Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

SVILUPPO DI UN'APPLICAZIONE ANDROID PER LA COMUNICAZIONE DECENTRALIZZATA

Relatore:
Chiar.mo Prof.
Luciano Bononi
Corelatore:
Chiar.mo Prof.
Luca Bedogni

Presentata da:
Mattia Maldini

Sessione di Luglio
Anno Accademico 2015/2016

A chi mi ha sempre considerato un fallito,
con i miei più profondi ringraziamenti.

Introduzione

La rete globale è senza alcun dubbio lo strumento più utilizzato e importante nel campo della diffusione delle informazioni e della comunicazione.

Il numero di persone che è in grado di accedervi è in costante aumento (circa il 40% della popolazione mondiale, `http://www.internetlivestats.com/`), il che tra le altre conseguenze contribuisce alla crescita sia sociale che economica di molti paesi in via di sviluppo. Social network e in generale tutti gli strumenti di comunicazione messi a disposizione dal web hanno già giocato ruoli cruciali in molte conquiste di diritti civili in diversi paesi nel mondo. Basti pensare che in Egitto, dopo le dimissioni del presidente Mubarak, a una neonata è stato il nome "Facebook", in un atto di ringraziamento al principale mezzo di coordinazione delle proteste popolari.

Oltre che consentire una comunicazione efficiente, il ruolo di internet è stato anche quello di mostrare a tutto il mondo situazioni di disagio che vorrebbero essere tenute nascoste da chi le causa; blog di denuncia sulla violazione di diritti umani possono essere strumenti potenti contro regimi autoritari che temono l'applicazione di sanzioni internazionali da parte dei paesi più sviluppati.

L'utilizzo del web ha fatto compiere un passo importante nella lotta per i diritti civili e la libertà di espressione. Nonostante questo non si tratta di una zona completamente sicura nella quale discutere e organizzarsi liberamente: se un qualunque cittadino può aprire un sito web ed esprimere un'opinione contraria a quella che si vorrebbe avesse, il suo governo ha certamente le risorse per impedirglielo. La censura dei media esiste da sempre, e internet

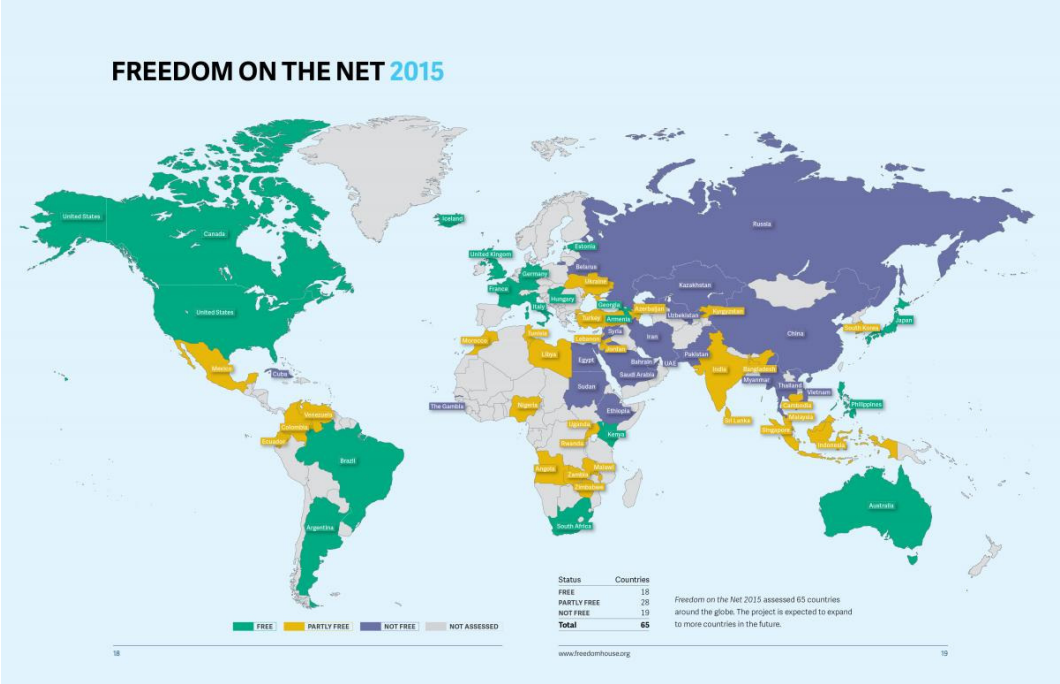

Figure 1: Mappa della libertà di espressione sul web relativa al 2015. Nell'ordine i paesi più virtuosi sono quelli verdi, gialli e blu.

non è un'eccezione. Non è difficile per le autorità filtrare o bloccare del tutto il traffico in rete, e non sempre i provider di servizi come Google riescono a garantire piena libertà ai propri utenti in tutte le parti del mondo.

Oltre a non essere assolutamente libero, Internet non si può considerare sicuro. Le informazioni che viaggiano sul web devono essere crittate con mezzi propri se si vuole avere una qualche garanzia che non vengano visualizzate da occhi indiscreti. I servizi di comunicazione messi a disposizione degli utenti si basano tutti su modelli centralizzati, il che significa che i messaggi inviati finiscono sempre nelle mani di terzi prima di arrivare al destinatario. Anche stipulando contratti di privacy all'installazione delle applicazione la maggior parte delle volte non si ha modo di scoprire cosa succede una volta che i dati lasciano il client. Senza mettere in dubbio la buona fede di chi li maneggia, questi potrebbero essere esposti semplicemente da meccanismi di sicurezza poco curati. Inoltre, avvenimenti recenti mettono in discussione la segretezza al di là di attacchi da parte di malintenzionati; ci si riferisce in particolare alla strage di San Bernardino, un attacco terroristico avvenuto nel dicembre 2015 per la cui seguente indagine l'FBI ha ufficialmente richiesto ad Apple di accedere ai dati privati del telefono dei terroristi.

Proprio sulla scia di casi come questi molti produttori e servizi hanno cominciato sempre di più a crittare i dati degli utenti in modo tale che nessuno - nemmeno loro stessi - sia in grado di accedervi. Ma anche in questo caso, le moderne tecniche di cifratura procedono di pari passo con gli sforzi per penetrarle, per cui quello che è sicuro oggi potrebbe non esserlo domani.

In conclusione, la rete globale non è un posto sicuro, né si può dare per scontato di avervi accesso senza filtri.

Si sta recentemente facendo strada l'idea di un tipo di comunicazione che non passi del tutto o in parte attraverso internet. Molto spesso le persone con cui si vuole comunicare si trovano a pochi chilometri di distanza, per cui non sarebbe necessario scomodare un server centrale con sede in un'altra città che si occupi di inoltrare i nostri messaggi.

Se si potessero trasmettere dati direttamente tra i dispositivi ai due capi della conversazione la maggior parte di questi problemi sarebbe superata:

- Si avrebbe una comunicazione più affidabile in quanto non dipendente da infrastrutture e servizi esterni per il proprio funzionamento.
- Sarebbe uno scambio più sicuro che idealmente non coinvolgerebbe nessuna entità "superiore"(posto di utilizzare un proprio strato di crittografia).
- Si guadagnerebbe in efficienza riducendo i salti da fare per arrivare a destinazione.

Ci si è proposto di implementare un'applicazione che offra un servizio di questo tipo. Di seguito vengono elencate le specifiche.

Specifiche

L'applicazione realizzata vuole fornire un servizio di diffusione di messaggi sicuro e indipendente da infrastrutture preesistenti (internet).

I messaggi vengono inviati a dei gruppi di utenti e non hanno un destinatario specifico. I gruppi devono essere semplici da creare e atri utenti devono poter essere invitati nei gruppi creati; i gruppi non hanno limite di dimensione.

In particolare la comunicazione tra i dispositivi dovrebbe avvenire senza intervento dell'utente. Le informazioni dovrebbero inoltre essere crittate in maniera che solo gli utenti all'interno del gruppo possano visualizzarle.

Non ci sono requisiti di latenza minima: lo scopo è concentrato più sulla diffusione dei messaggi rispetto alla comunicazione in tempo reale.

Contents

List of Figures

Chapter 1

Lavori correlati

Esistono diversi studi e proposte, quasi tutti in uno stato fortemente sperimentale, su servizi di messaggistica privata distaccata dal sistema centralizzato di internet. Ancora, più in generale, è ampio il campo di presentazioni di protocolli, modelli e sistemi per la formazione di reti a maglie parallele e con diversi gradi di separazione dalla rete globale.

Di seguito vengono elencati alcuni dei progetti che si ritengono più rilevanti e analoghi al lavoro svolto. Tutti gli esempi portati sono rilasciati sotto una licenza di software libero e il codice (o la descrizione) è liberamente consultabile.

RiseApp [1] è la proposta di applicazione che ha effettivamente ispirato questo lavoro. L'idea consiste appunto nel poter diffondere delle informazioni in un contesto in cui la connessione a internet è (volutamente) limitata da un agente repressivo. L'autore di RiseApp propone un sistema più ampio di quello qui descritto, coprendo anche il passaggio delle informazioni dai dispositivi tra cui sono state condivise a internet attraverso la rete Tor per garantire l'anonimato degli utenti.

La proposta non è stata implementata, per cui non viene descritto nel dettaglio il metodo di condivisione dei dati; si fa riferimento a dei non meglio precisati "punti di raccolta wireless" (termine che verrà utilizzato anche ai fini dell'applicazione discussa), a cui i dispositivi si collegherebbero per scambiare

i dati. Il progetto qui proposto è di fatto una possibile implementazione della parte *peer to peer* di RiseApp.

Un altro progetto che punta a fornire lo stesso servizio è Briar [2]. Anche questa è un'applicazione Android indirizzata ad attivisti e giornalisti in contesti di censura e repressione mediatica e che trasmette i suoi dati in assenza di internet usando comunicazioni dirette (Wifi e Bluetooth) oppure sulla rete Tor. La comunicazione diretta tra dispositivi dovrebbe avvenire attraverso una rete a maglie mirata soprattutto a ridurre al minimo il traffico per evitare il più possibile il rischio di intercettazioni. Briar va anche oltre, ponendosi come obbiettivo non solo la semplice diffusione di messaggi in forma anonima ma l'organizzazione in sistemi più complessi come blog o simili, fornendo un supporto completo per organizzazioni e discussioni libere e anonime.

Di fatto Briar dovrebbe coprire in toto i requisiti esposti da RiseApp e, potenzialmente, rendere superato il lavoro qui esposto. L'autore è venuto a conoscenza solo in un secondo momento della sua esistenza, ed essendo ancora in uno stato di sviluppo non è in grado di esprimersi in una valutazione più precisa.

Poi ci sono progetti che potrebbero essere sfruttati per implementare servizi di comunicazione sicura P2P (*peer to peer*) ; la maggior parte di essi si trova in uno stato di sviluppo iniziale non ancora fruibile, oppure non è utilizzabile per altri motivi.

Per esempio, Thali [3] è un framework che dovrebbe permettere di sviluppare applicazioni multi-piattaforma basate su connessioni P2P tra dispositivi limitrofi, con sincronizzazione con il cloud solo quando disponibile. Si tratterebbe di una piattaforma costruita su HTML5 e Javascript (in particolare Node.JS) in grado di fornire indipendenza dal dispositivo su cui si lavora e un approccio nativo di tipo P2P alla rete, con l'utilizzo opzionale di server e cloud.

Il tipo di progetto è molto più generale, ed è facile vedere che sarebbe triviale sviluppare le funzionalità di RiseApp con questo supporto, che ha

comunicazione sicura e peer to peer come filosofia di vita. Questo non appena Thali uscirà dallo stato sperimentale in cui si trova; nel frattempo, sono state preziose le considerazioni di alcuni dei suoi sviluppatori [4] nel filtrare le tecnologie utilizzabili per implementare il modello di comunicazione.

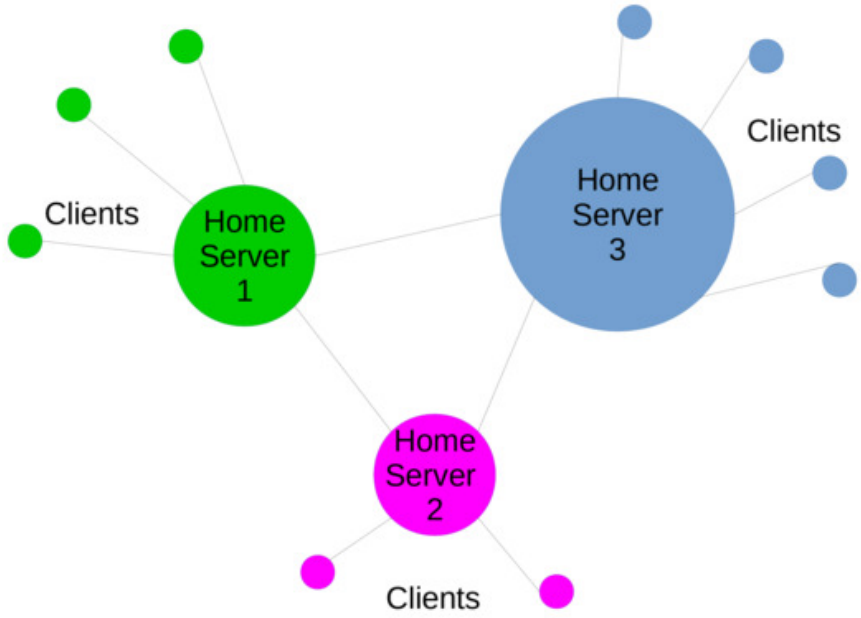

Figure 1.1: Struttura di Matrix

Matrix [5] è uno standard in qualche modo simile, che però si allontana dall'idea di connessioni strettamente peer to peer. Si tratta di un insieme di API HTTP costruite per supportare un nuovo ecosistema globale di comunicazioni in tempo reale, dove i servizi che lo usano fanno riferimento a più server sincronizzati tra loro piuttosto che a uno centralizzato, come mostra la figura 2.1. Molteplici server (**homeserver**) decentralizzati mantengono le informazioni gestendo arbitrariamente i conflitti di sincronizzazione e forniscono la loro visione dei dati ai client a loro connessi. Idealmente seguendo lo standard servizi diversi dovrebbero essere in grado di interoperare tra di

loro e trasmettere informazioni tra i loro utenti: nella figura 2.1 gli home-server 1, 2 e 3 potrebbero riferirsi a tre diverse applicazioni che si scambiano dati di uno stesso contesto.

Viene menzionato per la stessa volontà di decentralizzare le comunicazioni e renderle più sicure, ma lo scopo principale sembra essere quello di garantire l'interoperabilità dei servizi aderenti allo standard piuttosto che il distacco completo da una rete centrale. Per quanto riduca infatti la dipendenza da una rete centralizzata, non si preoccupa di coprire la comunicazione in assenza di accesso a internet.

Bitmessage [6] è un protocollo di comunicazione P2P per inviare messaggi a uno o a molti. Pone un accento particolare sulla segretezza non solo dei contenuti ma soprattutto dei metadati (indirizzo di mittente e destinatario) e sulla resistenza ad attacchi di *spoofing* e *man in the middle*; il tutto evitando di utilizzare sistemi basati su *trust chain*, che nelle motivazioni dell'autore soffrono troppo della debolezza di pochi anelli compromessi. Secondo questo sistema gli utenti dovrebbero formare una rete P2P che inoltra messaggi con un modello best effort. Ogni client che riceve un messaggio cerca di decodificarlo con le sue chiavi verificando se è lui il destinatario; in caso affermativo invia una risposta di acknowledgement, altrimenti lo reinoltra.

Il protocollo non è però pensato per una distribuzione in tempo reale su sistemi mobili: per evitare attacchi di tipo *denial of service* infatti ogni volta che un client vuole inviare un messaggio deve prima computare una *proof of work* che nel whitepaper si indica dover richiedere in media quattro minuti. In conclusione si tratta di un protocollo adatto a una rete P2P ma formata da peer statici, il che è in contrasto con la natura dinamica (pur non in tempo reale) di quello che si vuole realizzare.

Menzionato per l'originalità del modello concepito, Oppline [7] è un servizio di messaggistica breve basato sulla creazione di reti ad hoc tra smartphone in occasioni ed eventi che portano al radunarsi di molte persone in spazi relativamente ristretti.

Il problema affrontato si sposta di più sull'impossibilità di usare i metodi di comunicazione che si affidano a una rete globale (SMS via rete cellulare, GSM, o anche una connessione a internet tramite wifi) a causa appunto dell'eccessiva quantità di persone che cercano di connettersi. Il sistema proposto da Oppline trae invece beneficio da una quantità di utenti elevata e concentrata, e permette la comunicazione affidabile di messaggi brevi e sporadici attraverso l'*SSID beaconing* dei vari dispositivi. I dispositivi intervallano ciclicamente uno stato di ascolto degli hotspot vicini e uno di beaconing di un proprio hotspot: i messaggi vengono codificati nell'SSID e trasmessi ad altri dispositivi durante la fase di beaconing. I messaggi che possono essere inviati sono molto brevi (dei 32 byte concessi per la pubblicizzazione del proprio SSID i primi 16 vengono usati per il protocollo di comunicazione, e gli solo ultimi 16 possono contenere il messaggio vero e proprio), ma il sistema ha dei requisiti bassissimi e scala in maniera efficiente col numero di dispositivi. Non si può dire lo stesso della scalabilità sulla frequenza di messaggi trasmessi, che fa calare rapidamente la percentuale di pacchetti correttamente recapitati al destinatario.

L'applicazione che è stata sviluppata richiede un trasferimento di dati consistente, impossibile da gestire con pacchetti da 16 byte, e continuo, in contrasto con i problemi di Oppline con la creazione frequente di pacchetti.

Infine sono stati trovate diverse proposte di implementazioni e protocolli per la creazione di rete a maglie tra dispositivi mobili, che se supportate rappresenterebbero la soluzione perfetta, come descritto da [8] o [9]; la discussione di questi argomenti va oltre lo scopo di questo studio.

Chapter 2

Architettura

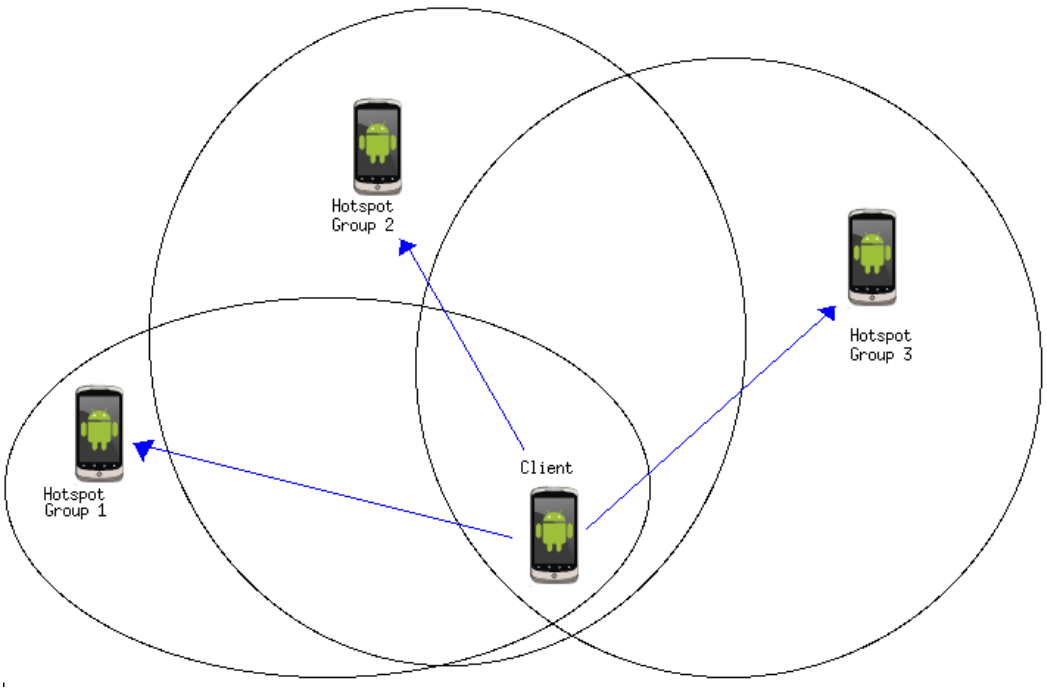

Figure 2.1: primo esempio di funzionamento

2.1 Descrizione generale

La descrizione dell'architettura fa riferimento allo stato attuale dell'applicazione; modifiche significative che sono state fatte nel corso dello sviluppo vengono coperte nel capitolo successivo.

Lo scopo è quello di fornire un servizio di comunicazione sicura e non intercettabile senza aver accesso a internet. Gli utenti fanno parte di gruppi privati all'interno dei quali possono scambiare informazioni tramite messaggi. Il funzionamento è completamente offline e anonimo, con un nickname personalizzabile e non univoco (di default il modello del dispositivo) come unica distinzione tra gli utenti. Un utente può in qualsiasi momento creare un nuovo gruppo e invitare chi vuole. Tutti gli utenti che appartengono a uno stesso gruppo condividono e visualizzano i suoi messaggi. Al contrario dispositivi al di fuori di quel gruppo non ricevono alcuna informazione appartenente a esso, che comunque non sarebbero in grado di decrittare. I messaggi che l'utente vuole inviare vengono inseriti nella memoria interna del dispositivo.

Periodicamente lo smartphone ricerca un punto di raccolta dell'applicazione, connettendosi e chiedendo lo scambio di dati dei gruppi che condivide con esso. L'operazione di ricerca avviene anche immediatamente dopo l'inserimento di un nuovo contenuto, e lo scambio dei dati avviene solo in seguito alla verifica che il ricevente appartenga davvero al gruppo in questione. Se il dispositivo non trova un access point per uno dei suoi gruppi, lo diventa per un certo intervallo di tempo.

Un esempio di funzionamento è descritto in figura 3.1: un dispositivo alla ricerca di informazioni fa una scansione delle reti e si collega a quelle dei gruppi a cui appartiene, ignorando le altre. Nel caso rimanessero dei gruppi di cui non ha trovato il punto di raccolta lo diventerebbe a sua volta.

La figura 3.2 mostra un caso d'uso più complesso, con un messaggio che passa da un dispositivo a un altro facendo due salti. Quando il primo nodo crea il messaggio (all'interno del gruppo B) inizia immediatamente la ricerca per dei punti di raccolta a cui inviarlo, trovando il primo hotspot del gruppo

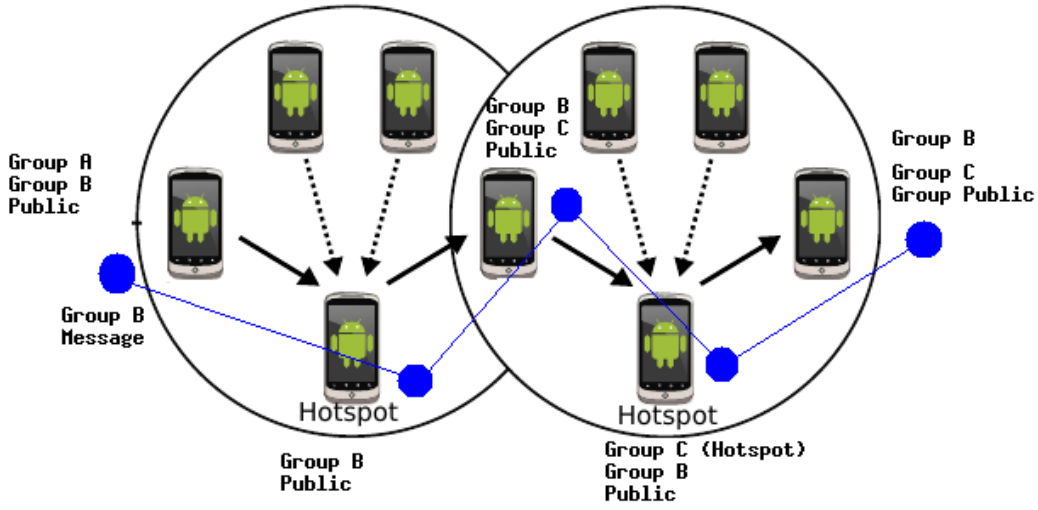

Figure 2.2: secondo esempio di funzionamento

B e passandoglielo.

L'hotspot comincia quindi a inoltrare il messaggio a tutti i client che si connettono che appartengono al gruppo B. Uno di essi dopo averlo ricevuto si connette con un altro punto di raccolta; nonostante sia un hotspot relativo al gruppo C condivide con esso l'appartenenza a B, per cui si scambiano anche il messaggio in questione. Da lì l'ultimo hotspot invierà i suoi contenuti (tra cui il messaggio iniziale) ai dispositivi che incontra. Si noti che mentre l'immagine evidenzia il passaggio dal primo dispositivo a sinistra all'ultimo a destra, il messaggio è stato inoltrato anche a tutti quelli collegati alle varie reti.

L'applicazione è pensata per la distribuzione di dati più che per una effettiva comunicazione; per questo viene accettato il meccanismo di trasmissione basato su una connessione periodica a degli access point (eventualmente non presenti), che può introdurre una certa latenza a seconda dell'intervallo con cui i dispositivi entrano nella fase di ricerca. Considerando la situazione per cui è stata pensata (cioè una manifestazione nella quale sia imperativo diffondere le informazioni raccolte senza poter accedere alla rete globale) si può

tranquillamente assumere un numero di dispositivi molto superiore al numero di gruppi. Infine, con un raggio medio di 30 metri per la copertura di un hotspot mobile, anche un numero limitato di dispositivi potrebbe coprire facilmente una piazza intera.

2.2 Struttura

L'applicazione nel suo stato attuale ha una struttura descritta in termini generali dagli schemi nelle figure 3.3 e 3.4 La activity principale (**MainActivity**) è anche quella iniziale. Contiene una **ListView** con i contenuti correntemente salvati sul telefono e permette di passare alle altre schermate. I messaggi mostrano il contenuto, il gruppo di appartenenza, l'autore e la data di creazione. Ogni gruppo viene associato a un colore tramite il calcolo di una funzione di hashing molto semplice sul suo nome.

I nuovi contenuti si inseriscono dalla **PostActivity**, accessibile dal bottone in basso a destra nella **MainActivity**. Si può creare un messaggio testuale e scegliere a quali gruppi (a cui si appartiene) inoltrarlo. Un messaggio inoltrato gruppi multipli viene creato e visualizzato in più copie da chi fa parte di due o più di suddetti gruppi.

Dal menù nella *Toolbar* dell'applicazione si accede invece alla gestione dei gruppi (**GroupsActivity**) o alle impostazioni. La Activity per la gestione dei gruppi consente di creare un nuovo gruppo o dal nulla o scannerizzandone il QR code, insieme ovviamente alla possibilità di mostrare il QR code di uno qualunque dei gruppi. Sia i gruppi che i contenuti si possono eliminare: vengono rimossi dal dispositivo ma nulla vieta di riottenerli.

Le impostazioni prevedono per ora soltanto la modifica del nickname e l'attivazione del servizio in background che trasferisce i dati. La classe **WifiAPManager** è una API non ufficiale rilasciata da uno sviluppatore indipendente e lievemente modificata [10].

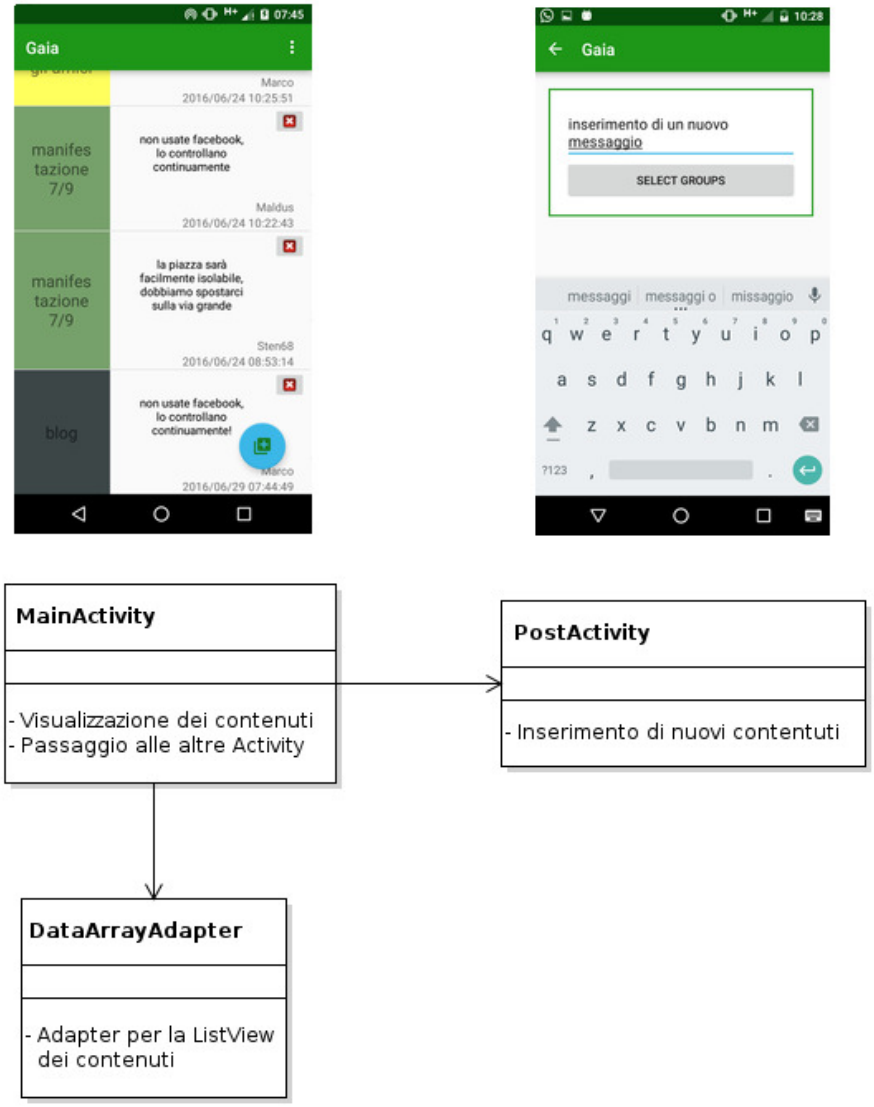

Figure 2.3: Schema delle Activity principali (1)

2.3 Requisiti

2.3.1 Permessi richiesti

Il Manifest Android richiede i seguenti permessi:

- *android.permission.INTERNET*: per utilizzare l'interfaccia di rete dell'SDK (socket Java).
- *android.permission.ACCESS_WIFI_STATE*: questo e il seguente permesso servono per la ricerca e la connessione ai punti di raccolta.
- *android.permission.CHANGE_WIFI_STATE*: come sopra.
- *android.permission.ACCESS_NETWORK_STATE*: questo e il seguente permesso servono per generare un hotspot e usarlo come punto di raccolta.
- *android.permission.CHANGE_NETWORK_STATE*: come sopra.
- *android.permission.ACCESS_COARSE_LOCATION*: anche se apparentemente scollegato dalle funzionalità richieste, questo permesso è necessario per accedere alla scansione delle reti Wifi, probabilmente perché è possibile calcolare una posizione geografica approssimativa in base alle reti disponibili.
- *android.permission.WRITE_SETTINGS*: permesso necessario per creare un hotspot e per ottenere la lista di client che sono connessi.

Per versioni di Android con SDK superiore a 23, deve essere l'utente a fornire esplicitamente i permessi per accedere alle impostazioni di sistema, per cui viene ridireziona alla modifica di suddetta opzione nel caso non vengano riscontrati.

La ricerca di reti Wifi richiede l'attivazione della locazione GPS per le versioni di Android successive a Marshmallow. Non è chiaro se si tratti di un problema o di una scelta progettuale.

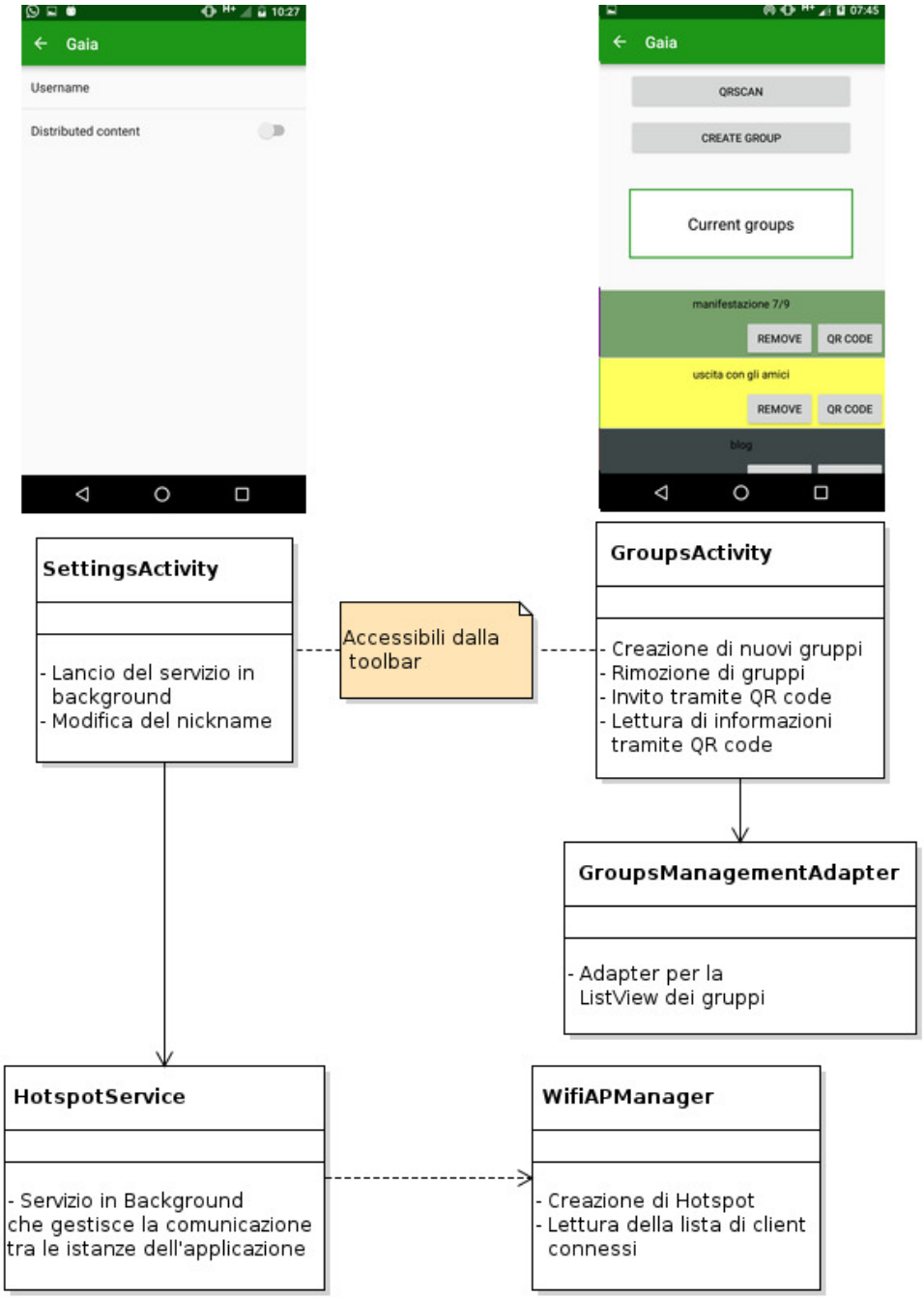

Figure 2.4: Schema delle Activity principali (2)

2.3.2 Requisiti di sistema

Per quanto riguarda l'aspetto prettamente tecnico, l'applicazione necessita di un dispositivo che supporti la tecnologia Wifi e la possibilità di fare da hotspot locale.

Come verrà spiegato nel capitolo successivo, l'invito a un gruppo avviene tramite lettura di un *QR code* contenente numero consistente di dati, per cui è necessaria una fotocamera sufficientemente potente da leggerlo (dispositivi dotati di una bassa risoluzione non dovrebbero comunque precludere la lettura, richiedendo soltanto una mano molto ferma e qualche tentativo in più).

Non vengono mai fatti sforzi computazionali tali da richiedere o anche solo consigliare dei requisiti minimi in questo aspetto.

2.3.3 Requisiti ambientali

Per garantire una comunicazione accettabile sono necessari almeno $N+1$ dispositivi, dove N è il numero totale di gruppi distinti di cui fanno parte i dispositivi considerati: questo perché ogni dispositivo diventa un hotspot per un certo gruppo se non ne trova altri; in una situazione con N gruppi ed N dispositivi, si rischia di avere un hotspot per ogni gruppo ma nessun dispositivo che si connetta a essi scambiando effettivamente le informazioni.

Inoltre bisogna considerare che un gruppo a cui appartengono soltanto pochi utenti ha molte meno possibilità di vedere condivisi i propri dati, per cui bisogna equilibrare tra ristrettezza dell'insieme di persone con cui si vuole comunicare ed efficacia di questa comunicazione.

In ogni caso, tutte le istanze dell'applicazione appartengono a priori al gruppo **public**, le cui informazioni vengono trasmesse in chiaro in qualsiasi caso, per garantire se necessario massima visibilità.

Chapter 3

Implementazione

In questo capitolo verranno descritti i processi centrali dello sviluppo dell'applicazione (trasferimento e crittografia), partendo dagli strumenti e le librerie utilizzate.

3.1 Strumenti Utilizzati

L'applicazione è stata sviluppata per Android in linguaggio Java. Di seguito vengono mostrati i componenti utilizzati per lo sviluppo dell'applicazione.

3.1.1 Android SDK

Android SDK (Software Development Kit) è il pacchetto contenente il set di API ufficiale della piattaforma e gli strumenti di sviluppo utili a compilare, testare e debuggare applicazioni per Android. Le librerie integrate nell'SDK coprono la stragrande maggioranza delle necessità che si possono avere nello sviluppare delle funzionalità. Nel corso di questo progetto è stata usata solo una libreria esterna a queste.

3.1.2 Android Studio

Android Studio è l'ambiente di sviluppo integrato ufficiale di Google per la piattaforma Android. È basato su IntelliJ IDEA ed è liberamente disponibile sotto licenza Apache 2.0. Nonostante Android Studio metta a disposizione un emulatore per ogni tipo di dispositivo il progetto qui descritto è stato testato completamente su dispositivi fisici, in particolare

- Motorola Moto G, Android 6.0
- ASUS_T00P, Android 4.4.2
- Huawei Y560-L01, Android 5.1.1

3.1.3 Zxing

Zxing (`https://zxingnet.codeplex.com/`) è una libreria open source per la generazione e l'elaborazione di barcode in una o due dimensioni.

È stata utilizzata per implementare la comunicazione delle chiavi crittografiche dei gruppi, come spiegato nella sezione 4.3.

3.2 Trasferimento dei dati

Per trasferire i dati tra i dispositivi senza passare da internet sono disponibili diverse possibilità. Purtroppo non molte di queste sono accettabili per lo scopo che si vuole raggiungere, e nessuna rappresenta la soluzione perfetta. Mentre una comunicazione ideale si otterrebbe con la realizzazione di una rete wifi a maglie, sarebbe anche più di quanto necessario all'applicazione che è si voluto realizzare. Essendo lo scopo principale la diffusione di dati, senza curarsi di mittenti o destinatari dei messaggi (esclusivamente in broadcast), sarebbe infatti bastata la capacità di stabilire connessioni temporanee e saltuarie con i dispositivi che si incontrano. In questo capitolo vengono discussi i due metodi effettivamente implementati nei loro pro e contro, insieme ad altre strade considerate ma non percorse in pratica.

3.2.1 Wifi Direct

Descrizione

La tecnologia wifi direct sembrava, inizialmente, la soluzione ideale per il trasferimento dei dati. Il wifi direct è uno standard (relativamente) nuovo di comunicazione tra dispositivi dotati di tecnologia Wifi. Lo scopo è quello di permettere connessioni ad hoc altamente dinamiche, cioè senza ruoli prestabiliti, che vengono invece negoziati nel momento della connessione.

In pratica il funzionamento è simile alla modalità a infrastruttura del Wifi tradizionale: si stabilisce una sorta di hotspot detto **P2P Group** formato da un access point (chiamato **Group Owner**, GO) e tanti client che si connettono a esso; quale dispositivo debba essere GO viene appunto deciso dinamicamente. Una volta che il gruppo è creato altri dispositivi possono connettervisi come se fosse un normale hotspot: in realtà la struttura è proprio utilizzabile anche da client legacy che non supportano wifi direct, che a quel punto vi si connetteranno secondo lo standard IEEE 802.11 come farebbero con un qualunque access point.

Ci sono tre casi d'uso diversi per la creazione di un gruppo P2P [11]:

Standard: Se i dispositivi non si conoscono e vogliono formare un nuovo gruppo dal nulla devono per prima cosa trovarsi e poi negoziare quale dei due debba adempiere al ruolo di **Group Owner**.

Per trovare i dispositivi adiacenti viene prima fatta una scannerizzazione tradizionale di access point Wifi (tra cui anche quelli di gruppi già esistenti); poi si porta avanti una ricerca specifica alternata tra ascolto e invio di *Probe request* su canali prestabiliti.

L'intervallo di tempo con cui i due stati vengono alternati è randomizzato per garantire che due peer riescano a ricevere una *Probe request* e rispondere con una *Probe response*, inizializzando così la connessione.

Una volta trovati, i due dispositivi danno il via ai negoziati per decidere quale dei due sarà il **Group Owner** e su quale canale operare. Questa fase consiste in un *three-way handshake* durante il quale si scambiano un parametro numerico che indica l'attitudine del mittente a essere **Group Owner** (modificabile dall'utente ma anche variabile in base a dati come la batteria rimanente, la potenza del segnale e i numero di peer che sono visibili a ogni partecipante).

Dopo una fase ulteriore che stabilisce i parametri di sicurezza del collegamento (detta di *WPS Provisioning*), il GO comincia ad agire come DHCP e fornisce un indirizzo IP a ogni client (il GO è sempre 192.168.49.1, tutti gli altri ricevono un indirizzo a caso nello spazio restante tra 192.168.49.2 e 192.168.49.254).

A questo punto la connessione è pienamente stabilita e la comunicazione può procedere.

Autonomo: Un dispositivo può decidere indipendentemente di creare un gruppo P2P (senza alcun client connesso) di cui diventerà ovviamente il GO.

Altri dispositivi possono poi trovarlo durante la fase di scansione tradizionale e cercare di connettervisi, partendo direttamente dalla fase WPS.

Saltando la negoziazione del GO la connessione è significativamente

più semplice, ma richiede che un peer decida a priori di fare da access point.

Persistente: Durante la creazione del gruppo i dispositivi possono salvare le "credenziali" (ovvero i parametri stabiliti dalle fasi di negoziazione) del gruppo per poterlo reistanziare rapidamente in un secondo momento.

In particolare se durante la connessione le entità in gioco riconoscono che il gruppo esisteva già possono ricrearlo senza ulteriore burocrazia con un *two-way handshake* detto *Invitation Procedure*.

In una prima implementazione dell'applicazione (implementazione comunque arrivata a uno stadio pienamente funzionante) era stato utilizzato il wifi direct, che Android supporta nelle sue API ufficiali a partire dalla versione 4.0 (Jelly Bean), SDK 14.

L'interazione con l'hardware wifi è possibile usando i metodi della classe *WifiP2pManager*, che permettono di scoprire i dispositivi vicini, connettersi e disconnettersi da essi.

A causa dell'utilizzo diretto dell'hardware tutti questi metodi comportano una risposta asincrona che deve essere gestita creando dei *listener* da registrare presso il *WifiP2pManager* e dei *BroadcastReceiver* che reagiscano ai cambiamenti nella connettività del dispositivo.

Lo scambio di informazioni in background è stato implementato attraverso un *AlarmReceiver* programmato per svegliarsi a intervalli più o meno regolari, cercare dispositivi limitrofi, connettersi e comunicare con essi. Si trattava di una comunicazione con gruppi di due o più dispositivi alla volta, a seconda di quanti fossero attivi nel momento in cui avveniva la ricerca; la comunicazione avveniva soltanto tra ognuno dei client e il Group Owner secondo un modello best effort.

Si noti che, teoricamente, lo standard wifi Direct dovrebbe permettere che un dispositivo sia connesso nello stesso momento a più gruppi diversi (non è però chiaro quanto questa possibilità sia tuttora supportata da Android),

creando di fatto una rete a maglie sulla quale si potrebbero inviare in broadcast le informazioni. Si è preferito mantenere il modello di comunicazione il più semplice possibile non essendo questo lo scopo principale dello studio.

Chiaramente il caso principe che si sarebbe dovuto verificare era quello di un gruppo standard, tra dispositivi che non si conoscono.

Problemi riscontrati

Pur riuscendo in pratica a collegare i dispositivi, nel corso dell'implementazione si sono presentate tre difficoltà principali che hanno portato alla decisione di utilizzare un altro metodo di comunicazione:

1. Considerando come già detto il caso Standard, un ostacolo di cui non si era tenuto presente è stata la necessità dei dispositivi che si vogliono connettere di portare avanti la scansione nello stesso momento, con una tolleranza di qualche secondo.

   La soluzione è stata banalmente cercare di far partire la ricerca nei vari smartphone il più possibile nello stesso istante, in generale al secondo 00; soluzione raggiungibile non senza problemi, in quanto Android mette a disposizione delle sveglie a orari esattamente precisi soltanto a partire dell'SDK 19, mentre tutte le precedenti tengono conto dell'orario stabilito ma si riservano il diritto di accorparne diversi per svegliare la CPU il minor numero di volte possibile, costringendo a ricorrere a un seppur breve busy waiting per cadere esattamente alla fine del minuto.

   Evidentemente si tratta di una soluzione assolutamente indesiderabile e inaffidabile.

2. Il problema più grave è però l'obbligo di accettazione da parte dell'utente per stabilire una connessione tra due smartphone che non si sono mai visti prima. Questo si traduce in un dialog che chiede conferma per permettere la connessione qualsiasi sia il contesto in cui è stata por-

tata avanti la scansione, compreso quello in cui l'applicazione non sia nemmeno attiva (portata avanti da un *AlarmReceiver*).

Le conseguenze possono anche essere piuttosto irritanti per l'utente, perché le richieste di conferma si sovrappongono in background se non vengono accettate o rifiutate, di fatto bloccando il telefono una volta accumulate in numero sufficiente.

È possibile rimuovere il dialog di conferma in un telefono a cui sono stati dati i diritti di root [12], ma ovviamente si è preferito continuare a lavorare in un ambiente il più possibile aderente alla versione Android di stock, senza fare assunzioni forti come quella di operare su un sistema operativo modificato.

Nel caso di una connessione a un gruppo già esistente si può aggirare il problema. Usando delle credenziali prestabilite si può accedervi come se fosse un normale access point [13] (usando quindi la classe per le normali connessioni Wifi *WifiManager* invece di *WifiP2pManager*) senza intervento dell'utente. Tuttavia quando viene creato un gruppo le sue credenziali sono generate casualmente dal sistema e non possono essere cambiate, per cui bisogna comunicarle attraverso altri canali. Tra le possibilità per farlo c'è il Bluetooth (che però soffre della stessa necessità di intervento dell'utente) o pubblicizzandole nel nome di un servizio offerto sempre tramite Wifi dal telefono con la modalità *service discovery* [1]

Nel complesso è stata giudicata troppo complessa come soluzione (i motivi verranno spiegati meglio nella prossima sezione) e per questo abbandonata.

Bisogna infine menzionare che una volta stabilita una connessione questa viene ricordata permanentemente dal dispositivo, che non richiede più alcuna conferma per reistanziarla. Non si può però supporre che gli

[1] Procedura di scambio di frame molto piccoli prima ancora della scansione dei peer, usati per determinare il tipo di servizi fornito senza doversi connettere.

utenti possano confermare a priori tutte le possibili connessioni prima di usare l'applicazione in background.

3. Il terzo ostacolo è sorto in seguito a considerazioni sulle future estensioni del progetto ad altre piattaforme; in particolare, IOS. I supporti forniti da IOS e Android al Wifi Direct non sono al momento compatibili, e l'impossibilità di inoltrare le informazioni tra i due sistemi operativi mobile più usati al momento di questa stesura appare come una limitazione da evitare a tutti i costi.

Dalle prime due difficoltà si evince come il servizio Wifi Direct (almeno nella sua implementazione nelle API di Android) sia pensato esclusivamente per un utilizzo sincrono, con un istante in cui i due dispositivi si devono connettere ben definito (connessione iniziata e conclusa con la diretta volontà dell'utente), una funzione limitata nel tempo da portare a termine per poi abbandonare il gruppo P2P, il tutto entro il ciclo di vita dell'applicazione - non in background.

Più in generale tutto il design dell'API segue questo stesso ragionamento, oltre a trovarsi in uno stato piuttosto acerbo.

Per esempio la connessione passa attraverso diverse fasi (grossomodo quelle descritte sopra) che devono essere gestite con un *BroadcastReceiver* che ascolti i cambiamenti nella connettività del sistema: cambiamenti notificati tramite eventi lanciati in maniera non sempre correttamente sequenziale e spesso ripetuti più volte senza significato, costringendo lo sviluppatore a preoccuparsi del giusto andamento del percorso di connessione (evitando di reagire a eventi multipli che indicano lo stesso avvenimento).

Ancora, ci sono due modi per ottenere la lista di dispositivi rilevati nelle vicinanze: richiedere la lista correntemente salvata (con il metodo *WifiP2pManager.requestPeers*, inspiegabilmente anch'esso asincrono) o richiedere una scansione, catturare l'evento *WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION* nel *BroadcastReceiver* e a quel punto richiedere la lista di peer come nel primo metodo.

Chiedendo direttamente la lista di peer si rischia di ricevere delle informazioni obsolete, mentre facendo una scansione si riceve una lista aggiornata al costo di qualche secondo in più.

Purtroppo l'evento che dovrebbe notificare l'aggiornamento della lista viene passato al Receiver solo se la lista cambia rispetto a quella che il sistema conosceva già: se la scansione viene completata e la lista non è cambiata non si ha modo di saperlo, rendendo difficile l'aggiornamento tra dispositivi che sono rimasti in raggio di ricezione l'uno dell'altro.

Quest'ultimo problema non è stato veramente risolto, ma si è semplicemente accettato che la comunicazione avvenisse soltanto tra dispositivi che si scoprono nuovamente (assunzione non troppo pesante in un contesto di movimento continuo come è quello dei dispositivi mobili).

Tenendo presente tutte le precedenti difficoltà, pur avendo raggiunto un prototipo funzionante si è deciso di tentare un'altra via.

3.2.2 Modalità a infrastruttura

A causa degli ostacoli presentati nell'utilizzo del Wifi Direct si è deciso di affrontare il problema con un metodo meno adatto ma di più facile utilizzo: la creazione di hotspot che facesse da infrastruttura ad hoc sulla quale effettuare il trasferimento di dati.

La possibilità di creare un gruppo autonomo Wifi Direct per evitare la conferma di connessione richiesta all'utente non è stata percorsa per tre motivi principali:

1. L'inerente complessità nella comunicazione per vie traverse delle credenziali di accesso a suddetto gruppo.

2. La vanificazione del vantaggio del Wifi Direct, cioè la comunicazione diretta; se è proprio necessario creare un access point a priori, tanto vale farlo con una tecnologia più facilmente gestibile (quella della modalità a infrastruttura).

3. Il fatto che il protocollo Wifi Direct non è pensato per creare infrastrutture centralizzate per più di quattro o cinque dispositivi, per cui non scala bene con il numero di connessioni.

Descrizione

Il concetto è molto semplice: come in precedenza il dispositivo si deve svegliare periodicamente cercando altri con cui scambiare informazioni. Usando la modalità a infrastruttura non è possibile però effettuare connessioni dirette tra due dispositivi.

Quando lo smartphone si attiva, per prima cosa fa una scansione delle reti: tra quelle trovate ricerca degli SSID corrispondenti a un hash (*SHA-1*) dell'id dei gruppi di cui fa parte.

Se trova un access point con queste caratteristiche ci si connette e scambia, solo con il fornitore della rete, i contenuti su tutti i gruppi che condivide con esso (ulteriori dettagli nella sezione 4.3). Una volta terminato il trasferimento continua con la scansione.

Se alla fine della procedura sono rimasti dei gruppi di cui non è stato trovato il corrispondente punto di raccolta questo viene temporaneamente creato dal dispositivo stesso. In questo modo con una densità sufficiente di nodi sarà sempre presente un punto di raccolta per ogni gruppo, compito svolto a turno da tutti i dispositivi in gioco.

Questa tecnica si porta dietro tutti i vantaggi della semplicità di una rete locale a cui potersi connettere.

Nel corso dell'implementazione è stata inoltre cambiata la modalità di "risveglio" per la ricerca di altri smartphone: vista la tecnica più invasiva, si è optato per un service che l'utente deve esplicitamente attivare e disattivare piuttosto che un alarm continuo.

La presenza di un access point per ogni gruppo è necessaria per garantire la comunicazione anche tra gli utenti che fanno parte di gruppi non molto diffusi, che possono quindi inviarsi messaggi anche se soltanto in coppia.

Questo metodo è apparso immediatamente più valido del Wifi Direct; tuttavia comporta anch'esso una serie non indifferente di problemi.

Problemi riscontrati

Il primo e più importante di questi è che, mentre il Wifi Direct ha una API con molti punti da migliorare, l'utilizzo della funzionalità di hotspot non è ufficialmente supportato da alcuna API di Android.

Trattandosi di una funzionalità problematica è comprensibile la scelta di mettere tutto nelle mani dell'utente: secondo la filosofia ufficiale, dovrebbe essere soltanto quest'ultimo a decidere se e quando attivare un hotspot. Questo si scontra ancora con la necessità di svolgere queste operazioni per la maggior parte del tempo in background.

Pur non essendo ufficialmente supportata, l'attivazione di un hotspot è comunque fattibile al prezzo di una scarsa documentazione e di una tecnica poco legittima. Il seguente spezzone di codice è tratto dalla classe *WifiAP-Manager*:

Listing 3.1: WifiAPManager

```
public boolean setWifiApEnabled(WifiConfiguration wifiConfig,
    boolean enabled) {
     try {
       if (enabled) { // disable WiFi in any case
         mWifiManager.setWifiEnabled(false);
       }

       Method method = mWifiManager.getClass().getMethod("
           setWifiApEnabled", WifiConfiguration.class, boolean.
           class);
       return (Boolean) method.invoke(mWifiManager, wifiConfig,
            enabled);
     } catch (Exception e) {
       Log.e(this.getClass().toString(), e.toString());
       return false;
     }
   }
}
```

È possibile dunque attivare l'hotspot accedendo al metodo *setWifiApEnabled* della classe *WifiManager* per riflessione (si veda l'Appendice A).

Per quanto in pratica funzioni, una soluzione del genere non può essere altro che temporanea. È alta infatti la probabilità che successive versioni di Android non permettano di sfruttare lo stesso meccanismo per i motivi più disparati, o addirittura per una precisa decisione di impedire questo tipo di utilizzo.

Mentre come già detto la volontà di lasciare all'utente l'attivazione dell'hotspot è comprensibile, lascia un po' perplessi la mancanza di un metodo per conoscere la lista dei client connessi a esso.

Il seguente codice mostra il metodo utilizzato, che legge il file di sistema **/proc/net/arp** (da cui la richiesta di permessi per accedere ai file di sistema) e ne estrapola il contenuto.

Listing 3.2: WifiAPManager

```
br = new BufferedReader(new FileReader("/proc/net/arp"));
          String line;
          while ((line = br.readLine()) != null) {
            String[] splitted = line.split(" +");

            if ((splitted != null) && (splitted.length >= 4))
                {
              // Basic sanity check
              String mac = splitted[3];

              if (mac.matches("..:..:..:..:..:..")) {
                boolean isReachable = InetAddress.getByName(
                    splitted[0]).isReachable(reachableTimeout)
                    ;

                if (!onlyReachables || isReachable) {
                  result.add(new ClientScanResult(splitted[0],
                      splitted[3], splitted[5], isReachable))
                     ;
                }
              }
            }
          }
```

Il dispositivo che sta attualmente svolgendo il ruolo di hotspot controlla i cambiamenti nella lista di dispositivi connessi, procedendo allo scambio dei contenuti ogni qualvolta ne arrivi uno nuovo.

Tra il momento in cui finisce la scansione della lista di access point e l'attivazione della modalità hotspot possono passare (a seconda del tipo di dispositivo) fino a cinque secondi. Può quindi accadere che due (o più) dispositivi diventino nello stesso momento punti di raccolta per uno stesso gruppo. In tal caso si ha una sincronizzazione dei messaggi meno efficiente poiché i vari client spartiranno le proprie informazioni tra i due access point.

Di seguito vengono menzionate alcune delle tecnologie considerate ma scartate per effettuare la comunicazione.

3.2.3 Bluetooth

Dopo alcune considerazioni iniziali il Bluetooth è stato rapidamente messo da parte come inadatto allo scopo.

Per prima cosa si tratta di una tecnologia inerentemente meno efficace del Wifi: il raggio d'azione è più breve e più facilmente intralciato da ostacoli fisici e interferenze; inoltre l'ampiezza di banda raggiunge al massimo i 3 Mb/s (Bluetooth 2.0). Lo stato attuale dell'applicazione permette soltanto di scambiare messaggi testuali, ma tra le possibilità future si è tenuto presente di contenuti più corposi (immagini o addirittura video) che richiederebbero per essere trasmessi con una connessione così lenta un tempo di contatto non sempre raggiungibile.

Una connessione Bluetooth è composta da un ciclo di due fasi. La prima crea un accoppiamento tra due dispositivi, mentre la seconda scopre se un dispositivo già accoppiato è presente e ci si connette. Come per la creazione di un hotspot, su Android l'attivazione del Bluetooth richiede l'intervento dell'utente. Questo però non limita allo stesso modo l'azione in background, in quanto si può chiedere a quest'ultimo di accenderlo e mantenerlo accesso insieme al servizio in background *hotspotService*. Non si può fare lo stesso con la modalità a infrastruttura perché il servizio ha bisogno di disabilitare e riabilitare l'hotspot durante la ricerca di altre reti Wifi. Perché due dispositivi si trovino prima di essere accoppiati uno dei due deve essere nello stato *discoverable* e l'altro deve cominciare la scansione. Qui risiede uno degli ostacoli principali, poiché anche per diventare "scopribile" dagli altri il telefono ha bisogno di una conferma del suo proprietario, il che riporta al problema iniziale dell'intervento utente.

Anche sorvolando su questo punto, a quanto pare sia la ricerca che la condizione di visibilità sono operazioni che richiedono un particolare consumo di energia, e per questo limitate a un certo lasso di tempo (dodici secondi

[14] per la prima, due minuti per la seconda). È chiaro che anche questa tecnologia è pensata per un utilizzo sincrono tra entità con ruoli subordinati molto ben definiti - molto di più del Wifi Direct. Una volta accoppiati i dispositivi si possono trovare immediatamente senza passare da condizioni di ricerca e visibilità, ma non si può pensare di aver collegato a priori tutte le istanze in gioco dell'applicazione. È possibile aggirare la fase di ricerca con una sorta di *service discovery* se si conosce l'UUID del servizio (che si può settare), in sostanza chiedendo all'API di cercare se quell'entità specifica è presente nelle vicinanze. Si tratta di una feature di Android non presente nelle specifiche Bluetooth, e non è chiaro quanto sia affidabile una scansione di questo tipo (per quanto tempo devo chiedere prima di essere sicuro che ci sia o meno?).

3.2.4 Bluetooth Low Energy

Menzionato soltanto per completezza, risolve notevolmente il problema del consumo di energia eccessivo, ma continua a portarsi dietro tutte le difficoltà precedentemente descritte. Inoltre non è ancora sufficientemente supportato (pienamente da Android soltanto a partire dalla versione 5).

3.2.5 Modalità ad hoc

Strettamente parlando, le connessioni tramite Wifi Direct non sono davvero peer to peer [15]. Semplicemente usano un protocollo per la formazione dinamica di gruppi, tipicamente tra due dispositivi.

La modalità ad hoc è un protocollo ufficialmente presente [16](chiamato IBSS, *Independent Basic Service Set*) e parte attiva dello standard IEEE 802.11. Prevede una totale assenza di gerarchia tra i nodi della rete che si viene a formare. Ogni nodo può creare una rete con un certo SSID e cominciare a pubblicizzarla o entrare in uno stato di ascolto per reti vicine.

La costruzione di una rete ad hoc è più semplice rispetto alla formazione di un gruppo Wifi Direct. Tutti i nodi già collegati a una rete inviano

regolarmente dei pacchetti *beacon*, contenenti il nome della rete (*SSID*), l'identificatore della rete (*BSSID*) e un timer. Un *BSSID* (Basic Service Set Identifier) è una stringa che identifica la rete in modo univoco; nella modalità a infrastruttura corrisponde all'indirizzo MAC dell'access point a cui si è collegati.

Per connettersi a una rete ad hoc basta ascoltare passivamente su canali prestabiliti ricevendo dei pacchetti beacon inviati dai nodi della rete, oppure ricevendoli come risposta in seguito all'invio di una *probe request*. Una volta ricevute le informazioni sulla rete il nuovo nodo passa a negoziare le credenziali per lo standard di cifratura usato dalla rete. Se la rete è semplicemente aperta invece può immediatamente cominciare a inviare pacchetti su di essa (attraverso i nodi già connessi) utilizzando il suo *BSSID*, senza ulteriore burocrazia. Il timer contenuto nei beacon serve a risparmiare banda impedendo che tutti i nodi della rete inoltrino per pubblicizzarla più messaggi del dovuto. Se un nodo (connesso alla rete) riceve un pacchetto beacon da un altro con un timer sufficientemente recente evita di crearne uno a sua volta. L'invio di questi pacchetti è parzialmente randomizzato per lasciare che ogni nodo della rete ne invii la sua parte.

In teoria si tratta del supporto ufficiale per creare una vera e propria rete a maglie; in pratica è presente a livello di hardware sui dispositivi Android ma non supportata nativamente dalle API. È possibile (e alcune applicazioni lo fanno) usare un supporto software non ufficiale, ma come in altri casi richiede la modifica del sistema operativo.

3.3 Crittografia

La comunicazione avviene su canali potenzialmente intercettabili, per cui tutte i contenuti che sono trasferiti viaggiano in forma crittata (la stessa forma crittata con cui sono salvati nei singoli dispositivi).

3.3.1 Meccanismi utilizzati

Per assicurare la confidenzialità delle informazioni queste vengono crittate usando crittografia simmetrica AES con chiavi a 256 bit. Ogni gruppo è costituito da una tripla formata da id del gruppo, chiave per la crittografia del gruppo, e nome del gruppo. Identificatore e chiave vengono generati casualmente al momento della creazione, mentre il nome viene inserito dall'utente che lo crea. Ogni gruppo usa la sua chiave per crittare e decrittare i messaggi che invia e riceve. Dopo aver creato un gruppo un utente può invitare altri a parteciparvi facendo leggere al loro dispositivo un QR code (generato dall'applicazione) contenente appunto la tripla che identifica il gruppo. I messaggi sono salvati nella memoria interna del telefono in formato JSON; contengono un campo per l'id del suo gruppo e uno per il contenuto crittato. Il trasferimento dei dati avviene secondo un protocollo che prevede uno scambio challenge-response per ogni gruppo di cui si richiedono le informazioni: un dispositivo invia a un altro dei contenuti solo se è sicuro che l'interlocutore conosce come lui la chiave privata del gruppo.

Quando viene stabilita una connessione tra un dispositivo e l'access point di un certo gruppo per prima cosa entrambi si scambiano i dati pubblici, senza fare alcun controllo. Dopodiché entrambi si inviano reciprocamente la lista di gruppi a cui rispettivamente appartengono, di cui individualmente trovano l'intersezione. Poi, per ognuno dei gruppi in comune a turno si inviano una challenge-response sotto forma di stringa casuale da crittare con la chiave del gruppo in questione. Se l'autenticazione viene confermata il dispositivo che l'aveva richiesta invia i suoi dati, altrimenti interrompe la connessione.

Si noti che la presenza di un punto di raccolta per gruppo serve a garantire

che ci sia per ognuno almeno un access point abilitato a scambiare le sue informazioni. Tutti gli hotspot possono però, ai fini di una diffusione più efficiente, scambiare i contenuti di tutti i gruppi che condivide con i suoi interlocutori.

3.3.2 Vulnerabilità

La generazione di chiavi e identificatori per i gruppi non tiene conto della loro univocità: è tecnicamente possibile che due gruppi vengano creati con la stessa chiave o lo stesso id. Considerato però

1. Quella che si intende essere la natura effimera dei gruppi: essendo così facile creare e invitare utenti in un gruppo, si assume che vengano usati come strutture "usa e getta", da cambiare ogni qualvolta sia necessario usare il servizio.

2. L'effettiva probabilità che un evento del genere si verifichi: insignificante, visto lo spazio dei valori possibili (stringhe di 256 bit e 256 byte per chiavi e identificatori rispettivamente).

3. I danni che provocherebbe: nel caso pessimo, utenti appartenenti a entrambi i gruppi avrebbero problemi a visualizzare correttamente i contenuti, oppure membri di un gruppo riceverebbero i contenuti crittati di un altro.

   Vista la natura aleatoria dell'incidente, non sarebbe in ogni caso sfruttabile per accedere a contenuti segreti.

Si è voluto privilegiare la semplicità di gestione al rigore di identificatori e chiavi assolutamente univoci.

La comunicazione delle chiavi di cifratura viene fatta attraverso un canale (lettura di un QR code) la cui sicurezza dipende dalle scelte dell'utente; una condivisione indiscriminata può portare facilmente a mostrare la chiave anche a soggetti che non dovrebbero conoscerla. Ricordando ancora una volta la facilità con cui si può abbandonare un gruppo "compromesso" passando a uno

nuovo si considera il meccanismo implementato a priori sicuro, nel senso che è difficile da parte di un malintenzionato forzarlo contro il volere di un utente legittimo. Come tutti i servizi, resta vulnerabile a un utilizzo improprio.

Riguardo all'autenticazione di un interlocutore con cui si vogliono scambiare dei dati, il meccanismo di challenge-response garantisce che il dispositivo a cui si inviano i dati abbia la chiave per leggerli (e quindi appartenga a quel gruppo). Un intruso potrebbe comunque interpretare il ruolo di access point e ottenere così delle coppie plaintext-cyphertext (attraverso la challenge-response di un client legittimo) da cui cercare di estrapolare la chiave. A oggi non si conoscono tuttavia dei metodi più efficienti di attacchi brute force controllando con il cyphertext se si ha raggiunto la chiave giusta, che richiedono risorse tali da renderli infattibili in pratica. Per eliminare questa vulnerabilità si potrebbero inviare i response cifrati senza specificare con quale chiave sono stati cifrati, lasciando a chi controlla l'incombenza di verificare con tutte le sue chiavi. Vista la robustezza di AES contro i known-plaintext attack è stata fatta una scelta a favore dell'efficienza e della semplicità del protocollo.

Una grave vulnerabilità del sistema è la natura aperta degli hotspot per la raccolta dei dati. Il protocollo di trasferimento dovrebbe in sé contrastare i tentativi di intrusione, ma una rete difesa da un qualsiasi standard di crittografia (anche WEP o WPA) porterebbe dei miglioramenti significativi. Purtroppo come già spiegato nella sezione precedente la creazione di un hotspot non è permessa dalla API ufficiale; si è riuscito comunque ad attivare un access point, ma i tentativi dell'autore non sono valsi a stabilire nulla di più complesso di una rete pubblica.

Infine, essendo le informazioni sui gruppi salvate nella memoria interna che Android riserva alle applicazioni, potrebbero tecnicamente essere rubate avendo accesso diretto al telefono. In realtà, chiunque può visualizzare informazioni che dovrebbero essergli precluse impossessandosi di un dispositivo con installata l'applicazione. Questo aspetto è stato sorvolato in virtù della

natura di *proof of concept* del lavoro e in quanto sarebbe di facile soluzione, crittando le informazioni sensibili con una passphrase inserita dall'utente - cosa che effettivamente avviene già se il telefono è protetto da un PIN, o addirittura da un file system crittato.

Chapter 4

Casi d'uso

Lo scopo dell'applicazione è quello di fornire la possibilità di diffondere (senza un destinatario specifico) dati sensibili in assenza di connessione a internet. Allo stato attuale si possono condividere soltanto contenuti testuali; nella descrizione di casi d'uso esemplificativi si suppone di poter trasferire anche immagini (passo ancora da implementare ma che non pone particolari difficoltà).

L'utilizzabilità del servizio è fortemente dipendente dalla sua diffusione. È adatto per comunicazioni in zone con un'alta densità di dispositivi, per esempio in un generico contesto urbano o in un evento che comporta un raccoglimento di persone. Uno dei suoi punti forti è la crittografia e la sicurezza contro intrusioni, per cui è utile per comunicazioni sensibili e anonime.

L'applicazione può essere usata per organizzare una manifestazione o altri tipi di associazione pacifica senza correre rischi per la propria attività online nel caso fosse monitorata. Tutti gli utenti interessati possono condividere un gruppo e tramite esso comunicarsi ora e luogo di un incontro in maniera anonima e sicura. Chiaramente più l'applicazione e il gruppo sono diffusi nell'area più le comunicazione sarà rapida e affidabile.

Una volta all'interno di una manifestazione, il servizio consente di diffondere il più possibile messaggi ma soprattutto immagini e media sensibili. Non soltanto tra i manifestanti: passando per la trasmissione diretta i mes-

saggi potrebbero arrivare direttamente ai giornalisti inviati da altri paesi, a cui magari viene impedito l'accesso alle zone calde. Più semplicemente potrebbe essere nell'interesse di tutti condividere dei gruppi di diffusione con i reporter fornendo una nuova fonte di materiale sull'avvenimento. Non è raro che le forze dell'ordine requisiscano telefoni e altri dispositivi di registrazione quando una persona viene arrestata o anche soltanto fermata, bloccando la pubblicazione di queste informazioni.

Il servizio non è adatto ad essere utilizzato in situazioni che richiedono risposte in tempo reale, a causa dell'elevata latenza nella trasmissione dei messaggi. In una versione futura o in quella attuale in maniera limitata però potrebbe essere utile per comunicare in un contesto in cui non siano accessibili altri mezzi. Per esempio in situazioni disastrose (terremoti, crisi urbane), nelle quali i normali mezzi di comunicazione crollano davanti al numero di richieste di molto superiore alla media.

Conclusioni

Risultati conseguiti

Lo scopo che si era prefissato è stato raggiunto: l'applicazione nel suo stato attuale implementa i requisiti iniziali, fornendo un servizio di diffusione di informazioni e scambio di messaggi in maniera sicura e anonima (non in tempo reale).

Il trasferimento di messaggi tramite infrastrutture temporanee create dai telefoni stessi è tutt'ora ritenuta la migliore tra le possibilità elencate in termini di utilizzabilità. Nonostante ciò il metodo di comunicazione tra i dispositivi rimane un nodo non risolto; tutte le tecnologie studiate o utilizzate si sono rivelate completamente o parzialmente inadeguate all'attuale stato dell'arte. La tecnica usata per attivare la funzionalità di hotspot è una pesante forzatura del supporto fornito dall'SDK ufficiale, e sarebbe ragionevole aspettarsi che possa smettere di funzionare a partire da un qualsiasi futuro aggiornamento.

In realtà l'autore spera vivamente che quel momento arrivi, augurandosi però che sia rapidamente seguito dal supporto che ora manca per implementare una comunicazione diretta in background. Il supporto alla modalità Wifi ad hoc permetterebbe di creare facilmente reti distribuite su cui comunicare liberamente. Anche se una rete distribuita è più di quanto serva per sincronizzare i propri messaggi con tutti i nodi limitrofi, la modalità ad hoc consentirebbe di collegare due smartphone in maniera molto più rapida e semplice rispetto al Wifi Direct.

L'evoluzione di Android in particolare come sistema operativo è tesa nell'equilibrare la sicurezza contro software maligni e lo sfruttamento delle potenzialità del sistema. Esistono come accennato diverse versioni modificate della versione di stock di Android (prima tra tutte CyanogenMod, `http://www.cyanogenmod.org/`) che permettono già di sfruttare appieno le possibilità di vere connessioni peer to peer e reti completamente distribuite. La politica di Google a riguardo sembra però piuttosto scettica e indirizzata a mantenere dei permessi limitati per garantire gli utenti contro abusi.

A causa della mancanza di protocolli di sicurezza *built-in* nel modello di reti ad hoc non ci si aspetta che sia quello il supporto più probabile a essere fornito in futuro. Il Wifi Direct invece è già di fatto supportato: basterebbe permetterne l'utilizzo anche mentre il telefono è in standby e senza l'intervento del suo proprietario. Sarebbe più che sufficiente per connessioni brevi e saltuarie con pochi dispositivi - secondo il primo modello implementato per il trasferimento dei dati. Inoltre in teoria anche quest'ultimo permette la creazione di reti distribuite grazie alla possibilità (offerta dallo standard ma a oggi non supportata dall'API) di essere connessi a più gruppi contemporaneamente.

In conclusione, in una prospettiva di miglioramento ed estensione del lavoro svolto fino ad ora ci si aspetta (e si spera) di dover cambiare ancora il metodo usato per connettere i dispositivi, in particolare usando lo standard Wifi Direct - fermo restando che allo stato attuale la modalità a infrastruttura è decisamente la scelta più plausibile.

Estensioni

Il lavoro è stato portato a termine in una fase comunque sperimentale; di seguito si elencano alcune delle caratteristiche con cui sarebbe possibile estenderlo:

- Gestione di contenuti multimediali oltre che testuali; la comunicazione di messaggi di solo testo non è molto utile ed è servita soltanto come prova della fattibilità del progetto.
- Salvataggio dei contenuti in maniera più efficiente, utilizzando SQLite (database nativo di Android).
- Introduzione di un ulteriore strato di crittografia per proteggere le chiavi contenute nel dispositivo, magari usando una *passphrase* inserita dall'utente.
- Miglioramento della fase di autenticazione nello scambio di dati, senza fornire a eventuali intrusi coppie *plaintext-ciphertext* con cui portare avanti (seppur improponibili) attacchi *brute force*.
- Cambiare o aggiornare il metodo di comunicazione non appena verrà fornito un supporto più adeguato.
- Utilizzare diversi metodi di comunicazione contemporaneamente; per esempio non c'è motivo di scomodare Wifi Direct o modalità ad hoc se molti dispositivi sono già connessi a una rete locale.

Bibliography

[1] https://ans.disi.unitn.it/users/maccari/riseapp

[2] https://briarproject.org/index.html

[3] http://thaliproject.org/

[4] Yaron Y. Goland, *Thali and the Mesh Mess*. http://www.goland.org/thalimesh/

[5] https://matrix.org/

[6] Jonathan Warren, *Bitmessage: A Peer-to-Peer Message Authentication and Delivery System*. 2012

[7] Okan Turkes, Hans Scholten, Paul J. M. Havinga, *Friend-to-Friend Short Message Service with Opportunistic Wi-Fi Beacons*. The Seventh IEEE Workshop on Pervasive Collaboration and Social Networking, 2016

[8] Andre Ippisch, Kalman Graffi, *An Android Framework for Opportunistic Wireless Mesh Networking*. IEEE NetSys'15, 2015

[9] Chao Yao, Hongliang Zhang, Lingyang Song, *Demo: WiFi Multihop - Implementing Device-to-Device Local Area Networks by Android Smartphones*. 2015

[10] https://github.com/nickrussler/Android-Wifi-Hotspot-Manager-Class

[11] Wi-Fi Alliance, *Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.2.* 2010

[12] Colin Funai, Cristiano Tapparello, Wendi Heinzelman, *Supporting Multi-hop Device-to-Device Networks Through WiFi Direct Multi-group Networking.* Cornell University Library, 2015

[13] `http://www.drjukka.com/blog/wordpress/?p=39`

[14] `https://developer.android.com/guide/topics/connectivity/bluetooth.html`

[15] `http://www.thinktube.com/tech/android/wifi-direct`

[16] IEEE Standards Association, *IEEE Std 802.11-2012.* 2012

Appendix A

Riflessione (Java)

Nei linguaggi di programmazione, la *type introspection* è l'abilità di un programma di esaminare le proprietà degli oggetti(i campi, i metodi, il tipo) a runtime. In Java ogni entità è o un oggetto o un tipo primitivo. Mentre i tipi primitivi sono un insieme fissato, si chiama oggetto qualsiasi istanza di una classe che erediti da *java.lang.Object*. Per ogni tipo di oggetto (la sua classe) la *Java Virtual Machine* crea una istanza fissata di *java.lang.Class*. L'oggetto *Class* a sua volta permette di esaminare a runtime le proprietà dell'oggetto; è quello che in Java implementa la possibilità di fare type introspection. La classe di ogni oggetto è ottenibile tramite il metodo *getClass()*; da quella si possono poi conoscere i campi, i metodi e le classi interne a quell'oggetto, compresi di modificatori di visibilità (public, private, ...) e lista di parametri.

La riflessione (*Reflection*) è invece l'abilità di fare modifiche a runtime utilizzando le informazioni fornite dall'introspezione. Per rendere più chiara la differenza si evidenzia che la riflessione non è possibile senza introspezione, mentre alcuni linguaggi supportano la seconda ma non la prima (per esempio il C++). In generale la riflessione può essere usata per modificare il comportamento di un programma a runtime, senza necessariamente conoscere tutte le informazioni necessarie al momento della compilazione. Un programma che fa uso esteso della riflessione può monitorare l'esecuzione di una parte di codice e modificare il proprio comportamento di conseguenza, utilizzando

codice che potrebbe non essere disponibile prima dell'esecuzione. Viene utilizzata spesso nelle fasi di testing: JUnit 4, un software per il testing automatico del codice, usa la riflessione per cercare i metodi delle classi taggati con il commento "*@Test*" e lanciarli. In particolare in Java è possibile usare la riflessione per accedere a metodi che sarebbero ristretti da modificatori di accesso o di visibilità: è possibile richiedere l'esecuzione di un metodo marcato come *private* o modificare un campo *final*.

Il seguente codice (preso da `https://docs.oracle.com/javase/tutorial/reflect/member/methodInvocation.html`) mostra un esempio di metodo che invoca i tutti i metodi privati il cui nome comincia con la parola "test" all'interno di un oggetto passato come parametro:

Listing A.1: Esempio di riflessione

```
public static void main(String... args) {
  if (args.length != 4) {
      err.format("Usage: java Deet <classname> <langauge> <
         country> <variant>%n");
      return;
  }

  try {
      Class<?> c = Class.forName(args[0]);
      Object t = c.newInstance();

      Method[] allMethods = c.getDeclaredMethods();
      for (Method m : allMethods) {
    String mname = m.getName();
    if (!mname.startsWith("test")
        || (m.getGenericReturnType() != boolean.class)) {
        continue;
    }
    Type[] pType = m.getGenericParameterTypes();
    if ((pType.length != 1)
        || Locale.class.isAssignableFrom(pType[0].getClass()))
            {
        continue;
```

```
    }

    out.format("invoking %s()%n", mname);
    try {
        m.setAccessible(true);
        Object o = m.invoke(t, new Locale(args[1], args[2],
            args[3]));
        out.format("%s() returned %b%n", mname, (Boolean) o);

    // Handle any exceptions thrown by method to be invoked.
    } catch (InvocationTargetException x) {
        Throwable cause = x.getCause();
        err.format("invocation of %s failed: %s%n",
             mname, cause.getMessage());
    }
      }
  } catch (ClassNotFoundException x) {
      x.printStackTrace();
  } catch (InstantiationException x) {
      x.printStackTrace();
  } catch (IllegalAccessException x) {
      x.printStackTrace();
  }
    }
```

In particolare, in questo progetto la riflessione è stata usata per invocare il seguente metodo della classe *WifiManager*:

Listing A.2: setWifiApEnabled

```
/**
 * Start AccessPoint mode with the specified
 * configuration. If the radio is already running in
 * AP mode, update the new configuration
 * Note that starting in access point mode disables station
 * mode operation
 * @param wifiConfig SSID, security and channel details as
 *        part of WifiConfiguration
```

```
     * @return {@code true} if the operation succeeds, {@code
        false} otherwise
     *
     * @hide Dont open up yet
     */
    public boolean setWifiApEnabled(WifiConfiguration wifiConfig,
       boolean enabled) {
        try {
            mService.setWifiApEnabled(wifiConfig, enabled);
            return true;
        } catch (RemoteException e) {
            return false;
        }
    }
```

I commenti che precedono il metodo sono delle annotazioni per Javadoc, un applicativo incluso nel Java Development Kit utilizzato per la generazione automatica della documentazione del codice sorgente. Android Studio usa Javadoc per facilitare la programmazione ai suoi utenti, ma non solo: l'annotazione *@hide* indica che il metodo non è accessibile attraverso l'SDK. Anche se esiste ed è pubblico quindi, tentare di invocarlo con un condice di questo tipo

Listing A.3: setWifiApEnabled

```
WifiManager manager = (WifiManager) getSystemService(Context.
   WIFI_SERVICE);
manager.setWifiApEnabled(null, true);
```

Porta ad un errore di compilazione.

Questo perché il file .jar dell'SDK Android che viene usata nell'applicazione non è il vero framework presente nella Dalvik Virtual Machine in esecuzione sui dispositivi, ma una sua versione ridotta che contiene

- Tutti i metodi originali **eccetto** quelli marcati con l'annotazione *@hide*.
- Tutti i metodi presenti sono implementati come semplici stub (*"throw*

new RuntimeException("Stub!")"), solo per mostrare a tempo di compilazione cosa sarà utilizzabile a runtime.

- Le annotazioni Javadoc.

Invocandolo per riflessione, come mostrato nel capitolo sull'implementazione, si passa oltre i controlli di compilazione e si trova il metodo a runtime nella classe *WifiManager* del framework effettivamente presente sul dispositivo.

Come nota finale, si osservi che il commento a fianco dell'annotazione che nasconde il metodo *setWifiApEnabled* recita "*Don't open up* ***yet***" (non **ancora** da aprire), il che fa presupporre che sia quantomeno nelle intenzioni degli sviluppatori di renderlo accessibile in futuro.