014 international conference on*. IEEE, 2014, pp. 23–30.

[27] S. M. Babu, A. J. Lakshmi, and B. T. Rao, "A study on cloud based internet of things: Cloudiot," in *Communication Technologies (GCCT), 2015 Global Conference on*. IEEE, 2015, pp. 60–65.

[28] S. Abdelwahab, B. Hamdaoui, M. Guizani, and A. Rayes, "Enabling smart cloud services through remote sensing: An internet of everything enabler," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 276–288, 2014.

[29] J. Zhou, T. Leppanen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, H. Jin, and L. T. Yang, "Cloudthings: A common architecture for integrating the internet of things with cloud computing," in *Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on*. IEEE, 2013, pp. 651–657.

[30] Y. Liu, K. A. Hassan, M. Karlsson, Z. Pang, and S. Gong, "A data-centric internet of things framework based on azure cloud," *IEEE Access*, vol. 7, pp. 53 839–53 858, 2019.

[31] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[32] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, "Research on the architecture of internet of things," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, vol. 5. IEEE, 2010, pp. V5–484.

[33] P. Sethi and S. R. Sarangi, "Internet of things: architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, 2017.

[34] "Cloud standards customer council," http://www.cloud-council.org, [Online] Accessed 18-Sep-2020].

[35] J. Clinton, A. Zollo, A. Marmureanu, C. Zulfikar, and S. Parolai, "State-of-the art and future of earthquake early warning in the european region," *Bulletin of Earthquake Engineering*, vol. 14, no. 9, pp. 2441–2458, 2016.

[36] J. E. Aranda, A. Jimenez, G. Ibarrola, F. Alcantar, A. Aguilar, M. Inostroza, and S. Maldonado, "Mexico city seismic alert system," *Seismological Research Letters*, vol. 66, no. 6, pp. 42–53, 1995.

[37] O. Kamigaichi, M. Saito, K. Doi, T. Matsumori, S. Tsukada, K. Takeda, T. Shimoyama, K. Nakamura, M. Kiyomoto, and Y. Watanabe, "Earthquake early warning in japan: Warning the general public and future prospects," *Seismological Research Letters*, vol. 80, no. 5, pp. 717–726, 2009.

[38] H. M. Brown, R. M. Allen, M. Hellweg, O. Khainovski, D. Neuhauser, and A. Souf, "Development of the elarms methodology for earthquake early warning: Realtime application in california and offline testing in japan," *Soil Dynamics and Earthquake Engineering*, vol. 31, no. 2, pp. 188–200, 2011.

[39] H. Chi, J. Park, and D. Sheen, "Feasibility study of eew application in korea," *AGUFM*, vol. 2008, pp. S13C–1833, 2008.

[40] N.-C. Hsiao, Y.-M. Wu, T.-C. Shin, L. Zhao, and T.-L. Teng, "Development of earthquake early warning system in taiwan," *Geophysical research letters*, vol. 36, no. 5, 2009.

[41] D. D. Given, E. S. Cochran, T. Heaton, E. Hauksson, R. Allen, P. Hellweg, J. Vidale, and P. Bodin, *Technical implementation plan for the ShakeAlert production system: An earthquake early warning system for the west coast of the United States*. US Department of the Interior, US Geological Survey, 2014.

[42] A. Kumar, H. Mittal, B. Chamoli, A. Gairola, R. Jakka, and A. Srivastava, "Earthquake early warning system for northern india," in *15th symposium on earthquake engineering, Indian Institute of Technology, Roorkee*, 2014, pp. 11–13.

[43] A. Mărmureanu, C. Ionescu, and C. Cioflan, "Advanced real-time acquisition of the vrancea earthquake early warning system," *Soil Dynamics and Earthquake Engineering*, vol. 31, no. 2, pp. 163–169, 2011.

[44] H. Zhang, X. Jin, Y. Wei, J. Li, L. Kang, S. Wang, L. Huang, and P. Yu, "An earthquake early warning system in fujian, china," *Bulletin of the Seismological Society of America*, vol. 106, no. 2, pp. 755–765, 2016.

[45] C. Satriano, L. Elia, C. Martino, M. Lancieri, A. Zollo, and G. Iannaccone, "Presto, the earthquake early warning system for southern italy: Concepts, capabilities and future perspectives," *Soil Dynamics and Earthquake Engineering*, vol. 31, no. 2, pp. 137–153, 2011.

[46] E. Weber, V. Convertito, G. Iannaccone, A. Zollo, A. Bobbio, L. Cantore, M. Corciulo, M. Di Crosta, L. Elia, C. Martino *et al.*, "An advanced seismic network in the southern apennines (italy) for seismicity investigations and experimentation with earthquake early warning," *Seismological Research Letters*, vol. 78, no. 6, pp. 622–634, 2007.

[47] B. Weber, J. Becker, W. Hanka, A. Heinloo, M. Hoffmann, T. Kraft, D. Pahlke, J. Reinhardt, and H. Thoms, "Seiscomp3—automatic and interactive real time data processing," in *Geophysical Research Abstracts*, vol. 9, no. 09,219, 2007.

[48] A. Lomax, C. Satriano, and M. Vassallo, "Automatic picker developments and optimization: Filterpicker—a robust, broadband picker for real-time seismic monitoring and earthquake early warning," *Seismological Research Letters*, vol. 83, no. 3, pp. 531–540, 2012.

[49] C. Satriano, A. Lomax, and A. Zollo, "Real-time evolutionary earthquake location for seismic early warning," *Bulletin of the Seismological Society of America*, vol. 98, no. 3, pp. 1482–1494, 2008.

[50] M. Lancieri and A. Zollo, "A bayesian approach to the real-time estimation of magnitude from the early p and s wave displacement peaks," *Journal of Geophysical Research: Solid Earth*, vol. 113, no. B12, 2008.

[51] "Iris: Sac data file format," https://ds.iris.edu/files/sac-manual/manual/file_format.html, accessed: 2020-09-30.

[52] A. Holland, "Earthquake data recorded by the mems accelerometer: Field testing in idaho," *Seismological Research Letters*, vol. 74, no. 1, pp. 20–26, 2003.

[53] R. Clayton, T. Heaton, M. Chandy, A. Krause, M. Kohler, J. Bunn, R. Guy, M. Olson, M. Faulkner, M. Cheng *et al.*, "Community seismic network," *Annals of Geophysics*, vol. 54, no. 6, pp. 738–747, 2011.

[54] E. S. Cochran, J. F. Lawrence, C. Christensen, and R. S. Jakka, "The quake-catcher network: Citizen science expanding seismic horizons," *Seismological Research Letters*, vol. 80, no. 1, pp. 26–30, 2009.

[55] Y.-M. Wu, "Progress on development of an earthquake early warning system using low-cost sensors," *Pure and Applied Geophysics*, vol. 172, no. 9, pp. 2343–2351, 2015.

[56] K. Fleming, M. Picozzi, C. Milkereit, F. Kuhnlenz, B. Lichtblau, J. Fischer, C. Zulfikar, and O. Ozel, "The self-organizing seismic early warning information network (sosewin)," *Seismological Research Letters*, vol. 80, no. 5, pp. 755–771, 2009.

[57] Q. Kong, R. M. Allen, L. Schreier, and Y.-W. Kwon, "Myshake: A smartphone seismic network for earthquake early warning and beyond," *Science advances*, vol. 2, no. 2, p. e1501055, 2016.

[58] F. Finazzi, "The earthquake network project: Toward a crowdsourced smartphone-based earthquake early warning system," *Bulletin of the Seismological Society of America*, vol. 106, no. 3, pp. 1088–1099, 2016.

[59] A. M. Zambrano, I. Perez, C. Palau, and M. Esteve, "Technologies of internet of things applied to an earthquake early warning system," *Future Generation Computer Systems*, vol. 75, pp. 206–215, 2017.

[60] J. Lee, I. Khan, S. Choi, and Y.-W. Kwon, "A smart iot device for detecting and responding to earthquakes," *Electronics*, vol. 8, no. 12, p. 1546, 2019.

[61] J. Won, J. Park, J.-W. Park, and I. Kim, "Bleseis: low-cost iot sensor for smart earthquake detection and notification," *Sensors*, vol. 20, no. 10, p. 2963, 2020.

[62] E. Panizzi, "The seismocloud app: Your smartphone as a seismometer," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*, 2016, pp. 336–337.

[63] M. Klapez, C. A. Grazia, S. Zennaro, M. Cozzani, and M. Casoni, "First experiences with earthcloud, a low-cost, cloud-based iot seismic alert system," in *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2018, pp. 262–269.

[64] Y. Behr, J. Clinton, P. Kästli, C. Cauzzi, R. Racine, and M.-A. Meier, "Anatomy of an earthquake early warning (eew) alert: Predicting time delays for an end-to-end eew system," *Seismological Research Letters*, vol. 86, no. 3, pp. 830–840, 2015.

[65] B. Dost, J. Zednik, J. Havskov, R. Willemann, and P. Bormann, "Seismic data formats, archival and exchange," in *New Manual of Seismological Observatory Practice (NMSOP)*. Deutsches GeoForschungsZentrum GFZ, 2009, pp. 1–20.

[66] W. Hanka, A. Heinloo, and K.-H. Jaeckel, "Networked seismographs: Geofon real-time data distribution," 2000.

[67] T. Yokotani and Y. Sasaki, "Comparison with http and mqtt on required network resources for iot," in *Control, Electronics, Renewable Energy and Communications (ICCEREC), 2016 International Conference on*. IEEE, 2016, pp. 1–6.

[68] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni, "A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks," in *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, 2015, pp. 931–936.

[69] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of mqtt and coap via a common middleware," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE, 2014, pp. 1–6.

[70] A. T. Ringler and J. R. Evans, "A quick seed tutorial," *Seismological Research Letters*, vol. 86, no. 6, pp. 1717–1725, 2015.

[71] M. Beyreuther, R. Barsch, L. Krischer, T. Megies, Y. Behr, and J. Wassermann, "Obspy: A python toolbox for seismology," *Seismological Research Letters*, vol. 81, no. 3, pp. 530–533, 2010.

[72] "Eclipse paho python client," https://www.eclipse.org/paho/clients/python/, [Online] Accessed 10-May-2020].

[73] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," Internet Requests for Comments, RFC Editor, RFC 5905, June 2010, http://www.rfc-editor.org/rfc/rfc5905.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5905.txt

[74] B. Matteo, "filterpicker," Nov. 2019, doi: 10.5281/zenodo.3609025. [Online]. Available: https://doi.org/10.5281/zenodo.3609025

[75] "Iris supported software: Ringserver," https://seiscode.iris.washington.edu/projects/ringserver, accessed: 2020-04-10.

[76] "Incorporated research institutions for seismology," http://www.iris.edu, accessed: 2020-04-10.

[77] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *The Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[78] M. Massa, S. Lovati, G. Franceschina, E. D'Alema, S. Marzorati, S. Mazza, M. Cattaneo, G. Selvaggi, A. Amato, A. Michelini *et al.*, "Ismd, a web portal for real-time processing and dissemination of ingv strong-motion data," *Seismological Research Letters*, vol. 85, no. 4, pp. 863–877, 2014.

[79] L. Chiaraluce, R. Di Stefano, E. Tinti, L. Scognamiglio, M. Michele, E. Casarotti, M. Cattaneo, P. De Gori, C. Chiarabba, G. Monachesi *et al.*, "The 2016 central italy seismic sequence: A first look at the mainshocks, aftershocks, and source models," *Seismological Research Letters*, vol. 88, no. 3, pp. 757–771, 2017.

[80] G. Festa, M. Picozzi, A. Caruso, S. Colombelli, M. Cattaneo, L. Chiaraluce, L. Elia, C. Martino, S. Marzorati, M. Supino *et al.*, "Performance of earthquake early warning systems during the 2016–2017 mw 5–6.5 central italy sequence," *Seismological Research Letters*, vol. 89, no. 1, pp. 1–12, 2018.

[81] A. Caruso, S. Colombelli, L. Elia, M. Picozzi, and A. Zollo, "An on-site alert level early warning system for italy," *Journal of Geophysical Research: Solid Earth*, vol. 122, no. 3, pp. 2106–2118, 2017.

[82] A. D'Alessandro, R. D'Anna, L. Greco, G. Passafiume, S. Scudero, S. Speciale, and G. Vitale, "Monitoring earthquake through mems sensors (mems project) in the town of acireale (italy)," in *2018 IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*. IEEE, 2018, pp. 1–4.

[83] A. D'Alessandro, S. Scudero, and G. Vitale, "A review of the capacitive mems for seismology," *Sensors*, vol. 19, no. 14, p. 3093, 2019.

[84] V. Vukmirica, I. Trajkovski, and N. Asanovic, "Two methods for the determination of inertial sensor parameters," *methods*, vol. 3, no. 1, 2018.

[85] A. D'Alessandro, G. Vitale, S. Scudero, R. D'Anna, A. Costanza, A. Fagiolini, and L. Greco, "Characterization of mems accelerometer self-noise by means of psd and allan variance analysis," in *2017 7th IEEE International workshop on advances in sensors and interfaces (IWASI)*. IEEE, 2017, pp. 159–164.

[86] P. Pierleoni, S. Marzorati, C. Ladina, S. Raggiunto, A. Belli, L. Palma, M. Cattaneo, and S. Valenti, "Performance evaluation of a low-cost sensing unit for seismic applications: field testing during seismic events of 2016-2017 in central italy," *IEEE Sensors Journal*, vol. 18, no. 16, pp. 6644–6659, 2018.

[87] A. Lomax, J. Virieux, P. Volant, and C. Berge-Thierry, "Probabilistic earthquake location in 3d and layered models," in *Advances in seismic event location*. Springer, 2000, pp. 101–134.

[88] P. P. Ray, "A survey of iot cloud platforms," *Future Computing and Informatics Journal*, vol. 1, no. 1-2, pp. 35–46, 2016.

[89] O. Mazhelis and P. Tyrvainen, "A framework for evaluating internet-of-things platforms: Application provider viewpoint," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 147–152.

[90] T. Pflanzner and A. Kertész, "A taxonomy and survey of iot cloud applications," *EAI ENDORSED TRANSACTIONS ON INTERNET OF THINGS*, vol. 3, no. 12, pp. Terjedelem–14, 2018.

[91] S. P. Ahuja, "On the use of system-level benchmarks for comparing public cloud environments," in *Handbook of Research on Cloud Computing and Big Data Applications in IoT*. IGI Global, 2019, pp. 24–38.

[92] L. Gillam, B. Li, J. O'Loughlin, and A. P. S. Tomar, "Fair benchmarking for cloud computing systems," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, p. 6, 2013.

[93] "Cloudharmony, transparency for the cloud," http://www.cloudharmony.com/, [Online] Accessed 29-Ottobre-2020].

[94] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: An iot benchmark for distributed stream processing systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4257, 2017.

[95] "Iot developer survey 2018," https://www.eclipse.org/org/press-release/iotdevsurvey2018.php, [Online] Accessed 28-Ottobre-2020].

[96] A. Souri, A. Hussien, M. Hoseyninezhad, and M. Norouzi, "A systematic review of iot communication strategies for an efficient smart environment," *Transactions on Emerging Telecommunications Technologies*, p. e3736, 2019.

[97] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," Internet Requests for Comments, RFC Editor, RFC 7252, June 2014, http://www.rfc-editor.org/rfc/rfc7252.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[98] P. Saint-Andre, "Extensible messaging and presence protocol (xmpp): Core," Internet Requests for Comments, RFC Editor, RFC 6120, March 2011, http://www.rfc-editor.org/rfc/rfc6120.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6120.txt

[99] J. J. Levandoski, P.-Å. Larson, and R. Stoica, "Identifying hot and cold data in main-memory databases," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 26–37.

[100] "Amazon cloudwatch," http://aws.amazon.com/cloudwatch, [Online] Accessed 20-June-2020].

[101] "Choose the right iot hub tier for your solution," https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-scaling, [Online] Accessed 20-June-2018].

[102] "Google cloud pub/sub," https://cloud.google.com/pubsub/, [Online] Accessed 15-May-2020].

[103] "Amazon simple notification service," https://aws.amazon.com/sns, [Online] Accessed 15-May-2020].

[104] "Microsoft azure message bus," https://azure.microsoft.com/services/service-bus/, [Online] Accessed 15-May-2020].

[105] P. Rydelek and J. Pujol, "Real-time seismic warning with a two-station subarray," *Bulletin of the Seismological Society of America*, vol. 94, no. 4, pp. 1546–1550, 2004.

[106] S. Horiuchi, H. Negishi, K. Abe, A. Kamimura, and Y. Fujinawa, "An automatic processing system for broadcasting earthquake alarms," *Bulletin of the Seismological Society of America*, vol. 95, no. 2, pp. 708–718, 2005.

[107] G. Cua and T. Heaton, "The virtual seismologist (vs) method: A bayesian approach to earthquake early warning," in *Earthquake early warning systems*. Springer, 2007, pp. 97–132.

[108] J. Pujol, "Earthquake location tutorial: graphical approach and approximate epicentral location techniques," *Seismological Research Letters*, vol. 75, no. 1, pp. 63–74, 2004.

[109] F. Gustafsson and F. Gunnarsson, "Positioning using time-difference of arrival measurements," in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, vol. 6. IEEE, 2003, pp. VI–553.

[110] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proceedings of the 6th international conference on Information processing in sensor networks*. ACM, 2007, pp. 254–263.

[111] S. Valenti, M. Conti, P. Pierleoni, L. Zappelli, A. Belli, F. Gara, S. Carbonari, and M. Regni, "A low cost wireless sensor node for building monitoring," in *2018 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*. IEEE, 2018, pp. 1–6.