Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Studio, Realizzazione e Simulazione di SaveMe, un'applicazione Android per il Disaster Recovery

Relatore:
Chiar.mo Prof.
Luciano Bononi
Correlatore:
Dott. Luca Bedogni

Presentata da:
Marco Giaccari

Sessione I
Anno Accademico 2014/2015

A Maria,
la mia ragazza,
che mi ha sempre sostenuto...

Introduzione

Quando avvengono disastri naturali, spesso la copertura dati viene a mancare e le infrastutture o sono danneggiate o sono sovraccariche; in questo modo è difficile comunicare sia da parte delle persone che hanno bisogno di aiuto, sia da parte dei soccorritori che cercano di organizzare i soccorsi.

Con questa tesi si è voluto realizzare un'applicazione Android che permetta agli utenti di segnalare il proprio bisogno di aiuto, anche se il device non ha una connessione internet attiva.

L'applicazione, sfruttando il Wi-Fi e il Wi-Fi Direct, cercherà di formare una rete di dispositivi, attraverso la quale i messaggi di aiuto degli utenti verranno scambiati tra i device.

Questa rete, man mano, si allargherà fino ad arrivare ad includere device che avranno una connessione dati attiva. Questi comunicheranno con il mio server, al quale manderanno tutti i messaggi che gli sono arrivati.

I dati del server, ossia i messaggi che sono stati mandati dagli utenti, verranno mostrati sul sito ltw1528.web.cs.unibo.it. Attraverso questo sito, i soccorritori potranno vedere la posizione degli utenti in stato di bisogno, cosicché potranno mandarli un messaggio di soccorso, che si propagherà nella rete formatasi in precedenza, ed organizzare i soccorsi in maniera ottimale.

Si è anche voluto fare uno studio simulativo per testare la scalabilità dell'applicazione e per raccogliere dati statistici, quali il delay medio tra l'arrivo del messaggio al device con connessione dati e il tempo in cui è stato creato, l'influenza sulla batteria del numero dei messaggi scambiati e il numero degli host, il delay tra il tempo di invio e il tempo di arrivo nello

scambio di messaggi al variare del numero degli host.

Indice

Introduzione **i**

1 Stato dell'arte **1**

1.1 Android 1

1.1.1 Architettura 2

1.1.2 Applicazioni 4

1.2 Wi-fi Direct 6

1.2.1 Formazione del Gruppo 7

1.2.2 Wi-Fi Direct su Android 9

1.3 Related Works 10

2 L'applicazione **13**

2.1 Scenario/Obiettivo 13

2.2 Realizzazione 14

2.2.1 Componenti Principali 14

2.2.2 Le fasi dell'applicazione 16

3 Il server e la pagina web **23**

3.1 Il Server 23

3.2 Sviluppo Sito Web 25

4 Studio simulativo **27**

4.1 OMNeT++ 27

4.1.1 I Moduli 29

4.1.2 I messaggi 30
4.1.3 Il linguaggio NED 30
4.1.4 I file .ini 30
4.1.5 MiXiM 31
4.2 La simulazione 32
4.2.1 I risultati 33

Conclusioni **41**

Bibliografia **43**

Elenco delle figure

1.1 Android Stack. Sorgente: [1] 2
1.2 Metodi della classe WifiP2pManager. Sorgente: [4] 9
1.3 Listener della classe WifiP2pManager. Sorgente: [4] 10
1.4 Intent della classe WifiP2pManager. Sorgente: [4] 10

2.1 Main Activity 15
2.2 Stati applicazione. [23] 16

3.1 Sito web 25

4.1 Simple e Compound Module 29
4.2 Delay medio tra la creazione di un messaggio di aiuto e l'arrivo ad un nodo con 3G 34
4.3 Delay medio tra la creazione di un messaggio di aiuto e l'arrivo ad un nodo con 3G 34
4.4 Media del numero di pacchetti inviati 35
4.5 Media del numero di pacchetti inviati 35
4.6 Media del numero di messaggi di Bonj inviati 36
4.7 Media del numero di messaggi di Bonj inviati 36
4.8 Media dei messagi inviati 37
4.9 Media dei messagi inviati 37
4.10 Consumo medio batteria 38
4.11 Consumo medio batteria 38
4.12 Delay tra host e host senza 3G 39

Capitolo 1

Stato dell'arte

1.1 Android

Android è un sistema operativo per sistemi mobili basato su kernel Linux e sviluppato da Google. Con un'interfaccia utente basata sulla manipolazione diretta, Android è stato progettato principalmente per dispositivi mobili touchscreen come smartphone e tablet, con interfacce utente specializzate per televisioni (Android TV), macchine (Android Auto) e orologi da polso (Android Wear).

Il Sistema Operativo usa tocchi di input che corrispondono vagamente alle reali azioni, come lo swiping, il tapping, il pinching, per manipolare gli oggetti sullo schermo, insieme ad una tastiera virtuale.

Il codice sorgente di Android è stato rilasciato da Google sotto licenza open source, ad esclusione dei driver non-liberi inclusi per i produttori di dispositivi.

Android è stato con noi in una forma o nell'altra da più di sette anni. Durante questo tempo abbiamo assistito ad un tasso assolutamente mozzafiato di cambiamento come nessun altro ciclo di sviluppo che sia mai esistito.

Ormai è un sistema operativo storicamente grande. Quasi un miliardo di device sono stati venduti, e 1.5 milioni sono attivati giornalmente.

Una delle caratteristiche più evidenti di Android è il fatto che le sue diver-

se versioni sono indicate a livello ufficiale con un numero di versione secondo gli standard informatici, ma che alla fine queste vengono distinte per il proprio "codename", tradizionalmente ispirato alla pasticceria e rigorosamente in ordine alfabetico: Cupcake (Android 1.5), Donut(1.6), Eclair (2.0/2.1), Froyo (2.2), Gingerbread(2.3/2.4), Honeycomb (3.0), Ice Cream Sandwich (4.0), Jelly Bean (4.1), KitKat (4.4), Lollipop (5.0) e infine Android M presentato nell'evento Google I/O 2015.

1.1.1 Architettura

Il sistema operativo Android è organizzato su più livelli di astrazione ed ha un'architettura a stack. Ogni strato, ed i corrispondenti elementi all'interno, sono strettamente integrati e accuratamente sintonizzati per fornire l'ottimale sviluppo delle applicazioni e un ambiente di esecuzione per i dispositivi mobili.

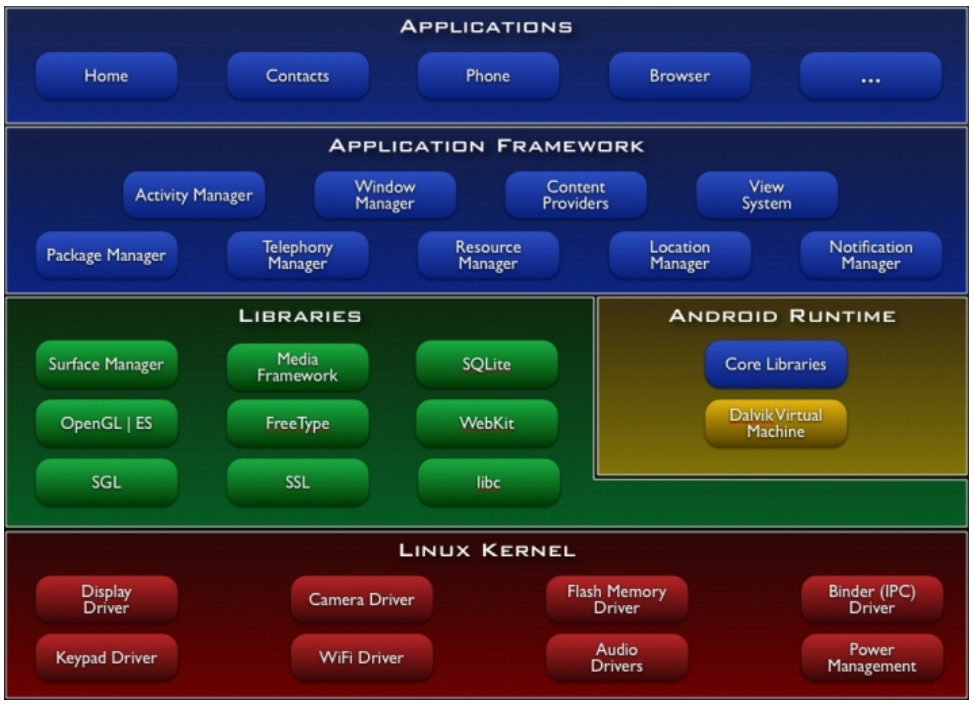


Figura 1.1: Android Stack. Sorgente: [1]

Possiamo dividere lo stack nei seguenti livelli:

- **Kernel Linux**
  Posizionato alla base dello stack, il Kernel Linux fornisce un livello di astrazione tra l'hardware del device e i livelli superiori dello stack. Basato sulla versione 2.6 di Linux, il kernel fornisce multitasking preemptive, servizi di basso livello di sistema, come la gestione della memoria, dei processi e dell'alimentazione, oltre a fornire anche driver per il display, il Wi-Fi e l'audio.

- **Librerie**
  Il livello successivo sono le librerie native di Android. È questo il livello che permette al device di manipolare diversi tipi di dati. Queste librerie sono scritte in C o C++. Tra le più importanti troviamo:

  - **Surface Manager**
  - **Media framework**
  - **SQLite**
  - **WebKit**
  - **OpenGL**

- **Android Runtime**
  Questa sezione dell'architettura Android si trova al secondo livello dello stack, assieme alle librerie. Questa sezione fornisce una componente chiave chiamata Dalvik Virtual Machine (DVM), che è un tipo di Java Virtual Machine (JVM) progettata ed ottimizzata per Android. La DVM, a differenza della JVM che esegue file di tipo .class, esegue file .dex. Questi sono costruiti da file di tipo .class al momento della compilazione e forniscono efficienza maggiore in ambienti con poche risorse. Google, nelle ultime release, ha introdotto una nuova macchina virtuale conosciuta come ART (Android RunTime). ART ha molti vantaggi rispetto alla DVM come la compilazione AOT, ossia la compilazione

in anticipo e una migliorata garbage collection che aumentano notevolmente le performance delle applicazioni. In questa sezione troviamo anche le librerie **core** che permettono agli sviluppatori di scrivere le Applicazioni Android usando il linguaggio standard di programmazione Java.

- **Application Framework**

  Questo livello fornisce molti servizi di livello superiore per le applicazioni sotto forma di classi Java, tra cui:

  - **Activity Manager**, che controlla tutti gli aspetti del ciclo di vita dell'applicazione;
  - **Content Providers**, che permette alle applicazioni di pubblicare e condividere dati con altre applicazioni;
  - **Resource Manager**, che fornisce l'accesso alle risorse come le stringhe, i colori e i layout;
  - **Notifications Manager**, che permette alle applicazioni di mostrare alert e notifiche all'utente;
  - **View System**, un insieme estendibile di viste usate per creare l'interfaccia utente;

- **Applicazioni**

  Al più alto livello dello stack sono situate le applicazioni. Queste comprendono sia applicazioni native preinstallate nel dispositivo, come l'applicazione per i messagi, il dialer, il browser, il manager dei contatti, sia applicazioni di terze parti installabili dall'utente.

1.1.2 Applicazioni

Con il termine applicazione (o app) ci si riferisce a software applicativi java-based installabili su Android. Come già detto, le applicazioni sono al più alto livello dell'architettura e gli elementi più importanti che le compongono sono [2] :

- Activity
- Service
- Content providers
- Broadcast receivers

Activity

Un Activity fornisce uno schermo attraverso il quale l'utente può interagire con l'applicazione al fine di fare qualcosa. Ad ogni activity è assegnata una finestra nella quale disegnare la propria interfaccia utente. La finestra tipicamente riempie lo schermo, ma può essere più piccola o può fluttuare su altre finestre. Un'applicazione, di solito, consiste di molteplici activity che sono vagamente legate le une alle altre e una di queste è definita come main; questa verrà presentata all'utente quando lancerà l'applicazione per la prima volta.

Service

Un Service può eseguire operazioni di lunga durata in background e non fornisce un'interfaccia utente. Può essere fatto partire da un altro componente dell'applicazione e può rimanere in background anche se l'utente cambia applicazione. Per lo scopo di questa tesi, i service sono di fondamentale importanza in quanto tutto si svolge in background.

Content Providers

I Content Providers gestiscono l'accesso ad un insieme strutturato di dati. Questi incapsulano i dati e forniscono meccanismi per la definizione della sicurezza su questi. I content providers sono le interfacce standard che connettono i dati in un processo con il codice presente in un altro processo.

Broadcast Receivers

Un Broadcast Receiver è un componente che rileva i messaggi di broadcast che arrivano dal sistema operativo, come per esempio un broadcast che annuncia che lo schermo si è spento, o che la carica della batteria è bassa ecc. Un'applicazione può "lanciare" un broadcast, per esempio per avvertire le altre applicazioni che qualcosa è stato scaricato e che è pronto ad essere utilizzato. Sebbene il Broadcast Receiver non disponga di un'interfaccia utente, può creare una notifica per avvertire l'utente che un certo evento broadcast è avvenuto.

1.2 Wi-fi Direct

Il Wi-Fi Direct è una nuova tecnologia definita da Wi-Fi Alliance [3], attraverso la quale i dispositivi compatibili possono collegarsi direttamente tra loro in modo rapido, sicuro, con la possibilità di eseguire compiti come la stampa, la sincronizzazione e lo scambio di dati.

Lo standard IEEE 802.11 è divenuto una delle più comuni vie d'accesso a Internet. Oramai è presente su un'infinità di dispositivi, come smartphone, TV, stampanti, automobili, ecc. Il Wi-Fi si basa sul modello base di Access Point che creano reti wireless a cui i dispositivi si possono collegare e comunicare. Il Wi-Fi Direct permette ai dispositivi di comunicare tra loro usando un metodo simile al Wi-Fi tradizionale, ma senza richiedere l'uso di un Access Point centrale. I device usano un "Software Access Point" (Soft AP).

La comunicazione diretta tra i dispositivi era già possibile con il Wi-Fi standard per mezzo della modalità ad-hoc. Tuttavia questo tipo di connettività non è mai stato in grado di marcare la propria presenza sul mercato a causa di diversi inconvenienti e limitazioni nei requisiti, come ad esempio la mancanza di un supporto al risparmio di energia.

La tecnologia Wi-Fi Direct utilizza un approccio differente: invece di utilizzare la modalità ad-hoc, si basa sull'infrastruttura dell' IEEE 802.11

e consente ai dispositivi di negoziare chi assumerà le funzionalità di AP. In questo modo si consente agli altri dispositivi di connettersi alla rete così creata.

1.2.1 Formazione del Gruppo

I dispositivi Wi-Fi Direct, formalmente conosciuti come P2P Device, comunicano costruendo un gruppo P2P, che è funzionalmente equivalente alle tradizionali infrastrutture network Wi-Fi. Il device che implementa le funzionalità dell'AP nel gruppo P2P è chiamato P2P Group Owner (P2P GO), invece i device che si comportano da client sono chiamati P2P Client.

Le funzionalità del GO e del Client sono dinamiche e sono negoziate nel momento dell'inizializzazione della rete. Due P2P device si trovano l'un l'altro, negoziano i loro ruoli e formano il gruppo P2P. Una volta stabilito il gruppo, altri P2P Client possono entrare a far parte del gruppo come in una normale rete Wi-Fi.

Ci sono diversi modi in cui due device possono formare un gruppo P2P. Abbiamo tre tipi di tecniche: Standard, Autonomous e Persistent.

Standard

In questo caso i device P2P devono prima trovarsi l'un l'altro, e poi "decidere" quale dispositivo sarà GO.

I dispositivi Wi-Fi Direct solitamente iniziano eseguendo una tradizionale scan Wi-Fi, per mezzo della quale si possono scoprire gruppi P2P già esistenti e reti Wi-Fi. Dopo questa scan un nuovo algoritmo di Discovery è avviato. Per prima cosa, un device P2P seleziona uno dei "Social Channels", tra i canali 1, 6 o 11 come il suo Canale di Ascolto. Dopodichè, il dispositivo alterna due stati:

- uno stato di ricerca, nel quale il dispositivo esegue una scansione attiva mandando una Probe Request in ognuno dei Social Channel.

- uno stato di ascolto, nel quale il device resta in attesa di Probe Request nel suo Canale di Ascolto.

Una volta che i due dispositivi P2P si sono trovati l'un l'altro, fanno partire la fase di Negoziazione del GO. Questa è implementata usando un three-way handshake, chiamato Go Negotiation Request/Response/Confirmation, dove i due device si accordano su quale dei due dovrà agire come GO e su quale canale lavorare. Allo scopo di scegliere quale dispositivo opererà come P2P GO, i dispositivi inviano un parametro numerico, il "GO Intent value", e il dispositivo che dichiara il valore più alto diventa P2P GO. Per prevenire conflitti quando due dispositivi dichiarano lo stesso GO Intent, un bit di parità è incluso nella Go Negotiation Request, che è settato in maniera casuale ogni volta che una GO Negotiation Request è mandata.

Autonomous

In questo caso, invece, un device decide in modo autonomo di essere GO. Per rendere visibile il gruppo appena formato, il dispositivo invia periodicamente messaggi di Beacon. Gli altri dispositivi possono scoprire il gruppo appena formato facendo una tradizionale scansione e successivamente si collegano al gruppo. In questo caso non c'è nessuna Go Negotiation e, inoltre, chi forma il gruppo non deve alternare i due stati di ascolto e ricerca.

Persistent

Durante il processo di formazione del gruppo, i device P2P possono dichiarare un gruppo come persistent, usando un flag presente nel pacchetto Beacon, nel Probe Responses e nel Go negotiation.

In questo modo, i device che formano il gruppo immagazzinano le credenziali e i ruoli della rete creata, per le seguenti re-instanziazioni del gruppo. Specificamente, dopo la fase di Discovery, se un device P2P riconosce che in passato aveva formato un gruppo di tipo persistent col corrispettivo device trovato, uno dei due dispositivi può usare la Invitation Procedure (un two-way handshake) per re-instanziare rapidamente il gruppo.

1.2.2 Wi-Fi Direct su Android

Dalla versione 4.0 di Android (API level 14), il Wi-Fi Direct è stato reso utilizzabile sui device Android, con l'hardware appropriato, attraverso le API del Wi-Fi P2P.

Le API del Wi-Fi P2P sono costituite dalle seguenti parti principali:

- **Metodi** che permettono la ricerca, la richiesta e la connessione ai nodi definiti nella classe **WifiP2pManager**;
- **Listener** che permettono di essere notificati in caso di successo o fallimento dei metodi della classe WifiP2pManager;
- **Intent** che notificano uno specifico evento riguardante il Wi-Fi P2P rilevato dal device, come ad esempio l'avvenuta connessione ad un peer o la scoperta di nuovi peer.

Di solito si utilizzano queste tre componenti principali contemporaneamente.

Esposti nella figura sottostante troviamo i metodi della classe WifiP2pManager.

Method	Description
`initialize()`	Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.
`connect()`	Starts a peer-to-peer connection with a device with the specified configuration.
`cancelConnect()`	Cancels any ongoing peer-to-peer group negotiation.
`requestConnectInfo()`	Requests a device's connection information.
`createGroup()`	Creates a peer-to-peer group with the current device as the group owner.
`removeGroup()`	Removes the current peer-to-peer group.
`requestGroupInfo()`	Requests peer-to-peer group information.
`discoverPeers()`	Initiates peer discovery
`requestPeers()`	Requests the current list of discovered peers.

Figura 1.2: Metodi della classe WifiP2pManager. Sorgente: [4]

Ad ognuno di questi metodi è collegato un listener, cosicché si potrà essere avvisati dell'avvenuto successo o fallimento della chiamata del corrispettivo metodo. I vari listener sono riportati nella figura 1.3.

Listener interface	Associated actions
`WifiP2pManager.ActionListener`	`connect()`, `cancelConnect()`, `createGroup()`, `removeGroup()`, and `discoverPeers()`
`WifiP2pManager.ChannelListener`	`initialize()`
`WifiP2pManager.ConnectionInfoListener`	`requestConnectInfo()`
`WifiP2pManager.GroupInfoListener`	`requestGroupInfo()`
`WifiP2pManager.PeerListListener`	`requestPeers()`

Figura 1.3: Listener della classe WifiP2pManager. Sorgente: [4]

Intent	Description
`WIFI_P2P_CONNECTION_CHANGED_ACTION`	Broadcast when the state of the device's Wi-Fi connection changes.
`WIFI_P2P_PEERS_CHANGED_ACTION`	Broadcast when you call `discoverPeers()`. You usually want to call `requestPeers()` to get an updated list of peers if you handle this intent in your application.
`WIFI_P2P_STATE_CHANGED_ACTION`	Broadcast when Wi-Fi P2P is enabled or disabled on the device.
`WIFI_P2P_THIS_DEVICE_CHANGED_ACTION`	Broadcast when a device's details have changed, such as the device's name.

Figura 1.4: Intent della classe WifiP2pManager. Sorgente: [4]

Le API del Wi-Fi Direct definiscono anche degli Intent che sono mandati dal sistema quando accadono certi eventi riguardanti il Wi-Fi P2P, come ad esempio quando un nuovo peer è scoperto o quando cambia lo stato del Wi-Fi del dispositivo. Un'applicazione può rilevare questi eventi creando un broadcast receiver che riceva gli intent riportati nella figura 1.4.

1.3 Related Works

Vari studi sono stati fatti per cercare il modo migliore di creare una soluzione al fine di fornire un mezzo di comunicazione multi-hop per scopi di soccorso post situazioni di disastro. Sono state proposte varie soluzioni tra cui architetture e insiemi di protocolli di routing [11] [12], una sensor-network con

strategie di raccolte dati e di routing finalizzate anche al risparmio energetico [13]; alcuni studi propongono l'utilizzo dei TV White Space per favorire le comunicazioni [14] [15].

Si sono fatti anche studi, con relativi contributi, sulla formazione di reti wireless "spontanee", formate da device mobili di utenti finali, quali smartphone e tablet, sempre allo scopo di favorire i soccorsi e dare adeguato aiuto. Uno dei contributi dati è un'architettura pervasiva che prevede la creazione di una rete ad-hoc dinamica tra i membri delle squadre di soccorso in possesso di device mobili [16] [17]. Sono state proposte anche varie applicazioni per fornire aiuto durante i momenti di post-disastro, come illustrato negli articoli [18] [19].

Tra gli studi fatti troviamo [20], [21] e [22] che si soffermano a studiare e valutare una STEM-Net, ossia una rete auto-adattiva, nella quale i suoi nodi, chiamati STEM-Node, cambiano il loro comportamento a seconda sia di vari fattori interni al device sia di vari fattori esterni quali il carico della rete, lo scopo che si vuole raggiungere, ecc..

Da quando il Wi-Fi Direct è stato introdotto sono stati fatti vari studi su di esso. Tra questi possiamo menzionare il lavoro [5]. In questo articolo si fa una valutazione sperimentale delle performance del Wi-Fi Direct in uno scenario reale, in particolare vengono mostrati nei risultati il delay fra la reciproca scoperta dei dispositivi e la reale connessione e le performance del protocollo per il risparmio della batteria.

Anche nell'articolo [6] vengono studiate le tempistiche relative alle varie fasi del Wi-Fi Direct, facendo variare il numero di dispositivi fino ad un massimo di sei.

In questi articoli, per attuare la connessione tra i device era sempre necessaria l'interazione con l'utente.

Negli articoli [7], [8], [9], viene mostrato il concetto di una hybrid cellular mobile adhoc network (Hybrid cellular MANET), implementando un sistema funzionante di microblogging per smartphone e tablet che utilizza il Wi-Fi Direct per la connessione tra i vari dispositivi. In questi lavori i device

riescono a connettersi in modo autonomo l'un l'altro ma per fare ciò hanno bisogno del root.

A differenza degli articoli precedenti, nell'articolo [10], gli autori sono riusciti a creare una rete di device in maniera autonoma, senza che i device abbiano bisogno del root, e a far comunicare più gruppi di dispositivi. In questo articolo, le sperimentazioni sono state fatte solo con sei dispositivi.

Capitolo 2

L'applicazione

2.1 Scenario/Obiettivo

L'obiettivo della mia applicazione è quello di formare una rete di device in modo autonomo, in cui un messaggio di aiuto mandato da un utente che necessita soccorso, si potrà propagare fino ad arrivare ad un device che abbia connessione 3G attiva e, quindi, la possibilità di comunicare con il server al quale manderà il messaggio.

I device Android sono progettati per essere dei dispositivi user-driven, ossia al centro di tutto vi è l'interazione con l'utente.

L'unico modo documentato per formare un gruppo di dispositivi peer-to-peer tra device Android è basato, anch'esso, sull'interazione con l'utente, infatti per procedere con la connessione e con l'autenticazione, una qualsiasi applicazione che sfrutta il Wi-Fi Direct deve attendere l'input e la conferma da parte di questo.

L'esigenza di avere periodiche interazioni con l'utente per formare le reti di dispositivi non è compatibile con lo scenario di disastro per il quale questa applicazione è stata pensata.

2.2 Realizzazione

L'implementazione del Wi-Fi Direct su Android permette la creazione di un Soft Access Point, che creerà una rete alla quale più dispositivi potranno connettersi conoscendo il SSID e la PreShared Key della rete.

Dato che sia il SSID che la Preshared Key non possono essere decisi dal programmatore, ma sono generati in maniera casuale dal sistema, questi devono essere mandati ai device che si vorranno connettere.

Queste due informazioni dovranno essere immagazzinate in un pacchetto che verrà mandato in broadcast attraverso la rete. Tutti i dispositivi, nel raggio del dispositivo che agisce da Access Point, riceveranno questo pacchetto e potranno connettersi contemporaneamente all'AP tramite Wi-Fi senza nessuna interazione da parte dell'utente.

2.2.1 Componenti Principali

Si può suddividere questa applicazione in cinque componenti principali:

- **Main Activity**: ha il compito di permettere all'utente di mandare il messaggio, far partire il Main Service e mostrare alcuni messaggi di log;
- **Main Service**: è la componente principale della mia applicazione. Si occupa di gestire l'avvicendamento delle fasi di Group Owner e Client, le quali verranno esposte nel dettaglio nella sezione successiva;
- **Service For Communication With Server**: è un service attivo solo nei dispositivi con connessione 3G attiva. Ha il compito di mandare periodicamente al server i messaggi arrivati fino a quel momento al dispositivo su cui è attivo, prendere le risposte di avvenuta ricezione e i messaggi preimpostati da parte dei soccorritori;
- **Broadcast Receiver**: si occupa di ricevere i broadcast da parte del sistema come il cambiamento di stato (attivo/disattivo) del Wi-Fi Direct, l'avvenuta trasformazione in Group Owner, o l'avvenuta connessione alla rete del GO selezionata in precedenza;

- **Database Manager**: si occupa della gestione del database dell'applicazione.

Come supporto all'applicazione si è realizzato un sito web e un server online, che verranno illustrati nel capitolo 3.

All'avvio l'applicazione si presenta in questo modo:

Figura 2.1: Main Activity

Tramite questa schermata l'utente potrà digitare il suo messaggio di aiuto nella textbox e cliccando il bottone "Invia" salverà il messaggio nel Database

dell'applicazione, facendo uso di una funzione del DB Manager. Insieme al testo del messaggio verrà salvata la posizione dell'utente.

2.2.2 Le fasi dell'applicazione

Per far partire "davvero" l'applicazione si dovrà premere il pulsante "Parti".

Una volta premuto partirà il Main Service che effettuerà ciclicamente una serie di operazioni rappresentate nella seguente figura:

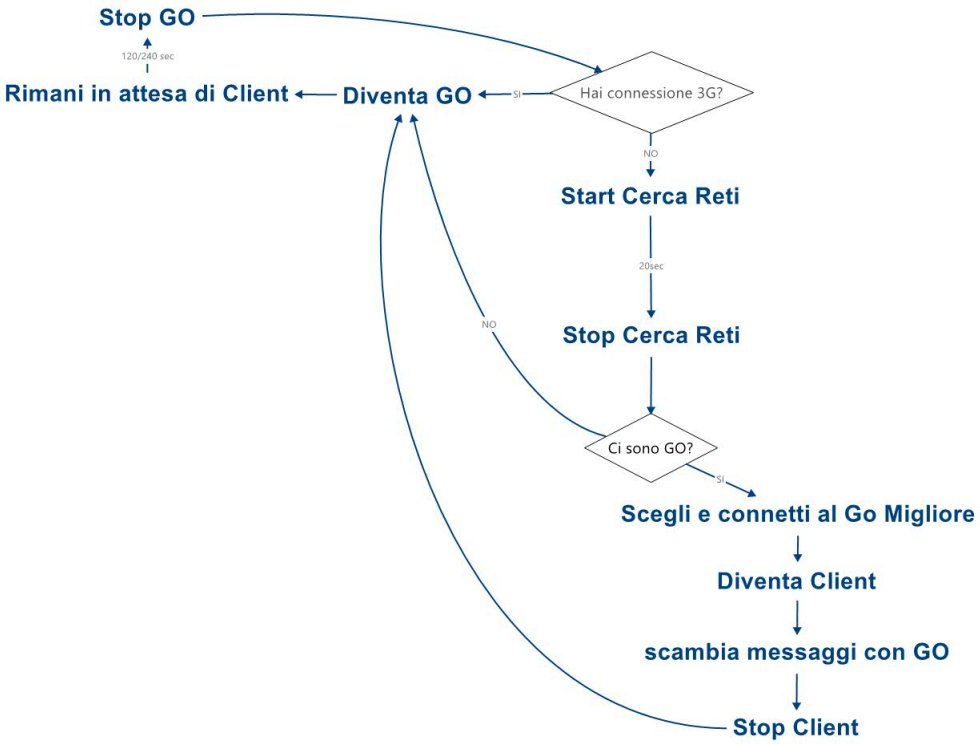


Figura 2.2: Stati applicazione. [23]

Appena il Main Service parte, ci sarà la fase di inizializzazione; viene definito il canale di comunicazione attraverso il quale l'applicazione comunicherà col Wi-Fi P2P framework e viene inizializzato il Broadcast Receiver, impostandolo a ricevere gli intent da parte del sistema di tipo:

- **WIFI_P2P_STATE_CHANGED_ACTION** per conoscere lo stato del Wi-Fi P2P del dispositivo (attivo/disattivo);
- **WIFI_P2P_CONNECTION_CHANGED_ACTION** per sapere quando il dispositivo è diventato Group Owner del gruppo e per ricavare le informazioni relative a questo quali il SSID e la PreShared Key;
- **SUPPLICANT_STATE_CHANGED_ACTION** per essere informati di quando il dispositivo si connette ad una rete.

Dopodiché il service verificherà la presenza di connessione 3G attraverso una richiesta http. Se questa va a buon fine significherà che il device ha una connessione 3G funzionante e andrà subito nella fase Diventa GO, nella quale partirà anche il Service che gestirà la comunicazione col server. Al contrario, se la richiesta http fallisce, si andrà nella fase Diventa Client.

Fase Diventa GO

Attraverso la funzione **createGroup()** si manda al framework del Wi-Fi P2P una richiesta di creazione di un nuovo gruppo. In questo modo si vuole creare un gruppo che ha come GO il dispositivo in questione e che agisce come Soft Access Point.

Dopo aver chiamato la funzione, si utilizza un timer per impostare quanto dovrà durare questa fase. Se il dispositivo ha connessione 3G allora questa fase durerà 240 secondi, altrimenti 120. La temporizzazione avviene tramite uno **ScheduledExecutorService**.

Il Broadcast Receiver, una volta catturato il messaggio da parte del sistema che notificherà l'avvenuta creazione del gruppo, salverà in locale il nome di questo, ossia il SSID della rete, e la Preshared Key necessarie per il collegamento al device. Ora il device registrerà la sua presenza sulla rete attraverso le funzioni **WifiP2pDnsSdServiceInfo.newInstance()** e **addLocalService()**.

Oltre al SSID e alla Preshared Key, il GO manderà anche un'altra informazione: un valore numerico che indica la sua distanza dal nodo che ha la

connessione dati attiva, ossia il numero di dispositivi dai quali il messaggio di aiuto deve passare prima di arrivare al dispositivo con il 3G e quindi arrivare al server. Questo valore è impostasto inizialmente a -1 se il dispositivo non ha connessione o a 0 se ce l'ha. Ogni volta che il dispositivo si connetterà ad un GO con distanza diversa da -1, la sua distanza verrà cambiata salvando il valore della distanza del GO a cui si è connessi incrementata di una unità.

Nel frattempo si fa partire un Thread che istanzia un ServerSocket e rimane in attesa dei Client.

Ogni volta che un Client si collegherà al dispositivo, partirà un altro Thread e comincerà lo scambio di messaggi. Questa fase verrà in seguito descritta nel dettaglio.

Una volta finito il tempo a disposizione del Go, si partirà dall'inizio rifacendo il controllo della connessione.

Service For Communication With Server

Una volta partito, attraverso una funzione del DB Manager, si prenderanno tutti i messaggi da parte degli utenti arrivati fino a quel momento al dispositivo e tramite una richiesta HTTP di tipo POST si manderanno uno alla volta al server web. Se la richiesta sarà andata a buon fine allora si salverà nel DataBase una risposta di avvenuta ricezione per ogni messaggio, la quale sarà mandata indietro nella rete di dispositivi. Una volta completata questa fase, il service farà una richiesta di tipo GET, aggiungendo ai parametri della richiesta la sua posizione, per ricevere i messaggi da parte dei soccorritori, relativi ai messaggi di aiuto nel raggio di 500 metri dal dispositivo. La procedura si ripeterà ciclicamente facendo uso di un ScheduledExecutorService.

Fase Diventa Client

Il primo passo è quello di instanziare una **WifiP2pDnsSdServiceRequest** e i vari Listener necessari per la ricezione dei pacchetti contenenti le informazioni dei vari GO. Il dispositivo rimarrà in ascolto per un tempo pari a

20 secondi e ogni qualvolta catturerà un messaggio da un GO memorizzerà le informazioni relative a questo.

Una volta terminato il tempo a disposizione, si procede alla scelta del GO migliore.

Se non si è ricevuto nessun pacchetto, allora si procede con la Fase Diventa GO. Se, invece, sono stati ricevuti pacchetti si chiede al sistema, attraverso la funzione **getScanResults()** della classe WifiManager, l'ultima scan disponibile delle reti WI-Fi, nella quale compariranno anche le reti formate dai vari GO che il dispositivo riesce a "vedere". Da questa lista viene presa la potenza del segnale della varie reti formate dai GO da cui il dispositivo ha ricevuto i pacchetti.

Dopodiché si creeranno due liste: una con tutti i GO che hanno una distanza maggiore o uguale a 0, ossia una lista con tutti i dispositivi attraverso i quali il messaggio riuscirà ad arrivare al server e una lista con i GO con distanza pari a -1.

Se la prima lista non è vuota allora si prenderanno tutti i Go con distanza pari a zero, ossia i GO che hanno connessione 3G attiva. Tra questi si sceglierà quello che avrà la potenza del segnale maggiore tra tutti e superiore ad una soglia preimpostata di potenza minima pari al 30 per cento. Se nessun Go passerà questa prima selezione, si andrà a cercare il GO migliore con soglia sempre maggiore al 30 per cento e con distanza compresa tra il 30, poi 60 e infine 100 per cento della differenza tra la distanza massima presente nella lista e quella minima. Se anche stavolta nessun GO rientra nei parametri, allora, si sceglierà il GO più potente senza tener conto né della distanza, né della soglia di potenza minima.

Se la prima lista è vuota allora si passa alla lista dei dispositivi che hanno distanza -1. Si fa una prima selezione scorrendo la lista e chiedendo al database se il SSID del GO selezionato è presente nella lista delle ultime reti a cui ci si è connessi. Se non è presente allora è un dispositivo candidato a cui connettersi. Una volta finito di scorrere la lista, se il risultato di questa scrematura non è nullo, allora ci si connette al più potente tra i candidati, al-

trimenti non si tiene conto della presenza nel Database e viene scelto sempre il più potente.

Può succedere che i Go che hanno mandato i pacchetti non siano presenti nella lista delle connessioni Wi-Fi disponibili. In questo caso si inizia la fase Diventa GO.

Una volta scelto il GO migliore si inizia la procedura di connessione. Si crea una variabile di tipo **WifiConfiguration** e le si assegnano il SSID del GO e la Preshared Key. Attraverso la funzione **addNetwork()** si aggiunge la rete definita dalla precedente variabile. Dopodiché si attiva la connessione facendo uso della funzione **enableNetwork()**. A questo punto si fa partire un timer settato a 40 secondi per dire quando terminare la Fase di Client e cominciare la Fase di Go.

Una volta fatte queste operazioni, il Broadcast Receiver rimarrà in attesa del messaggio del sistema di avvenuta connessione alla rete Wi-Fi e, una volta verificato che ci si è davvero connessi alla rete del GO scelto, farà partire un Thread, il quale istanzierà un socket per collegarsi a quello del GO, che era in attesa di connessioni. Ora che il Client si è connesso al GO, può iniziare la Fase dello scambio di messaggi e risposte.

Fase dello scambio di messaggi

Durante questa fase al dispositivo possono arrivare tre tipologie di messaggi:

- i messaggi di aiuto generati dagli utenti ;
- i messaggi di avvenuta ricezione generati ogni qual volta che un messaggio arriva al server;
- i messaggi preimpostati da parte dei soccorritori;

La struttura dei messaggi degli utenti è la seguente:

- **tipo**: un intero pari a zero;

- **idMex**: è una stringa formata dall'id del dispositivo che ha generato il messaggio e il timestamp in cui è stato generato;
- **timestampOrigine**: è il timestamp della creazione del messaggio;
- **testo**: una stringa contenente il messaggio digitato dall'utente;
- **latitudine**;
- **longitudine**;

I messaggi di avvenuta ricezione e i messaggi da parte dei soccorritori hanno la stessa struttura dei messaggi degli utenti, solo che il tipo è pari, rispettivamente, a uno o due; il testo del messaggio è un testo preimpostato per entrambi i tipi e la latitudine e la longitutine sono settate a zero.

Nel Database, oltre a questi campi, viene memorizzato anche il timestamp di ricezione del messaggio.

Quando entrambi i dispositivi, GO e Client, hanno effettuato la connessione tramite socket, allora entrambi, attraverso una **write** mandano il loro id. Dopodiché chiedono al Database il timestamp dell'ultimo aggiornamento che hanno avuto dal dispositivo opposto. Se con tale dispositivo non c'è mai stato uno scambio di messaggi, il Database ritornerà zero, altrimenti il timestamp corrispondente all'id. Una volta ottenuto verrà mandato al dispositivo opposto e si attenderà che l'altro dispositivo mandi a sua volta il suo timestamp. A questo punto si prende il timestamp attuale e lo si manda all'altro dispositivo e si attende che faccio lo stesso.

Ora partiranno due Thread, uno per la gestione dei messaggi in uscita e l'altro per la gestione dei messaggi in entrata. Il primo chiederà al Database i vari tipi di messaggi che hanno un timestamp di ricezione maggiore o uguale rispetto al timestamp dell'ultimo aggiornamento da parte del dispositivo (mandato precedentemente dal dispositivo opposto); se questo timestamp è pari a zero allora si prenderanno tutti i messaggi prensenti nel Database. I vari campi dei messaggi vengono concatenati in un'unica stringa, separandoli tramite uno /. Una volta avuta la lista dei messaggi si comincerà ad inviare

all'altro dispositivo. Dopo che si invierà l'ultimo messaggio, si manderà il messaggio finale definito da una stringa con valore 123. Per quanto riguarda il secondo Thread si limiterà a leggere e memorizzare in una lista temporanea i messaggi mandati dall'altro dispositivo fino a quando non leggerà il messaggio finale.

Una volta conclusi i due Thread si inseriscono nel Database tutti i messaggi letti dal secondo di questi e si imposta come timestamp di ricezione il secondo timestamp mandato dall'altro dispositivo.

Durante la memorizzazione dei messaggi di tipo 1, ossia messaggi di avvenuta ricezione, se è presente il messaggio dell'utente con lo stesso valore del campo idMex allora quest'ultimo verrà eliminato dal Database. Lo stesso avviene durante la memorizzazione dei messaggi di tipo 2, ma questa volta vengono eliminati sia i messaggi di tipo 0 che di tipo 1. In questi due casi, se nell'idMex del messaggio è presente, come sottostringa, l'id del dispositivo su cui è arrivato, l'utente sarà avvisato tramite notifica che il suo messaggio è stato ricevuto dal server, oppure che qualche soccorritore ha preso in carica il suo stato di aiuto e che si sta azionando per salvarlo.

Dopo aver fatto ciò, se il dispositivo era nella fase GO continua ad attendere altri Client; invece, se il dispositivo era nella fase Client, allora comincia la Fase Diventa GO.

Capitolo 3

Il server e la pagina web

Come già descritto, una volta che i messaggi arrivano ad un device che ha una connessione dati attiva, vengono mandati al server web. Questi messaggi verranno visualizzati sulla pagina web ltw1528.web.cs.unibo.it, tramite la quale i soccorritori potranno vedere la lista dei messaggi, mandare un messaggio preimpostato per dire all'utente che i soccorsi si stanno mobilitando ed archiviare il messaggio una volta che il soccorso è andato a buon fine.

3.1 Il Server

Nel database verranno memorizzati i messaggi provenienti dai dispositivi con connessione. La struttura della tabella nella quale verranno salvati è la seguente:

- **id**: un intero che indica l'identificatore univoco all'interno del Database;
- **status**: un intero che indica lo stato in cui si trova il messaggio; possiamo avere tre diversi stati:
  - **status 0**: indica che il messaggio è stato appena ricevuto dal server e che l'utente che ha inviato tale messaggio è in una situazione di bisogno;

  - **status 1**: indica che un soccorritore è andato in aiuto dell'utente che ha mandato il messaggio;
  - **status 2**: indica che un soccorritore è riuscito a svolgere il suo compito e il messaggio è stato archiviato;

- **idMex**: l'identificatore univoco del messaggio
- **timestamp**: il timestamp di ricezione del messaggio;
- **testo**: il testo digitato dall'utente;
- **latitudine**;
- **longitudine**;

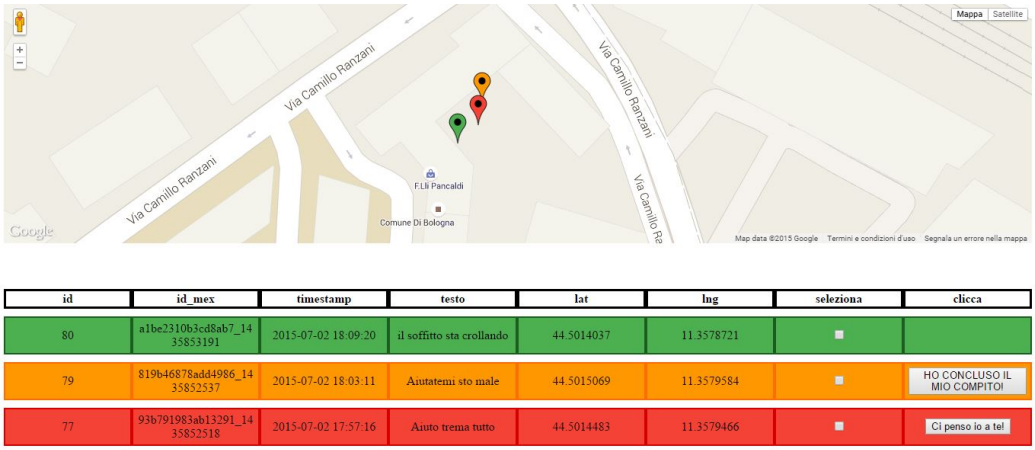
All'interno del Database vi è anche una tabella in cui verranno memorizzati i messaggi preimpostati da parte dei soccorritori, ogni qual volta uno di questi prenderà in incarico l'utente, o un gruppo di utenti, che hanno mandato i messaggi di aiuto. La struttura di questa tabella è la seguente:

- **id**: l'identificatore univoco della riga;
- **idMex**: lo stesso del messaggio di aiuto a cui questa risposta si riferisce;
- **testo**: il messaggio preimpostato;
- **timestamp**: il timestamp dell'origine del messaggio;
- **latitudine**;
- **longitudine**;

Questi ultimi due campi si riferiscono alla latitudine e alla longitudine dell'utente che ha inviato il messaggio e vengono salvate in quanto, quando il service del dispositivo con la connessione dati attiva chiederà al server i messaggi da parte dei soccorritori, verranno dati quelli che rientrano in un certo raggio a partire dalla posizione del dispositivo.

3.2 Sviluppo Sito Web

Il sito si presenta come una semplice pagina web: nella parte superiore di questa vi è una mappa dove verranno visualizzati dei marker che indicheranno la posizione degli utenti che hanno mandato i messaggi di aiuto; nella parte inferiore invece comparirà una tabella. Questa verrà costruita dinamicamente tramite richieste AJAX di tipo GET al server, fatte periodicamente ogni dieci secondi.



id	id_mex	timestamp	testo	lat	lng	seleziona	clicca
80	a1be2310b3cd8ab7_1435853191	2015-07-02 18:09:20	il soffitto sta crollando	44.5014037	11.3578721	☐	
79	819b46878add4986_1435852537	2015-07-02 18:03:11	Aiutatemi sto male	44.5015069	11.3579584	☐	HO CONCLUSO IL MIO COMPITO!
77	93b791983ab13291_1435852518	2015-07-02 17:57:16	Aiuto trema tutto	44.5014483	11.3579466	☐	Ci penso io a te!

Figura 3.1: Sito web

Come si può vedere dall'immagine le righe della tabella possono avere colori diversi, i quali indicano lo stato del messaggio:

- **rosso** se il messaggio ha stato zero;
- **arancione** se ha stato uno;
- **verde** se ha stato due

Anche i marker relativi ai messaggi avranno colori diversi a seconda dello stato del messaggio che rappresentano. Anche il testo del pulsante, posizionato nell'ultima colonna a destra della tabella, e le successive azioni corrispondenti cambieranno a seconda dello stato.

Il soccorritore che vedrà un messaggio con stato zero, ossia una riga rossa della tabella, potrà cliccare il pulsante con testo “Ci penso io a te!”; in questo caso un messaggio preimpostato verrà salvato nel Database e si cambierà lo stato del messaggio da zero a uno. Il soccorritore potrà anche selezionare contemporaneamente più messaggi e salvare i relativi messaggi. .

Una volta concluso il suo compito potrà tornare sul sito e cliccare sul pulsante “Ho concluso il mio compito!” per archiviare il messaggio.

Capitolo 4

Studio simulativo

Dopo aver sviluppato l'applicazione, si sono fatti alcuni test con tre dispositivi Android, uno dei quali avente connessione dati attiva. Questi riuscivano a comunicare tra loro e scambiarsi le varie tipologie di messaggio (messaggio da parte dell'utente, messaggio di avvenuta ricezione, messaggio dai soccorritori).

Pur avendo più dispositivi a disposizione rispetto a quelli usati per i test, sarebbe complicato testare la scalabilità dell'applicazione su uno scenario reale. Perciò si è deciso di simulare lo scenario e l'applicazione tramite il programa OMNeT++ [24] e il framework Mixim [25].

4.1 OMNeT++

OMNeT++ è un framework modulare ed estendibile composto da librerie in C++ per lo sviluppo di simulazioni ad eventi discreti. Funzionalità specifiche come il supporto per sensor-network, reti wireless ad-hoc, protocolli Internet ecc., sono fornite tramite modelli framework, sviluppati come progetti indipendenti. È un prodotto open-source nato nel 2003, che può essere usato sotto licenza GNU, ovvero senza scopi di lucro, e che viene principalmente usato in ambito di ricerca accademica.

Una delle sue principali caratteristiche è la modularità dei suoi componenti, ossia ogni entità presente è rappresentata mediante un modulo, e tutti i moduli sono organizzati in maniera gerarchica e comunicano tra di loro tramite scambio di messaggi attraverso porte e connessioni. I moduli sono ottimamente realizzati e totalmente riutilizzabili al fne di combinarli tra loro per ottenere la configurazione desiderata.

Le simulazioni possono essere fatte in diverse interfacce utente, sia grafiche, che servono sopratutto per dimostrazioni e per debugging, sia a linea di comando.

Omnet fornisce anche due tool grafici, PLOVE e SCALARS, per visualizzare il contenuto degli output della simulazione scritti su appositi file dati generati automaticamente, ma anche estendibili dall'utente.

La sua area di applicazione primaria è la simulazione di network di comunicazione, ma grazie alla sua architettura è usato in altre aree tra cui:

- Modellazione di protocolli;
- Modellazione di code nelle reti;
- Modellazione di multiprocessori e altri sistemi hardware distribuiti;
- Convalida di architetture hardware;
- Valutazione delle performance dei sistemi software complessi;
- In generale, modellazione e simulazione di qualunque sistema ad eventi discreti con modalità basata su scambio di messaggi;

OMNeT++ mette a disposizione delle classi per modellare:

- moduli, porte, connessioni e parametri;
- messaggi e packets, questi ultimi sono un'estensione dei messaggi usati per le reti di comunicazione;
- array e code;

- classi per la collezione dei dati;
- statistiche e distribuzioni;

4.1.1 I Moduli

Come già detto, un modello implementato in OMNeT++ è costituito da una gerarchia di moduli annidati che comunicano tra loro attraverso lo scambio di messaggi. I moduli che racchiudono la vera e propria logica di tutto vengono chiamati **simple module**; sono scritti in C++ attraverso delle librerie di supporto messe a disposizione da OMNeT++. I simple modules possono essere raggruppati in **compound modules**, senza alcun limite di gerarchia, attraverso l'uso di un linguaggio di alto livello denominato **NED**. L'intero modello, chiamato **network**, è esso stesso un compound module. Nella figura sottostante, i rettangoli con sfondo grigio sono simple modules, mentre quelli con sfondo bianco sono compound modules. Le frecce che collegano i piccoli quadratini rappresentano, rispettivamente, le connessioni e le porte.

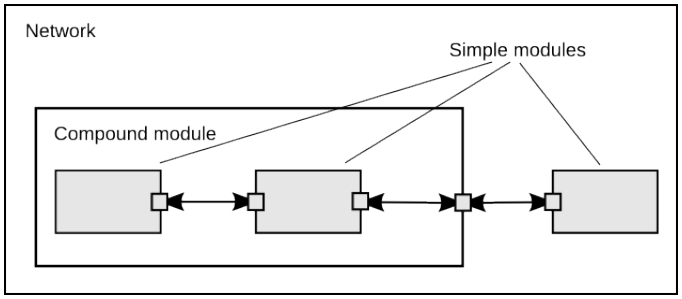


Figura 4.1: Simple e Compound Module

I moduli possono avere dei parametri, usati principalmente per passare dati di configurazione ai simple module e per aiutare a definire la topologia del modello di simulazione. I parametri possono essere stringhe, valori numerici o booleani. Questi possono essere assegnati nei file NED, nei file di configurazione o possono essere chiesti all'utente nel momento del lancio della simulazione.

4.1.2 I messaggi

I messaggi scambiati tra moduli possono contenere dati di arbitraria natura, in aggiunta agli attributi classici come il timestamp. I simple module, di solito, mandano messaggi attraverso le porte, ma è possibile mandarli direttamente al modulo di destinazione. Possono anche essere degli auto-messaggi; in questo caso vengono detti **self-message** e vengono gestiti attraverso timer. Le porte sono le interfacce di input e di output dei moduli: i messaggi vengono spediti attraverso una porta di output e vengono ricevuti attraverso una porta di input.

Gli header dei pacchetti sono descritti nei **Message Definition File**, ossia file semplici con estensione msg. Questi file vengono tradotti in classi e header C++ del tipo <nome_pacchetto>_m.h e <nome_pacchetto >_m.cc. Per poter utilizzare all'interno della propria applicazione i pacchetti definiti in questo modo bisogna includere l'header del file del pacchetto.

4.1.3 Il linguaggio NED

Il termine NED si riferisce a **Network Description**, ed è un linguaggio di alto livello usato dall'utente per descrivere la struttura di un modello.

Particolarità di questo linguaggio è la struttura ad albero.

Il NED permette di creare moduli semplici e di connetterli ed assemblarli tra loro così da formare moduli composti, ottenere sottoclassi, aggiungere parametri, porte e nuovi sotto-moduli.

4.1.4 I file .ini

Questo tipo di file contiene la configurazione ed i dati di input per le simulazioni. I dati al suo interno sono raggruppati in sezioni il cui nome, racchiuso tra parentesi quadre, indica l'identificatore univoco della simulazione. In questi file si fa riferimento ai parametri dei moduli attraverso il loro path completo o al loro nome gerarchico; quest'ultimo consiste in una

lista di nomi di moduli separati da un ".". I parametri vengono assegnati attraverso l'operatore "=" e la loro risoluzione avviene nel modo seguente:

- se il parametro è già assegnato nel NED, questo non può essere sovrascritto;
- se dopo l'operatore di assegnamento c'è un valore, allora questo viene assegnato come parametro;
- se dopo l'operatore di assegnamento è presente l'identificatore **default**, viene assegnato il valore di default per il tipo di parametro;
- se è presente l'identificatore **ask**, il valore da assegnare viene chiesto all'utente;
- se il parametro non viene assegnato in questo file, prenderà il valore di default, sempre se questo è definito;
- se non si verifica nessun caso di quelli precedenti, il parametro viene dichiarato **non assegnato** e verrà gestito a seconda della politica dell'interfaccia utilizzata.

4.1.5 MiXiM

MiXiM è un OMNeT++ framework di modellazione creato per simulare reti wireless mobili o stabili (wireless sensor networks, body area networks, ad-hoc networks, vehicular networks, etc.)

MiXim fornisce modelli dettagliati e protocolli, nonché infrastrutture di supporto. Questi possono essere divisi in cinque gruppi:

- **Modelli di Ambiente**: in una simulazione solo le parti rilevanti del mondo reale dovrebbero essere riflesse, come per esempio ostacoli che disturbano le comunicazioni wireless;

- **Connettività e Mobilità**: quando due moduli si muovono, la loro influenza sugli altri nodi nella rete cambia. La simulazione deve tener conto di questi cambiamenti e fornire adeguate rappresentazioni grafiche;

- **Ricezione e Collisioni**: per le simulazioni di tipo wireless, i movimenti di oggetti e nodi hanno grande influenza sulla ricezione dei messaggi. Il gestore della ricezione ha il compito di decidere come un segnale trasmesso subisce cambiamenti anche tenendo conto degli altri segnali;

- **Supporto alla Sperimentazione**: questo supporto è necessario per aiutare i ricercatori a comparare i risultati con uno stato ideale, aiutandoli a trovare un modello adatto alle loro implementazioni;

- **Librerie di Protocolli**: una ricca raccolta di librerie permette ai ricercatori di confrontare le loro idee con altre già implementate.

4.2 La simulazione

Per attenersi alla realtà si è deciso di simulare il nostro scenario con un numero di dispositivi proporzionale alle varie densità demografiche delle regioni italiane, partendo da quelle meno popolate sino ad arrivare a quelle con maggiore densità.

Sempre per attenersi ad uno scenario veritiero si è data una certa mobilità agli host. Quelli che avranno una mobilità pari a zero saranno gli host che avranno bisogno di aiuto e che, quindi, manderanno il messaggio di soccorso.

Gli host nelle varie simulazioni avranno una certa probabilità di avere il 3G e quindi dovranno anche simulare la comunicazione col server e la ricezione dei messaggi di avvenuta ricezione.

Rispetto all'applicazione, nella simulazione si è modificata in minima parte la fase Diventa Client. L'host avrà un tempo limitato in cui potrà diventare Client settato a venti secondi. L'host, in questi venti secondi, rimarrà in attesa dei messaggi di presenza dei GO per un tempo pari a 2, 5, 10 e 20 secondi.

Dopo la scelta del GO migliore, la connessione e lo scambio di messaggi, se il tempo limite, impostato a venti secondi, non è finito allora ricomincerà la ricerca dei GO.

La temporizzazione delle varie fasi dell'applicazione, fase GO e fase Client, viene effettuata attraverso schedulazione di automessaggi attraverso la funzione **scheduleAt()**.

Nell'host con indice zero ci sarà un modulo a cui tutti gli altri host manderanno dei messaggi utilizzando la funzione **sendDirect()**, la quale permette di mandare messaggi diretti tra moduli senza ritardi. Questi messaggi saranno mandati in tre occasioni differenti:

- ogni qualvolta un host crea il messaggio di aiuto;
- ogni volta che un host simula la generazione di un messaggio di avvenuta ricezione da parte del server;
- ogni volta che all'host che aveva generato il messaggio di aiuto arriva il messaggio del server con l'idMex contentente l'id dell'host;

Ogni volta che un messaggio di questi arriva al modulo, questo aumenterà il contatore relativo e lo salverà in un vettore tramite la funzione **record()**. Quando si arriverà ad avere lo stesso numero di messaggi di aiuto e di messaggi di avvenuta ricezione da parte del server arrivati agli host che hanno inviato il messaggio, allora il modulo chiamerà la funzione **endSimulation()**. Se questo non avviene, allora la simulazione terminerà una volta passati 5000 secondi.

4.2.1 I risultati

Come detto in precedenza il tempo di ricerca dei Group Owner è stato cambiato rispetto all'applicazione Android. Diminuendo il tempo di ricerca, inversamente, è aumentato il numero di connessioni tra Client e Group Owner; questo ha portato a far sì che il messaggio di aiuto creato da un utente in stato di bisogno arrivi prima al device con connessione 3G.

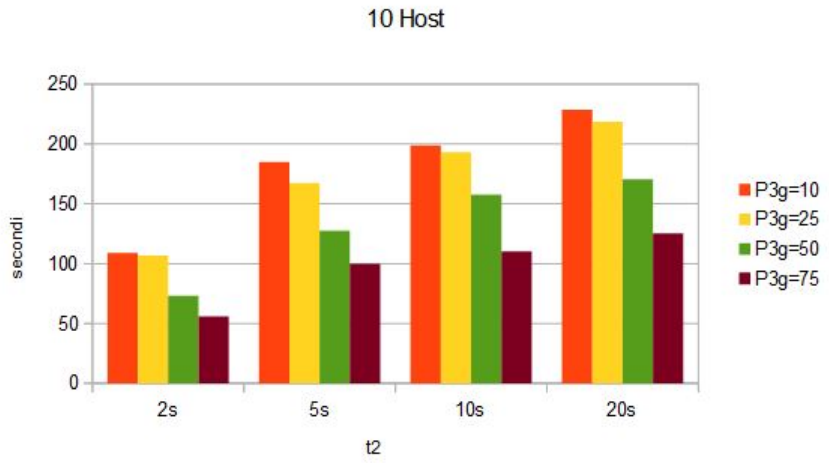


Figura 4.2: Delay medio tra la creazione di un messaggio di aiuto e l'arrivo ad un nodo con 3G

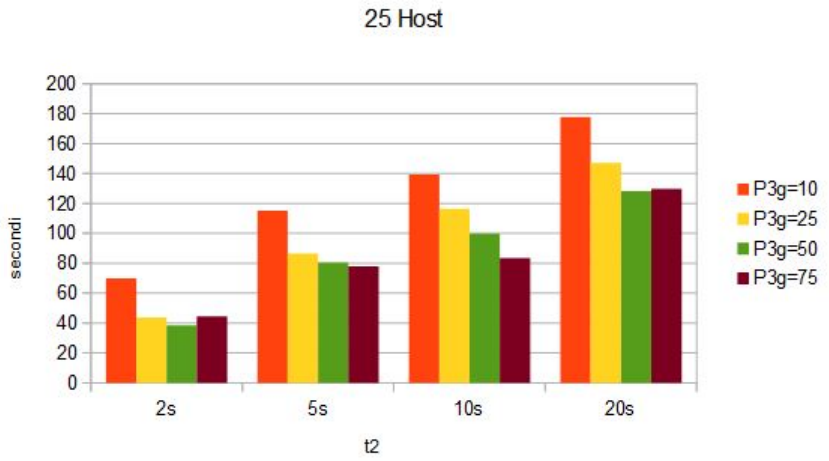


Figura 4.3: Delay medio tra la creazione di un messaggio di aiuto e l'arrivo ad un nodo con 3G

Questo si può vedere chiaramente dai grafici 4.2 e 4.3. Il primo mostra i risultati ottenuti facendo le simulazioni con 10 dispositivi, mentre il secondo con 25. Come ci si aspettava, aumentando il numero dei device il tempo che intercorre tra la creazione del messaggio di aiuto e l'arrivo ad un device con connessione 3G diminuisce. Possiamo vedere come il tempo diminuisca

anche all'aumentare della percentuale di device con una connessione 3G.

Più connessioni implicano un maggiore numero di messaggi scambiati tra gli host, come è possibile vedere dai grafici 4.4 e 4.5.

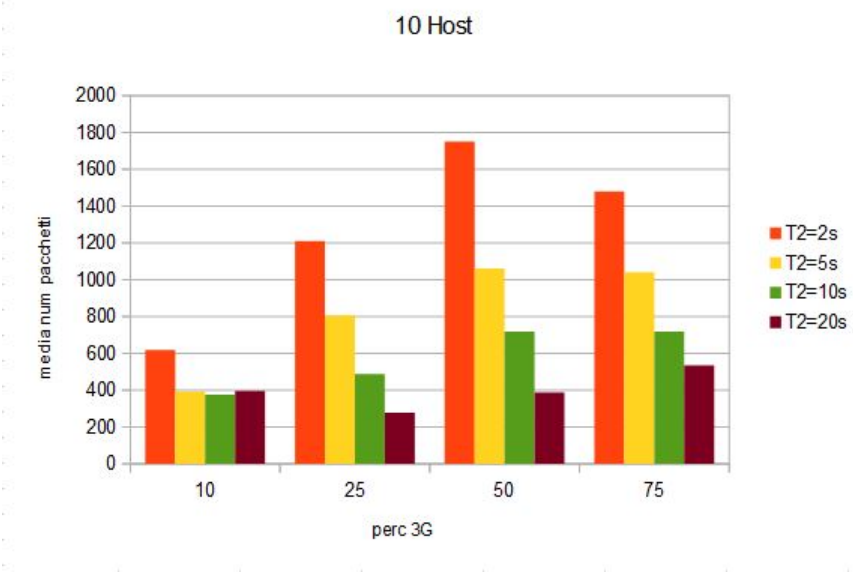


Figura 4.4: Media del numero di pacchetti inviati

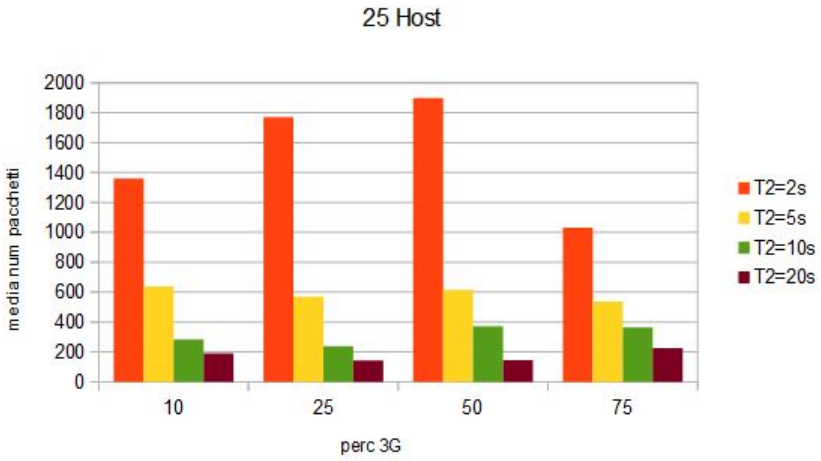


Figura 4.5: Media del numero di pacchetti inviati

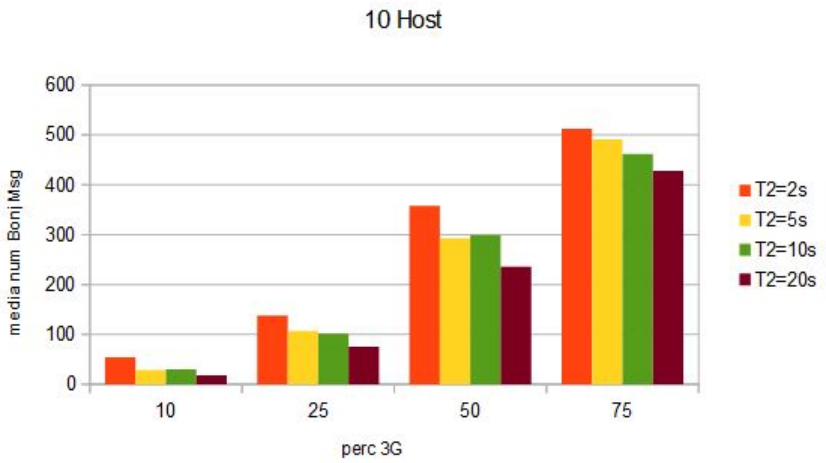


Figura 4.6: Media del numero di messaggi di Bonj inviati

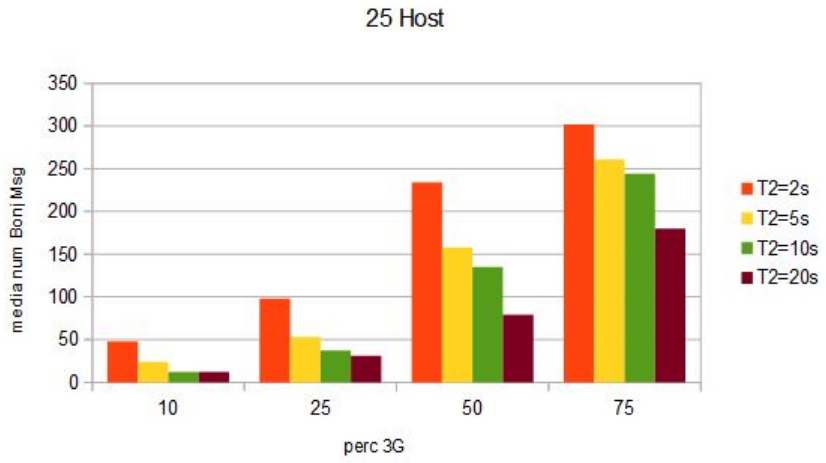


Figura 4.7: Media del numero di messaggi di Bonj inviati

I grafici 4.6,4.7 e 4.8,4.9 indicano la media, rispettivamente, del numero dei messaggi di Bonjour mandati da dispositivi aventi connessione 3G e del numero dei messaggi di aiuto scambiati tra i dispositivi non aventi una connessione.

Si è fatto anche uno studio sul consumo medio della batteria. Dai grafici 4.10 e 4.11 si può notare come il numero di pacchetti inviati, e quindi anche ricevuti, influisca sul batteria, infatti questo grafico ha lo stesso andamento

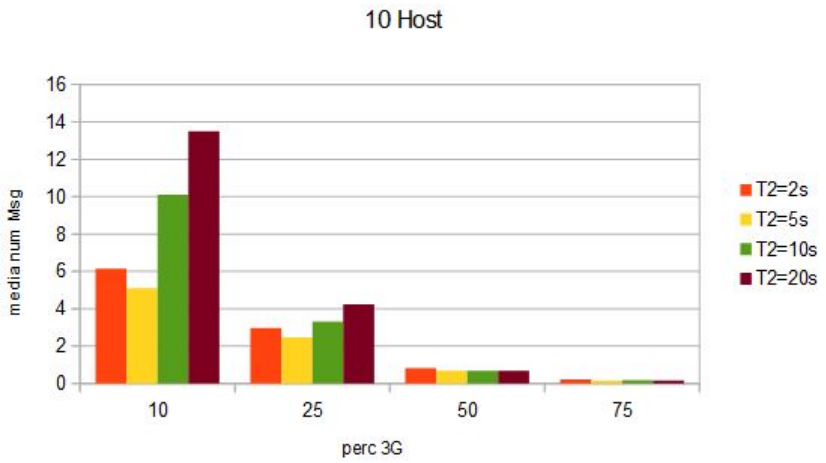


Figura 4.8: Media dei messagi inviati

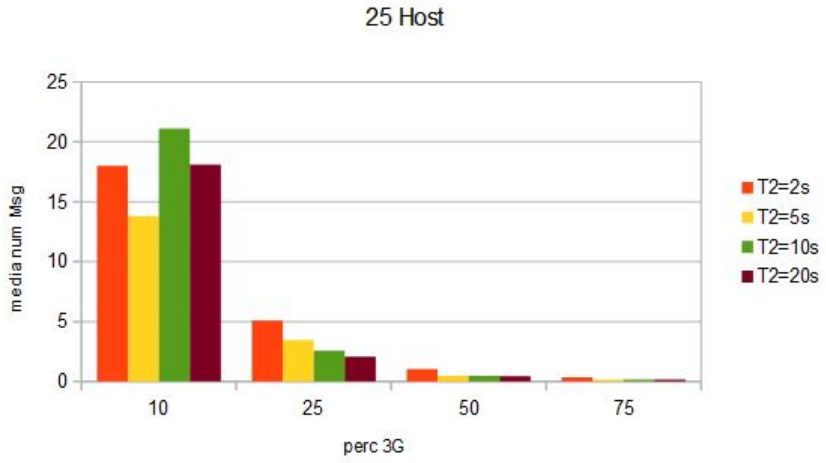


Figura 4.9: Media dei messagi inviati

di 4.4 e 4.11.

Si è anche studiato il delay medio tra host e host che intercorre tra l'invio di un messaggio e la sua ricezione 4.12.

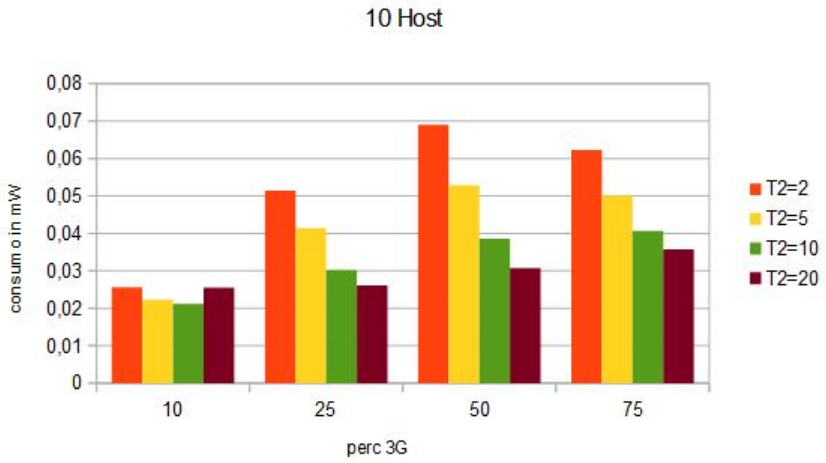


Figura 4.10: Consumo medio batteria

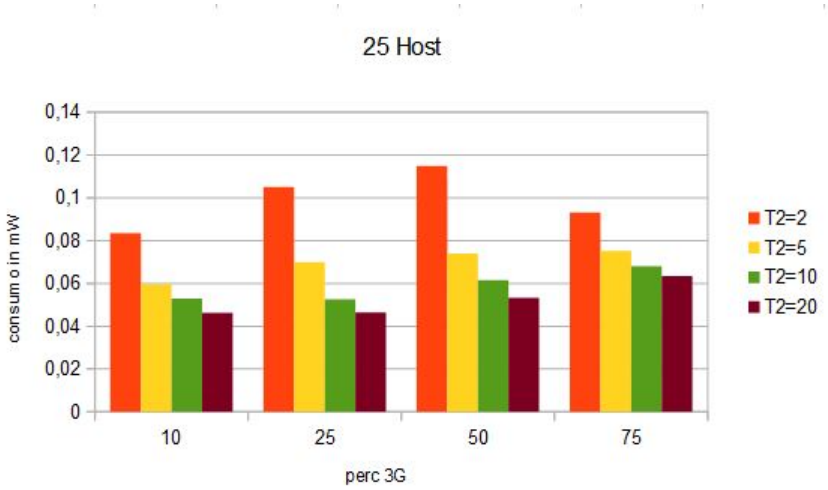


Figura 4.11: Consumo medio batteria

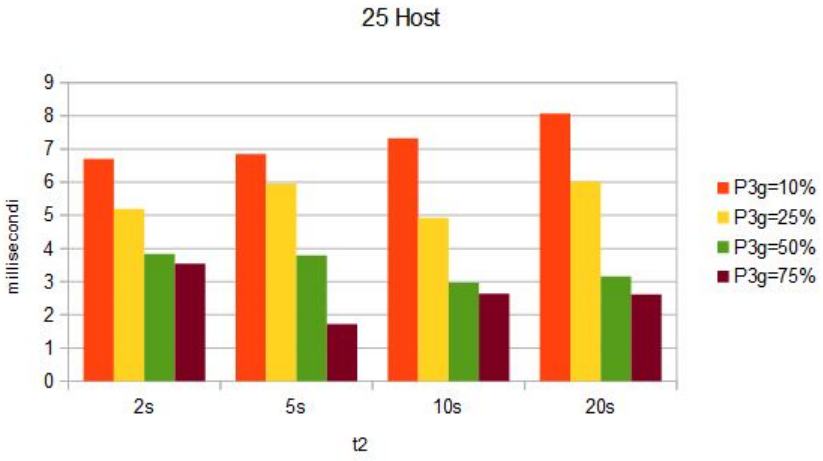


Figura 4.12: Delay tra host e host senza 3G

Conclusioni e sviluppi futuri

Con questa tesi si è voluto implementare un'applicazione che permetta agli utenti, in stato di bisogno, di comunicare con il mondo "esterno" anche in uno scenario post-disastro in cui le comunicazioni potrebbero essere difficoltose.

Attraverso le simulazioni si è visto come il meccanismo di comunicazione e scambio di messaggi implementato funzioni efficientemente anche con un numero crescente di dispositivi.

Molti sono i miglioramenti che si potrebbero apportare all'applicazione. Sarebbe molto interessante riuscire a implementare un'applicazione, basandosi su quella creata in questa tesi ed estendendola, per la creazione di una STEM-Net. Sfruttando le caratteristiche descritte negli articoli [20], [21] e [22], il device diventerebbe uno STEM-Node e in questo modo non ci sarebbero ciclicamente solo le due fasi Client e Group Owner ma il device deciderebbe cosa fare a seconda di vari fattori, in modo da aumentare l'efficienza sia in termini di prestazioni sia in termini di consumo energetico del device.

Come descritto, nella mia applicazione il messaggio di aiuto è caratterizzato solo da una stringa di testo immessa dall'utente. In un futuro, facendo riferimento al lavoro [19], si potrebbe espandere la mia applicazione aggiungendo varie caratteristiche :

- si dovrebbero aggiungere al messaggio varie categorie e sotto-categorie per descrivere la natura della richiesta;
- si dovrebbero creare profili per ogni utente in cui questo descriverà le sue competenze;

In questo modo quando un messaggio arriverà su un device di un utente che abbia le competenze necessarie per riuscire ad aiutare l'utente che ha mandato originariamente il messaggio, allora comparirà una notifica e i due utenti potranno interagire l'uno con l'altro. In questo modo anche un utente "normale" potrà aiutare altri utenti e facilitare il lavoro dei soccorritori.

Bibliografia

[1] https://code.google.com/p/androidteam/wiki/AndroidSystemArch

[2] http://developer.android.com/guide/components/fundamentals.html

[3] http://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[4] http://developer.android.com/guide/topics/connectivity/wifip2p.html

[5] Camps-Mur D. , Garcia-Saavedra A. , Serrano P. , *"Device-to-device communications with Wi-Fi Direct: overview and experimentation"*, Wireless Communications, IEEE (Volume:20 , Issue: 3), Giugno 2013

[6] Marco Conti, Franca Delmastro, Giovanni Minutiello, Roberta Paris , *"Experimenting opportunistic networks with WiFi Direct"*, Wireless Days (WD), 2013 IFIP, Novembre 2013

[7] Chenyu Zheng, Douglas C. Sicker, Lijun Chen, *"Self-organized context-aware hybrid MANETs"*, Wireless On-demand Network Systems and Services (WONS), 2013 10th Annual Conference on, Marzo 2013

[8] Chenyu Zheng, Douglas C. Sicker, Lijun Chen, *"Hybrid cellular-MANETs: An energy-aware routing design"*, Wireless On-demand Network Systems and Services (WONS), 2014 11th Annual Conference on, Aprile 2014

[9] Chenyu Zheng, Lijun Chen, Douglas Sicker, Xinying Zeng , *"Hybrid cellular-MANETs in practice: A microblogging system for smart devices*

in disaster areas", Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International, Agosto 2014

[10] Yufeng Duan, Carlo Borgiattino, Claudio Casetti, Carla Fabiana Chiasserini, Paolo Giaccone, Marco Ricca, *"Wi-Fi Direct Multi-group Data Dissemination for Public Safety"*, WTC 2014; World Telecommunications Congress 2014; Proceedings of, Giugno 2014

[11] S. M. George, W. Zhou, H. Chenji, M. Won, Y. O. Lee, A. Pazarloglou, R. Stoleru, P. Barooah, *"DistressNet: a wireless ad hoc and sensor network architecture for situation management in disaster response"*, Communications Magazine, IEEE (Volume:48 , Issue: 3), Marzo 2010

[12] N. Uchida, K. Takahata, Y. Shibata, *"Evaluation of cognitive wireless networks in rural area for disaster information network"*, Computational Science and Its Applications (ICCSA), 2011 International Conference on, Giugno 2011

[13] N. Pogkas, G. E. Karastergios, C. P. Antonopoulos, S. Koubias, G. Papadopoulos, *"Architecture design and implementation of an ad-hoc network for disaster relief operations"*, Industrial Informatics, IEEE Transactions on (Volume:3 , Issue: 1), Febbraio 2007

[14] A. Gorcin, H. Arslan, *"Public safety and emergency case communications: opportunities from the aspect of cognitive radio"*, New Frontiers in Dynamic Spectrum Access Networks, 2008. DySPAN 2008. 3rd IEEE Symposium on, Ottobre 2008

[15] G. P. Villardi, G. T. F. De Abreu, H. Harada, *"TV white space technology"*, Vehicular Technology Magazine, IEEE, Aprile 2012

[16] T.Catarci,M.DeLeoni,A.Marrella,M.Mecella,B.Salvatore,G.Vetere,S.Dustdar, L. Juszczyk, A. Manzoor, H.-L. Truong, *"Pervasive software environments for supporting disaster responses"*,Internet Computing, IEEE (Volume:12 , Issue: 1), Gennaio-Febbraio 2008

[17] D. D. Deb, S. Bose, S. Bandyophyay, *"Coordinating disaster relief operations using smart phone/PDA based peer-to-peer communication"*,International Journal of Wireless & Mobile Networks (IJWMN) Vol. 4, No. 6, Dicembre 2012

[18] T.Wada, T.Takahashi, *"Evacuation guidance system using everyday use smarthphones"*,Signal-Image Technology & Internet-Based Systems (SITIS), 2013 International Conference on, Dicembre 2013

[19] O. Mokryn, D. Karmi, A. Elkayam, T. Teller, *" Help Me: Oppurtinistic Smart Rescue Application and System"*,Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean, Giugno 2012

[20] Gianluca Aloi, Luca Bedogni, Marco Di Felice, Valeria Loscrì, Antonella Molinaro3, Enrico Natalizio, Pasquale Pace, Giuseppe Ruggeri, Angelo Trotta and Nicola Roberto Zema, *"STEM-Net: an evolutionary network architecture for smart and sustainable cities"*,Transactions on Emerging Telecommunications Technologies,Volume 25, Issue 1, Gennaio 2014

[21] Marco Di Felice, Luca Bedogni, Angelo Trotta, Luciano Bononi, Fabio Panzieri, Giuseppe Ruggeri, Gianluca Aloi, Valeria Loscrì, Pasquale Pace, *"Smartphones Like Stem Cells: Cooperation and Evolution for Emergency Communication in Post-Disaster Scenarios"*,Communications and Networking (BlackSeaCom), 2013 First International Black Sea Conference on, Luglio 2013

[22] Gianluca Aloi, Luca Bedogni, Luciano Bononi, Orazio Briante, Marco Di Felice, Valeria Loscrì, Pasquale Pace, Fabio Panzieri, Giuseppe Ruggeri, Angelo Trotta, *"STEM-NET: How to Deploy a Self-Organizing Network of Mobile End-User Devices for Emergency Communication"*, Computer Communications, Febbraio 2015

[23] Figura realizzata con XMind https://www.xmind.net/

[24] http://omnetpp.org/

[25] http://mixim.sourceforge.net/

Ringraziamenti

Ringrazio Luca Bedogni per il suo sostegno e per essere stato sempre presente.

Ringrazio i miei genitori per avermi sempre spinto ad andare avanti e fare quello che mi piaceva.

Un ringraziamento alla mia ragazza, Maria, per essermi sempre stata accanto, anche se lontana, in tutti i momenti del mio percorso.

Grazie ai miei amici,Fausto, Francesca, Gioia, Stefano e Danilo. Senza di loro questi anni non sarebbero stati gli stessi. Un ringraziamento particolare a Fausto per essere stato al mio fianco sin dal primo giorno.

Un ringraziamento anche ai miei coinquilini per aver sopportato i miei deliri di questi ultimi mesi.

Infine grazie anche a tutti coloro che non ho citato che mi sono stati accanto nel mio percorso.