Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il management

LA-MQTT IN UN CONTESTO MULTI-BROKER: PROGETTAZIONE, IMPLEMENTAZIONE ED ANALISI

Relatore:
Dott. Federico Montori

Presentata da:
Andrea Iannoli

Correlatore:
Chiar.mo Prof. Marco Di Felice

Sessione II
Anno Accademico 2022/2023

Abstract

L'avvento dell'Internet delle cose (IoT) ha introdotto un significativo cambiamento nel nostro rapporto con il mondo fisico, spingendo verso una crescente connessione di dispositivi in ambienti diversificati, dai contesti domestici a quelli industriali e urbani. Questa interconnessione ha evidenziato l'urgenza di protocolli di comunicazione efficienti e scalabili. Tra questi, MQTT (Message Queuing Telemetry Transport) si è affermato come uno standard dell'interscambio di dati nell'IoT.

Tuttavia, la sua mancanza di sensibilità spaziale ha generato sfide pratiche nell'affrontare dispositivi in movimento e nell'indirizzamento dei messaggi in base alla posizione geografica. Questa carenza ha ispirato l'evoluzione di LA-MQTT, un'estensione progettata per affrontare tale limitazione, mantenendo la retrocompatibilità con MQTT standard.

Il nucleo di LA-MQTT si basa su tre pilastri fondamentali: il protocollo MQTT, la preservazione della privacy nei servizi basati sulla posizione (LBS), e la consapevolezza della posizione nei sistemi publish-subscribe. L'analisi della letteratura su questi temi e dei loro punti d'incontro evidenzia il contesto in cui LA-MQTT si colloca.

Esplorando LA-MQTT in un contesto single-broker, la tesi approfondisce il modello del sistema, l'architettura software, e le librerie client e backend. Tali implementazioni mostrano il ruolo chiave della sensibilità spaziale nella comunicazione publish-subscribe di MQTT ma evidenziano i limiti di scalabilità dell'operare in un sistema single-broker.

Successivamente, questa tesi propone una soluzione che permette ad LA-

MQTT di operare in un contesto multi-broker, grazie al quale LA-MQTT affronta i problemi di scalabilità attraverso l'utilizzo di bridge e la distribuzione del lavoro tra broker. L'implementazione multi-broker comprende l'uso di bridge, e modifiche alle librerie client, e backend di LA-MQTT standard.

Un caso d'uso concreto, JAM, illustra l'applicabilità di LA-MQTT multi-broker in contesti reali. L'analisi delle tecnologie utilizzate e lo sviluppo di JAM evidenziano le potenzialità pratiche di LA-MQTT multi-broker.

Infine, l'evaluation della tesi adotta un metodo mirato, analizzando e valutando i risultati ottenuti. La sezione di valutazione include delle considerazioni finali sulla performance e l'efficacia di LA-MQTT.

La tesi si conclude con la considerazione critica di LA-MQTT in un contesto multi-broker, esplorando le sfide rimanenti e suggerendo possibili sviluppi futuri per migliorare l'efficienza e la scalabilità del protocollo in ambienti IoT complessi.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

L'avvento dell'IoT ha segnato una trasformazione significativa nel modo in cui interagiamo con il mondo fisico. Con un numero crescente di dispositivi connessi, dalla casa intelligente agli ambienti industriali fino alle smart city, la necessità di un protocollo di comunicazione efficiente, scalabile e leggero è diventata sempre più evidente. In questo contesto, MQTT (Message Queuing Telemetry Transport) si è affermato come uno dei protocolli principali per lo scambio di dati nell'IoT.

L'IoT si basa sulla connessione e sulla collaborazione di dispositivi eterogenei distribuiti in ambienti fisici diversi. Questa interconnessione consente la raccolta di dati da sensori, l'attuazione di azioni tramite attuatori e la creazione di un ecosistema intelligente e reattivo. Tuttavia, man mano che il numero di dispositivi cresce e la loro distribuzione diventa più complessa, sorge la necessità di considerare non solo la comunicazione tra i dispositivi ma anche il contesto spaziale in cui operano.

1.1 MQTT

MQTT è diventato uno standard di fatto nel campo dell'IoT grazie alla sua leggerezza, scalabilità e affidabilità. MQTT segue un modello di comunicazione publish-subscribe, in cui i dispositivi possono agire sia come "publi-

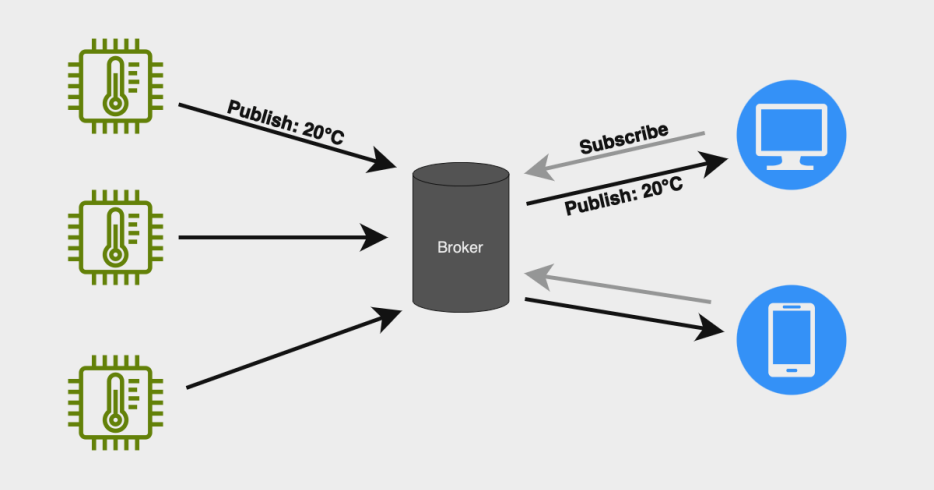

Figura 1.1: Architettura MQTT

shers" che inviano messaggi, sia come "subscribers" che ricevono i messaggi. Inoltre, esiste un concetto di "topic" che agisce come un canale virtuale attraverso il quale i messaggi vengono trasmessi. Il cuore del sistema MQTT è il broker. Il broker è un server che gestisce la ricezione e la distribuzione dei messaggi. Quando un dispositivo pubblica un messaggio su un certo "topic", il broker instrada il messaggio a tutti i dispositivi che si sono sottoscritti a quel "topic". Tuttavia, mentre MQTT ha dimostrato di essere efficace nelle comunicazioni Point-to-Point e broadcast, non tiene conto del contesto spaziale in cui operano i dispositivi.

1.2 I problemi di MQTT

La mancanza di sensibilità spaziale in MQTT può comportare problemi pratici. Ad esempio, potrebbe essere difficile gestire dinamicamente i dispositivi in movimento o garantire che i messaggi vengano indirizzati in base alla loro posizione geografica. Questa carenza può anche ostacolare lo sviluppo di applicazioni avanzate, come il monitoraggio in tempo reale di sistemi distribuiti su vasta scala. Immaginiamo, ad esempio, un'applicazione di monitoraggio ambientale in cui sensori sono distribuiti in un'area geogra-

fica vasta. La posizione di ciascun sensore può essere altrettanto importante quanto i dati che raccoglie.

Inoltre è da notare che la mancanza di sensibilità spaziale può causare inefficienze, aumentando l'onere di rete e compromettendo la qualità del servizio (QoS) a causa di notifiche inviate a consumatori geograficamente distanti dalla fonte dei dati.

1.3 LA-MQTT

Per affrontare questa sfida, F.Montori et al. [20], hanno proposto LA-MQTT, un'estensione al protocollo MQTT progettata per supportare comunicazioni di pubblicazione e sottoscrizione consapevoli dello spazio negli scenari IoT. LA-MQTT è progettato per essere broker-agnostic e completamente retrocompatibile con MQTT standard, offrendo una soluzione che integra la consapevolezza spaziale senza compromettere l'efficienza e la compatibilità. LA-MQTT introduce la sensibilità spaziale nel paradigma di pubblicazione e sottoscrizione di MQTT, considerando la posizione geografica dei consumatori (MCs) e dei fornitori di dati basati sulla posizione (LDSs). L'articolo attraverso il quale F.Montori et al. propongono LA-MQTT[20], mette inoltre, in evidenza, l'importanza di garantire la privacy della posizione dei consumatori nel tempo, sottolineando l'implementazione di un algoritmo basato sull'apprendimento automatico che gestisce il trade-off ottimale tra la privacy della posizione e la QoS delle notifiche spaziali.

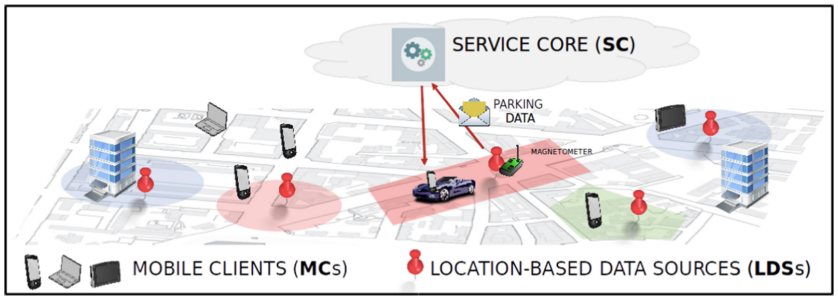

Figura 1.2: Scenario LA-MQTT. Tratto da [20]

1.3.1 I tre pilastri di LA-MQTT

È giusto sottolineare i tre principali pilastri di ricerca sui quali si basa LA-MQTT, ovvero: il protocollo MQTT, la preservazione della privacy nei LBS(Location Based Service) e la location awareness nei sistemi publish-subscribe. È quindi utile analizzare fino a che punto la letteratura si sia spinta su questi tre temi, avendo un occhio di riguardo sui loro punti d'incontro.

Ricerca correlata di MQTT

Il protocollo MQTT[1] è, come già detto, un protocollo IoT publish-subscribe client-server per lo scambio di messaggi, proposto per la prima volta nel 1999 ed ora, ormai, entrato a far parte degli standard ISO[9] e OASIS[1]. La sua leggerezza, apertura e facilità di implementazione lo hanno reso ampiamente utilizzato in contesti IoT e Machine-to-Machine (M2M), offrendo tre livelli di Quality of Service (QoS) per la consegna dei messaggi. La sua popolarità è evidenziata dalla diversità di implementazioni di broker MQTT e dalle discussioni attive nella comunità di ricerca [18]. La ricerca sulla sicurezza ha proposto estensioni al protocollo, affrontando sfide come le comunicazioni in tempo reale o la mancanza di sicurezza, con alcune soluzioni che aggiungono funzionalità di autorizzazione, autenticazione e crittografia [5][16]. Le comparazioni delle prestazioni di MQTT rispetto ad altri protocolli di messaggistica IoT, come CoAP o AMQP, sono state oggetto di studio [15][17][22]. La letteratura si è concentrata anche su versioni più recenti come MQTT 5, che introduce miglioramenti e il supporto per nuove funzionalità [9].

La privacy nei Location-based Service

Nei servizi basati sulla posizione (LBS), la massimizzazione della Qualità del Servizio (QoS) spesso richiede la conoscenza della posizione dei potenziali client. Tuttavia, questo solleva preoccupazioni sulla privacy, fortunatamente esistono diverse strategie online e distribuite proposte per affrontare questo

[1]Sito ufficiale del protocollo: https://mqtt.org

problema. Le strategie includono il mascheramento della posizione, l'uso di tecniche basate su "dummy updates" e l'invio selettivo di informazioni [3][12][13][19]. Questa sezione esplora soluzioni specifiche per la privacy nella gestione delle informazioni geografiche, escludendo strategie offline come k-anonymity e differential privacy.

La posizione nei sistemi Publish-Subscribe

La scalabilità è un problema che non è direttamente affrontato dal protocollo MQTT standard. In scenari densamente popolati da produttori e consumatori, la riduzione coerente dei messaggi è cruciale. Sono state proposte soluzioni distribuite che dividono l'area geografica in settori o cluster e utilizzano broker dedicati per ciascuno di essi [10][11]. Tuttavia, queste soluzioni non considerano la consapevolezza spaziale nella comunicazione di pubblicazione e sottoscrizione.

Sono stati proposti anche approcci specifici per la localizzazione utilizzando MQTT, come MQTT-g [2], che gestisce la rilevanza della posizione dei topic per i sottoscrittori. Altri approcci utilizzano l'estensione MQTT5 per la geolocalizzazione [8]. Tuttavia, nessuna di queste soluzioni è pienamente compatibile con le implementazioni standard di MQTT e non affronta argomenti come la privacy dei sottoscrittori.

Sono state anche proposte soluzioni di pubblicazione e sottoscrizione basate sulla posizione al di fuori del contesto MQTT, ma spesso mancano di compatibilità con le implementazioni standard esistenti.

1.4 Altri problemi di scalabilità

Sebbene LA-MQTT, tenendo conto della posizione dei client, risolva dei problemi di efficienza e migliori la scalabilità del protocollo MQTT in un ambiente IoT, il protocollo non rimane privo di problemi di efficienza che ne compromettono la scalabilità. Bisogna considerare che in un contesto

reale con un ambiente spaziale molto esteso avere un singolo broker potrebbe portare a problemi di efficienza.

La soluzione al problema è cercare di far operare il protocollo in un contesto multi-broker. Questo è possibile grazie alle numerose implementazioni del protocollo sul mercato che consentono la creazione di gateway tra diversi broker, questa funzione prende il nome di bridge. Esistono lavori in quest'ambito che propongono un approccio al bridging dinamico [14] ed altri che vedono il bridging come soluzione ai problemi di dinamicità della rete rendendo le richieste del client uniformi e indipendenti dalla topologia della rete [21]. Lo svantaggio di queste proposte è l'assenza di retrocompatibilità con MQTT standard in quanto prevedono la riscrittura dell'implementazione del broker.

1.5 LA-MQTT in un contesto multi-broker

Questa tesi propone una soluzione per rendere LA-MQTT operabile in un contesto multi-broker. Si analizzeranno quindi, le fasi che hanno portato a un ulteriore sviluppo delle librerie LA-MQTT portandole a sfruttare la tecnologia dei bridge nel migliore dei modi per permettere ai broker di far parte di un ambiente spatial aware ed interoperabile. La soluzione proposta in questa tesi affonda le proprie basi nella prima versione di LA-MQTT[20], permettendo di arrivare ad una soluzione finale broker-agnostic. L'unica discriminante rimane l'utilizzo di un broker che offra, nativamente o tramite estensioni di terze parti, la funzione di bridge.

Capitolo 2

LA-MQTT in un contesto single-broker

Per capire a pieno come si è arrivati ad una versione di LA-MQTT capace di operare in un contesto multi-broker, è necessario sapere quali sono le basi da cui si è partiti per arrivare alla soluzione proposta in questa tesi. Come già accennato nell'introduzione la soluzione proposta da questa tesi si basa sui lavori di ricerca che hanno portato alla creazione di LA-MQTT e alla sua funzione in un contesto single-broker[20]. Il seguente capitolo ha quindi la funzione di illustrare il funzionamento base di LA-MQTT.

2.1 Modello del Sistema

Consideriamo uno scenario IoT generico, come illustrato in Figura 1.2. Lo scenario coinvolge due attori principali: i Client Mobili (MC), che fungono da consumatori di servizi/dati, e le fonti di dati basate sulla posizione (LDS), che fungono da fornitori di servizi/dati. In particolare, gli MC si riferiscono a smartphone o dispositivi IoT dotati di un sensore GPS.

Per convenzione, siano M e N, rispettivamente, il numero di MC e LDS attivi. Inoltre, sia $P_i = \langle \text{lat}_i, \text{lon}_i \rangle$ la posizione attuale (latitudine e longitudine) di un detereminato MC_i. Analogamente a molti ambienti IoT,

supponiamo che la comunicazione tra MC e LDS segua un paradigma di pubblicazione-sottoscrizione, in cui un MC registra l'interesse per un canale di dati specifico e riceve notifiche quando nuove istanze di dati appartenenti a tale canale sono prodotte da qualsiasi LDS. Ogni canale è identificato da un topic (una stringa univoca) che descrive il tipo di servizio. Sia T_i l'elenco dei canali di interesse per un determinato MC_i.

A differenza delle soluzioni precedentemente esistenti, questo approccio considera vincoli spaziali per la diffusione dei dati IoT. Ciò comporta il basare l'identificazione degli MC a cui destinare un messaggio sia sui canali a cui sono iscritti che sulle loro posizioni attuali. Ad esempio, consideriamo un'applicazione di smart mobility in cui un conducente riceve notifiche in tempo reale su posti auto disponibili nelle vicinanze. Per supportare questa funzionalità, ogni LDS è associato a un argomento t_s che specifica sia il tipo di dati (ad esempio, dati riguardanti i parcheggi) che una regione di interesse, definita come "geofence" (g_s), che stabilisce la regione geografica in cui i suoi dati sono di interesse e devono essere consegnati.

Nella nostra notazione, c_s rappresenta il contenuto da consegnare rispetto a t_s, ovvero il payload del messaggio. Gli LDS generano un nuovo contenuto c_s ogni f_s secondi ma potremmo pensare anche ad una situazione in cui il nuovo contenuto viene mandato ogni qual volta il sensore dell'LDS raccoglie un dato nuovo ed interessante. Nella Figura 1.2, un magnetometro statico in $\mathrm{LDS}_{\mathrm{park}}$ notifica la disponibilità di un posto auto libero agli MC iscritti all'argomento t_{park} e situati all'interno della geofence rettangolare g_{park}. I dati di disponibilità c_{park} del posto auto vengono generati ogni f_{park} secondi.

La comunicazione tra MC e LDS avviene attraverso un'infrastruttura di terze parti chiamata Service Core (SC). Il disaccoppiamento logico tra produttori e consumatori di dati, facilitato dal SC, consente agli MC di concentrarsi sul topic di interesse piuttosto che sugli indirizzi di rete delle fonti di dati. Per implementare la comunicazione di pubblicazione-sottoscrizione, gli MC devono notificare periodicamente al SC remoto la propria posizione GPS (P_i). Sia f_i l'intervallo di tempo tra due eventi di pubblicazione GPS

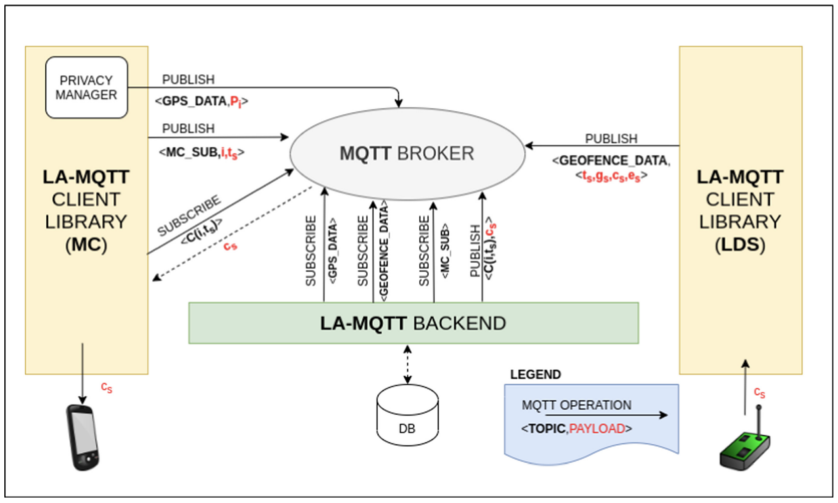

Figura 2.1: Architettura software di LA-MQTT. Tratto da [20]

consecutivi per MC_i.

Definiamo un messaggio generato da un LDS come rilevante per MC_i se sussiste un:

1. $t_s \in T_i$, cioè l'argomento dell'LDS è di interesse per MC_i

2. $P_i \in g_s$, cioè MC_i si trova nella regione di geofence di LDS_s

Qui, l'operatore $\in$ denota l'inclusione spaziale di un punto 2D all'interno di una regione 2D.

Dato che gli MC condividono continuamente le loro posizioni GPS con l'SC, sorgono preoccupazioni sulla privacy. L'SC potrebbe potenzialmente associare un id utente alla posizione attuale o tracciare le traiettorie degli utenti nel tempo. A tal proposito, gli studi di F. Montori et al. hanno sviluppato anche meccanismi di privacy che consentano agli MC di alterare le loro posizioni GPS prima di trasmetterle al SC, senza influire significativamente sulle prestazioni.

2.2 Architettura Software ed implementazione

Nel tentativo di superare la sfida delineata nella sezione precedente, F. Montori et al. propongono un protocollo di pubblicazione-sottoscrizione denominato LA-MQTT, un'estensione del noto protocollo MQTT progettata per supportare le comunicazioni IoT spatial-aware. In particolare, LA-MQTT è stato sviluppato seguendo questi principi:

1. Compatibilità standard, senza apportare modifiche al formato dei messaggi del protocollo MQTT.

2. Compatibilità con il broker, implementando l'estensione come un componente aggiuntivo esterno senza richiedere supporto specifico dal broker.

3. Supporto dettagliato per il geofencing, consentendo la definizione di regioni di geofence di qualsiasi forma, non solo circolari come illustrato in [2].

4. Attenzione verso la privacy, con l'introduzione di meccanismi e algoritmi lato client volti a garantire il miglior equilibrio nel trade-off riguardante la privacy già accennato in precedenza.

L'architettura software di LA-MQTT è rappresentata nella Figura 2.1. Essa è composta da tre componenti principali: un broker MQTT legacy[1], la libreria client LA-MQTT e la libreria backend LA-MQTT. Per quanto riguarda il broker, le operazioni di LA-MQTT sono state testate con il broker Eclipse Mosquitto[2], noto per la sua diffusione nelle comunità di ricerca IoT e la sua licenza open source. Tuttavia, è importante sottolineare che LA-MQTT è

[1]Con "MQTT standard" o "MQTT legacy" indichiamo la versione base di MQTT, priva di estensioni

[2]Uno dei broker open-source più utilizzati (https://mosquitto.org)

broker-agnostic, consentendo un'integrazione senza soluzione di continuità con altri strumenti disponibili sul mercato.

La libreria client LA-MQTT rappresenta uno strato software leggero che offre funzionalità di geo-comunicazione sia per gli MC che per gli LDS, consentendo operazioni come la pubblicazione di una nuova geofence (per gli LDS) o la sottoscrizione ad essa (per gli MC).

Il backend è responsabile dell'elaborazione dei messaggi LA-MQTT e dell'invio delle notifiche spaziali associate a eventi di ingresso in specifiche geofence. Può essere implementato sullo stesso host del broker MQTT o su un host diverso.

2.3 Libreria Client

La libreria client LA-MQTT è utilizzata sia dai dispositivi degli *MC* che dagli *LDS*, poiché permette di eseguire operazioni di pubblicazione-sottoscrizione spatial-aware verso il broker MQTT remoto. Il trasferimento dei dati sottostante è gestito dal protocollo MQTT legacy: non sono state apportate modifiche né ai formati dei messaggi MQTT né alla sequenza dei messaggi. Tuttavia, la libreria client LA-MQTT introduce una nuova API per le comunicazioni basate sulla posizione tramite le primitive di pubblicazione-sottoscrizione MQTT. Ciò è stato implementato definendo, esclusivamente per i messaggi LA-MQTT, quanto segue: (1) una denominazione convenzionale di riferimento per i topic MQTT e (2) una struttura uniforme del payload, codificata mediante il linguaggio JSON. La Tabella 2 riassume l'elenco dei topic e i formati dei payload per ciascuna delle tre principali funzionalità offerte dalla libreria client LA-MQTT:

- **Position Publish:** L'API consente al *MC* di inviare la sua posizione GPS corrente al broker MQTT. L'operazione è mappata su un'azione di pubblicazione MQTT di base, dove il topic è vincolato a una stringa predefinita (`GPS_DATA`), e i valori di latitudine e longitudine sono inclusi

nel payload. Prima di eseguire questa azione, il client può attivare eventuali meccanismi di privacy.

- **Topic Subscription:** L'API consente al MC di sottoscrivere un topic. Di conseguenza, l'MC viene notificato ogni volta che un LDS genera un nuovo contenuto per quel topic, a condizione che siano soddisfatti i vincoli di posizione. L'operazione è intrinsecamente suddivisa in due azioni. La prima è l'azione di sottoscrizione base di MQTT: il topic di interesse è costruito concatenando il topic fornito dall'utente, un id del MC e un prefisso predefinito. In questo modo, LA-MQTT gestisce sottoscrizioni private per ogni MC e topic di interesse; questa scelta è giustificata dalla necessità di fornire una diffusione filtrata dei messaggi IoT verso quei MC che soddisfano i requisiti basati sulla posizione. Indichiamo con $C(i, t_s)$ il nuovo argomento creato per MC_i e LDS_s. La seconda azione, oltre alla sottoscrizione MQTT, è una richiesta di pubblicazione MQTT emessa da MC_i. Tale richiesta è emessa con il topic predefinito `MC_SUB` e il payload $\langle t_s, i \rangle$: lo scopo è notificare al backend la sottoscrizione eseguita da MC_i.

- **Geofence Publish:** L'API consente all'LDS di pubblicare messaggi che devono essere recapitati a tutti gli MC interessati situati all'interno di una geofence target. Il topic MQTT è vincolato a una stringa predefinita (`GEOFENCE_DATA`), mentre il payload è composto da quattro campi: l'argomento LDS t_s, il contenuto pubblicato c_s, la geofence target g_s e un evento di mobilità e_s. Il contenuto pubblicato è il contenuto dipendente dall'applicazione (c_s) che deve essere alla fine ricevuto dai MC (ad esempio, rapporto sulla disponibilità di posti auto rispetto all'esempio in Figura 1.2, includendo anche la posizione del posto auto). La geofence g_s deve essere codificata attraverso il formato GEOJSON RFC 7946[3]. Anche se abbiamo affermato che un topic è rilevante per un MC quando quest'ultimo si trova all'interno della relativa geofence,

[3]https://geojson.org/

possiamo immaginare una relazione spaziale più complessa. Per riflettere ciò, aggiungiamo l'evento di mobilità e_s che cattura la condizione in base alla quale la notifica deve essere inviata. Attualmente, sono supportate due opzioni di base $e_s \in \{\text{IN} \mid \text{OUT}\}$, a seconda che il MC sia all'interno o all'esterno della regione di geofence.

Tabella 2.1: Operazioni Publish-subscribe di LA-MQTT

API	Soggetto	MQTT OP	Topic	Payload
Position Publish	MC	Publish	GPS_DATA	{position: P_i}
Topic Subscription	MC	Subscribe	$C(i, t_s)$	*
	MC	Publish	MC_SUB	{mc: i, topic: t_s}
Geofence Publish	LDS	Publish	GEOFENCE_DATA	{topic: t_s, content: c_s, region: g_s, event: e_s}
Content publish	Backend	Publish	$C(i, t_s)$	{content: c_s}

2.4 Libreria Backend

Il backend LA-MQTT è eseguito come processo in background, sulla stessa macchina del broker MQTT o su una macchina separata. All'avvio del sistema, la libreria di backend si iscrive al topic `GPS_DATA`, ricevendo notifiche ogni volta che un nuovo aggiornamento della posizione GPS viene trasmesso da qualsiasi MC_i. L'informazione $\langle i, P_i \rangle$ viene memorizzata su un database MongoDB. Allo stesso modo, si iscrive al topic `GEOFENCE_DATA`, venendo notificata ogni volta che un nuovo messaggio è inviato da qualsiasi LDS_s. Allo stesso modo, l'informazione $\langle t_s, g_s, c_s, e_s \rangle$ viene immagazzinata su un database MongoDB.

Per identificare l'elenco dei potenziali destinatari per ciascun messaggio generato da LDS_s, il backend necessita di conoscere T_i, ossia l'elenco delle iscrizioni di ciascun MC_i. Tuttavia, il broker MQTT potrebbe non rendere disponibili tali informazioni tramite un'API. Per ovviare a ciò, il backend si sottoscrive al canale `MC_SUB`, ricevendo notifiche ogni volta che MC_i si registra al topic t_s.

All'interno del backend è incluso un modulo di geoprocessing, attivato ad ogni pubblicazione di un nuovo `GPS_DATA` da qualsiasi MC_i, che verifica se il vincolo spaziale dell'evento è rispettato (ad esempio, $P_i \sqsubset g_s$ se $e_s = \text{IN}$) e $t_s \in T_i$. Se entrambe le condizioni sono soddisfatte, il backend pubblica un nuovo messaggio sul topic MQTT privato $C(i, t_s)$, utilizzando c_s come payload MQTT. Inoltre, un processo simile è eseguito quando un LDS_s pubblica un nuovo contenuto c_s: in questo caso, il modulo itera su tutti i MC_j con $0 < j \leq M$, pubblicando un nuovo messaggio con payload c_s su tutti i canali $C(j, t_s)$ per cui $t_s \in T_j$ e la condizione e_s è verificata.

Infine, è da notare che un MC potrebbe spostarsi all'interno di una geofence g_s; tuttavia, dovrebbe ricevere una sola notifica per ogni contenuto c_s prodotto da LDS_s. Per affrontare ciò, il backend include un modulo di caching che tiene traccia della storia delle notifiche inviate da LDS_s a MC_i, implementato attraverso la memorizzazione di:

- un numero progressivo N_s, che conteggia i messaggi `GEOFENCE_DATA` prodotti da LDS_s;
- una struttura dati di cache $\langle N_s(i), i \rangle$, dove $N_s(i)$ rappresenta il numero progressivo dell'ultimo messaggio prodotto da LDS_s e consegnato a MC_i.

Durante l'elaborazione di un aggiornamento GPS da MC_i, il modulo verifica che $N_s(i)$ sia inferiore a N_s; solo in tal caso, a condizione che $t_s \in T_i$ e $P_i \sqsubset g_s$, il messaggio viene pubblicato sul broker MQTT, con topic $C(i, t_s)$ e payload c_s.

2.5 Considerazioni parziali

LA-MQTT presenta inoltre un modulo dedito alla privacy, il quale fa parte delle librerie client e viene utilizzato nel momento in cui un MC compie un'azione di publish position. Essendo la parte relativa alla privacy molto interessante ma non al fine delle soluzioni proposte in questa tesi, rimando a

[20] per una descrizione dettagliata della parte in oggetto. A questo punto, le conoscenze di base necessarie per capire a pieno le seguenti fasi che hanno portato alla soluzione proposta in questa tesi sono state esposte e dovrebbero essere state acquisite dal lettore. Dunque, nei prossimi capitoli, vedremo quali sono state le fasi di sviluppo successive che hanno permesso ad LA-MQTT di operare in un contesto multi-broker.

Capitolo 3

LA-MQTT in un contesto multi-broker

3.1 L'idea iniziale e la progettazione

I presupposti iniziali sono quelli aumentare l'efficienza e di conseguenza la scalabilità di LA-MQTT rendendo LA-MQTT multi-broker. Per fare ciò inizialmente ci si è quindi interrogati sulla fattibilità del tutto. L'intero processo, dall'idea iniziale a quella infine adottata, si può riassumere nell'analisi progressiva di tre scenari.

3.1.1 Scenario 1: l'utilità del bridge

Inizialmente si è pensato a quale delle varie entità di LA-MQTT dovesse scalare assieme al numero di broker per rendere l'aggiunta della componente multi-broker sensata ed efficiente. Come è possibile notare nella Fig. 3.1, il primo scenario preso in analisi, propone un contesto multi-broker in cui l'unica entità del protocollo LA-MQTT a scalare era per l'appunto il broker. Questa soluzione ha una criticità principale: il pontenziale rischio di sovraccarico dell'entità Backend, il quale rimane un pericolo, si sopprime solamente il rischio di un sovraccarico delle entità Broker.

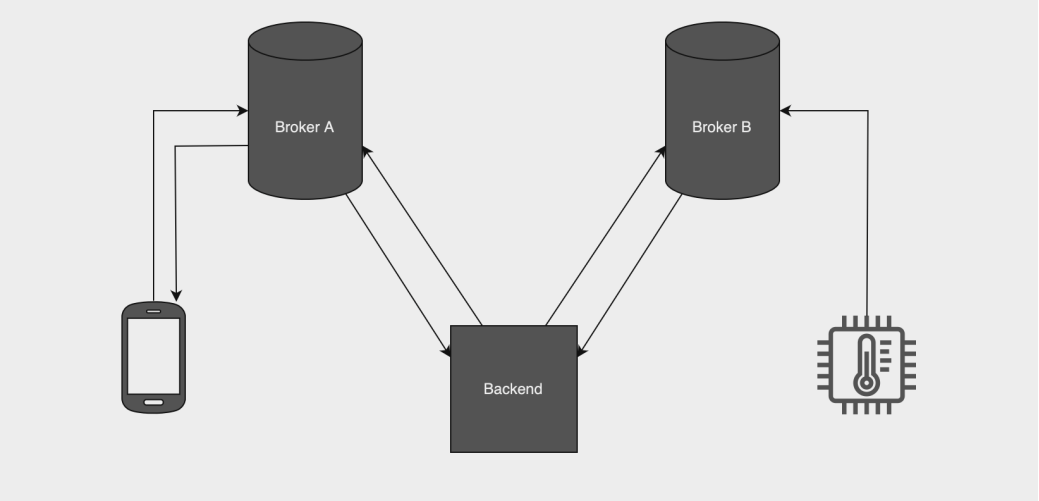

Figura 3.1: Scenario n.1

Le considerazioni del paragrafo precedente portano a doversi interrogare su quali entità del protocollo debbano scalare assieme al numero di Broker presenti. La soluzione, a questo punto ovvia, è quella di far scalare il numero di Backend assieme a quello dei Broker. In particolare per evitare modifiche radicali di LA-MQTT, la soluzione che sembra più opportuna è quella di far scalare il numero di Backend in rapporto 1:1 con i Broker in modo tale da mantenere invariate le librerie facendo corrispondere ad ogni Broker un rispettivo Backend il quale si preoccuperà di svolgere il lavoro inerente al suo Broker di competenza.

Il problema rimanente è la comunicazione tra Broker differenti. Quest'ultima, nello scenario 1, era basata sull'unicità dell'entità Backend e sulle sue molteplici connessioni con la totalità dei Broker. C'è bisogno, quindi, di un gateway per lo scambio di messaggi tra broker differenti. Fortunatamente a questo problema, ci viene in soccorso una feature implementata dalla maggior parte dei broker presenti sul mercato: il Bridge.

3.1.2 Scenario 2: distribuzione del lavoro

Come detto nella precedente sezione la soluzione per la comunicazione tra diversi Broker risiede nella feature dei Bridge. In MQTT, un bridge è un componente che consente di collegare due broker MQTT tra loro. I bridge MQTT sono utilizzati per estendere la copertura di una rete MQTT o per collegare due reti MQTT separate. Il concetto di bridge in MQTT è simile a quello di un ponte in una rete di computer, poiché il suo ruolo è facilitare la comunicazione tra due segmenti di rete separati. I bridge MQTT consentono a dispositivi in una rete MQTT di comunicare con dispositivi in un'altra rete MQTT, anche se le due reti sono fisicamente distanti o gestite da broker MQTT separati. Un bridge MQTT riceve i messaggi da un broker e li inoltra a un altro broker, consentendo così la comunicazione tra i due. Questo è utile in scenari, come il nostro, in cui si hanno diverse istanze di broker MQTT che devono essere collegate per permettere la comunicazione tra dispositivi connessi a broker diversi.

Per configurare un bridge MQTT, è necessario specificare i dettagli del broker di destinazione (indirizzo IP, porta, ecc.) e stabilire le regole di filtraggio per determinare quali messaggi devono essere inoltrati attraverso il bridge. La configurazione può variare a seconda dell'implementazione specifica del broker MQTT utilizzato. Per avere idea di una configurazione tipo di un Bridge MQTT si può fare riferimento alla configurazione di uno dei Broker MQTT più diffusi sul mercato: Eclipse Mosquitto[1].

[1]Guida alla configurazione di un Broker Eclipse Mosquitto: https://mosquitto.org/man/mosquitto-conf-5.html

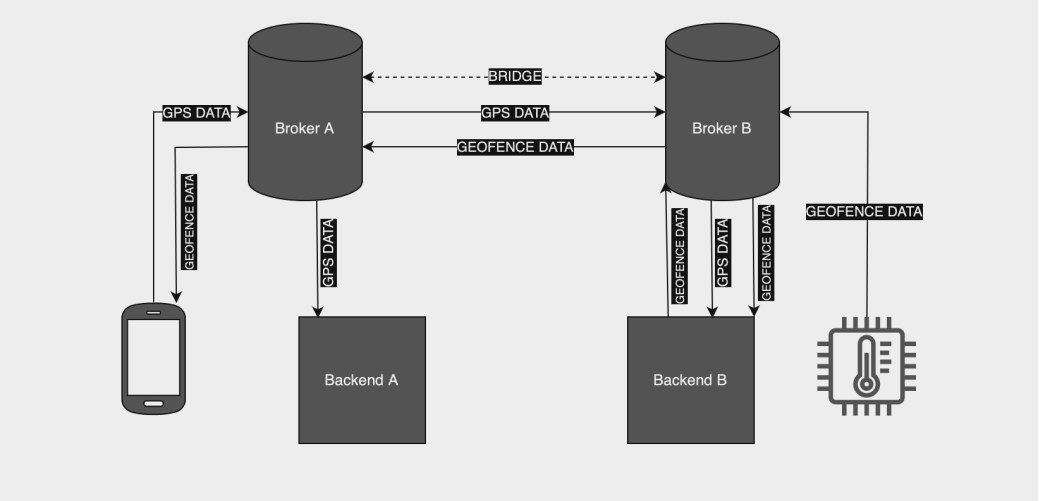

Figura 3.2: Scenario n.2

L'utilizzo di un Bridge risolve il nostro principale problema di comunicazione tra due Broker ma ci pone un altro interrogativo: come distribuire il lavoro di geoprocessing tra le varie istanze dei Backend e di conseguenza quali sono le informazioni da far passare nel collegamento del Bridge?

Prima di arrivare allo scenario finale, poi adottato nella soluzione proposta in questa tesi, si è passati per uno scenario spinto dalla domanda precedente, ovvero lo scenario illustrato nella Fig. 3.2. Nello scenario in figura si può notare che il peso del geoprocessing è spostato sul Backend del Broker che riceve i dati dall'LDS. Questo porterebbe a due problemi:

1. La distribuzione del lavoro di geoprocessing tra le varie istanze dei Backend è meno equa. Il Backend che riceverà più dati dagli LDS avrà sempre una quantità di lavoro maggiore degli altri.

2. Il passaggio di un dato in più sul collegamento del Bridge, ovvero la posizione dell'utente.

Questi problemi hanno portato all'elaborazione dello scenario finale, poi adottato come soluzione.

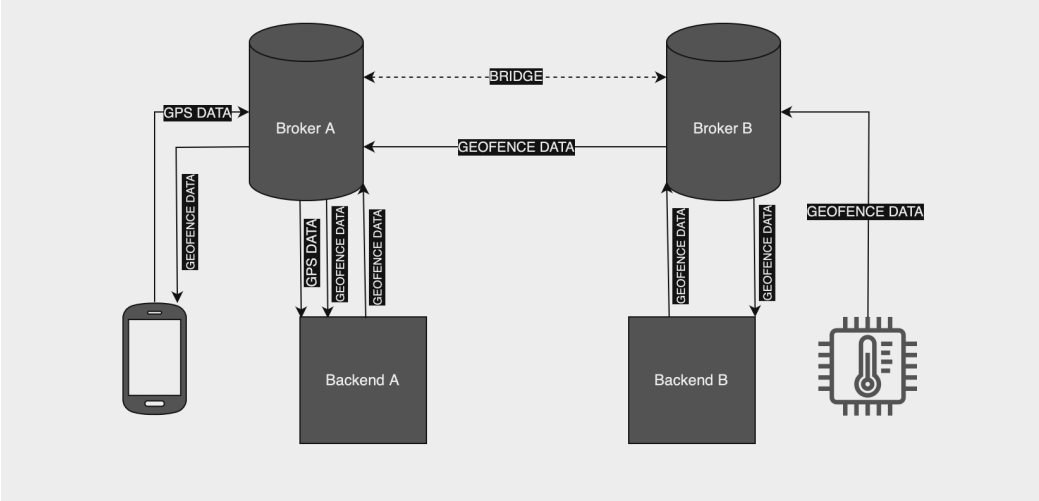

Figura 3.3: Scenario n.3

3.1.3 Scenario 3: soluzione finale

I problemi precedenti hanno portato a spostare il carico di lavoro di geo-processing sull'istanza del Backend competente del Broker al quale l'MC è connesso. Questo porta al dover passare sul bridge solamente l'eventuale dato prodotto da un LDS. Inoltre si avrà una situazione in cui le istanze dei Backend avranno una quantità di lavoro proporzionale ai dispositivi collegati al loro Broker di competenza. Lo scenario dell'architettura utilizzata nella soluzione finale è quello nel diagramma illustrato in Fig. 3.3

3.2 Implementazione di LA-MQTT multi-broker

L'implementazione di LA-MQTT multi-broker ha come base le librerie di LA-MQTT single-broker. Le modifiche che sono state fatte per permettere ad LA-MQTT di operare in un contesto multi-broker sono diverse. La parte di libreria Client per gli MC non ha subito modifiche a differenza di quella per gli LDS. Anche la libreria Backend ha subito diversi cambiamenti. In questa sezione verranno quindi esposte le varie modifiche e i vari dettagli implementativi senza tralasciare come è stato gestito l'utilizzo dei bridge.

3.2.1 Bridge

Per spiegare l'impostazione del sistema di bridging verrà fatto riferimento alla configurazione del broker Eclipse Mosquitto. La configurazione del Broker prevede la dichiarazione del Bridge, all'interno della quale bisognerà definire alcuni campi come l'indirizzo del broker con il quale si vuole che la connessione venga stabilita. È estremamente consigliato settare il campo `try_private` a `true`, è consigliato nella guida alla configurazione di mosquitto ed aiuterebbe i Broker a distinguere i messaggi provenienti da dei Bridge in modo da prevenire la creazione di eventuali loop. Analogamente se si utilizza qualsiasi altro Broker sul mercato è fortemente consigliato impostarlo in modo da avere qualche forma di anti-looping, in quanto la formazione di loop è uno dei rischi maggiori quando si hanno dei Broker connessi in mesh.

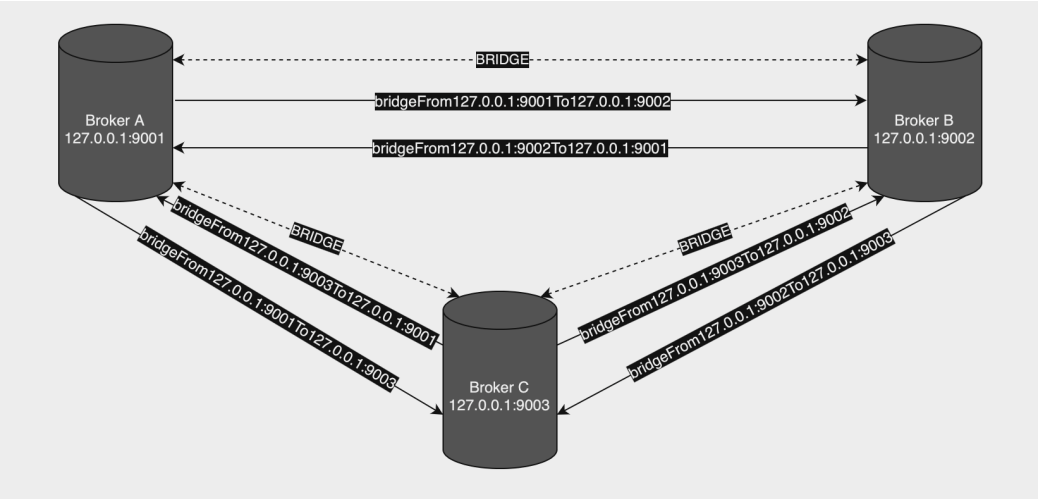

Figura 3.4: Bridging tra Broker

Un passo molto importante nella dichiarazione del bridge è il filtraggio dei messaggi. È possibile filtrare i messaggi in base al topic ed in base alla provenienza del messaggio. In ogni dichiarazione di Bridge bisogna quindi indicare un topic per i messaggi in entrati ed un topic per i messaggi in uscita. I topic in entrata ed in uscita devono avere la seguente forma: bridgeFrom<Indirizzo ip broker di partenza>To<Indirizzo ip broker di arrivo>. Nel diagramma il-

lustrato nella Fig. 3.4 è possibile vedere come è gestito il bridging di tre Broker, attraverso l'orientamento delle frecce è inoltre possibile capire se il topic sia in entrata o in uscita.

3.2.2 Libreria Client: LDS

Le modifiche che sono state apportate alla libreria Client, in particolare alla parte utilizzata dagli LDS, riguardano il payload dei messaggi e i valori delle variabili passate allo startup dell'LDS.

Per quanto riguarda il payload, sono stati aggiunti solamente tre campi, `notifiedBrokers`, `bounce` e `maxBounces`. Il campo `notifiedBrokers` è un array contenente tutti gli ip dei Broker che hanno già ricevuto il messaggio. I campi `bounce` e `maxBounces` sono utilizzati per regolare e limitare la propagazione del messaggio all'interno della rete di Broker.

I valori passati al momento dello startup all'LDS oltre a differire di caso d'uso in caso d'uso come è possibile osservare nel Cap.4, devono contenere alcuni valori necessari ai fini del funzionamento di LA-MQTT multi-broker. Bisognerà fornire l'ip del Broker a cui l'LDS deve collegarsi e il valore che si vuole assegnare alla variabile `maxBounces` che corrisponde al numero massimo di volte che il messaggio verrà inoltrato.

3.2.3 Libreria Backend

L'implementazione multi-broker per la libreria Backend consiste in varie modifiche le quali adempiono al nuovo compito del Backend, ovvero quello di inoltrare eventuali messaggi provenienti dagli LDS verso i topic in uscita dei vari Bridge.

In questa nuova estensione del protocollo LA-MQTT si assume che al Backend nel momento dello startup vengano fornite diverse informazioni. In particolare, deve essere fornito: l'indirizzo ip del Broker e il path di un file json di configurazione. Il file json di configurazione è un file, aperto ad espansioni future, che contiene un array, `bridgedBrokers`, il quale al suo interno ospita

gli indirizzi ip di tutti i Broker con i quali il Broker corrispondente al Backend che si sta avviando ha un Bridge.

All'arrivo di un geofence update da un LDS o da un Bridge il Backend controlla il valore della variabile `bounce` e se `bounce` $<$ `maxBounces`, procede ad inoltrare il messaggio verso i Broker adiacenti (ovvero i Broker collegati tramite Bridge) a meno che l'indirizzo ip del Broker adiacente verso cui si sta inoltrando il messaggio non sia già presente nell'array `notifiedBrokers`. Prima dell'eventuale inoltro, viene aggiornato il payload del messaggio, incrementando il valore di `bounce` e aggiungendo all'array `notifiedBrokers` l'indirizzo ip dei Broker adiacenti verso i quali si sta inoltrando il messaggio.

Capitolo 4

Un esempio di caso d'uso: JAM

4.1 Cosa è JAM?

Figura 4.1: Logo di JAM

Nel contesto delle città sempre più intelligenti e connesse, l'esigenza di ottimizzare la gestione del traffico e dei parcheggi diventa sempre più rilevante. Il progetto "JAM" nasce con l'obiettivo di affrontare questa sfida, offrendo una soluzione innovativa ed efficiente per la ricerca di parcheggi in tempo reale nel contesto urbano di una smart city.

JAM è quindi un app di parking assistance che sfrutta il protocollo MQTT, ed in particolare utilizza le librerie LA-MQTT multi-broker. JAM si distingue per la sua capacità di visualizzare la mappa dettagliata della città, evidenziando mediante segmenti colorati lo stato dei parcheggi nelle diverse strade. Inoltre, l'app consente agli utenti di ricevere notifiche in-app

relative alle strade di loro interesse, rendendo l'esperienza di parcheggio più personalizzata e efficiente.

JAM utilizza il protocollo MQTT per lo scambio di aggiornamenti sulla disponibilità di parcheggi nelle varie strade urbane. Nello specifico JAM utilizza la versione di LA-MQTT multi-broker, proposta come soluzione in questa tesi.

4.2 Gli obiettivi di JAM: città più smart e green

Oltre a semplificare radicalmente l'esperienza di trovare un parcheggio in città, l'app di JAM si pone due obiettivi fondamentali. In primo luogo, mira a semplificare la vita quotidiana dei cittadini, offrendo loro strumenti intuitivi e pratici per gestire il processo di parcheggio. In secondo luogo, e non meno importante, JAM si propone di contribuire attivamente alla riduzione del traffico stradale urbano, riducendo il numero di veicoli in circolazione alla ricerca di parcheggio. Questo approccio non solo migliora l'efficienza complessiva del sistema di trasporto urbano, ma contribuisce anche a una riduzione significativa dell'inquinamento ambientale, promuovendo un ambiente più sano e sostenibile.

In conclusione, il progetto "JAM" rappresenta una risposta concreta e avanzata alle sfide legate alla gestione del traffico e dei parcheggi nelle smart cities. JAM si inserisce come un elemento chiave nella trasformazione delle città verso modelli più intelligenti, efficienti e sostenibili.

4.3 Showcase

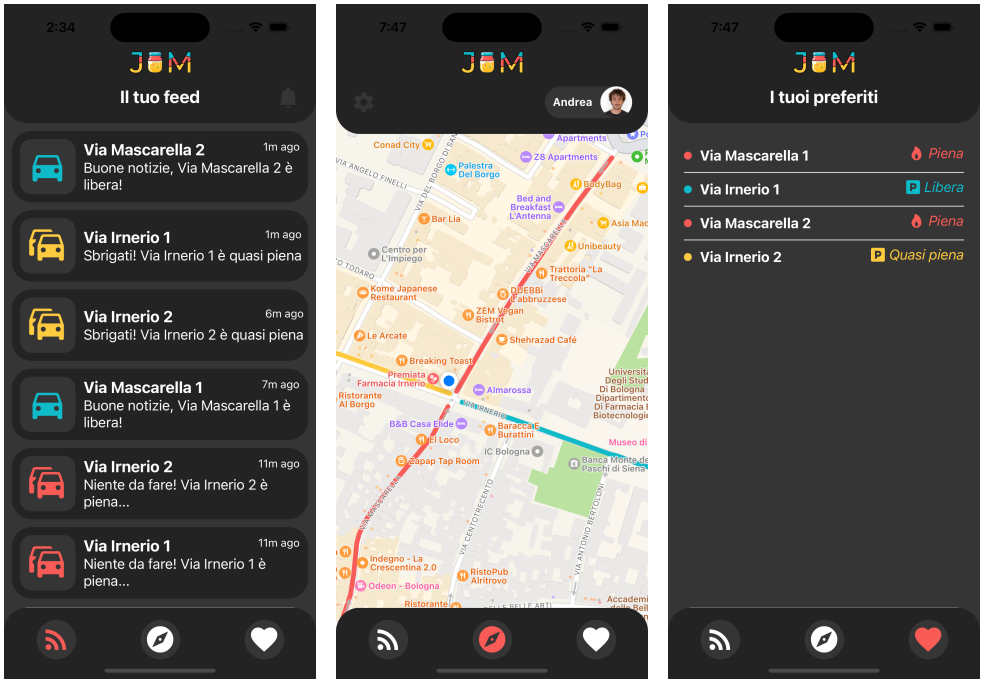

Figura 4.2: Tab principali di JAM

L'applicazione si articola in tre tab (Fig. 4.2): Feed, Explore e Favourites. La tab principale, Explore, ci permette di visualizzare la mappa centrata sulla nostra posizione con all'interno dei segmenti di strada colorati in base alla disponibilità di parcheggi. Alla pressione di un determinato segmento si aprirà una finestra modale attraverso la quale possiamo aggiungere il segmento ai preferiti per raggiungerlo facilmente e ricevere notifiche sulle sue variazioni di disponibilità. La Tab contenente i preferiti, Favourites, ci permette di visualizzare lo stato di tutti i nostri segmenti di strada preferiti e alla pressione di uno di essi accederemo alla schermata dedicata ad uno specifico segmento dove potremo visualizzare il segmento sulla mappa e potremo rimuoverlo dai preferiti. Infine la Tab Feed, ci permette di visualizzare le notifiche in-app relative alle variazioni della disponibilità di parcheggi dei nostri preferiti. All'interno di quest'ultima schermata è possibile anche decidere se abilitare o meno le Push Notification.

Inoltre l'app è disponibile in inglese ed italiano ed offre un servizio di autenticazione. Altre immagini relative all'app e al suo sviluppo sono presenti in appendice.

4.4 Sviluppo

4.4.1 Tecnologie utilizzate

Per lo sviluppo dell'app, che si distingue per la sua natura ibrida, sono stati utilizzati React Native e Expo. La scelta di React Native ha garantito un notevole vantaggio in termini di efficienza dello sviluppo, riducendo i tempi di sviluppo complessivi.

In aggiunta, l'utilizzo di Expo ha ulteriormente semplificato il processo di sviluppo. Expo è un framework basato su React Native che fornisce una serie di strumenti e servizi che semplificano la gestione delle risorse, la distribuzione e il testing dell'applicazione. Questa scelta si è rivelata cruciale per la creazione di un'applicazione ibrida che possa garantire una user experience uniforme su entrambe le piattaforme, con un accesso agevole alle funzionalità native di ciascun sistema operativo.

4.4.2 LA-MQTT multi-broker in JAM

Lato app (MC)

All'avvio dell'applicazione una volta effettuata l'autenticazione viene inizializzato lo `SpatialMQTTClient`, viene effettuata la connessione al Broker competente e viene utilizzato il metodo per MC `subscribeGeofence` per effettuare l'operazione subscribe di MQTT. Come argomenti nella funzione `subscribeGeofence` oltre al nome del topic (parking) viene passato un `MQTTReceiver`, o meglio, un oggetto di una classe che implementa l'interfaccia `MQTTReceiver`.

La classe che implementa l'interfaccia MQTTReceiver in JAM ha vari compiti. A grandi linee, la classe si occupa di contenere al suo interno diversi

React State e diverse funzioni per settare i relativi state. La maggior parte degli state fa poi parte di un React Context, il quale contenendo la totalità dei React Component dell'app fornisce a quest'ultimi i React State, che vengono poi utilizzati da diversi componenti all'interno dell'app attraverso l'hook `useContext` di react.

Uno dei React State di maggiore importanza per l'app e per lo sfruttamento del protocollo è `streetInfos`, un dizionario che ha come chiavi i nomi dei segmenti di strada e come valori, degli oggetti contenenti informazioni sul relativo segmento di strada, estratte dal payload del messaggio contenente l'aggiornamento dello status del segmento.

All'arrivo di un messaggio il receiver controlla che una delle seguenti situazioni sia vera

1. Il messaggio contiene un cambiamento effettivo dello status di un segmento di strada già esplorato, quindi già presente nel dizionario `streetInfos`.

2. Il messaggio contiene lo status riguardante un segmento di strada di cui siamo appena entrati nella geofence, quindi non esplorato e non presente nel dizionario `streetInfos`.

Nel primo caso viene aggiunta una nuova entry a `streetInfos` mentre nel secondo caso viene aggiornata la entry già presente. L'utente in entrambi i casi viene notificato con una Push Notification (se abilitate) e con una notifica in-app che si andrà ad aggiungere nel tab *Feed*. È da sottolineare che queste modifiche vengono fatte attraverso il rispettivo setter del React State in modo tale da propagarle attraverso il React Context per tutti i React Component dell'app e triggerare eventuali `useEffect` che hanno come *deps* uno di questi React State.

L'app manda la posizione dell'utente al Broker ogni 20 secondi, così che il Backend possa effettuare le sue azioni di geoprocessing ed eventualmente mandare nuovi messaggi all'app (MC).

Il payload dei messaggi contiene come content una stringa che ospita varie informazioni separate da un token (il carattere "-"), queste informazioni sono rispettivamente:

- `streetName`: il nome del segmento di strada
- `latStart`: latitudine di inizio del segmento
- `lngStart`: longitudine di inizio del segmento
- `latEnd`: latitudine di fine del segmento
- `lngEnd`: longitudine di fine del segmento
- `status`: status della disponibilità di parcheggi del segmento. I possibili status sono:
  - full
  - almost full
  - free

La schermata Home ed il suo rispettivo ed omonimo React Component permettono all'utente di visualizzare la mapview centrata sulla sua attuale posizione. Il React Component recupera, attraverso il React State `streetInfos`, le varie informazioni sui vari segmenti di strada e i loro status. Prosegue poi, creando delle Polylines nella mapview in modo indiretto attraverso il React Component MapViewDirections del package react-native-maps-directions. Quest'ultimo attraverso le coordinate di inizio e di fine del segmento è in grado di tracciare una Polyline sulla mappa la quale viene colorata in base allo status della via e segue il path definito dall'API directions di google-maps tra il punto di inizio e di fine.

Lato LDS

L'LDS allo startup tramite lo script npm `start` richiede, oltre all'indirizzo ip del Broker a cui collegare LDS, anche un file json contenente alcune

informazioni sul rispettivo segmento di strada che l'LDS andrà a coprire. Il file json di configurazione deve contenere le seguenti informazioni:

- `streetName`
- `latCenter`
- `lngCenter`
- `latStart`
- `lngStart`
- `latEnd`
- `lngEnd`

La descrizione di ognuno di questi campi è identica a quella già fornita nella sezione precedente nella descrizione del formato del content del payload. Abbiamo però due informazioni aggiuntive, ovvero `latCenter` ed `lngCenter`. Quest'ultime rappresentano le coordinate del punto centrale del segmento di strada e sono utilizzate nel geoprocessing.

4.5 Possibili sviluppi

Una parte mancante dello sviluppo, in quanto strettamente legata al territorio, è la parte riguardante il trigger effettivo di un `geofenceUpdate`. L'LDS è tendenzialmente un SBC (Single Board Computer) collegato a diversi sensori. Sorge quindi il bisogno di dover scegliere quale sia la miglior tipologia di sensori a cui affidarsi per il rilevamento della presenza o meno di un autoveicolo nello stallo del parcheggio.

C'è però da considerare il contesto territoriale in cui si opera. Molti lavori a riguardo propongono sistemi per aree di parcheggio delimitate [6][7], con un entrata ed uscita gestita da un lettore RFID, attivato dall'automobilista attraverso la propria card RFID nel momento dell'entrata e dell'uscita

dal parcheggio. Sistemi come questo sono poco pratici e difficilmente adottabili in un ambiente urbano dove gli stalli sono spesso collocati di fianco alla carreggiata. C'è quindi bisogno di utilizzare un metodo che non si basi sull'interazione dell'automobilista e che sia pratico. Ci sono diverse tipologie di sensori che possono aiutarci a raggiungere il nostro obiettivo, come evidenziato in [4]:

- **Sensore radar**: impiega la tecnologia a onde continue modulate in frequenza per rilevare veicoli parcheggiati e in movimento. La sua adattabilità alle condizioni meteorologiche estreme lo rende ideale per la rilevazione all'aperto a lunga distanza. Tuttavia, la vicinanza di due veicoli può causare una rilevazione errata.

- **Sensore ultrasonico wireless**: utilizza onde ultrasoniche ad alta frequenza ed è alimentato da una rete wireless. È indicato per parcheggi su larga scala, superando i problemi di disagio associati all'ultrasuono udibile e consentendo una misurazione ottimizzata grazie alla sua alta direzionalità. Tuttavia, è consigliato per parcheggi al chiuso a causa della sua suscettibilità alle condizioni meteorologiche.

- **Sensore magnetometrico wireless**: rileva cambiamenti nel campo magnetico causati dall'ingresso di veicoli, eliminando la necessità di una scatola di controllo esterna o cablaggi complessi. La sua natura piccola e autocontenuta facilita la manutenzione e l'installazione rapida, soprattutto rispetto ai sensori induttivi alimentati elettricamente. La sua flessibilità nell'installazione, la sensibilità ai veicoli di grandi dimensioni e la possibilità di installazione sia sopra che sotto il terreno lo rendono una soluzione economica e adattabile ai vari contesti.

Il magnometro risulta essere il migliore tra i sensori poiché offre un'installazione semplice e una manutenzione rapida. La tecnologia magnetica wireless elimina la necessità di una scatola di controllo esterna o cablaggi complessi, riducendo notevolmente i costi di installazione. Inoltre, la possibilità di

installazione sia sopra che sotto il terreno offre una flessibilità di implementazione che lo rende una soluzione versatile e efficiente, garantendo precisione nella rilevazione dei veicoli.

Capitolo 5

Evaluation

Nel seguente capitolo si andranno ad analizzare dei grafici frutto di test effettuati sul protocollo per vedere come quest'ultimo reagisce quando deve effettuare operazioni di inoltro su dei Bridge. In particolare andremo ad analizzare diverse situazioni aumentando gradualmente il numero di messaggi inoltrati su un Bridge in modo da poter analizzare i vari delay e trarre i limiti del protocollo.

5.1 Metodo di valutazione

Il metodo utilizzato per valutare il protocollo è consistito nell'invio di un numero elevato di messaggi su un singolo Broker per vedere come quest'ultimo reagisse nell'inoltro eventuale di questi messaggi. Sono stati quindi inviati 100000 messaggi ad un Broker ed è stata poi fatta variare la percentuale di messaggi da inoltrare(*bridged*) e da non inoltrare(*not bridged*). Le due tipologie di messaggi sono state mandate in modo misto ma distribuite equamente all'interno dei 100000 messaggi finali.

I messaggi *bridged* sono stati mandati attraverso la funzione `geofenceUpdate` mentre i messaggi *not bridged* sono stati mandati attraverso la funzione `publicPosition`. La scelta di utilizzare funzioni della libreria LA-MQTT

e non funzioni apposite per la misurazione è stata fatta per valutare il protocollo avvicinandosi il più possibile ad una situazione reale.

All'interno del payload del messaggio mandato dall'LDS è stato inserito un timestamp del momento dell'invio. Il timestamp è poi stato utilizzato per calcolare il delay finale di ogni messaggio.

I test sono stati effettuati con il Broker Eclipse Mosquitto e i messaggi sono stati dilazionati di $1ms$ l'uno dall'altro per evitare che il Broker inizi a "droppare" messaggi o che, data l'implementazione interna del Broker, quest'ultimo aspetti prima di mandare i messaggi verso dei bridge e dia priorità ai messaggi da non inoltrare causando un delay dei messaggi "bridged" iniziali di $\sim 20ms$.

Infine è da notare che i messaggi sono stati considerati come ricevuti una volta raggiunto il Backend di competenza. Il tempo di inoltro del messaggio dal Backend all'MC è stato quindi omesso dal calcolo del delay di ogni messaggio, questo perché è direttamente proporzionale al numero di MC connessi al Broker e non è rilevante ai fini della nostra analisi e della soluzione proposta in questa tesi.

5.2 Analisi e valutazione dei risultati

I test sono stati effettuati con le seguenti percentuali di messaggi *bridged*: 20, 40, 60, 80. Per iniziare l'analisi si può partire dai risultati ottenuti con la percentuale più bassa analizzata, ovvero pari al 20%. Visualizzando il boxplot presente in Fig. 5.1 si può intuire la distribuzione dei dati risultanti dal testing del protocollo. I dati sono distribuiti su un range abbastanza grande e per questo motivo essendo di difficile rappresentazione nel boxplot sull'asse Y è stata utilizzata la scala logaritmica che ci permette di visualizzare al meglio la situazione. Sebbene il range sul quale i nostri dati sono distribuiti è abbastanza ampio si può notare che si ha una maggiore concentrazione sotto i $2ms$. Vedremo successivamente che anche facendo variare la percentuale di messaggi *bridged* la distribuzione dei dati relativi ai nostri risultati

non cambia molto. I dati sopra gli $1.7ms$ vengono classificati come punti individuali(outliers) in quanto hanno meno frequenza all'interno del dataset. Per confermare quest'idea si può semplicemente far affidamento sulla media aritmetica del dataset visualizzabile in rosso nel grafico in Fig. 5.1 in cui si può per l'appunto constatare facilmente che seppure esistano delle misurazioni del delay nel nostro dataset che eccedono di molto, la media di delay è accettabile e si attesta attorno ai $\sim 4.2ms$ per i messaggi *bridged* e $\sim 0.9ms$ per i messaggi *not bridged*.

In conclusione non rimane che cercare il motivo della presenza di outliers. Per farlo basta visualizzare i lineplot in Fig. 5.2 e in Fig. 5.3 i quali illustrano la media mobile del delay con una finestra di 500 misurazioni. Grazie a questi due grafici possiamo notare che gli outliers sono dovuti da picchi periodici del delay, più regolari nel caso dei messaggi *not bridged*, meno regolari nei messaggi *bridged*. Inoltre, possiamo notare che i messaggi *bridged* soffrono di un grande delay nella fase iniziale.

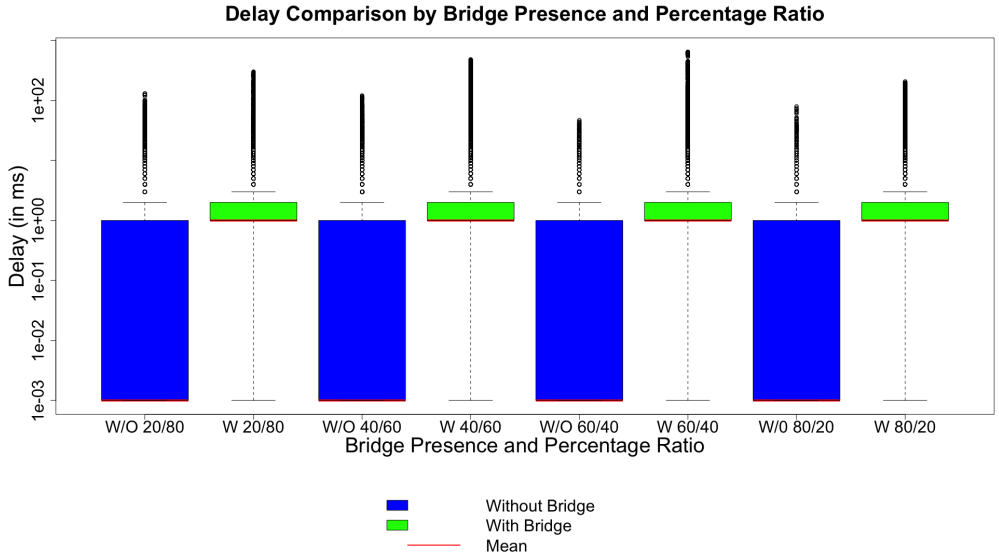

Figura 5.1: Boxplot delay dei messaggi *bridged* e *not bridged* (in scala logaritmica)

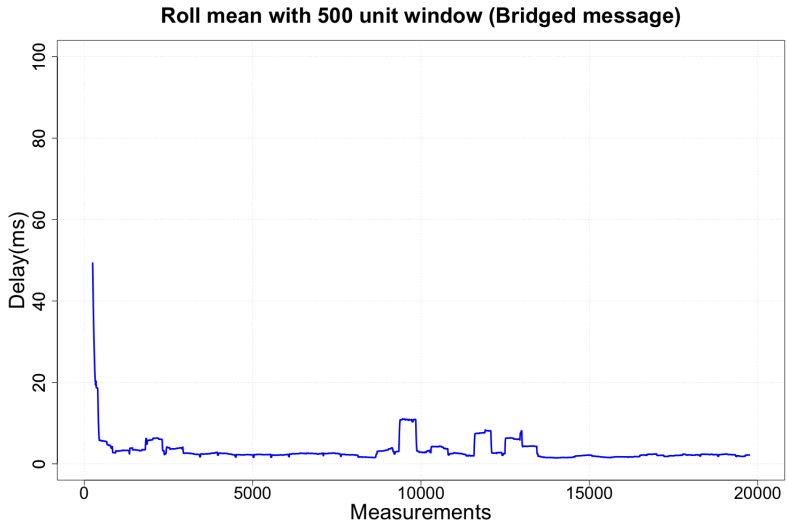

Figura 5.2: Media mobile dei messaggi *bridged* (20%/80%)

5.2.1 Considerazioni finali

Dal grafico illustrato nella Fig. 5.10 si possono trarre interessanti considerazioni finali sulla valutazione del protocollo.

Il trend del delay è proporzionale alla quantità di messaggi nel caso dei messaggi *not bridged*. I messaggi *bridged* contribuiscono ad accrescere il delay totale in modo differente. Possiamo notare che il trend, nel caso di quest'ultima tipologia di messaggi, è crescente in modo regolare e più o meno proporzionale fino a che il rapporto tra i messaggi non raggiunge il 60% *bridged*/40% *not bridged*, per poi calare nel caso del rapporto 80%/20%. I risultati ottenuti nel caso del rapporto 80%/20% sono interessanti, in quanto con test ripetuti il delay tende ad oscillare molto attestandosi sempre maggiore del rapporto 40%/60% e sempre minore del rapporto 60%/40%.

Una delle motivazioni di un delay maggiore nel rapporto 60%/40% è dovuto ad un maggiore picco di delay iniziale come visibile nel grafico in Fig. 5.6. Un altra possibile motivazione è dovuta dai picchi di delay periodici già citati nei paragrafi precedenti ma che in questo caso sono maggiori. Probabilmente

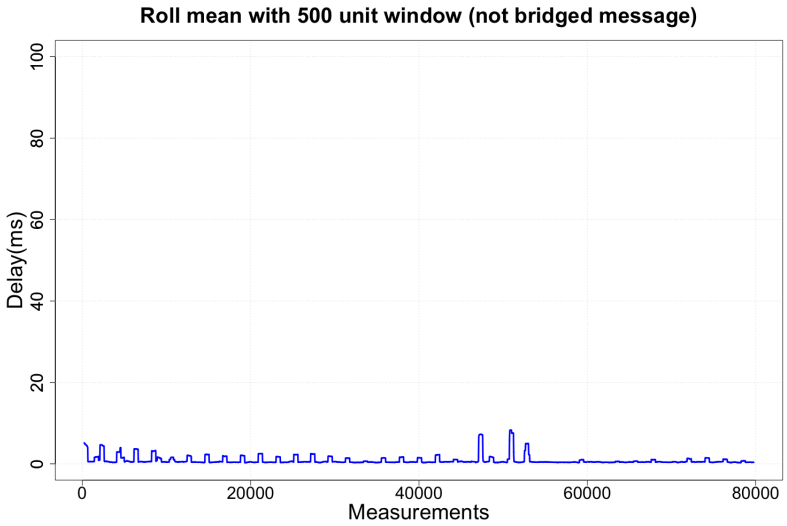

Figura 5.3: Media mobile dei messaggi *not bridged* (20%/80%)

ciò è dovuto dal numero quasi equo di messaggi che porta il Broker a prioritizzare i messaggi *not bridged*. Sebbene ciò non era un problema finché i messaggi *bridged* erano il 40% del totale, inizia a causare problemi quando questa percentuale supera la metà del totale. Il disturbo dato dalla prioritizzazione dei messaggi *not bridged* crea picchi di delay più frequenti mentre annulla quasi del tutto i pochi picchi di delay dei messaggi *not bridged* come illustrato in Fig. 5.7.

La situazione, come già detto migliore quando ci spostiamo su rapporti di percentuali con quasi la totalità di messaggi *bridged*. Possiamo osservare ciò nel rapporto 80%/20% che riporta il delay totale a valori accettabili e presenta meno picchi di delay (Fig. 5.8 e Fig. 5.9).

In conclusione, si deduce che per sviluppi futuri bisognerà riporre attenzione nell'evitare la creazione di situazioni come quella descritta nei paragrafi precedenti. Probabilmente il modo migliore sarebbe fare un tuning più accurato dei parametri del Broker, in quanto agendo direttamente sull'implementazione del Broker per risolvere il problema, si perderebbe la componente broker-agnostic di LA-MQTT.

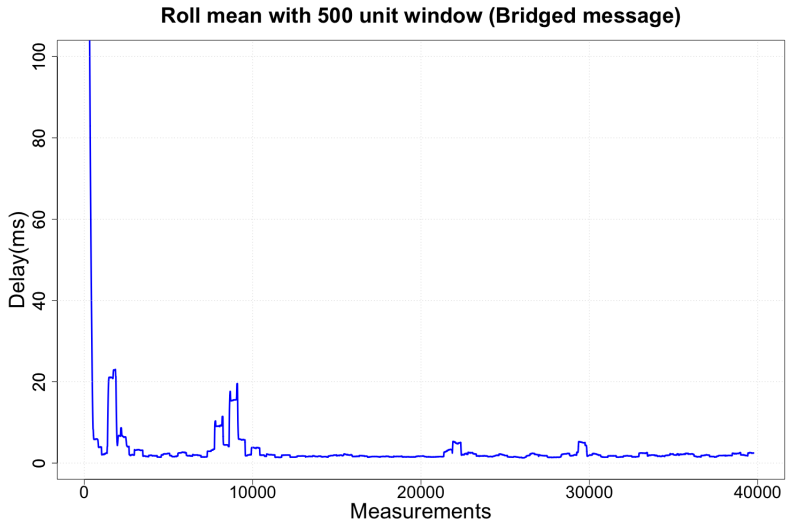

Figura 5.4: Media mobile dei messaggi *bridged* (40%/60%)

Figura 5.5: Media mobile dei messaggi *not bridged* (40%/60%)

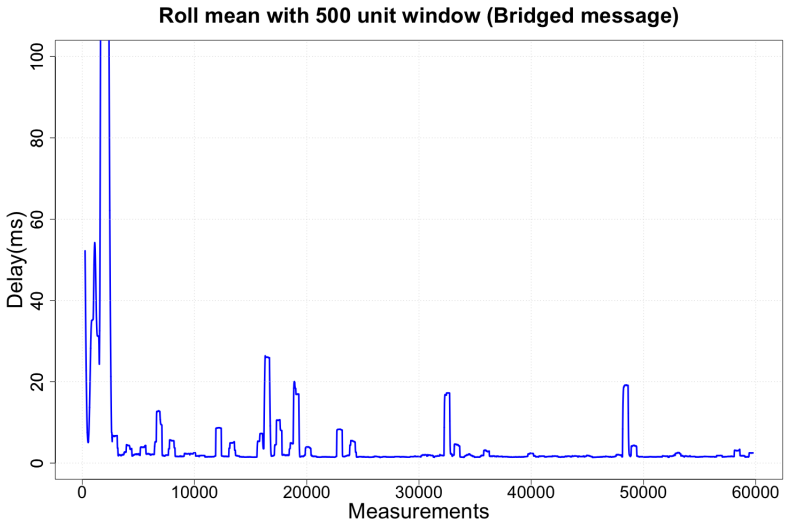

Figura 5.6: Media mobile dei messaggi *bridged* (60%/40%)

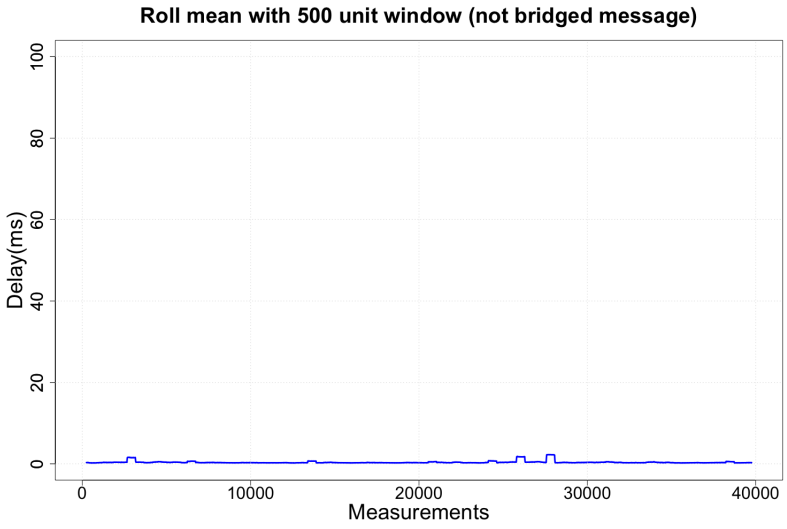

Figura 5.7: Media mobile dei messaggi *not bridged* (60%/40%)

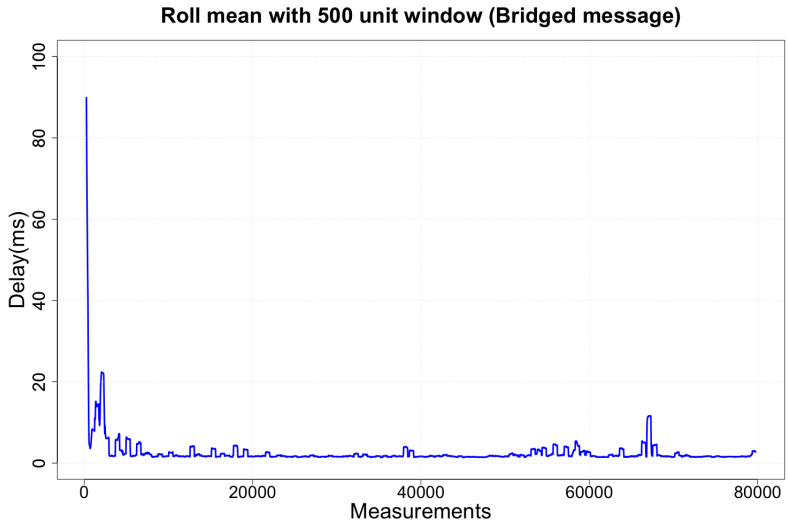

Figura 5.8: Media mobile dei messaggi *bridged* (80%/20%)

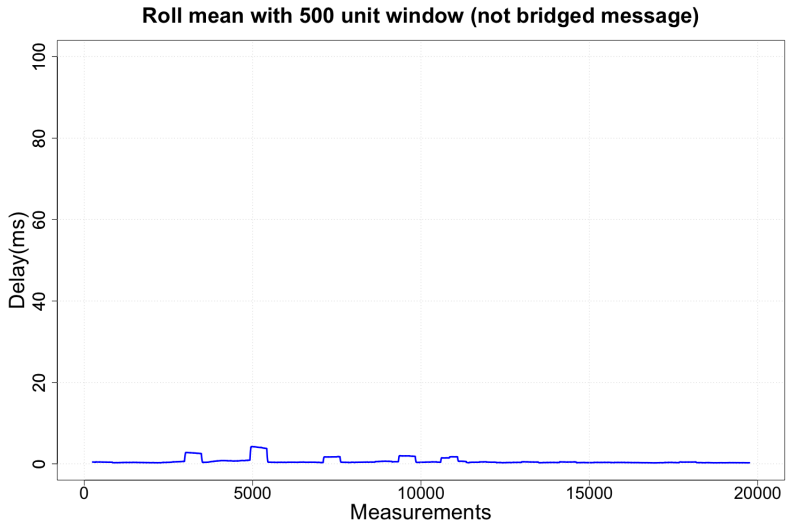

Figura 5.9: Media mobile dei messaggi *not bridged* (80%/20%)

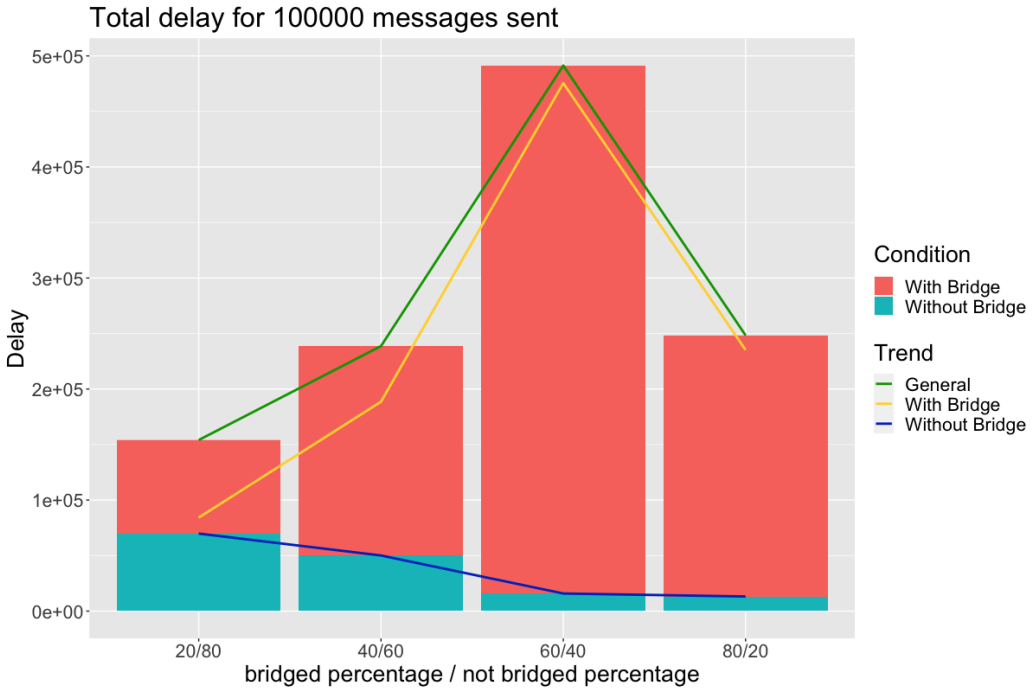

Figura 5.10: Delay totale(in ms) per 100000 messaggi inviati

Conclusioni

In chiusura, questa tesi offre una riflessione approfondita sui risultati emersi durante l'analisi di LA-MQTT standard e nell'ideazione e l'implementazione di LA-MQTT multi-broker. L'evoluzione di MQTT, mirata a introdurre la sensibilità spaziale nei sistemi IoT, ha dimostrato di affrontare le sfide legate alla mancanza di considerazione del contesto geografico.

Il contesto single-broker pur avendo evidenziato il ruolo cruciale della sensibilità spaziale nella comunicazione publish-subscribe di MQTT, non ha saputo rispondere alle sfide di scalabilità.

Nel contesto multi-broker ed in particolare nella soluzione proposta in questa tesi, LA-MQTT multi-broker, si è dimostrato efficace nel migliorare la scalabilità, affrontando problemi di efficienza nella gestione di reti IoT estese. Tuttavia, rimangono sfide aperte, e possibili sviluppi futuri potrebbero concentrarsi sulla dinamicità e sulla gestione ottimizzata del bridging tra broker.

Dall'analisi di JAM come caso d'uso, emerge l'applicabilità e la versatilità di LA-MQTT multi-broker in scenari reali, aprendo la strada a implementazioni future e a nuove possibilità di ricerca.

Le valutazioni effettuate evidenziano i successi raggiunti, ma anche le aree che richiedono ulteriori indagini. La tesi sottolinea l'importanza di considerare la sensibilità spaziale in scenari IoT, ponendo le basi per ulteriori ricerche volte a ottimizzare l'efficienza e la scalabilità delle comunicazioni nei contesti distribuiti e geograficamente diversificati.

In conclusione, LA-MQTT multi-broker si presenta come una solida ri-

sposta alle sfide presenti nei protocolli tradizionali di comunicazione IoT, aprendo nuove prospettive per la progettazione e l'implementazione di reti intelligenti, reattive e scalabili.

Appendice A

Altro su JAM

Qui di seguito sono presenti altre immagini riguardanti lo sviluppo e l'applicazione finale di JAM.

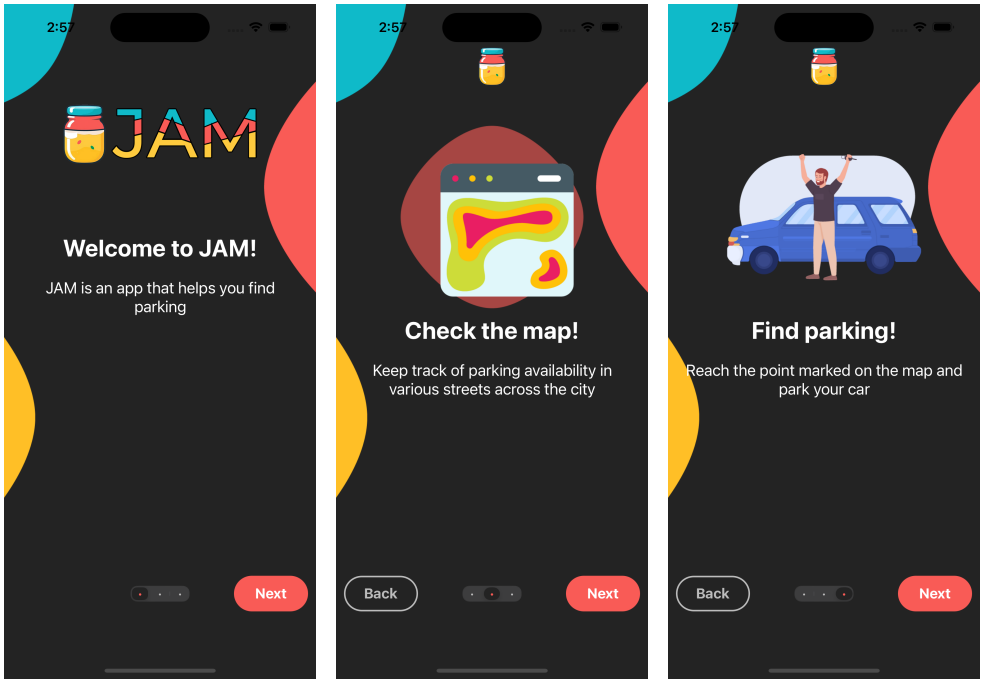

Figura A.1: Schermate onboarding di JAM

Figura A.2: Schermate autenticazione di JAM

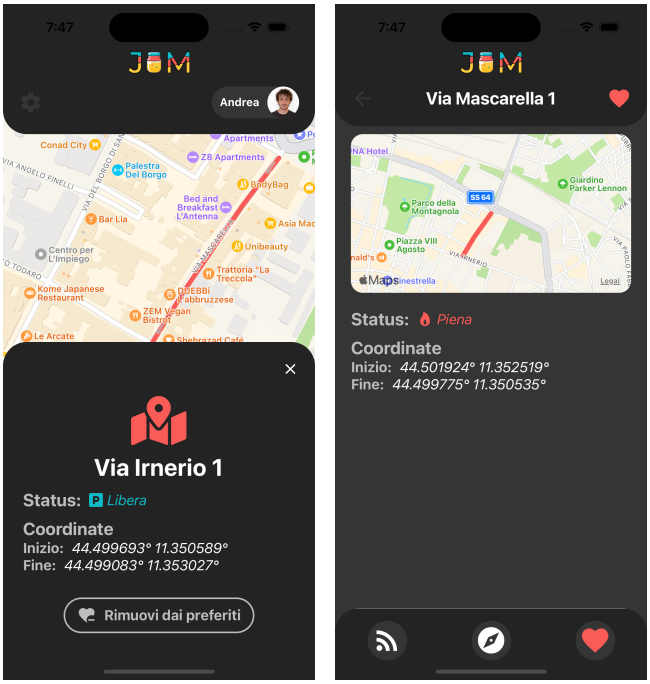

Figura A.3: Finestra modale e schermata di una specifica strada di JAM

Figura A.4: Primo mockup di JAM

Bibliografia

[1] Andrew Banks and Rahul Gupta. Mqtt version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, 2014-10-29. OASIS Standard.

[2] Robert Bryce, Thomas Shaw, and Gautam Srivastava. Mqtt-g: A publish/subscribe protocol with geolocation. In *2018 41st international conference on telecommunications and signal processing (TSP)*, pages 1–4. IEEE, 2018.

[3] Chi-Yin Chow, Mohamed F Mokbel, and Xuan Liu. Spatial cloaking for anonymous location-based services in mobile peer-to-peer environments. *GeoInformatica*, 15(2):351–380, 2011.

[4] Yi-Yun Chu and Kai-Hao Liu. Iot in vehicle presence detection of smart parking system. In *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, pages 56–59, 2020.

[5] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Computing Surveys (CSUR)*, 51(6):1–29, 2019.

[6] Albertus Ega Dwiputra, Handry Khoswanto, Raymond Sutjiadi, and Resmana Lim. Iot-based cars parking monitoring system. In *MATEC Web of Conferences*, volume 164, pages 1–11. Petra Christian University, 2018.

[7] BM Kumar Gandhi and M Kameswara Rao. A prototype for iot based car parking management system for smart cities. *Indian Journal of Science and Technology*, 9(17):1–6, 2016.

[8] Felicien Ihirwe, Giovanni Iovino, and Davide Di Ruscio. Towards an mqtt5 geo-location extension for location-aware applications. In *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pages 100–105. IEEE, 2021.

[9] ISO/IEC. Information technology - message queuing telemetry transport (mqtt) v3.1.1, 2016. ISO/IEC 20922:2016.

[10] Ryo Kawaguchi and Masaki Bandai. A distributed mqtt broker system for location-based iot applications. In *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–4. IEEE, 2019.

[11] Ryo Kawaguchi and Masaki Bandai. Edge based mqtt broker architecture for geographical iot applications. In *2020 International Conference on Information Networking (ICOIN)*, pages 232–235. IEEE, 2020.

[12] Hidetoshi Kido, Yutaka Yanagisawa, and Tetsuji Satoh. An anonymous communication technique using dummies for location-based services. In *ICPS'05. Proceedings. International Conference on Pervasive Services, 2005.*, pages 88–97. IEEE, 2005.

[13] Hai Liu, Xinghua Li, Hui Li, Jianfeng Ma, and Xindi Ma. Spatio-temporal correlation-aware dummy-based privacy protection scheme for location-based services. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[14] Edoardo Longo, Alessandro E.C. Redondi, Matteo Cesana, Andrés Arcia-Moret, and Pietro Manzoni. Mqtt-st: a spanning tree protocol for distributed mqtt brokers. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2020.

[15] Jorge E Luzuriaga, Miguel Perez, Pablo Boronat, Juan Carlos Cano, Carlos Calafate, and Pietro Manzoni. A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 931–936. IEEE, 2015.

[16] Lukas Malina, Gautam Srivastava, Petr Dzurenda, Jan Hajny, and Radek Fujdiak. A secure publish/subscribe protocol for internet of things. In *Proceedings of the 14th international conference on availability, reliability and security*, pages 1–10, 2019.

[17] Stefan Mijovic, Erion Shehu, and Chiara Buratti. Comparing application layer protocols for the internet of things via experimentation. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–5. IEEE, 2016.

[18] Biswajeeban Mishra and Attila Kertesz. The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, 8:201071–201086, 2020.

[19] Federico Montori and Luca Bedogni. A privacy preserving framework for rewarding users in opportunistic mobile crowdsensing. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–6. IEEE, 2020.

[20] Federico Montori, Lorenzo Gigli, Luca Sciullo, and Marco Di Felice. La-mqtt: Location-aware publish-subscribe communications for the internet of things. *ACM Transactions on Internet of Things*, 3(3):1–28, 2022.

[21] Alexandre Schmitt, Florent Carlier, and Valerie Renault. Data exchange with the mqtt protocol: Dynamic bridge approach. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–5, 2019.

[22] Dinesh Thangavel, Xiaoping Ma, Alvin Valera, Hwee-Xian Tan, and Colin Keng-Yan Tan. Performance evaluation of mqtt and coap via a common middleware. In *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*, pages 1–6. IEEE, 2014.

Ringraziamenti

Innanzitutto, ci tengo a ringraziare il Prof. Montori che mi ha accompagnato lungo la stesura di questa tesi. Grazie per i preziosi consigli e per avermi fatto scoprire e appassionare al mondo dell'IoT.

Uno speciale ringraziamento va ai miei genitori e alla mia famiglia ed in particolare alla persona più forte che io conosca, mia madre.

A mia sorella Isa, perché sebbene siamo completamente diversi, mi ha sempre aiutato e mi è sempre stata affianco quando ha potuto, grazie.

A mio fratello Giuseppe per non avermi fatto sentire l'unico pazzo della famiglia con mille idee per la testa e sopratutto per esserci stato quando ne avevo più bisogno, grazie.

A tutti gli altri per avermi fatto sentire a casa anche quando tornavo dopo mesi, grazie.

Vorrei poi ritagliare un posto in queste pagine per ringraziare la persona che in questi tre anni mi è sempre stata accanto dovendo spesso mettersi da parte. Abbiamo superato tutto assieme, anche la pioggia ed il freddo di Dublino, grazie Simona.

Ringrazio inoltre tutti i miei amici di corso dal primo all'ultimo.

A Moro, Mazzo e Ronto: quante giornate e notti passate a programmare, non mi sembra vero, grazie per ogni singolo istante.

A Ness e Letizia: grazie per avermi fatto constantemente compagnia durante questo percorso.

A Pietro per la bellissima convivenza e per le lunghissime e innumerevoli chiacchierate, grazie.

Infine, questa pagina segna per me la fine di una corsa durata più di tre anni. Pochi sanno che rischiava di non essere mai scritta. Nell'estate di tre anni fa mi ero ormai arreso al pensiero di dover rinunciare a questo sogno perché irrealizzabile. Qualcuno però mi spinse ad iscrivermi comunque. Vorrei ringraziarlo. Grazie Rita. Sono contento che tu sia riuscita a raggiungere i tuoi obiettivi e anche se qualche mese fa silenziosamente sei diventata dottoressa spero che tu sia orgogliosa di me quanto io lo sia di te.