Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

SIMULAZIONE IMMERSIVA REAL-TIME DI SCENARI MOBILE CROWDSENSING

Relatore:
Dott.
Federico Montori

Presentata da:
Antonino Zocco

Sessione 2
Anno Accademico 2020/2021

Ai miei Genitori *e mia* Sorella
che mi hanno insegnato a lottare e a
non arrendermi di fronte gli ostacoli della vita...

Sommario

Il Mobile CrowdSensing (MCS) è una tecnica per l'acquisizione di dati provenienti da dispositivi mobili come gli smartphone. Il simulatore CrowdSenSim è l'unico simulatore general purpose, ma non fornisce una modalità immersiva real-time. La modalità immersiva ha lo scopo di dare all'utente una visione microscopica e non soltanto aggregata dei risultati di una simulazione. In questo elaborato viene progettata la modalità real-time sul simulatore e implementata un'applicazione per rendere la simulazione immersiva, cioè permettere all'utente di essere partecipe della simulazione visualizzando la sua posizione sulla mappa e i suoi spostamenti oltre ai dati che sono stati prodotti dai sensori del suo smartphone simulato. Segue poi un'analisi dei risultati sui messaggi prodotti dal simulatore dove viene analizzato l'andamento dei messaggi pubblicati su 3 città, la frequenza dei messaggi pubblicati per ogni utente e il variare del tempo per il caricamento della lista degli utenti attivi.

Introduzione

Grazie alla diffusione dei dispositivi mobili come smartphone e tablet e alle loro capacità di acquisire dati attraverso i sensori, oggi è possibile parlare del Mobile Crowdsensing (MCS). Il Mobile Crowdsensing è una tecnica che sfrutta i sensori dei dispositivi mobili per acquisire un'elevata quantità di dati dagli smartphone degli utenti, la lettura delle informazioni ha un basso costo energetico e l'invio delle informazioni ha un ridotto consumo di banda per l'utente che mette a disposizione il suo dispositivo mobile.
Tali informazioni vengono raccolte per poi essere elaborate e trarre delle conclusioni. Per avere dati validi è necessario disporre di un gran numero di partecipanti e spesso non è semplice trovarli, per risolvere questo problema si usano i simulatori.
In questa tesi vengono integrate nuove funzionalità al simulatore CrowdSenSim2.0, un simulatore per lo sviluppo di MCS in ambienti urbani reali e con un numero di partecipanti variabile, dal quale si estraggono le informazioni per poi essere inviate all'applicazione attraverso il protocollo MQTT (Message Queue Telemetry Transport).
L'obiettivo è quello di ottenere dati dalla simulazione in tempo reale e tramite essi, visualizzare nell'applicazione la posizione di un utente sulla mappa e poter monitorare i suoi movimenti durante tutta la simulazione.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

1.1 Mobile Crowdsensing

Per Mobile Crowdsensing si intende una tecnica per misurare o analizzare dati riguardanti un determinato fenomeno o evento, la raccolta dei dati avviene sfruttando un gruppo più ampio possibile di utenti possessori di un dispositivo mobile e attraverso i sensori dei dispositivi mobili vengono acquisite informazioni sull'ambiente esterno.
Il termine Mobile Crowdsensing è stato coniato da Raghu Ganti, Fan Ye e Hui Lei in questo articolo del 2011 [6]. Il Mobile Crowdsensing (MCS) è diventato molto popolare negli ultimi anni, grazie all'incredibile diffusione dei dispositivi mobili in scala mondiale su tutta la popolazione. I dispositivi mobili oggi sono presenti costantemente nella quotidianità di ogni persona e questo ha permesso la diffusione del MCS. Il MCS è un paradigma che raccoglie dati su ambienti urbani reali attraverso i sensori dei dispositivi mobili dei cittadini, come smartphone, tablet e smartwatch per poi processarli e analizzare i dati elaborati.
Il MCS ha un'architettura formata da 4 strati:

- **Livello applicazione** coinvolge gli aspetti di alto livello delle campagne MCS come la fase di progettazione e organizzazione della campagna, le strategie di reclutamento degli utenti e la pianificazione ed esecuzione delle attività .
- **Livello dati** coinvolge tutti i componenti responsabili dell'archiviazione, analisi ed elaborazione dei dati.

- **Livello comunicazione** mostra i metodi utilizzati per fornire i dati acquisiti dai dispositivi mobili ai server.

- **Livello rilevamento** descrive i sensori integrati ai dispositivi mobili o esterni, specializzati nella cattura di determinati dati.

Un problema che interessa particolarmente il MCS è il consumo di batteria e di banda dei dispositivi utilizzati per l'acquisizione e l'invio dei dati, l'utilizzo di determinati algoritmi consente di ridurre i consumi di batteria e banda, ma anche se minimo, questo rappresenta un aspetto negativo del MCS.
L'architettura di un'applicazione MCS ha due componenti specifici, il primo è dato dall'applicazione per l'acquisizione e la diffusione dei dati dei sensori e il secondo è dato dal backend o cloud per l'analisi dei dati. Esempi di applicazioni che utilizzano il MCS sono:

COMMON SENSE il progetto Common Sense [4] si occupa di misurare l'inquinamento di una città utilizzando dei sistemi di rilevamento partecipativo che consentono agli individui di misurare la propria esposizione personale, ai gruppi di aggregare l'esposizione dei propri membri e agli attivisti di mobilitare l'azione della comunità di base. Il progetto ha come scopo principale di analizzare la qualità dell'aria ed elaborando questi dati trarre delle conclusioni sull'esposizione della comunità all'inquinamento a seconda del loro stile di vita.

PARK NET l'applicazione ParkNet [7] rileva i parcheggi disponibili nelle città utilizzando il sensore GPS e i dispositivi di rilevamento a ultrasuoni installati sulle auto. I dati raccolti vengono inviati a un server che costruisce una mappa dei parcheggi disponibili nelle vicinanze dell'utilizzatore dell'applicazione, in modo da facilitare la ricerca di un parcheggio.

Nelle applicazioni per il MCS si possono identificare degli attori comuni quasi a tutte le applicazioni e sono:

- **Amministratori della campagna** sono rappresentati da coloro che avviano le campagne di rilevamento, si occupano della progettazione, implementazione e distribuzione del sistema oltre che dalla sua manutenzione.

- **Partecipanti** sono gli utenti che installano l'applicazione sui propri dispositivi mobili e contribuiscono volontariamente alle campagne di rilevamento.

- **Utenti finali** sono coloro che accedono ai dati finali raccolti, possono essere i partecipanti stessi, nel caso in cui decidono di partecipare e consultare i loro dati, ma soprattutto gli amministratori che verificano i risultati ottenuti, scienziati nel caso in cui si deve analizzare un fenomeno o medici specializzati.

I partecipanti al MCS di solito utilizzano un'applicazione mobile generata per quel determinato caso, non esiste un'applicazione mobile generale per la gestione del MCS.

1.1.1 Incentivi

Il successo di una campagna MCS dipende dall'adesione degli utenti a partecipare a essa, questo rappresenta un grosso problema in termini economici e di tempo per il reclutamento degli utenti, esistono due tipi d'incentivazione e sono:

- **Incentivazione del crowdsourcer**[13] il crowdsourcer ha il controllo assoluto sul pagamento totale agli utenti e gli utenti possono solo personalizzare le loro azioni per soddisfare il crowdsourcer.

- **Incentivazione dell'utente**[13] ogni utente annuncia il prezzo più basso a cui è disposto a vendere un servizio, il crowdsourcer seleziona tra i candidati un gruppo di utenti e paga a ciascuno di essi per svolgere quell'attività un prezzo non inferiore al prezzo che è stato fissato dall'utente.

L'obiettivo del crowdsourcer è quello di massimizzare gli utili, in quanto non ha spese al di fuori del pagamento degli utenti perchè i dispositivi sono di proprietà di questi quindi non ha problemi legati all'usura dei dispositivi, gli utente a loro volta vogliono un incentivo sufficiente per il loro svolto. Nell'articolo [13] sono stati studiati i due modelli e si è giunti alla conclusione che sicuramente nel modello incentrato sul crowdsourcer, il piano di rilevamento di un utente è incentrato sul tempo di rilevamento. Un utente che partecipa al crowdsensing guadagnerà un pagamento non inferiore al suo costo. Tuttavia, deve

competere con altri utenti per un pagamento totale fisso. Nel modello utente ogni utente chiede un prezzo per il proprio servizio. Se selezionato, l'utente riceverà un pagamento non inferiore al prezzo richiesto. A differenza del modello incentrato su crowdsourcer, il pagamento totale non è fisso per il modello incentrato sull'utente. Quindi, gli utenti hanno un maggiore controllo sul pagamento nel modello incentrato sull'utente.

1.1.2 Privacy

Gli utenti manifestano diffidenza ad aderire alle campagne MCS a causa del problema della privacy [3]. Tutti i sistemi che rilevano dati attraverso i sensori, acquisiscono dei dati sensibili di un utente, questo comporta da parte dei sistemi un'accurata gestione dei dati e protezione di essa. I sistemi MCS si occupano di rendere anonimi le informazioni acquisite per proteggere la privacy degli utenti, inoltre per aumentare la protezioni degli utenti, molti sistemi effettuano campionamenti con intervalli temporali più ampi quando rilevano il dispositivo in luoghi sensibili come la casa o l'ufficio, questa soluzione ottimizza anche il consumo energetico e di banda da parte del dispositivo perchè viene ridotto il volume di dati da trasmettere.

1.1.3 Raccolta dati

Una soluzione alternativa estremamente valida alle campagne MCS con utenti reali è data dai simulatori. I simulatori permettono di valutare le prestazioni di sistemi MCS in ambienti urbani reali con un'ampia partecipazioni di utenti riducendo al minimo i costi e risolvendo il problema della maledizione del rilevamento[10], cioè l'incapacità delle applicazioni MCS di controllare i processi di rilevamento in modo che i dati non siano troppo sparsi o densi.
I dati sparsi possono portare a informazioni insufficienti che influiscono sulla capacità di prendere decisioni sui dati, al contrario i dati densi possono portare a intuizioni che non rispettano lo stato del mondo reale e portare a previsioni imprecise.

1.2 Applicazioni mobili

Con il termine applicazione mobile si identifica un'applicazione software destinata all'uso di dispositivi mobili che sfrutta le componenti hardware del dispositivo per processore informazioni. Le applicazioni per dispositivi mobili negli ultimi anni hanno subito un'importante crescita, tale crescita è correlata allo sviluppo dei dispositivi, che anno dopo anno vengono potenziati e quindi diventa possibile sviluppare applicazioni con un maggior carico sulle componenti hardware. Un aspetto molto importante delle applicazioni per dispositivi mobili riguarda l'interfaccia, nell'articolo [11] si parla di come l'usabilità delle applicazioni sia essenziale e in particolar modo gli utenti tendono sempre a scegliere le applicazioni in base all'interfaccia grafica, più sono intuitive più sono utilizzate. Il MCS necessità di applicazioni mobili che utilizzano i sensori, in quanto attraverso di esse deve raccogliere i dati catturati dai sensori e salvarli nel dispositivo, per poi inviarli a un dispositivo cloud o backend dove verranno elaborati.
Un'applicazione mobile può essere:

- **Personale** l'applicazione di rilevamento dati riguarda un solo individuo.
- **Community sensing** l'applicazione di rilevamento dati riguarda una vasta scala d'individui.

Come si può leggere nell'articolo [9], l'acquisizione dei dati in un'applicazione community sensing può essere partecipativa, opportunistica oppure ibrida, questo dipende dal grado di coinvolgimento degli utenti.

- L'approccio **partecipativo** consiste nell'interagire con gli utenti affinchè essi eseguano delle richieste, come registrare un suono attraverso il microfono.
- L'approccio **opportunistico** non prevede delle richieste dirette all'utente, ma le applicazioni autonomamente catturano le informazioni rilevate dai sensori a cui è stato concesso il permesso di condividere le informazioni rilevate, come il campionamento della posizione attraverso il GPS.
- L'approccio **ibrido** utilizza le migliori pratiche dei due approcci precedentemente illustrati.

Le applicazioni mobili per il MCS possono essere classificate in tre categorie a seconda dell'ambiente nel quale vengono misurati i fenomeni:

1. MCS **ambientale**, riguardano gli ambienti naturali, come la misurazione dell'inquinamento in una città.
2. MCS **infrastrutturale**, riguardano le infrastrutture pubbliche, come la misurazione della congestione del traffico.
3. MCS **sociale**, riguardano le attività sociali degli individui, come la condivisione di dati legati al tempo di attività fisica svolta in un giorno.

La raccolta dei dati nelle applicazioni mobili avviene seguendo degli step che portano dalla lettura dei dati dai sensori alla loro condivisione, gli step sono 3 e sono:

1. **Collezione** dove le informazioni vengono recuperate dai sensori.
2. **Salvataggio** le informazioni vengono condivise per poi essere elaborate.
3. **Condivisione** i dati vengono caricati e resi disponibili per tutti gli utenti.

Il ciclo di vita delle applicazioni [12] di MCS può essere diviso in 4 fasi:

1. **Creazione dell'applicazione** in questa fase viene creata un'applicazione che verrà utilizzata dall'utente per la raccolta dati. Durante questa fase vengono definiti i sensori incaricati del campionamento dei dati.
2. **Logica di assegnazione delle attività** una volta realizzata l'applicazione, l'organizzatore del MCS dovrà occuparsi del reclutamento di partecipanti per la fase di raccolta dati. Ai partecipanti vengono assegnate delle attività da svolgere durante le giornate per catturare le informazioni necessarie.
3. **Esecuzione attività individuali** i partecipanti sono a conoscenza del loro compito e lo eseguono contemporaneamente ad altri partecipanti. Ogni attività può essere pensata come il susseguirsi di 3 passi: rilevamento, computazione e trasmissione.
4. **Integrazione dati** in questa ultima fase vengono raccolti i dati generati e aggregati in modo da renderli processabili all'applicazione.

La capacità di queste applicazioni di acquisire dati attraverso i sensori è davvero elevata, questo permette alle applicazioni di poter gestire una grande mole di dati, ma rende pur sempre necessario la partecipazione di un numero di utenti relativamente alto e come detto precedentemente questo rappresenta un problema per il MCS e quindi l'utilizzo dei simulatori diventa fondamentale.

1.3 Simulazione

Come detto precedentemente per l'utilizzo del paradigma del MCS spesso una buona soluzione è data dall'utilizzo di un simulatore, questo perchè i meccanismi di reclutamento richiedo degli incentivi economici importanti e un coinvolgimento dei partecipanti più o meno significativo. Per queste ragioni sono stati sviluppati simulatori di ambienti urbani reali che simulano attraverso algoritmi il comportamento degli utenti in funzione delle attività necessarie per il rilevamento dei dati. I simulatori per il MCS nascono grazie al simulatore Network Simulator 3 (NS-3), un simulatore di reti utilizzato per l'implementazione di simulatori di MCS. NS-3 è uno strumento di simulazione altamente dettagliato al tal punto da limitare l'acquisizione di dati con migliaia di utenti e durata della simulazione superiore a poche ore. Il simulatore NS-3[1] è implementato interamente in C++, è costruito come una libreria e dunque le sue funzionalità possono essere collegate a un programma principale in C++ che definisce la tipologia di simulazione e conduce la simulazione effettiva. Inoltre possiede anche wrapper Python che permettono di condurre la simulazione utilizzando il linguaggio di programmazione Python.
L'architettura di base emula il comportamento reale di sistemi di rete complessi. Le astrazioni chiave definite dal simulatore corrispondono ai concetti più comunemente usati nel networking, come nodi, applicazioni e canali di trasmissione. Quindi lo strumento principale d'interazione durante la simulazione è il nodo, che incapsula il comportamento di qualsiasi dispositivo che si connette alla rete. Ogni nodo esegue applicazioni che ne definiscono il comportamento. Nel caso del simulatore NS-3 i nodi sono rappresentati dagli smartphone e l'applicazione determina i movimenti dell'utente. I dati verranno raccolti attraverso l'applicazione e inviate attraverso un canale di comunicazione wireless agli altri nodi in modo da poter elaborare i risultati finali.

Un simulatore di MCS molto importante è CupCarbon [8], un simulatore di Wireless Sensor Network (WSN) multi-agente. Le reti possono essere progettate utilizzando il framework OpenStreetMap (OSM) distribuendo manualmente i sensori direttamente sulla mappa e questo rende complicato la gestione di simulazioni con un grande numero di utenti. Il simulatore può essere utilizzato per studiare il comportamento di una rete e i suoi costi.
Negli ultimi anni il simulatore CrowdSenSim è diventato uno dei più importanti simulatori di MCS, in quanto riesce a gestire il movimento di centinaia di migliaia di utenti in una simulazione, il che permette di avere simulazioni in ambienti reali che non si discostano notevolmente dalla realtà. Il simulatore presta una notevole attenzione al consumo della batteria, infatti implementa una serie di algoritmi con obiettivo principale quello di ridurre i consumi energetici dei dispositivi mobili. Durante la simulazione CrowdSenSim [5] calcola a runtime una serie di statistiche, compreso il consumo di energia e la quantità di dati generati e fornisce al ricercatore uno strumento di visualizzazione per visualizzare i risultati.
Dal CrowdSenSim nasce il CrowdSenSim2.0, un simulatore che attraverso delle modifiche è riuscito a sopperire ai limiti della precedente versione e ha avuto l'introduzione di funzionalità estremamente utili e interessanti come la possibilità di supportare dati crowdsensing ibridi, e una modifica a livello architetturale importante è data dall'introduzione del PathChanger che permette i cambi di direzione durante la simulazione da parte degli utenti, inserendo una variabile ricorrente nella routine dei partecipanti reali anche sui partecipanti simulati.

1.4 Motivazioni

L'uso di simulatori permette di testare le campagne di Crowsensing in modo economico e rapido, ma una simulazione pur introducendo tutte le variabili possibili non può essere paragonato a un test reale, in quanto nelle simulazioni si perde un aspetto molto importante rappresentato dall'esperienza unica dell'utente. I sistemi di Crowdsensing possono essere partecipativi od opportunistici a seconda che l'utente fornisca volontariamente o si comporti normalmente con la differenza che i dati vengono raccolti dal suo

dispositivo. Nell'articolo [2] viene proposto un sistema ibrido, in cui i dati vengono rilevati con un approccio opportunistico, inoltre gli utenti che hanno contribuito alla raccolta dei dati vengono contattati e sono invitati a fornire informazioni aggiuntive seguendo così un approccio partecipativo. Un esempio potrebbe essere quello di un fenomeno sismico, utilizzando un approccio ibrido si riesce a catturare dai sensori i dati riguardanti l'intensità del terremoto, la posizione in cui è avvenuto e altre informazioni ottenibili grazie all'approccio opportunistico, ma solo grazie all'approccio partecipativo degli utenti con l'invio di immagini o altri documenti ottenuti da testimoni oculari sarebbe possibile avere un quadro più chiaro della situazione, soprattutto in caso di zone fortemente colpite dal fenomeno sismico o zone scarsamente coperte dai sensori. L'utilizzo di un'architettura ibrida permette di rendere la simulazione immersiva, cioè avere una simulazione di eventi reali mediante l'utilizzo del computer. La simulazione viene definita immersiva in quanto gli utenti seppur simulati attraverso l'uso dell'applicazione riesco a impersonarsi negli utenti simulati nell'ambiente reale. Gli utenti godono di una serie di parametri che rendono il loro comportamento il più simile possibile a quello di un utente reale. Del dispositivo simulato si possono decidere il numero di sensori da utilizzare per ricavare i dati, ogni dispositivo avrà un livello di carica differente e verrà simulato il consumo della batteria, ogni utente percorrerà il percorso a una velocità differente, inoltre è possibile scegliere il numero di utenti che parteciperanno alla simulazione, la durata e l'ora d'inizio della simulazione, tutte informazioni che permettono alla simulazione di ridurre il gap dalla realtà.

Capitolo 2

CrowdSenSim 2.0

CrowdSenSim 2.0 è un simulatore utilizzato per l'analisi di sistemi MCS in ambienti urbani reali, di dimensioni variabili e con un numero di partecipanti variabile. L'architettura mostra le novità rispetto la versione precedente, tali novità consentono la raccolta di dati con un approccio ibrido. Il simulatore in questa sua versione permette le simulazioni stateful, cioè un ordinamento temporale degli eventi. Il simulatore genera un insieme di partecipanti che si muovono lungo la rete stradale della città scelta, per ogni utente viene generato un percorso con punto di partenza casuale, ogni utente fornisce dati attraverso la lettura dei sensori dai loro dispositivi mobili simulati, i dati vengono riportati attraverso la connessione dati dei dispositivi o alla connessione a una rete Wi-Fi.

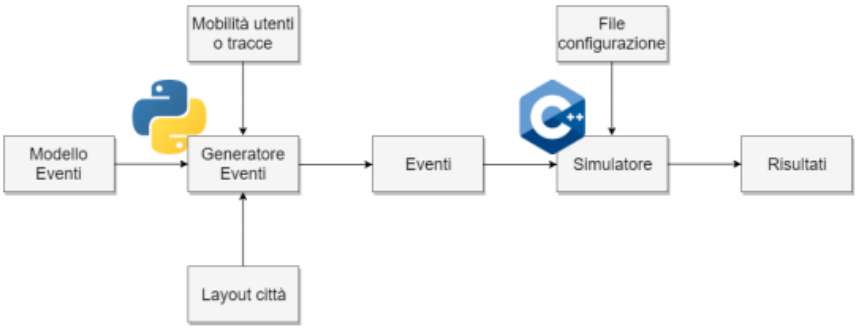

Figura 2.1: Architettura modulare simulatore

2.1 Architettura

L'architettura si può dividere in due moduli:

- **Generazione eventi**
- **Simulatore**

2.1.1 Generazione eventi

Il modulo di generazione eventi consiste nella creazione di eventi, per eventi si intende la posizione di un partecipante in un determinato momento. Ogni evento ha un id utente, id evento, stringa MGRS (Military Grid Reference System), coordinate di latitudine e longitudine e orario in cui avviene. Questo modulo è scritto con il linguaggio di programmazione Python e sfrutta le librerie implementate per la creazione dell'ambiente urbano (Layout città), il generatore di eventi riceve informazioni anche relative la mobilità degli utenti come punto di partenza, velocità e traiettorie. Per ogni utente vengono generati degli eventi che seguono un modello ben preciso e quindi tutti gli eventi hanno la stessa impostazione, infine gli eventi vengono passati al simulatore.

2.1.2 Simulatore

Il modulo del simulatore è scritto in C++. Vengono creati gli smartphone degli utenti da simulare, a ognuno di essi viene assegnata una capacità e un livello di carica della batteria differente. Vengono recuperati gli eventi ordinati. Per salvare i dati generati dai dispositivi dei partecipanti viene creata una matrice a 3 dimensioni composta dai valori dei dati dei sensori, dal MGRS di una certa area e da un determinato istante di tempo in cui è avvenuto l'evento. Attraverso il fil di configurazione, vengono impostati dei parametri del simulatore relativi alla simulazione come il numero di sensori da utilizzare e l'antenna da utilizzare per l'invio dei dati ricavati dai sensori.

2.2 Funzionalità simulatore

All'interno del simulatore si trova una cartella config, nella quale sono presenti i file per la configurazione del simulatore `Simulation_config.txt` e `Event_Generation_config.txt`, nei quali è possibile scegliere:

- **Numero utenti** il numero di utenti totale che parteciperanno alla simulazione.
- **Numero giorni della simulazione** indica il numero di giorni in cui si svolgerà la simulazione.
- **Orario inizio simulazione** orario in ore e minuti d'inizio della simulazione per ogni giorno di simulazione
- **Orario fine simulazione** orario in ore e minuti di fine della simulazione per ogni giorno di simulazione
- **Tempo minimo e massimo di percorrenza** indica la quantità di tempo minima e massima che sarà impiegata da ogni utente a camminare durante la simulazione.
- **Distribuzione utenti** rappresenta la distribuzione degli utenti durante la simulazione, si può scegliere tra distribuzione uniforme e distribuzione normale.
- **Tipologia antenna** la tecnologia utilizzata dai dispositivi mobili per inviare i dati ricevuti dai sensori.
- **Numero simulazioni** rappresenta il numero di simulazioni che verranno effettuate.
- **Numero sensori** indica il numero di sensori presente per ogni smartphone.
- **Lista algoritmi da utilizzare** in questo campo va inserita la lista degli algoritmi per MCS che dovranno essere utilizzati durante la simulazione.
- **Funzione in tempo reale** se impostato a 1 si sceglie di utilizzare la funzionalità real-time che esegue la simulazione rispettando le tempistiche reali, se impostato a 0 esegue una simulazione veloce.

La possibilità di modificare tutte queste informazioni permette di personalizzare la simulazione. La funzionalità per la simulazione in tempo reale non era presente nel simulatore, attraverso delle modifiche al simulatore è stata introdotta per permettere l'analisi degli eventi.

2.3 Algoritmi

Il simulatore è in grado di applicare sulla lettura dei dati i seguenti algoritmi:

- **Invio sempre** ogni dispositivo invia sempre i dati non appena li legge, viene utilizzato per effettuare il conto dei messaggi totali inviati.
- **Deterministic Distributed Algorithm (DDF)** ha come scopo principale quello della riduzione dei consumi energetici e di banda, la scelta di lettura dei dati dipende da un'analisi del contesto in cui si trova, viene analizzato il livello di carica della batteria e la quantità di dati che ha già letto per garantire l'equità tra i vari utenti.
- **Piggyback CrowdSensing (PCS)** ha come scopo la riduzione al minimo dei consumi energetici e di banda, l'invio avviene solo quando si ha una connessione con un server dovuta a un'altra applicazione o a una telefonata, in assenza di connessione vengono salvati nel dispositivo, non può essere utilizzato in casi in cui si richiede un'analisi real time dei dati.
- **Probabilistic Distributed Algorithm (PDA)** questo algoritmo si basa su design probabilistico per l'acquisizione dei dati dato dalla probabilità d'invio. Il sistema centrale utilizza un indice di soddisfazione (SI) calcolato attraverso il rapporto tra numero di dati inviati e numero di dati richiesti, più questo valore sarà vicino a 1 minore sarà la probabilità d'inviare o acquisire dati, garantisce in questo modo l'equità di lavoro tra i vari utenti. Esitono diverse varianti dell'algoritmo di PDA e sono:
  - **Basic** calcola la probabilità d'invio utilizzando il valore del SI.

- **Asymptotic Opportunistic algorithm for Satisfaction Index (AO-S)** utilizza un valore come booster del SI per aumentare la probabilità d'invio.
- **Asymptotic Opportunistic algorithm for Fairness (AO-F)** aumenta la probabilità d'invio all'aumentare del tempo nel quale il dispositivo non ha inviato dati.
- **Asymptotic Opportunistic for Joined Fairness and Satisfaction index (AO-JFS)** è la combinazione degli aspetti positivi di tutti gli algoritmi, ha le prestazioni migliori rispetto ai casi precedenti.

In ogni algoritmo di PDA è possibile inserire una finestra di lookahead, il cui scopo è di attendere il tempo rappresentato dalla sua dimensione. Così facendo si riduce il consumo energetico, perchè i dati vengono inviati tutti in un unico messaggio e non in messaggi differenti per ogni sensore.
Oltre agli algoritmi per l'invio dei dati abbiamo anche delle modalità per la ricezione del SI:

- **Active mode** questa modalità si occupa di mantenere sempre aggiornato il valore del SI e del booster (se presente), infatti all'inizio di ogni slot temporale vengono aggiornati i valori del SI e booster con i valori appena ricevuti derivanti dalla fine dello slot temporale precedente.

- **Power safe mode** in questa modalità il valore del SI e del booster sono ricevuti dai dispositivi solo dopo il primo invio di dati, in questo modo si risparmia molta energia per l'invio e la ricezione di dati, ma il valore del SI non è aggiornato.

- **Oracle mode** questa modalità tra tutte quelle elencate è sicuramente la più dispendiosa dal punto di vista di consumo di energia per un dispositivo mobile, ma anche la più precisa, in quanto ogni dispositivo in qualsiasi momento conosce il valore esatto del SI ed eventualmente del booster, aggiornati all'ultima ricezione del server, questo significa che per ogni valore SI che il server riceve, con questa modalità viene aggiornato anche sul dispositivo.

2.4 Generazione percorsi

La simulazione dopo aver scelta la città, effettua una serie di attività che porteranno poi alla generazione dei percorsi degli utenti nelle strade delle città. All'interno della mappa vengono generati dei nodi lungo la rete stradale della città, attraverso la libreria OSMnx per ogni coppia di nodi unita da un arco vengono generati dei nodi fittizi all'interno del grafo, i quali vengono posizionati a distanza di 3 metri l'uno dall'altro, questi nodi rappresentano i possibili punti di partenza dei vari utenti. Per conoscere la posizione dei vari punti è necessario conoscere l'angolo di Azimut, che è l'angolo che si forma partendo dal punto cardinale nord fino al secondo punto.

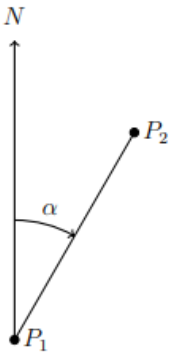

Figura 2.2: Angolo di Azimut

Le coordinate vengono convertite da gradi a radianti. Con l'utilizzo della formula inversa di Vincenty è possibile ottenere l'angolo α dalle coordinate dei due punti (P_1 e P_2).
Date le due coordinate $P_1 = (\Phi_1, L_1)$ e $P_2 = (\Phi_2, L_2)$ e considerando $\lambda = L_2$ - L_1 si ha che :

$$\alpha = \arctan\left(\frac{\cos\Phi_2 \sin\lambda}{\cos\Phi_1 \sin\Phi_2 - \sin\Phi_1 \cos\Phi_2 \cos\lambda}\right)$$

In questo modo conoscendo il punto iniziale P_1, l'angolo α e la distanza tra P_1 e P_2 è possibile ottenere le coordinate dei vari punti intermedi p_1, p_2,..., p_n. Ogni punto che viene inserito conterrà l'ID dei due nodi P_1 e P_2 e la relativa distanza, che sarà calcolata mediante l'utilizzo della funzione `great_circle_vec` della libreria OSMnx, la quale

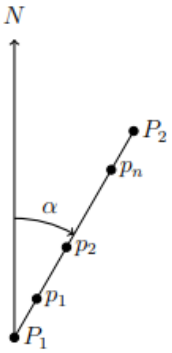

Figura 2.3: Calcolo coordinate punti intermedi con angolo di Azimut

prende in input latitudine e longitudine di due punti e restituisce la distanza tra essi. Una volta creato l'insieme di tutti i punti sulla mappa, viene estratto un nodo in modo casuale, questo sarà il punto iniziale del percorso dell'utente. In maniera casuale vengono anche generati l'ora di partenza, il tempo e la velocità dell'utente, il tutto rispettando le direttive imposte al simulatore con il file di configurazione. Conoscendo la velocità dell'utente e il tempo della sua attività è possibile risalire alla distanza percorsa, attraverso la funzione `single_source_dijkstra` della libreria NetworkX, la quale prende in input un nodo e una distanza e genera tutti i percorsi partendo da quel punto con distanza minore o uguale a quella indicata nella funzione, otteniamo così una serie di percorsi validi per il nostro utente, da questa lista di percorsi utili, ne viene estratto uno che sarà il percorso per il nostro utente. Una volta generato il cammino dell'utente in esso vengono create le posizioni nelle quali è possibile per l'utente leggere i dati provenienti dai sensori del suo smartphone. Il tempo di lettura dei dati è impostato di default a 10 secondi, questo significa che l'intervallo tre le varie letture dati sarà di almeno 10 secondi.

2.5 Lettura dati

Dopo aver ottenuto latitudine e longitudine dei vari punti nei quali l'utente può leggere i dati, questi vengono salvati nel file `UserMovimentList_0.txt` che si trova all'interno

della cartella `Inputs/saved/Olist`. Il file contiene le seguenti informazioni:

- **ID-user** il numero identificativo dell'utente che esegue la lettura.
- **MGRS** la stringa MGRS della posizione nella quale avviene la lettura.
- **Day, Hour, Minute, Second** giorno, ora, minuti e secondi nel quale si verifica la lettura.
- **Lat, Long, Alt** le coordinate nel quale avviene l'evento di lettura.

All'inizio della simulazione vengono creati gli smartphone per gli utenti. Ogni smartphone sarà riconducibile a un solo utente e avrà una capicità di batteria e livello di carica differente, in modo da simulare dispositivi differenti. Successivamente verranno letti tutti i possibili eventi di lettura e verranno salvati in una lista, gli eventi saranno poi ordinati cronologicamente. A questo punto può iniziare la simulazione effettiva.

Capitolo 3

Implementazione simulatore

In questo capitolo è illustrato il lavoro svolto in questa tesi sul simulatore. Viene spiegato nel dettaglio l'organizzazione del sistema, la nuova funzionalità che permette la gestione degli eventi in real-time e la condivisione dei dati tramite il protocollo MQTT (Message Queue Telemetry Transport).

3.1 Architettura

Al simulatore è stata introdotta una funzionalità che permette di avviare una simulazione in real-time, qui vengono creati gli smartphone simulati e generati gli eventi con i relativi dati dei vari utenti. I dati devono essere resi disponibili allo smartphone reale in cui viene eseguita l'applicazione, per permettere questa condivisione di dati è stato utilizzato il protocollo MQTT, con il quale avviene la pubblicazione delle informazioni dal simulatore al broker MQTT. Le pubblicazioni sul broker MQTT avvengono raggruppando i dati in topic, nel momento della pubblicazione deve essere indicato il topic nel quale vanno pubblicati i dati. Il broker MQTT, poi, si occuperà della pubblicazione dei dati a tutti i dispositivi iscritti a quel topic. L'iscrizione a un topic avviene attraverso l'applicazione con la quale è possibile iscriversi al topic selezionato. Una volta avvenuta l'iscrizione al topic, lo smartphone riceverà tutti i dati che vengono pubblicati su di esso.

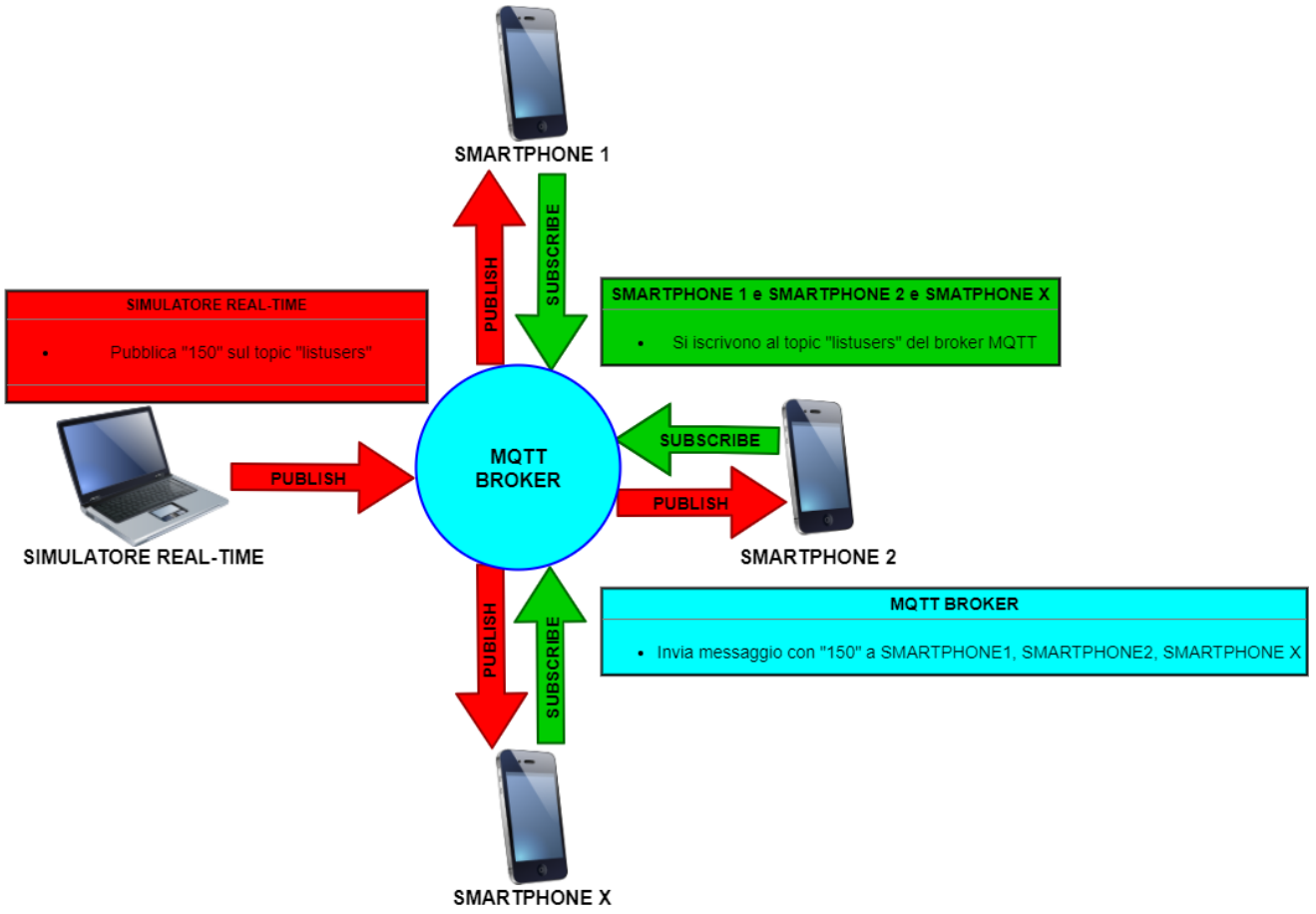

Figura 3.1: Architettura del sistema

3.2 Funzionalità real-time

La nuova funzionalità introdotta con questa tesi permette di gestire gli eventi rispettando il loro valore temporale, dunque un secondo nel simulatore corrisponderà a un secondo fuori dal simulatore. In precedenza il simulatore CrwodSenSim2.0 non prevedeva una simulazione in tempo reale, ma una simulazione veloce, in cui il tempo di simulazione dipendeva prevalentemente dalle prestazioni della macchina che eseguiva il simulatore. La gestione degli eventi real-time è permessa soprattutto grazie al lavoro di ordinamento degli eventi, avendo quindi gli eventi ordinati temporalmente è possibile farli eseguire rispettando l'ordine e i tempi di attesa tra il verificarsi di un evento e quello

successivo. Gli eventi si verificano nell'intervallo di tempo indicato nel file di configurazione, si possono verificare più eventi contemporaneamente e tra l'evento attuale e quello successivo può essere presente un intervallo di uno o più secondi.
La gestione degli eventi in real-time è attivabile attraverso il file di configurazione `Simulation_config.txt` nel quale si ha un campo con il quale si può decidere se avviare una simulazione veloce, impostando a 0 il valore del campo o avviare una simulazione con velocità reale impostando il valore del campo a 1. Sul file `simulation.cc` è dichiarata una variabile booleana che preleva il valore dal file di configurazione e attraverso un controllo indirizza il codice sulla gestione degli eventi veloce o real-time. La gestione in modalità real-time si basa sull'utilizzo della funzione `usleep` e della funzione `gettimeofday` della libreria `time.h` con le quali viene assegnato rispettivamente il valore di attesa al processore prima di eseguire l'evento successivo e il valore attuale espresso in microsecondi. Si effettua un'analisi di ogni evento, per ogni evento viene estratto il valore in secondi nel quale si verifica l'evento attuale e quello successivo, i due valori vengono confrontati, se sono uguali significa che l'evento successivo `timeE1` avviene nello stesso istante temporale di quello corrente `timeE0` quindi deve essere eseguito senza nessuno sleep, se sono diversi allora significa che l'evento successivo deve avvenire dopo una pausa, il valore della pausa viene calcolato sottraendo dal valore `nowE` il valore `startE` ottenendo esattamente il valore in microsecondi relativo al tempo da inizio simulazione al momento in cui viene analizzato l'evento in questione e il valore viene sottratto al tempo dell'evento successivo per ottenere il numero esatto di microsecondi da attendere prima d'iniziare a eseguire l'evento successivo, in modo da rendere la simulazione in tempo reale.

Calcolo tempo di attesa

Confronto tra tempo evento attuale ed evento successivo, se diversi allora calcolo il tempo di attesa.

```
1 if(timeE0 != timeE1){
2     gettimeofday(&nowE,NULL);
3     double tempoInMicrosecondi=
4     ((nowE.tv_sec*1000000+nowE.tv_usec)-
```

```
5     (startE.tv_sec*1000000+startE.tv_usec));
6     if(timeE1 != 0){
7         double timeForSleep = (timeE1 - tempoInMicrosecondi);
8         usleep(timeForSleep);
9     }
10 }
```

3.3 Condivisione dati

Un'altra modifica apportato sul simulatore con questa tesi riguarda la condivisione dei dati generati. Lo scopo era quello di condividere le informazioni generate dal simulatore con l'applicazione mobile in tempo reale, dunque era necessario l'utilizzo di un protocollo che potesse gestire più client contemporaneamente e una grande mole di dati, per questo si è pensato di utilizzare il protocollo MQTT per la pubblicazione dei messaggi del simulatore sull'applicazione mobile che è stata implementata.

3.3.1 Protocollo MQTT

Il protocollo MQTT è un protocollo di comunicazione che ha come scopo la riduzione al minimo del consumo della banda nelle comunicazioni tra dispositivi, inoltre ottimizza il consumo energetico dei dispositivi alimentati a batteria.
Il protocollo per scambiare informazioni sfrutta un meccanismo di pubblicazione e sottoscrizione (publish e subscribe) di messaggi tramite un apposito message broker in grado di gestire migliaia di client contemporaneamente.
Se un client vuole comunicare con un altro client, il primo pubblica un messaggio su un certo argomento (topic) sul message broker, il message broker ha il compito di filtrare e distribuire le comunicazioni tra publisher e subscriver, quindi è necessario che il secondo client sia iscritto al topic. Ogni client può iscriversi a più topics e ogni volta che viene pubblicato un messaggio su un topic il message broker lo distribuisce a tutti i client iscritti al topic.
Il topic è una stringa UTF-8 con struttura simile a quella di una directory, i vari livelli sono separati da uno "/" come nella struttura ad albero delle cartelle.

Per la trasmissione dei messaggi tramite MQTT è necessario conoscere IP, porta del broker e parametri di autenticazione, se richiesti. Nel mio elaborato ho utilizzato come broker mosquitto, un software che implementa il protocollo MQTT.

3.3.2 Utilizzo mosquitto

La libreria `mosquitto.h` implementa tutte le funzioni necessarie per utilizzare il protocollo MQTT. Le funzioni che ho utilizzato in questo elaborato sono:

- **mosquitto_lib_init** è la prima funzione che deve essere attivata in quanto inizilizza la libreria.

- **mosquitto_new** crea una nuova istanza client mosquitto, richiede in input come parametri l'id, un valore booleano per eliminare i messaggi alla disconessione e un puntatore utente che sarà passato come riferimento delle callback.

- **mosquitto_connect** è la funzione utilizzata da un client per connettersi a un MQTT Broker, il parametro host serve per identificare il broker attraverso il suo ip o hostname, il parametro port identifica la porta network utilizzata, da standard è la 1883.

- **mosquitto_publish** questa funzione è utilizzata per la pubblicazione dei messaggi, come parametri vengono specificati il nome del topic, la lunghezza del messaggio in byte, il messaggio, il QoS (Quality of Service) e infine un valore booleano per il retain dove viene specificato se mantenere l'ultimo messaggio.

- **mosquitto_disconnect** con questa funzione si effettua la disconnessione al broker.

- **mosquitto_destroy** è utilizzata per liberare la memoria associata alle istanze del client mosquitto.

- **mosquitto_lib_cleanup** è utilizzata per liberare le risorse associate alla libreria.

3.3.3 Implementazione

L'implementazione della condivisione dei dati è avvenuta sul file `simulation.cc` nel quale utilizzando la libreria `mosquitto.h` sopra illustrata, è stato possibile usufruire delle funzioni del protocollo MQTT. I topic creati per la pubblicazione dei messaggi sono:

- **IDUtente** per ogni ID utente è stato creato un topic nominato con il numero dell'id cioè per una simulazione di 10000 utenti vengono creati 10000 topic, ciascuno nominato con l'id dell'utente che andrà da 1 a 10000. In questo modo per ogni utente ho un topic nel quale ci saranno i dati relativi allo smartphone con quell'id. Nel topic vengono pubblicate le coordinate della posizione, il percorso effettuato, l'utilizzo dei sensori e la loro probabilità di utilizzo.

- **user** in questo topic vengono pubblicati per ogni evento l'id dell'utente e il tempo dell'evento in cui è avvenuto l'evento.

- **number** questo topic ha un solo messaggio prelevato dal file di configurazione, infatti in questo topic viene pubblicato il numero di sensori utilizzati nella simulazione, si sfrutta il retain affinchè il messaggio venga pubblicato una sola volta, e ogni client quando si collega riceva l'ultimo messaggio inviato, dunque per ogni connessione al topic viene ricevuto lo stesso messaggio che non cambierà nel corso della simulazione.

- **lastuser** il messaggio sarà pubblicato nel topic solo al verificarsi di una condizione, cioè che sia l'ultimo evento della simulazione, se la condizione non viene soddisfatta nel topic non viene pubblicato nulla. Quando viene analizzato l'ultimo evento verrà pubblicato un messaggio che contiene le informazioni relative all'id dell'utente e all'istante temporale in cui si è verificato l'evento.

- **IDUtente+finish** per ogni ID utente è stato creato un topic nominato con il numero dell'id concatenato con la parola finish, in questo topic per ogni utente viene pubblicato un messaggio che contiene le coordinate di latitudine e longitudine relative all'ultima posizione dell'utente.

3.4 Polyline

Un'ulteriore piccola modifica al simulatore ha riguardato l'introduzione di una polyline per rappresentare il percorso che viene effettuato dagli smartphone degli utenti a partire dal punto iniziale fino alla posizione attuale in quel istante della simulazione. Una polyline è una serie di segmenti che uniscono i vari punti, cioè le varie posizioni degli utenti, cercando in questo modo di poter risalire approssimativamente al percorso dell'utente. Risalire al percorso dell'utente inviando la successione di punti della polyline non è possibile, in quanto occuperebbe troppo spazio, la soluzione è rappresentata dalla codifica della polyline in una stringa, la codifica avviene utilizzando la libreria `polylineencoder.h` con la quale la successione di punti viene codificata in una stringa dove le lettere e i simboli rappresentano le posizioni, in modo da facilitare l'invio della polyline. La polyline verrà inviata allo smartphone, ma prima di essere utilizzata, deve essere decodificata. La decodifica avviene utilizzando l'API di Google che permette di decodificare la stringa e riottenere i vari punti (latitudine e longitudine). Nella classe `Smartphone` è stato aggiunto un vettore inizializzato vuoto, che ha lo scopo di raggruppare gli eventi per ogni smartphone. Ogni evento analizzato viene inserito nella lista dello smartphone con quell'id e viene estratta la posizione con le coordinate di latitudine e longitudine. Utilizzando la libreria `polylineencoder.h` il punto viene aggiunto alla struttura `encoder`, l'insieme dei punti viene codificato in una stringa, aggiungendo le varie posizioni si genera una stringa che rappresenta il cammino di quell'utente, tale informazione viene inviata codificata con le informazioni relative all'utente e verrà gestita nell'applicazione mobile dove verrà decodificata e i punti estratti saranno utilizzati nell'applicazione.

Generazione polyline

Aggiungo alla lista dello smartphone in questione le coordinate della posizione.

```
1 sm = smM->find((*eventsL)[e].id_user)->second;
2 sm->lista.push_back((*eventsL)[e]);
3 for(int i=0;i<sm->lista.size();i++){
4     encoder.addPoint(sm->lista[i].loc->lat,sm->lista[i].loc->lon);
```

```
5 }
6 std::string res = encoder.encode();
```

Capitolo 4

Implementazione applicazione mobile

In questo capitolo viene illustrato il lavoro svolto in questa tesi per la realizzazione dell'applicazione mobile per dispositivi Android. L'applicazione è stata sviluppata interamente nell'ambiente Android Studio. Lo scopo dell'applicazione è quello di connettersi al Broker MQTT, una volta stabilita la connessione con il broker si visualizzerà una lista di utenti attivi, attraverso il tocco di uno degli utenti visualizzati nella lista si effettuerà l'iscrizione al topic di quel utente, in questo modo l'applicazione riceverà tutti i dati che vengono pubblicati in quel topic, tra i dati pubblicati ci saranno quelli relativi la latitudine e longitudine dell'utente, quindi sarà possibile visualizzare in tempo reale la posizione dell'utente sulla mappa. Quando la simulazione terminerà l'applicazione effettua la disconnessione al broker e rimanda l'utente all'activity iniziale.

Le applicazioni Android sono organizzate in activity, per activity si intende lo schermo dell'applicazione dove è possibile scambiarsi informazioni. Le activity sono composte da componenti grafiche, le View, alle quali sono associati eventi, come button ed edittext, ai quali reagisce l'applicazione. L'applicazione sviluppata ha 3 activity principali:

- **MainActivity**
- **SettingsActivity**
- **MapsActivity**

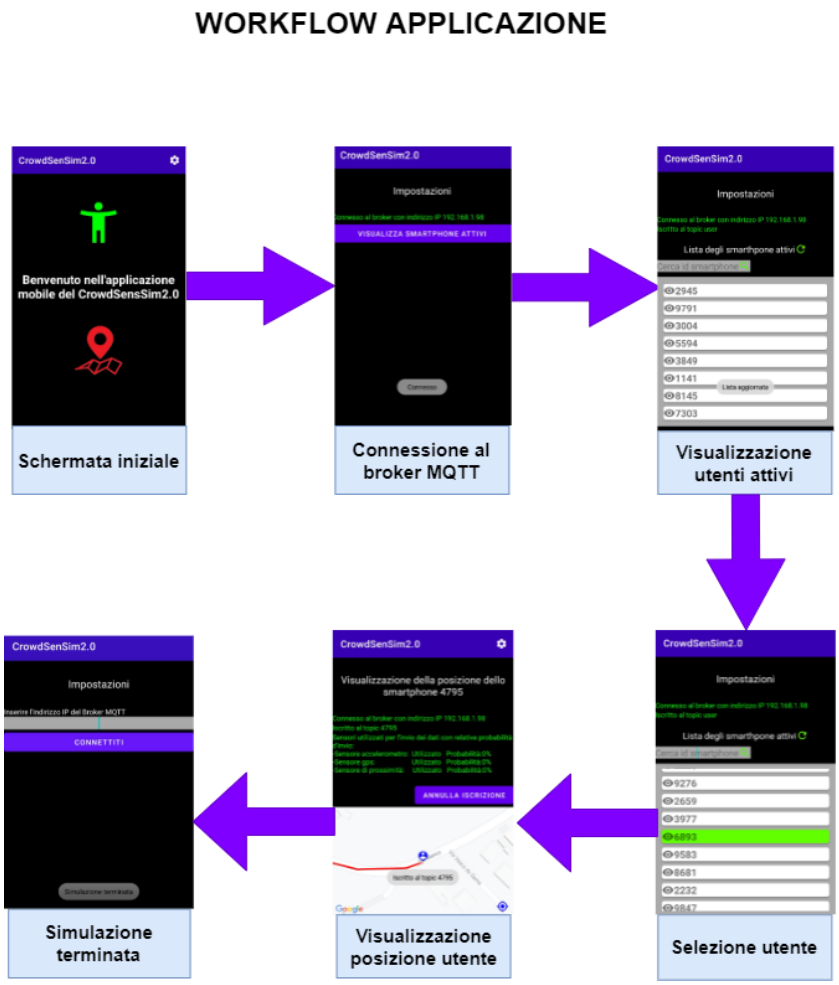

Figura 4.1: Workflow applicazione

4.1 MainActivity

La prima activity con la quale l'utente deve interagire è la MainActivity. Questa activity è stata realizzata a solo scopo introduttivo e quindi è stata gestita prevalentemente la parte grafica, in cui è rappresentata la schermata di benvenuto dell'applicazione CrowdSenSim2.0 con un messaggio di benvenuto al centro e sopra e sotto di esso due loghi per rendere l'idea del lavoro svolto dall'applicazione. L'implementazione più significativa di questa activity è rappresentata dalla `toolbar`, una barra degli strumenti posizionata nella parte alta dello schermo con la quale l'utente può interagire toccando l'immagine

delle impostazioni e con questa azione l'applicazione attraverso un intent, un messaggio che contiene informazioni con cui è possibile comunicare con un'altra componente o attivare una componente, come ad esempio in questo caso in cui l'intent indirizza l'utente a una nuova activity, la SettingsActivity.

Figura 4.2: MainActivity

4.2 SettingsActivity

La SettingsActivity è l'activity in cui avviene la connessione al Broker MQTT, l'iscrizione al topic user e la scelta dell'id dello smartphone di cui si vuole visualizzare la posizione sulla mappa. Nella prima schermata visualizzata è necessario instaurare la connessione al Broker MQTT, questa operazione avviene inserendo nella `editText` l'indirizzo IP del broker e toccando il `button CONNETTITI`.
Questa operazione genera un evento a cui è associata la richiesta di connessione al broker, se la richiesta va a buon fine viene notificato nello schermo l'avvenuta connessione e viene visualizzata una nuova pagina, altrimenti viene notificato nella stessa pagina che la connessione non è andata a buon fine. Una volta stabilita la connes-

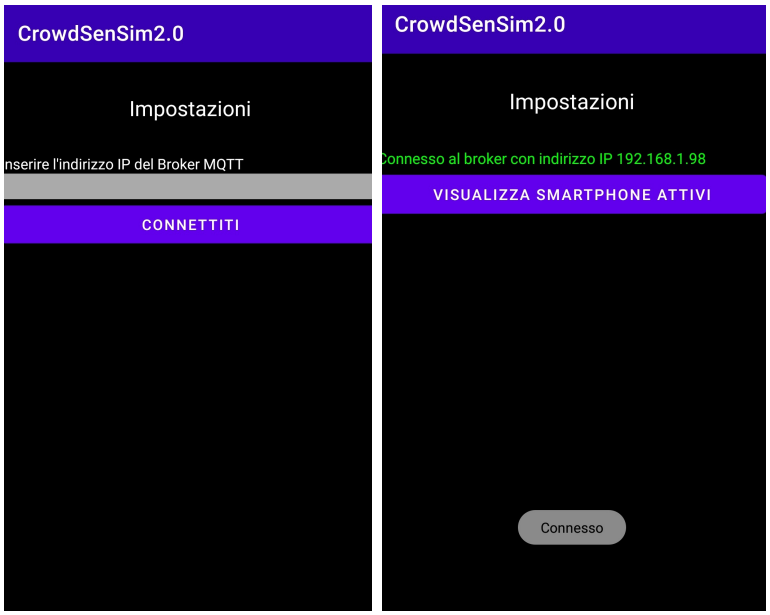

Figura 4.3: Connessione al broker

sione viene visualizzata una nuova schermata nella quale è necessario interagire con il `button VISUALIZZA SMARTPHONE ATTIVI`. Tale operazione avvierà il processo d'iscrizione al topic "user" del broker. In questo topic come illustrato precedentemente vengono inviati gli id degli utenti che compiono l'evento e l'istante temporale nel quale si è verificato l'evento.
Nella schermata vengono visualizzate le informazioni relative all'indirizzo IP del broker a cui ci siamo iscritti e al topic, inoltre viene mostrata una barra di ricerca attraverso la quale è possibile ricercare tra la lista degli smartphone attivi uno specifico utente e infine la lista degli smartphone attivi in una `RecyclerView`.Attraverso la `RecyclerView` vengono mostrati tutti gli utenti attivi in quel momento, per utenti attivi si intendono gli utenti che hanno pubblicato un messaggio negli ultimi 10 secondi.

La `RecyclerView` visualizza gli elementi di un ArrayList attraverso le `CardView`, ogni item ha l'id dell'utente e un `ImageButton` con il simbolo di un occhio che permette la visualizzazione di quell'item. Ho utilizzato due ArrayList, il primo `mExampleList` nel quale vengono aggiunti tutti gli utenti che vengono pubblicati nel topic "user" e il secondo `mExampleListVisual` nel quale vengono scartati gli utenti non attivi e mantenuti

Figura 4.4: Visualizzazione lista utenti attivi

solo gli ultimi 100 utenti attivi, a questo arraylist è associato l'adapter, cioè l'oggetto che permette la visualizzazione degli elementi della lista. La selezione degli utenti attivi avviene attraverso un algoritmo che analizza l'istante temporale dell'evento attuale (E_1) e lo confronta con l'istante temporale del primo elemento (E_0) dell'arraylist `mExampleListVisual` cioè l'elemento meno recente della lista, seguendo questa formula:

$$\text{se } E_1 > (E_0 + 9) \text{ allora rimuovi l'elemento } E_0 \text{ dalla lista e aggiungi } E_1$$
$$\text{altrimenti se } E_1 <= (E_0 + 9) \text{ allora aggiungi } E_1$$

se la prima condizione si verifica allora la formula viene ripetuta sul nuovo elemento in testa, fino a quando ha esito negativo e quindi non viene rimosso nessun altro elemento, questo perchè in questo modo si verifica che tutti gli elementi presenti nell'arraylist siano ancora validi, non basta controllare solo il primo elemento perchè possono esistere più elementi con lo stesso istante temporale.
Inizialmente nella `RecyclerView` viene caricato un solo elemento, attraverso l'`ImageButton` con il logo dell'aggiornamento è possibile ricaricare la `RecyclerView` con tutti gli utenti attivi. L'arraylist `mExampleListVisual` è associato all'`Adapter`, un oggetto che permet-

te il collegamento tra una lista e l'`AdapterView`, quindi non appena vengono effettuate delle modifiche all'arraylist viene invocata la funzione `NotifyDataSetChanged` in modo da poter visualizzare la nuova lista aggiornata.
Attraverso la barra di ricerca è possibile cercare all'interno della lista la presenza di un determinato id utente, il tutto avviene inserendo il numero dell'id nella barra di ricerca. L'esito della ricerca può avere due esiti:

- **Ricerca fallita** nel caso in cui l'utente ricercato non è attivo in questo momento o non esiste.
- **Ricerca riuscita** nel caso in cui viene trovato tra gli utenti attivi l'utente, in questo caso viene visualizzata una nuova lista che contiene il solo elemento ricercato.

Figura 4.5: Ricerca utente

Nella lista degli smartphone attivi si può notare la presenza di item con uno sfondo di colore differente, gli item con sfondo differente sono gli id degli utenti di cui ho già visualizzato il percorso nella mappa, in questo modo posso sempre tenere sotto controllo gli utenti che ho già visualizzato. La modifica dello sfondo degli item è stata realizzata sulla

classe `Adapter` nella `OnBindViewHolder` nella quale è stato creato un arraylist `mUser` che contiene l'id di tutti gli utenti selezionati, questo arraylist viene poi passato alla SettingsActivity e inserito in testa all'arraylist `mExampleListVisual` e in base alla dimensione dell'arraylist `mUser`, colora le prime posizioni dalla 0 alla dimensione di `mUser`-1 della `RecyclerView`. Un'altra modifica sempre su questa classe avviene sulla selezione di un elemento, infatti quando attraverso l'`ImageButton` con il simbolo dell'occhio visualizziamo un elemento questo si colora di verde, per mettere in risalto l'elemento selezionato.

Figura 4.6: Selezione di un utente dalla lista

4.2.1 Listati della SettingsActivity

Connessione al broker

Si legge da input l'indirizzo IP e si effettua il tentativo di connessione.

```
1 mqttAndroidClient[0] = new MqttAndroidClient(getApplicationContext(),
2   "tcp://"+s_ipAddress+":1883", clientId);
```

```
3     try {
4         mqttAndroidClient[0].connect(this, new IMqttActionListener() {
5             @Override
6             public void onSuccess(IMqttToken asyncActionToken) {
7                 ...
8                 //NOTIFICA CONNESSIONE RIUSCITA
9                 ...
10            }
11            @Override
12            public void onFailure(IMqttToken asyncActionToken,Throwable
                  exception) {
13                ...
14                //NOTIFICA CONNESSIONE FALLITA
15                ...
16            }
17        });
18    } catch (MqttException e) {
19        ...
20        ...
21 }
```

Popolamento ArrayList

Inserimento degli utenti attivi alla lista e rimozione degli utenti non attivi.

```
1 if(mExampleList.isEmpty()){
2     ...
3     //AGGIUNGI ELEMENTO
4     ...
5     mAdapter.notifyDataSetChanged();
6 }
7 else{
8     boolean controllo = true;
9     while(controllo){
```

```
10          if(Integer.parseInt(array[1])>Integer.parseInt(mExampleList.get(0)
11          .getmText2())+9){
12              mExampleList.remove(0);
13          }
14          else{
15              controllo = false;
16          }
17      }
18      mExampleList.add(new ExampleItem(array[0],array[1],s_ipAddress));
19  }
```

Aggiornamento RecyclerView

Caricamento della nuova lista degli utenti attivi.

```
1  public void Aggiorna(){
2      for (int j=0;j<mUser.size();j++){
3          mExampleListVisual.add(j,mUser.get(j));
4      }
5      if(!mExampleList.isEmpty()){
6          int n = mExampleList.size();
7          if(n<100){
8              for(int i=0;i<n-mUser.size();i++){
9                  ...
10                 //AGGIUNGI UTENTE IN TESTA ALLA LISTA VISUALIZZATA
11                 ...
12             }
13         }
14         else{
15             for(int i=0;i<100-mUser.size();i++){
16                 ...
17                 //AGGIUNGI UTENTE IN CODA ALLA LISTA VISUALIZZATA
18                 ...
19             }
```

```
20          }
21      }
22      mAdapter.notifyDataSetChanged();
23  }
```

Ricerca utenti

Ricerca di un determinato utente nella lista degli utenti attivi.

```
1  for(int i=0;i<mExampleList.size();i++){
2      if(mExampleList.get(i).getmText1().equals(s_sm_id)){
3          ...
4          //UTENTE TROVATO
5          ...
6      }
7      if(i == mExampleList.size()-1){
8           ...
9          //UTENTE NON TROVATO
10         ...
11     }
12 }
```

4.3 MapsActivity

Il passaggio alla MapsActivity avviene selezionando un elemento della `RecyclerView` che invoca un intent che permette l'apertura della nuova activity. Per una corretta gestione dello stack delle activity è necessario gestire gli stati delle activity. Ogni acitivity ha un suo ciclo di vita, nel momento in cui viene creata si trova nello stato `onCreate`, se attraverso un intent invochiamo il passaggio dalla SettingsActivity alla MapsActivity, lo stato della SettingsActivity va in `onPause` e la MapsActivity si trova nello stato `onCreate` se è la prima volta che viene visualizzata nello stack delle acitivty, se è già presente nello stack delle activity allora va nello stato `onResume`, quindi in ogni activity è necessaria la gestione dello stato per non avere dei comportamenti inaspettati.

Nella MapsActivity viene visualizzata la mappa e nello specifico viene visualizzata la posizione attuale dell'utente selezionato, inoltre vengono fornite all'utente informazioni riguardo al broker a cui si è connessi, al topic a cui si è iscritti e infine informazioni riguardo l'utilizzo dei sensori con le relative probabilità d'invio dei dati raccolti.
La mappa viene creata attraverso le API di Google che forniscono una mappa dettagliata nella quale è possibile inserire dei marker per personalizzarla. Questa activity riceve dalla SettingsActivity informazioni riguardo l'id dell'utente che si deve monitorare, dunque grazie a questa informazioni avviene l'iscrizione al topic di quell'utente, questo permette di monitorare i dati di quell'utente, cioè vedere la sua posizione e come vengono utilizzati i sensori del suo dispositivo. Per ogni messaggio ricevuto sul quel topic viene aggiornata

Figura 4.7: MapsActivity

la posizione dell'utente, che è rappresentata da un marker con la forma di una persona, la posizione viene aggiornata a intervalli minimi di 10 secondi, altre informazioni visualizzabili in questa schermata sono quelle relative ai sensori, in questo caso si ha l'i-

scrizione al topic "number", nel quale è indicato il numero di sensori utilizzati in questa simulazione, inoltre viene riportato il nome del sensore, se è utilizzato o meno e infine la probabilità d'invio dei dati da parte di quel sensore, questi dati vengono generati dal simulatore per ogni evento, quindi per ogni evento queste informazioni possono subire dei cambiamenti. L'activity possiede anche un `button ANNULLA ISCRIZIONE` che permette di annullare l'iscrizione al topic, questo evento comporta l'annullamento dell'iscrizione al topic con quell'id e il passaggio nello stato `onPause` della MapsActivity in modo da poter tornare nella SettingsActivity e poter selezionare un nuovo utente da visualizzare. Quando si sta osservando un utente è possibile visualizzare la mappa e spostare la visuale senza nessun impedimento in luoghi di nostro interesse, ma se vogliamo rapidamente tornare a visionare l'utente si può utilizzare un `ImageButton` in basso a destra sulla mappa che serve a rilocalizzare l'utente e spostare il focus sulla sua posizione attuale.

Un altro aspetto molto importante della MapsActivity deriva dal tracciamento del

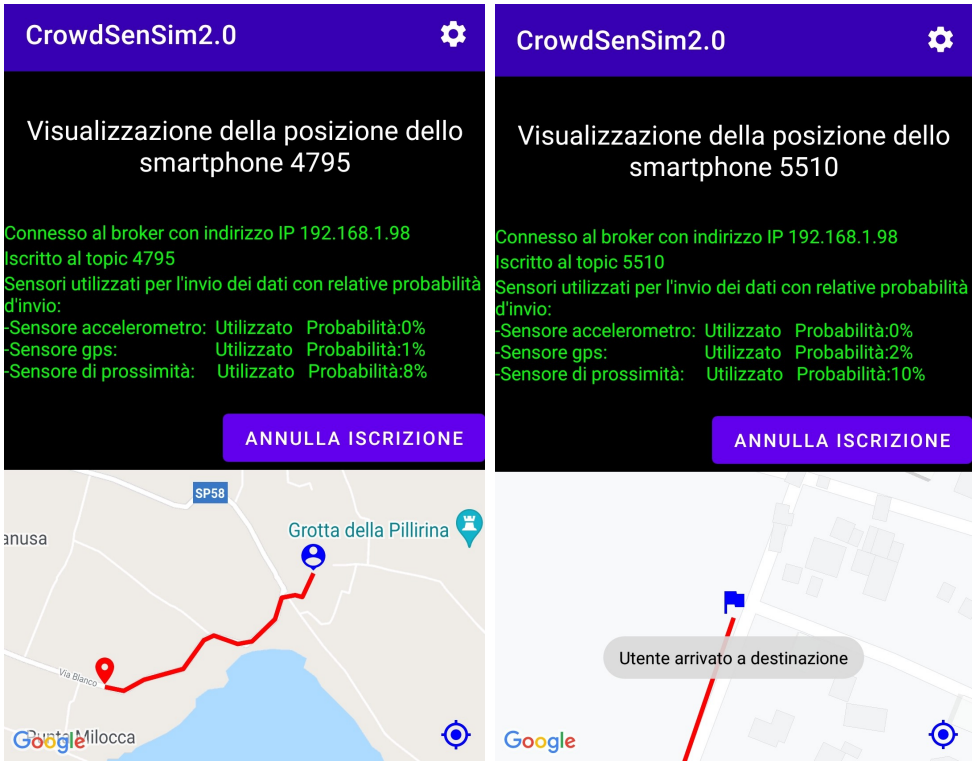

Figura 4.8: Visualizzazione percorso e destinazione di un utente

percorso svolto dall'utente, questa implementazione inizia sul simulatore attraverso la polyline codificata che viene mandata nello stesso messaggio con la posizione dell'utente, viene estratta dall'array e decodificata, ottenendo così i punti che compongono il percorso e tracciare una linea in modo da rappresentare il percorso, inoltre dalla polyline vengono estratti il primo punto, che rappresenta il punto di partenza del nostro utente che viene indicato sulla mappa con un marker di colore rosso con un segnaposto e l'ultimo punto che rappresenta la posizione attuale con il marker blu della persona, relativo all'ultimo punto, quindi della posizione attuale è necessario fare una precisazione, il percorso dell'utente generato ha un inizio e una fine, non è quantificabile in tempo l'arrivo dell'utente, poichè questo varia, dunque viene sfruttato un topic che per ogni utente pubblica l'ultima posizione del suo percorso, dunque per ogni posizione vengono confrontate le coordinate della posizione attuale e della destinazione, se queste sono uguali significa che la posizione attuale è la destinazione dell'utente e quindi il marker che indica la posizione attuale viene sostituito da un marker che indica il punto di arrivo.
Infine un quando il simulatore pubblica come messaggio l'ultimo evento, quando si ritorna alla SettingsActivity avviene la disconnessione al broker in quanto la simulazione è terminata.

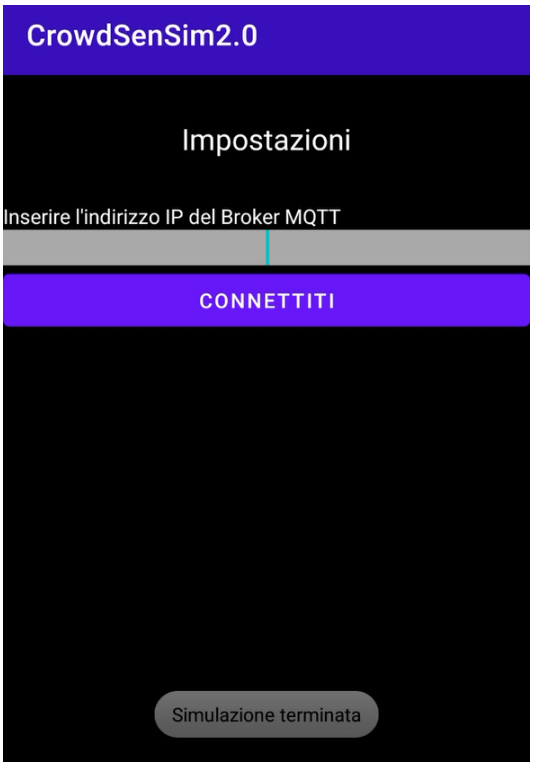

Figura 4.9: Fine simulazione

4.3.1 Listati della MapsActivity

Gestione posizioni sulla mappa

Aggiunta di marcatori nella mappa per rappresentare punto di partenza, punto di arrivo o posizione attuale e percorso effettuato.

```
1 LatLng position = new LatLng(latitudine,longitudine);
2 LatLng arrivo =
    PolyUtil.decode(poly_cod).get(PolyUtil.decode(poly_cod).size()-1);
3 try{
4     if((last_lat==latitudine)&&(last_lon==longitudine)){
5         ...
6         //UTENTE ARRIVATO A DESTINAZIONE
7         ...
```

```
8     }else{
9         ...
10        //POSIZIONE ATTUALE UTENTE
11        ...
12    }
13 } catch (NumberFormatException e) {
14            e.printStackTrace();
15 }
16 LatLng partenza = PolyUtil.decode(poly_cod).get(0);
17 ...
18 //SEGNA PUNTO DI PARTENZA
19 ...
20 mMap.addPolyline(polyop.addAll(PolyUtil.decode(poly_cod)).color(Color.RED));
```

Capitolo 5

Risultati

In questo capitolo vengono analizzati i risultati ottenuti attraverso dei grafici. Sono state prese in considerazioni simulazioni con 100, 250, 1000, 2500 e 10000 utenti in 3 città differenti, Bologna, Siracusa e Priolo Gargallo. La scelta delle città avviene per analizzare su territori di dimensioni differenti l'andamento delle simulazioni, infatti Bologna ha un territorio di circa 140 km^2, Siracusa di circa 207km^2 e Priolo Gargallo di circa 56 km^2.
Una prima analisi ha dimostrato che il numero di messaggi pubblicati dal simulatore per ogni simulazione cresce in maniera piuttosto omogenea, infatti, come era possibile immaginarsi mantenendo inalterati i parametri della simulazione, si evidenziano piccole differenze relative al numero di messaggi totali pubblicati attraverso il protocollo MQTT, la città con più messaggi pubblicati risulta essere Siracusa. Il grafico evidenzia che al crescere del numero degli utenti in una simulazione aumentano anche i messaggi pubblicati.

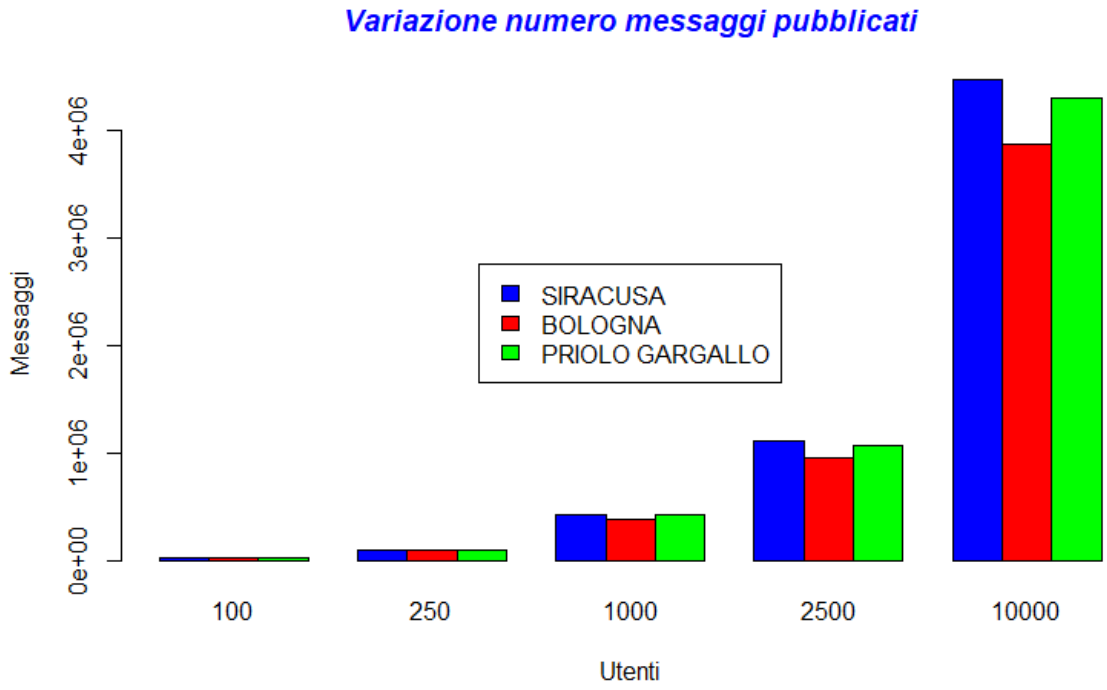

Figura 5.1: Grafico con numero di messaggi pubblicati per ogni simulazione simulazione

Il secondo grafico tende ad analizzare il numero di messaggi pubblicati per ogni utente nelle simulazioni con 10000 utenti per le 3 città, quindi sono stati prelevati dai messaggi del grafico precedente solo quelli che facevano riferimento agli utenti. Per ciascun utente sono stati contati solo i messaggi pubblicati dal suo smartphone ed è stato possibile realizzare questo istogramma che rappresenta la frequenza dei messaggi inviati da ogni utente.

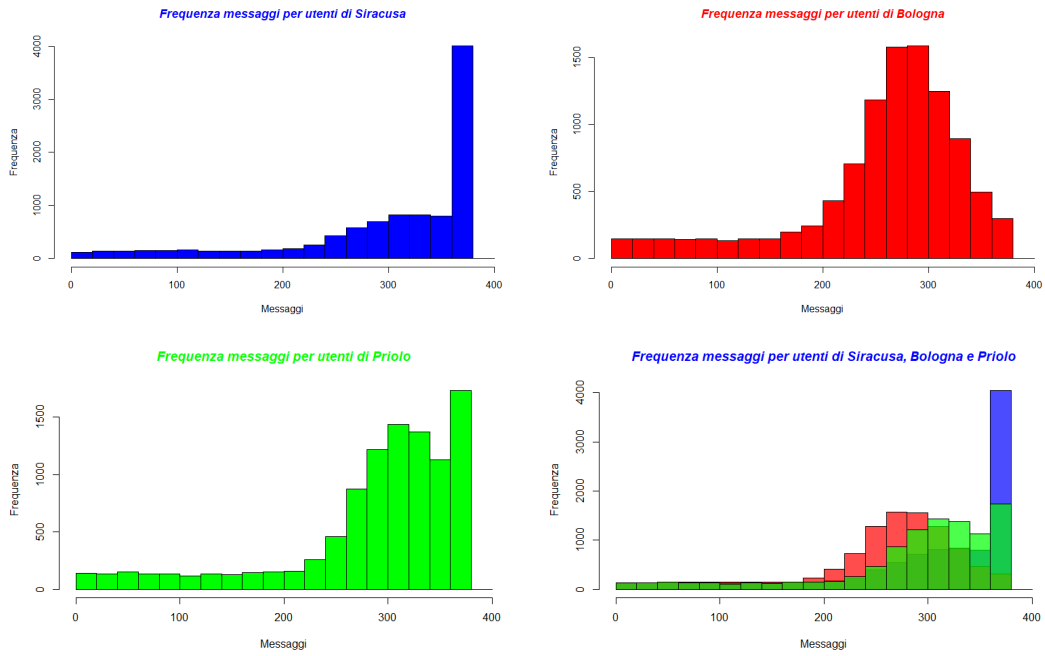

Figura 5.2: Grafico con frequenza numero di messaggi pubblicati a Siracusa, Bologna e Priolo

In questo grafico si evidenzia come per Bologna e Priolo Gargallo ci siano risultati molto simili tra di loro, per quanto riguarda la città di Siracusa si ha un picco nell'intervallo 360-379, questo significa che per molti utenti sono stati raccolti tra i 360 e i 379 messaggi nella simulazione di Siracusa, questo dato potrebbe essere spiegato attraverso i dati sul territorio e la forma irregolare della città. Per Bologna e Priolo Gargallo i dati raccolti ci indicano una forma più regolare, nello specifico per la città di Bologna, dove il grafico assume un aspetto quasi simmetrico, meno regolare per Priolo Gargallo dove il grafico presenta anche questa volta il picco massimo nell'intervallo 360-379 ma una distribuzione abbastanza regolare ngli intervalli precedenti. Infine un ultimo grafico è stato realizzato per analizzare il caricamento della recycler view, quindi un'analisi sull'applicazione. L'obiettivo di questo studio era capire la stabilità dell'applicazione al livello temporale per il caricamento della lista degli utenti attivi all'aumentare degli utenti nella simulazione. Sono stati prelevati dei dati relativo il tempo per l'aggiornamento e caricamento della recycler view che contiene gli utenti attivi in simulazioni con 100, 250,

1000, 2500 e 10000 utenti per le 3 città. Per ogni simulazione sono stati campionati 14 valori in millisecondi e i 2 estremi (il valore minore e maggiore) sono stati scartati. Dei restanti 12 valori si è calcolata la media e realizzato un grafico, nel quale si rappresenta l'andamento della variazione del tempo per ogni città al variare degli utenti. In questo grafico si evidenzia come l'aumento degli utenti per ogni simulazione non rappresenti un problema per l'applicazione, la quale mantiene ottime prestazioni.

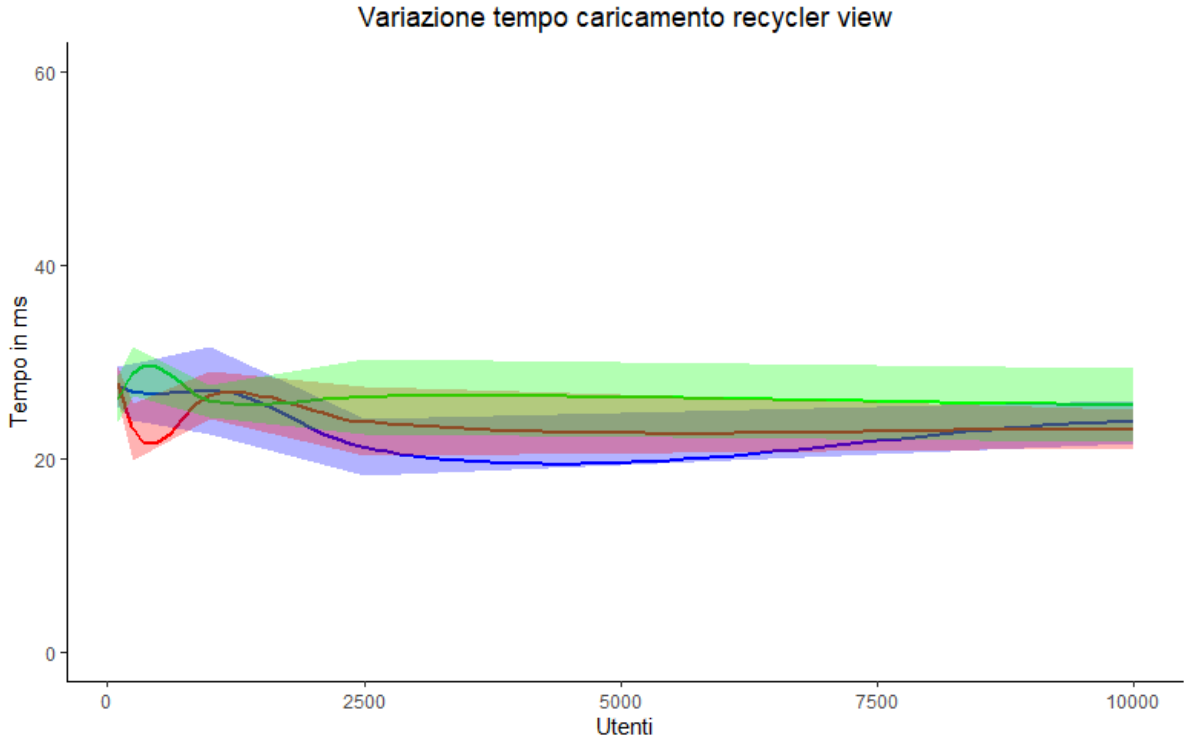

Figura 5.3: Grafico con variazione tempo per il caricamento della recycler view

Conclusioni

In questa tesi si è andati a modificare il simulatore CrowdSenSim2.0 per essere in grado di eseguire simulazioni con la modalità in tempo reale, sulla base di questa modifica sono state aggiunte funzionalità per la realizzazione dell'applicazione mobile, ad esempio sul simulatore è stato introdotto il protocollo MQTT per la condivisione dei dati. Una volta che dal simulatore vengono condivisi i dati vengono gestiti nell'applicazione mobile, che permette di visualizzare attraverso una mappa le posizioni simulate e il percorso effettuato dall'utente. Inoltre dall'applicazione è possibile visualizzare quali sensori hanno contribuito per l'invio dei dati e la probailità che i dati vengano inviati. Inoltre attraverso l'applicazione è possibile scegliere l'ID dell'utente di cui si vogliono visualizzare le informazioni. Principalmente l'implementazione dell'applicazione è avvenuta usufruendo del protocollo MQTT, scegliendo i vari topic a cui iscriversi per ricevere informazioni e gestirle. Attraverso i grafici si è gunti alla conclusione che l'applicazione è ben sviluppata e riesce a gestire 10000 utenti senza evidenti problemi, mantenendo elevate le prestazioni sull'operazione più rilevante compiuta, cioè la gestione della recylcer view. Per quanto riguarda il simulatore si è studiata la frequenza dei messaggi pubblicati per ogni utente, dove si è evidenziato un comportamento differente per una città di dimensioni superiori e una forma irregolare, mentre per città con forme più regolari e un territorio inferiore il comportamento è molto simile. Sempre sul simulatore si è studiato l'andamento dei messaggi pubblicati e si è evidenziata una crescita abbastanza regolare dei messaggi all'aumentare dei partecipanti alla simulazione.

Bibliografia

[1] Simulazione crowdsensing utilizzando ns-3. In *Workshop internazionale sui cittadini nelle reti di sensori*.

[2] Marco Avvenuti, Salvatore Bellomo, Stefano Cresci, Mariantonietta Noemi La Polla, and Maurizio Tesconi. Hybrid crowdsensing: A novel paradigm to combine the strengths of opportunistic and participatory crowdsensing. In *Proceedings of the 26th international conference on World Wide Web companion*, pages 1413–1421, 2017.

[3] Delphine Christin, Andreas Reinhardt, Salil S Kanhere, and Matthias Hollick. A survey on privacy in mobile participatory sensing applications. *Journal of systems and software*, 84(11):1928–1946, 2011.

[4] Prabal Dutta, Paul M Aoki, Neil Kumar, Alan Mainwaring, Chris Myers, Wesley Willett, and Allison Woodruff. Common sense: participatory urban sensing using a network of handheld air quality monitors. In *Proceedings of the 7th ACM conference on embedded networked sensor systems*, pages 349–350, 2009.

[5] Claudio Fiandrino, Andrea Capponi, Giuseppe Cacciatore, Dzmitry Kliazovich, Ulrich Sorger, Pascal Bouvry, Burak Kantarci, Fabrizio Granelli, and Stefano Giordano. Crowdsensim: a simulation platform for mobile crowdsensing in realistic urban environments. *IEEE Access*, 5:3490–3503, 2017.

[6] Raghu K Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *IEEE communications Magazine*, 49(11):32–39, 2011.

[7] Suhas Mathur, Tong Jin, Nikhil Kasturirangan, Janani Chandrasekaran, Wenzhi Xue, Marco Gruteser, and Wade Trappe. Parknet: drive-by sensing of road-side parking statistics. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 123–136, 2010.

[8] Kamal Mehdi, Massinissa Lounis, Ahcène Bounceur, and Tahar Kechadi. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. In *7th International ICST Conference on Simulation Tools and Techniques, Lisbon, Portugal, 17-19 March 2014*, pages 126–131. Institute for Computer Science, Social Informatics and Telecommunications, 2014.

[9] Federico Montori, Luca Bedogni, Claudio Fiandrino, Andrea Capponi, and Luciano Bononi. Performance evaluation of hybrid crowdsensing systems with stateful crowdsensim 2.0 simulator. *Computer Communications*, 161:225–237, 2020.

[10] Federico Montori, Prem Prakash Jayaraman, Ali Yavari, Alireza Hassani, and Dimitrios Georgakopoulos. The curse of sensing: Survey of techniques and challenges to cope with sparse and dense data in mobile crowd sensing for internet of things. *Pervasive and Mobile Computing*, 49:111–125, 2018.

[11] Fatih Nayebi, Jean-Marc Desharnais, and Alain Abran. The state of the art of mobile application usability evaluation. In *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4. IEEE, 2012.

[12] Leye Wang. *Facilitating mobile crowdsensing from both organizers' and participants' perspectives*. Theses, Institut National des Télécommunications, May 2016.

[13] Dejun Yang, Guoliang Xue, Xi Fang, and Jian Tang. Incentive mechanisms for crowdsensing: Crowdsourcing with smartphones. *IEEE/ACM transactions on networking*, 24(3):1732–1744, 2015.

Ringraziamenti

Mi è doveroso dedicare questo spazio alle persone che hanno contribuito e mi hanno accompagnato in questi anni al raggiungimento di questo obiettivo.

In primis, un ringraziamento speciale al mio relatore Montori Federico, per la sua immensa pazienza, per i suoi indispensabili consigli e per le conoscenze trasmesse durante il periodo di tirocinio e della stesura della tesi.

Un ringraziamento va a tutti i miei familiari, le vostre telefonate mi hanno sempre riempito il cuore e fatto sentire a casa, nei momenti di tristezza mi hanno fatto tornare un sorriso e dato la forza per continuare, nei momenti di gioia mi hanno reso fiero dei miei sforzi, grazie a tutti i nonni, zii e cugini. Vi voglio bene!

Un pensiero alle bisnonne che purtroppo non ci sono più, mi dispiace non essere riuscito a condividere anche con voi questo giorno, ma sono sicuro che oggi sareste tutte fiere di me. Mi mancate!

Ringrazio infinitamente il mio collega e amico Alessio Di Dio con il quale ho condiviso interamente questa esperienza, abbiamo condiviso la casa, le gioie universitarie e anche i momenti di difficoltà, sei stato una spalla su cui ho potuto sempre contare, capace di strapparmi una risata anche in momenti difficili, grazie!

Grazie ai miei amici Gianluca, Enrico, Daniele e Angelo per essere sempre stati presenti, grazie per avermi ascoltato ogni volta che ne ho avuto di bisogno, grazie per tutti

i consigli utili che mi avete dato, grazie per essere sempre presenti da 7 anni, talvolta le amicizie si evolvono e diventa riduttivo definirle tali, per questo vi ringrazio fratelli!

Ringrazio la mia fidanzata Valentina per avermi sopportato e supportato, sei stata al mio fianco durante i progetti infiniti e ogni volta che chiedevo un tuo parere leggevo il terrore nei tuoi occhi. Grazie per essermi stata davvero accanto in momenti in cui mi sono chiuso in me stesso, sei riuscita a darmi la forza di andare avanti. Grazie per tutto il tempo che mi hai dedicato anche quando io non riuscivo a dedicartene. Grazie amore mio per essere anche oggi al mio fianco!

Grazie a mia sorella, per avermi sempre supportato, in particolare negli ultimi mesi in cui eravamo a casa, hai visto i miei momenti di difficoltà e sei riuscita ad aiutarmi, così come hai sempre fatto in tutti questi anni. Sei stata la prima persona ad avermi fatto visita a Bologna e fatto capire cosa realmente stavo facendo, cosa significasse essere lontani da casa e non avere un contatto con le persone più importanti della mia vita. Sei sempre stata un riferimento della mia vita, grazie di esistere!

Vorrei ringraziare coloro che mi hanno dato la possibilità di raggiungere questo traguardo. Ringrazio di cuore i miei genitori, per avermi dato la possibilità di studiare a Bologna facendo dei sacrifici economici e non solo, li ringrazio perchè non hanno mai dubitato di me e sono sempre riusciti a farmi sentire vicino a loro, in questi anni non è mai esistito un giorno in cui non ci siamo sentiti, un giorno in cui non mi sono sentito amato. Grazie per tutti i valori che mi avete trasmesso, grazie per tutte le parole e gesti di questi anni, grazie per esserci sempre stati, per esserci oggi e per esserci domani, so sempre di poter contare su di voi. Spero che questo traguardo possa almeno in parte ripagare tutti i vostri sacrifici e rendervi almeno un minimo fieri di me così come lo sono io di voi. Grazie per essere la mia fonte d'ispirazione!

Un sentito grazie a tutti voi e tutti coloro che ho avuto il piacere di conoscere in questi magnifici anni, amici, coinquilini, colleghi e vicini di casa, grazie per avermi accompagnato in questo percorso!