Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Comunicazione P2P tra smartphone attraverso tecnologia Wi-Fi Direct: una valutazione sperimentale

Relatore:
Chiar.mo Prof.
Dr. Marco Di Felice

Presentata da:
Carlo Tamburrelli

Sessione III
Anno Accademico 2013/2014

Indice

Introduzione **9**

Capitolo 1
1 Lo standard Wi-Fi Direct **11**
1.1 Formazione di un gruppo 11
1.1.1 Gruppo standard 13
1.1.2 Gruppo autonomous 15
1.1.3 Gruppo persistent 15
1.2 Struttura del pacchetto 16
1.3 Applicazioni sullo standard 19
1.4 Salvaguardare l'energia della batteria 21
1.4.1 Opportunistic Power Save 21
1.4.2 Notice of absence 22

Capitolo 2
2 Sistema operativo Android **23**
2.1.1 Architettura stratificata 24
2.1.2 Compilazione 25
2.1.3 Componenti Android 25
2.1.4 Ciclo di vita di un'activity 26
2.1.5 Il cuore dell'applicazione 27
2.2 Wi-Fi Direct su Android 27
2.2.1 Principi generali 28
2.2.2 Un esempio pratico 29
2.2.3 Dettagli aggiuntivi 33
2.3 Scenario fra due dispositivi 33

Capitolo 3
3 Definizione delle metriche e degli scenari **35**
3.1 Metriche di valutazione 35
3.1.1 Packet Delivery Ratio 35
3.1.2 Throughput 36

3.1.3 Round Trip Time 36
3.2 Panoramica del software 36
3.2.1 Server side 37
3.2.2 Client side 39
3.3 Funzione di ripartizione del tempo di Discovery e Group Formation 41
3.3.1 Panoramica del software 41

Capitolo 4
4 Risultati sperimentali **45**
4.1 Tipologie di valutazione 45
4.1.1 Confronto tra Wi-Fi ed altre tecnologie 45
4.1.2 Valutazione del Wi-Fi Direct 47
4.1.3 Calcolo della Discovery e Group Formation 49

Capitolo 5
5 Sviluppo di un simulatore per la tecnologia Wi-Fi Direct **53**
5.1 Panoramica del software 53
5.1.1 Architettura 54
5.1.2 Problemi affrontati 55
5.2 Risultati ottenuti 57

Conclusioni 61
Appendice A 63
Bibliografia 65

Elenco delle figure

Capitolo 1

Figura 1.1: esempio di due scenari utilizzando lo standard Wi-Fi Direct. 12
Figura 1.2: schema dei possibili casi per eleggere il P2P GO in base ai valori Intent dei dispositivi 14
Figura 1.3: fase di WPS provisioning 14
Figura 1.4: formazione di un gruppo standard. 15
Figura 1.5: formazione di un gruppo autonomo 15
Figura 1.6: formazione di un gruppo persistente 16
Figura 1.7: formato del pacchetto 16
Figura 1.8: formato generale dell'attributo P2P 17
Figura 1.9: elenco dei possibili p2p attributi 18
Figura 1.10: pacchetto P2P catturato dal programma Omnipeek 19
Figura 1.11: esempio di uso del protocollo di risparmio energetico Opportunistic Power Save 21
Figura 1.12: esempio di uso del protocollo di risparmio energetico Notice of absence .. 22

Capitolo 2

Figura 2.1: architettura stratificata in Android. 24
Figura 2.2: processo di compilazione di un file java in Android. 25
Figura 2.3: il ciclo di vita di un'activity in Android 26
Figura 2.4: metodi della classe Wi-FiP2pManger 28
Figura 2.5: lista dei listener associati ai vari metodi della classe WifiP2pManager 29
Figura 2.6: elenco di permessi necessari per il funzionamento dell'applicazione nel file manifest 29
Figura 2.7: configurazione iniziale di Wi-Fi Direct 30
Figura 2.8: ritaglio di codice della gestione dei vari intent rilevati dal BroadcastReceiver 30
Figura 2.9: implementazione della notifica di successo o fallimento del listener in ascolto sul metodo discoverPeers(). 31
Figura 2.10: richiesta della lista di Peers col metodo requestPeers. 31
Figura 2.11: implementazione della notifica di successo o fallimento del listener in

ascolto sul metodo connect(). 32
Figura 2.12: diagramma di sequenza per la fase di Group Formation 34

Capitolo 3

Figura 3.1: formazione di un gruppo attraverso la primitiva createGroup() contenuta all'interno di una funzione convenzionalmente chiamata crea_gruppo() 37
Figura 3.2: esempio di messaggistica utilizzata per svegliare il thread2 tramite il thread1 38
Figura 3.3: interfaccia grafica dell'applicazione del dispositivo che funge da P2P GO 38
Figura 3.4: ciclo eseguito dal thread2 per ricevere pacchetti dal P2P Client. Nella porzione di codice cerchiata in rosso è possibile notare il calcolo delle metriche 39
Figura 3.5: array matrix 40
Figura 3.6: frammento di codice eseguito dal thread2 per la ricezione dei pacchetti e il calcolo del Round Trip Time 41
Figura 3.7: alla sinistra viene mostrata la schermata dell'applicazione durante la ricerca dei dispositivi, alla destra quella visualizzata subito dopo averne trovato uno. 42
Figura 3.8: schermata dell'applicazione che mostra i risultati ottenuti. 43

Capitolo 4

Figura 4.1: grafico Packet Delivery Ratio per Localhost, Wi-Fi ed Ethernet 46
Figura 4.2: grafico Throughput per Localhost, Wi-Fi ed Ethernet 46
Figura 4.3: grafico Round Trip Time per Localhost, Wi-Fi ed Ethernet 47
Figura 4.4: PDR indoor 48
Figura 4.5: PDR outdoor 48
Figura 4.6: Throughput indoor 48
Figura 4.7: Throughput outoor 48
Figura 4.8: RTT indoor 49
Figura 4.9: RTT outoor 49
Figura 4.10: grafico della funzione di ripartizione della fase di Discovery 50
Figura 4.11: grafico della funzione di ripartizione della fase di Group Formation 50
Figura 4.12: grafico della funzione di ripartizione del completamento di entrambe le fasi 51

Capitolo 5

Figura 5.1: rete P2P simulata tra i nodi. Solo i nodi 3 e 6 hanno terminato la fase di trasferimento 55

Figura 5.2: inizializzazione dei nodi 56
Figura 5.3: porzione di codice dell'implementazione del metodo di sostituzione per i 4 tipi di equazione 57
Figura 5.4: probabilità del completamento delle fasi al variare della velocità dei nodi nell'area 58
Figura 5.5: probabilità del completamento delle fasi al variare della quantità di pacchetti da inviare 59

Introduzione

Oggigiorno circa il 15% della popolazione mondiale è munito di uno smartphone. La diffusione sempre crescente di questi dispositivi ha incentivato gli studi ed ha favorito la nascita di nuove tecnologie inerenti alle applicazioni mobili. Un tema particolarmente importante che riguarda questa disciplina è quello che concerne la comunicazione tra dispositivi e le tecnologie sottostanti: tra quest'ultime una delle più importanti è quella Wi-Fi, che permette la comunicazione tra dispositivi senza l'utilizzo di cavi.
Ci sono due modalità principali di connessione Wi-Fi: quella mediata (da un Access Point) e quella immediata (peer-to-peer).
Ci sono altri standard che permettono al dispositivo di collegarsi ad altri apparecchi senza l'ausilio di un Access Point. La Wi-Fi Alliance, che ha lanciato la tecnologia Wi-Fi, ha realizzato ultimamente uno standard chiamato Wi-Fi Direct, per permettere connessioni tra i device in modo diretto.

Ci si è occupati in questa tesi prevalentemente dello standard Wi-Fi Direct, illustrandone l'architettura e i relativi scenari di utilizzo. Successivamente la tesi si è focalizzata sul valutare sperimentalmente l'efficienza di Wi-Fi Direct per contesti di comunicazione P2P tra dispositivi: sono stati costruiti degli applicativi, che istallati sui dispositivi, hanno permesso di ricavate sperimentalmente delle metriche di valutazione quali Packet Delivery Ratio, Throughput e Round Trip Time. In seguito, è stato sviluppato un ulteriore applicativo per esaminare i tempi necessari agli smartphone per attuare le fasi fondamentali per la loro interazione. Infine, servendosi dei dati sperimentali accumulati, è stato costruito un Simulatore in grado di riprendere uno scenario composto di entità mobili forniti di tecnologia Wi-Fi Direct.
Per gli esperimenti, è stato utilizzato il sistema operativo Android.

Nel primo capitolo è stata descritta l'architettura dello standard Wi-Fi Direct, presentandone anche alcuni scenari applicativi.
Nel secondo capitolo è stata descritta l'architettura di Android e successivamente

è stato presentato un esempio implementativo di Wi-Fi Direct sul sistema operativo.
Nel terzo capitolo sono stati presentati gli applicativi che hanno permesso di calcolare le metriche di valutazione e le tempistiche relative alle funzioni di interazione tra i due dispositivi.
Nel quarto capitolo sono stati graficati e discussi i risultati ottenuti dagli applicativi.
Nel quinto capitolo è stato presentato il simulatore, mostrando infine i risultati ottenuti.

Capitolo 1

Lo standard Wi-Fi Direct

Il Wi-Fi Direct è uno standard Wi-Fi che permette la comunicazione diretta tra due dispositivi senza l'ausilio di un Access Point, permettendo in questo modo di sostituire perfettamente i vecchi approcci utilizzati finora (come la tecnologia Bluetooth o reti ad-hoc) e di migliorarne l'efficienza; essenziale per la sua funzione è che il raggio di distanza sia tale da permettere al Wi-Fi Direct di dialogare con i vari device nelle vicinanze. Inoltre, un vantaggio di questo standard è la possibilità di far partecipare alla stessa rete differenti device di diversa natura (stampanti, schermi, smartphone ecc.).
Per permettere una comunicazione attraverso questo standard, è necessario che almeno uno dei due device disponga della tecnologia Wi-Fi Direct. Questo standard, è anche chiamato Wi-Fi P2P, ovvero una rete peer-to-peer wireless. In una rete di questo genere non esiste nessuna gerarchizzazione tra i nodi e, a differenza di una tipica rete Wi-Fi, in cui i device hanno dei ruoli "statici" (Client o Access Point), in una rete Wi-Fi P2P i ruoli diventano dinamici; un utente potrebbe svolgere entrambi i ruoli di client e Access Point (anche simultaneamente se ha a disposizione più schede di rete).

1.1 Formazione di un gruppo

I device che utilizzano lo standard Wi-Fi Direct comunicano attraverso l'instaurazione di gruppi P2P (p2p groups), in ognuno dei quali esiste un solo device che implementa le stesse funzionalità dell'Access Point (AP-like) ed è chiamato P2P Group Owner (P2P GO) e, invece, tutti gli altri device collegati alla stessa rete, sono chiamati P2P Client.

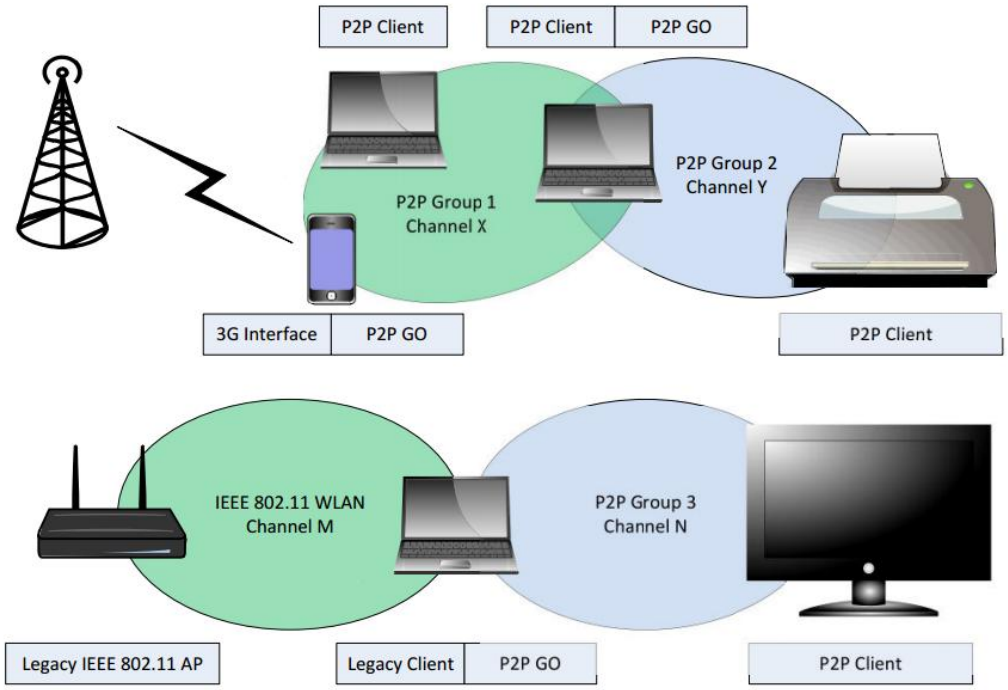

Figura 1.1: Esempio di due scenari utilizzando lo standard Wi-Fi Direct. *Sorgente : [1]*

La fig. 1.1 rappresenta due tipologie di scenari che si possono verificare utilizzando il Wi-Fi Direct:

-la parte superiore della figura mostra uno smartphone che condivide la sua connessione 3G con due laptop attraverso l'instaurazione di un gruppo. Lo smartphone agisce da P2P GO e i due laptop hanno il ruolo di P2P Client. Al fine di estendere la connessione 3G dello smartphone, uno dei due laptop potrebbe formare un nuovo gruppo per richiamare altri dispositivi; in questo caso, il laptop agisce sia da P2P GO nel nuovo gruppo e sia da P2P Client nel vecchio gruppo.

-Un altro scenario molto simile potrebbe essere il caso in cui un laptop utilizzi una connessione ad internet attraverso un dispositivo *legacy* (che non dispone dello standard Wi-Fi Direct), in modo tale da condividere i contenuti in streaming con un televisore disposto nei paraggi, mediante la formazione di un gruppo.

Con il Wi-Fi Direct si ha la possibilità di instaurare fino a tre tipi di gruppi differenti:

standard, *autonomous* e *persitent.*

1.1.1 Gruppo standard

Inizialmente lo standard Wi-Fi Direct esegue una procedura di Discovery. Tale procedura è composta da due sottofasi: *scan* e *find*.
Durante la sottofase scan, viene eseguita una tradizionale scansione col protocollo 802.11 per rilevare la presenza di esistenti gruppi P2P e reti Wi-Fi. In seguito a questa scansione, viene eseguito un nuovo algoritmo (sottofase find):
all'inizio ogni device seleziona uno dei canale di ascolto fra 1, 6 e 11 chiamato canale di listen ed in seguito i device si alternano fra due stati, quello di search e quello di listen. In quello di search si esegue un invio di richieste d'indagine (chiamate richieste Probe) ad ognuno dei canali (1, 6 e 11). Nello stato di listen, invece, i device sono in ascolto sul proprio canale di richieste Probe. È da notare che l'intervallo di tempo impiegato dai device sullo stato di listen è del tutto casuale (compreso tra i 100 e i 300 ms), questo permette ai dispositivi di convergere sullo stesso canale.
L'esecuzione di queste due fasi (scan e find) permette di elencare tutti i device attualmente disponibili nelle vicinanze e tutte le relative informazioni associate ad essi per comunicare. Una volta che si decide su quale dispositivo connettersi, viene eseguita una nuova operazione: Go Negotiation; durante questa fase i due device devono "negoziare" i loro ruoli, ovvero decidere chi ha il ruolo di P2P GO e chi quello di P2P Client, attraverso il meccanismo chiamato three-way handshake. Entrambi i device si inviano un parametro numerico (chiamato il valore intent GO) compreso tra 0 e 15. Per evitare che i due device dichiarano lo stesso numero intent GO, si utilizza un bit aggiuntivo (tie-breaker). Il dispositivo che dichiara il valore più alto diventa P2P GO. Inoltre, sempre in questa fase, viene deciso tra i due device il canale che deve utilizzare il gruppo per

comunicare.

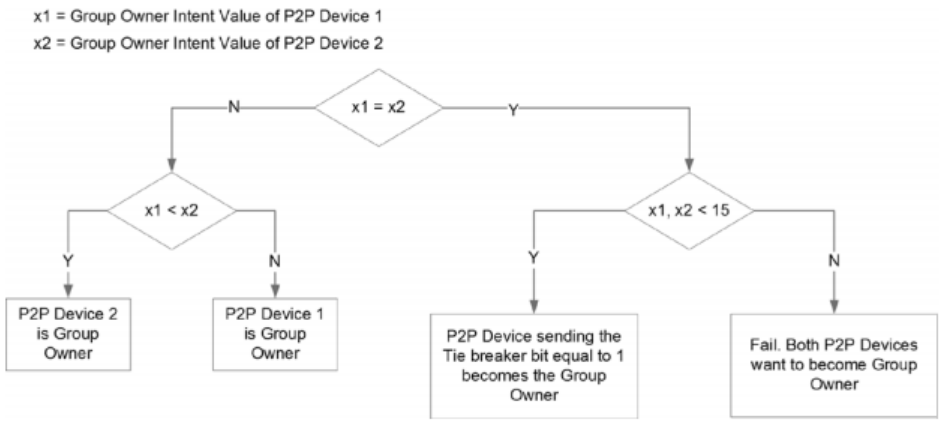

Figura 1.2*:* schema dei possibili casi per eleggere il P2P GO in base ai valori Intent dei dispositivi. *Sorgente : [2]*

Dopo questa fase, si deve stabilire un canale di sicurezza per la comunicazione, attraverso la fase di WPS Provisioning. Questa fase stabilisce una connessione sicura attraverso l'introduzione di un PIN o la pressione di un tasto. Le operazioni di WPS sono composte da due fasi:

- il P2P GO è incaricato di generare tutti i dati che servono al P2P client per avere una connessione sicura e protetta (es: security key, credenziali di accesso);
- dopo che il client riceve queste informazioni si riconnette con le sue nuove credenziali di autenticazione.

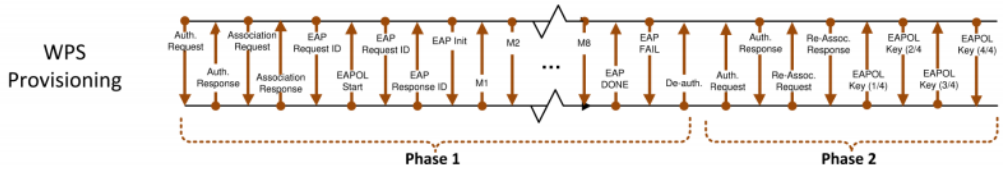

Figura 1.3*:* fase di WPS provisioning. *Sorgente : [1]*

Dopo questa ulteriore fase, un server dhcp, avviato sul dispositivo che funge da P2P GO, imposta gli indirizzi ai vari client del gruppo appena creato (proprio come un AP).

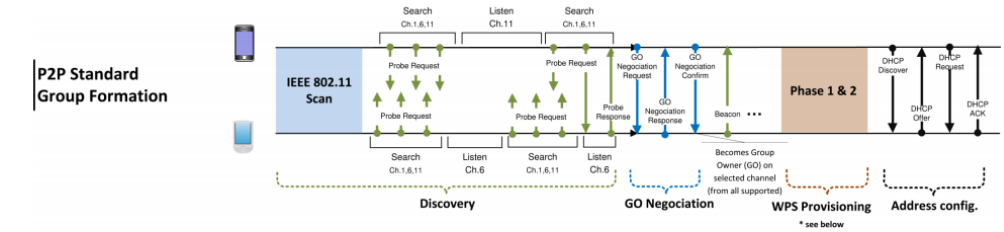

Figura 1.4: formazione di un gruppo standard. *Sorgente : [1]*

1.1.2 Gruppo autonomous

Un device potrebbe autonomamente crearsi un gruppo P2P, dove diventa automaticamente e senza nessuna fase ulteriore, P2P GO. Per rendere visibile il gruppo appena formato, il dispositivo inizia ad inviare Beacon (un segnale periodico inviato dagli AP ai device). Gli altri dispositivi possono scoprire il gruppo appena formato facendo una tradizionale scansione e successivamente si collegano al gruppo procedendo alla fase di WPS e alla configurazione dei vari indirizzi (dhcp). In questo caso non c'è nessuna Go Negotiation e, inoltre, chi forma il gruppo non deve alternarsi con i due stati (listen e search).

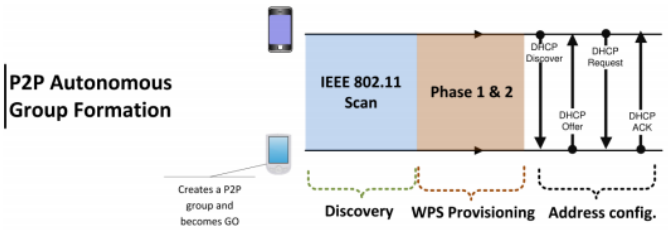

Figura 1.5: formazione di un gruppo autonomo. *Sorgente : [1]*

1.1.3 Gruppo persistent

I device, utilizzando un apposito flag come attributo presente nei Beacon, nei Probe e nella Go negotiation, possono riformare un gruppo che è stato creato precedentemente. Ovviamente con la successiva restaurazione del gruppo P2P, i dispositivi continuano ad avere gli stessi ruoli del gruppo formato in precedenza. Dopo la procedura di Discovery iniziale, se un device riconosce di aver formato un gruppo persistent in passato con altri nodi della rete (*peer)*, il device può usare

la procedura di invito per riprendere velocemente il gruppo. In questo caso la fase di WPS è significativamente ridotta, dato che la restaurazione del gruppo permette il riutilizzo delle credenziali e inoltre la Go Negotiation viene sostituita dalla procedura di invito.

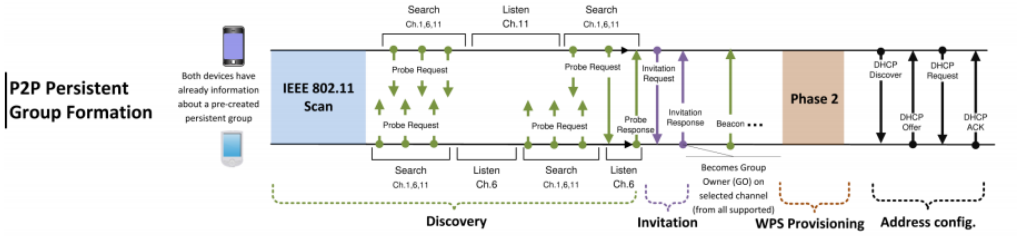

Figura 1.6: formazione di un gruppo persistente. *Sorgente : [1]*

1.2 Struttura del pacchetto

Ogni device comunica attraverso l'invio e la ricezione di pacchetti, chiamati *Information Element* (P2P IE).

Questi pacchetti permettono ai device di scambiarsi le informazioni necessarie per la comunicazione e le varie procedure di instaurazione di un gruppo.

Il formato di tale pacchetto è descritto dall'immagine seguente:

Field	Size (octets)	Value (Hexadecimal)	Description
Element ID	1	0xDD	IEEE 802.11 vendor specific usage.
Length	1	variable	Length of the following fields in the IE in octets. The Length field is a variable, and set to 4 plus the total length of P2P attributes.
OUI	3	50 6F 9A	WFA specific OUI.
OUI Type	1	0x09 (to be assigned)	Identifying the type or version of P2P IE. Setting to 0x09 indicates WFA P2P v1.0.
P2P Attributes	variable		One of more P2P attributes appear in the P2P IE.

Figura 1.7: formato del pacchetto. *Sorgente : [2]*

Un campo del pacchetto degno di nota è il *P2P Attributes*, in cui risiedono tutte le informazioni necessarie al destinatario del pacchetto per comunicare con il mittente.

Come si può notare, è anche l'unico campo che non possiede una grandezza fissa in termini di byte.

Esistono un totale di 220 P2P Attributes e ognuno di essi possiede determinate informazioni a seconda di ogni scopo; il formato di ogni P2P Attributes può essere generalizzato in questa immagine:

Field	Size (octets)	Value (Hexadecimal)	Description
Attribute ID	1	variable	Identifying the type of P2P attribute. The specific value is defined in Table 6.
Length	2	variable	Length of the following fields in the attribute.
Attributes body field	variable		Attribute-specific information fields.

Figura 1.8: formato generale di un P2P attributo. *Sorgente : [2]*

Elenco dei P2P Attributes:

Attribute ID	Notes
0	Status
1	Minor Reason Code
2	P2P Capability
3	P2P Device ID
4	Group Owner Intent
5	Configuration Timeout
6	Listen Channel
7	P2P Group BSSID
8	Extended Listen Timing
9	Intended P2P Interface Address
10	P2P Manageability
11	Channel List
12	Notice of Absence
13	P2P Device Info
14	P2P Group Info
15	P2P Group ID
16	P2P Interface
17	Operating Channel
18	Invitation Flags
19 – 220	Reserved
221	Vendor specific attribute

Figura 1.9: elenco dei possibili p2p attributi. *Sorgente : [2]*

Attraverso un apposito programma dedicato allo sniffer dei pacchetti, è possibile visualizzare il pacchetto durante il suo funzionamento:

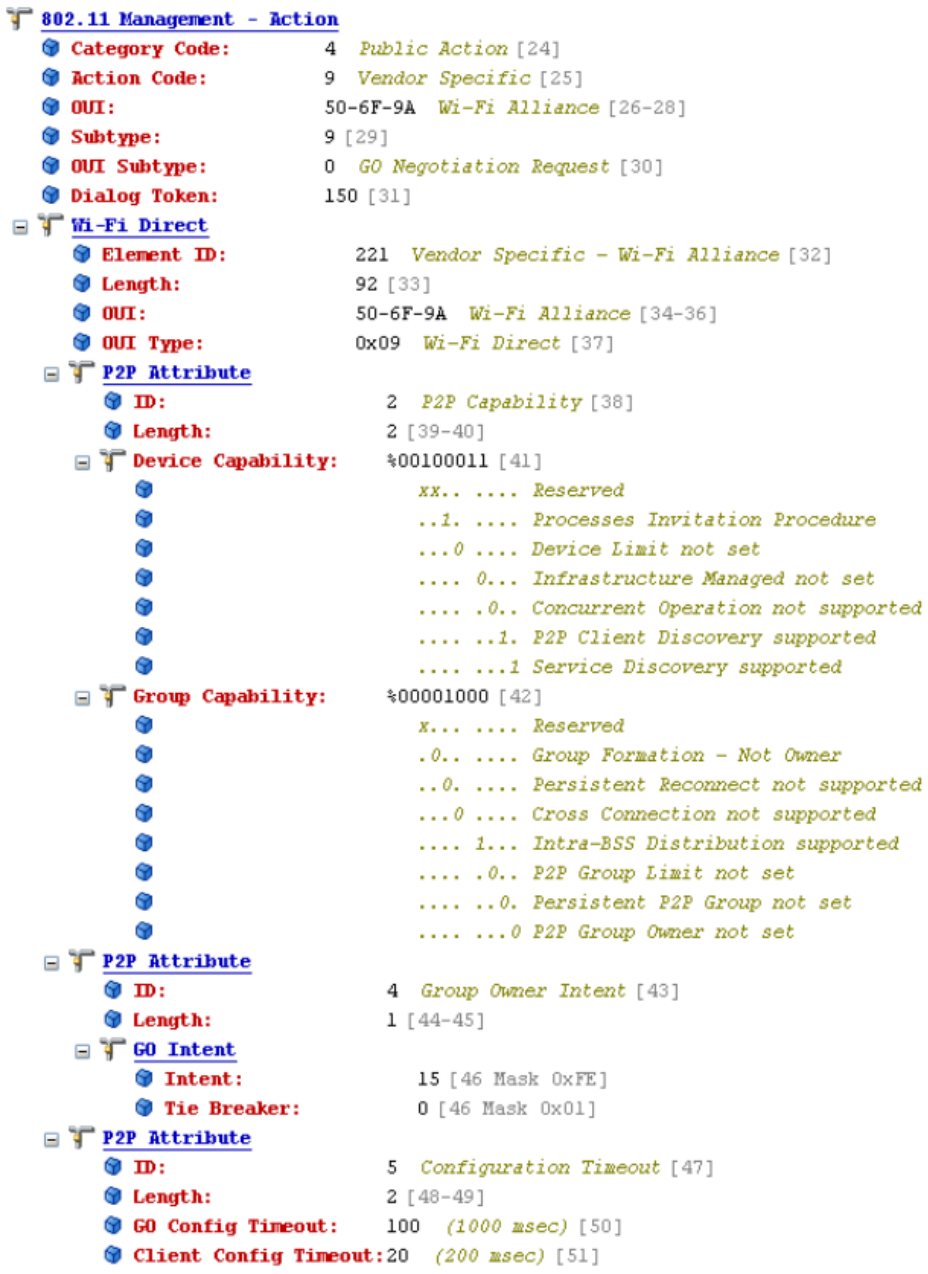

Figura 1.10: pacchetto P2P catturato dal programma Omnipeek. *Sorgente : [2]*

1.3 Applicazioni sullo standard

Il trasferimento e la condivisione dei contenuti in modo semplice e veloce ha reso Wi-Fi Direct una delle principali tecnologie di comunicazione diretta tra

dispositivi. Ormai la maggior parte degli apparecchi moderni supporta lo standard Wi-Fi Direct; oggigiorno è facile immaginare diversi device che fanno utilizzo di tale standard per connettersi e formare reti di condivisione di file.

Può essere usato in differenti ambiti: condivisione di file, sincronizzazione, socializzazione, video games e tutto ciò senza la necessità di trovare una connessione a internet per comunicare con i device.

In Android, solamente se entrambi i dispositivi supportano la tecnologia Wi-Fi Direct, è possibile attivarla nel menù di impostazione del sistema operativo e ricercare da subito i peer disponibili e richiedere quindi una connessione fra loro; in seguito, dopo aver stabilito la connessione con un dispositivo, basterà scegliere un file e, dopo aver scelto dal menu a tendina la voce "Wi-Fi Direct" per condividerlo, partirà la procedura di trasferimento.

Inoltre lo standard, grazie alla sue potenzialità, è stato utilizzato per diversi scopi; tra questi ricordiamo la classe delle applicazioni che utilizzano Wi-Fi Direct per la condivisione di file:

SuperBeam permette di utilizzare la potenza e la velocità di Wi-Fi per inviare e ricevere file in due modalità: la modalità diretta permette il trasferimento dei file usando Wi-Fi Direct, mentre l'altra, la modalità indiretta, permette il trasferimento collegando i dispositivi allo stesso router o access point. Un applicazione molto simile può essere anche "Wi-Fi Shoot".

In aggiunta, numerose industrie che si occupano della produzione di televisori, stampanti e di altre apparecchiature tecnologiche, hanno esteso il mercato delle applicazioni mobili sfruttando la tecnologia Wi-Fi Direct per far comunicare tali apparecchi in modo più semplice e veloce. Ad esempio, applicazioni come iMediaShare e DiXiM permettono la condivisione di contenuti tra TV e cellulari, HP ePrint consente invece ai dispositivi mobili di collegarsi direttamente alle stampanti HP per stampare documenti senza avere una connessione a internet.

Anche in ambito social, esiste un'intera gamma di applicazioni che utilizzano questo standard, come ad esempio Wi-Fi Social che permette di comunicare attraverso la tecnologia Wi-Fi Direct con i dispositivi nelle vicinanze, rendendo visibile il device dagli altri o scegliendo con chi parlare, anche in modo del tutto

anonimo; ciò può essere utile per diversi motivi: dalla semplice socializzazione, oppure anche in casi di necessità in cui altri sistemi di comunicazione risultano inaccessibili o non abbastanza veloci per contattare altri device nelle vicinanze.

1.4 Salvaguardare l'energia della batteria

La batteria del device che funge da P2P GO è di vitale importanza nei gruppi e quindi Wi-Fi Direct mette a disposizione due metodi per ridurre il consumo della batteria.

1.4.1 Opportunistic Power Save

Il P2P GO mandando il Beacon ai propri client, definisce in un campo il CTWindow, ovvero la minima quantità di tempo entro il quale il P2P GO resta attivo. In sintesi, resta attivo dopo quel tempo solamente se c'è almeno un client che invia un flag al P2P GO (chiamato PM – Power Manager) col valore 0 per dire che non è in stato sleep. Nel caso invece il P2P GO si accorge che tutti i client sono in stato sleep, sia perché lo annunciano col flag PM col valore 1 o sia perché erano in stato sleep già durante il precedente Beacon, allora il P2P GO dopo il prefissato tempo minimo, cambia il suo stato in sleep.

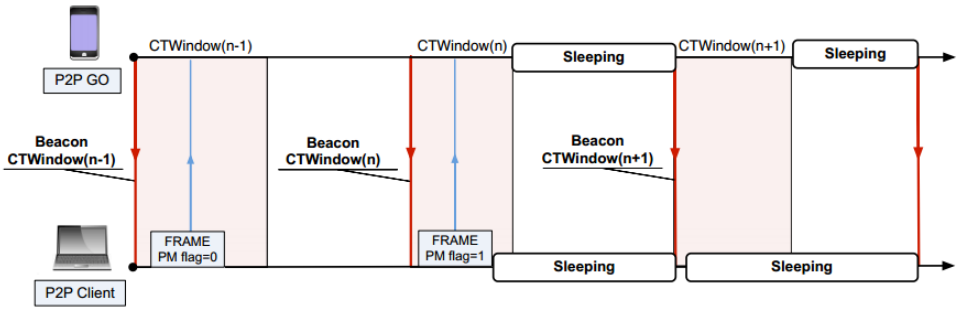

Figura 1.11: esempio di uso del protocollo di risparmio energetico Opportunistic Power Save. *Sorgente : [1]*

1.4.2 Notice of absence

Il P2P GO sempre attraverso i Beacon, diffonde delle informazioni ai client sulle modalità di comunicazione che devono rispettare. In questo caso, il P2P GO informa ai client che non è disponibile per alcuni intervalli di tempo, chiamati *absence periods*. Le informazioni che il P2P GO invia ai client sono composte da 4 parametri:

Durata: indica la durata dei periodi (absence periods);

Intervallo: durata del periodo più lo spazio che intercorre tra un periodo e l'altro;

Tempo: tempo d'inizio del primo absence period;

Count: il numero di absence periods.

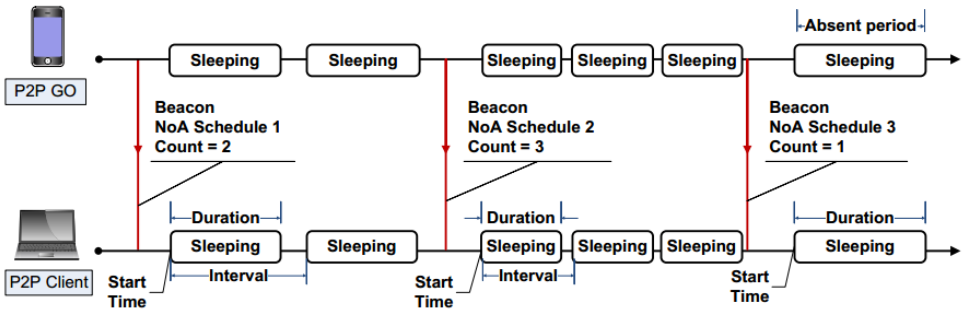

Figura 1.12: esempio di uso del protocollo di risparmio energetico Notice of absence. *Sorgente : [1]*

Capitolo 2

Sistema operativo Android

Android è un sistema operativo per smarthphone e tablet, basato sul kernel Linux. Ogni *release* del sistema ha convenzionalmente un nome simbolico per diversificarla dalle altre versioni: la 1.5 venne chiamata Cupcake, la 1.6 Donut, la 2.1 Eclair, la 2.2 Froyo, la 2.3 Gingerbread, la 3.0 Honeycomb, la 4.0 Ice Cream Sandwich, la 4.1 Jelly Bean, la 4.4 Kit Kat. Attualmente l'ultima versione del sistema operativo è la 5.0, chiamata Lollipop. La politica di licenza di tale sistema è di tipo open source, questo permette la libera distribuzione del codice sorgente. Inoltre Android risulta essere l'oggetto di studio di una vasta comunità di sviluppatori, i quali producono applicazioni con l'intento di ampliare le funzionalità dei dispositivi. Qui di seguito, ci si è interessati del lato programmatico di Android, soffermandosi sulla sua architettura e sull'intero processo di vita di un'applicazione per poi analizzare un esempio implementativo della tecnologia Wi-Fi Direct su Android, discutendo così delle principali procedure e classi che sorreggono tale standard.

2.1.1 Architettura stratificata

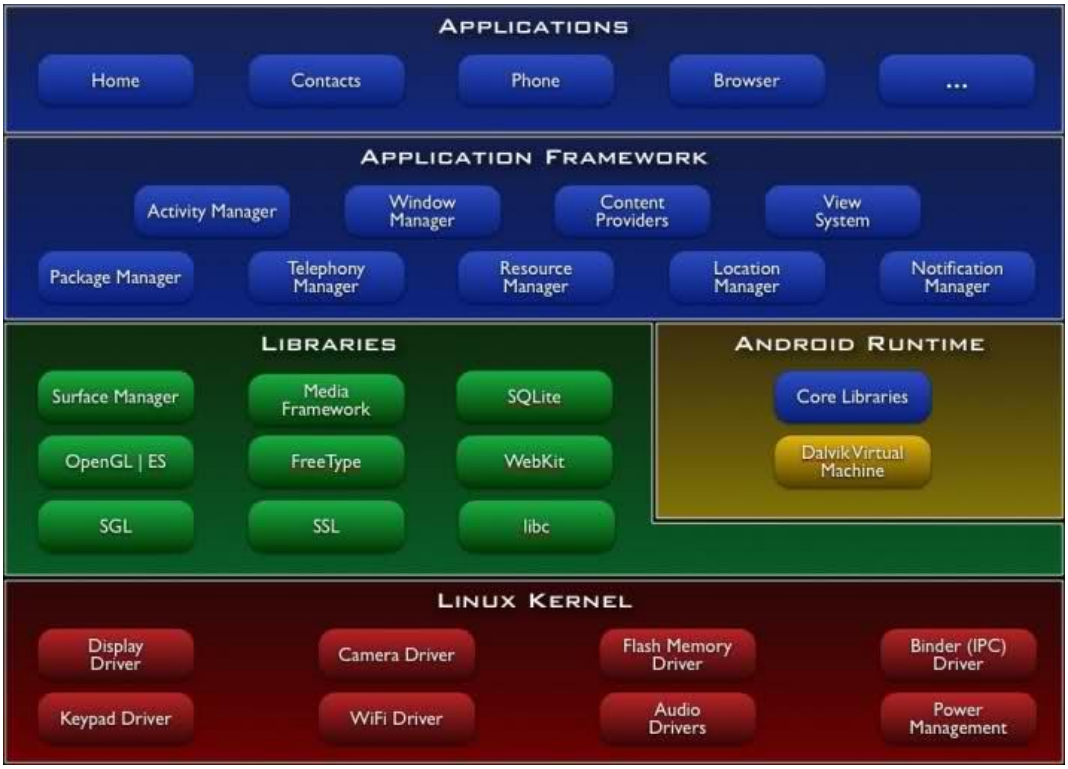

Figura 2.1: architettura stratificata di Android. *Sorgente : [3]*

L'architettura del sistema operativo Android si divide nei seguenti livelli, partendo dal più basso:

-il Linux Kernel corrisponde all'astrazione di tutto l'hardware del dispositivo, che include le principali componenti: Bluetooth, Wi-Fi, GPS, fotocamera, touchscreen;

-Libraries riguarda l'insieme degli strumenti fondamentali utilizzati in ambito di sviluppo per gestire al meglio le applicazioni;

-Android Runtime contiene al suo interno una macchina virtuale simile alla Java Virtual Machine che è ottimizzata in modo da sfruttare la poca memoria situata nei dispositivi mobili;

-Application Framework è composto dai gestori e dalle varie applicazioni di base del sistema operativo come: file system, notifiche, gestione delle telefonate, gestione delle schermate o activity;

-nel livello più alto risiedono tutte le applicazioni installate ed eseguibili in modalità utente.

2.1.2 Compilazione

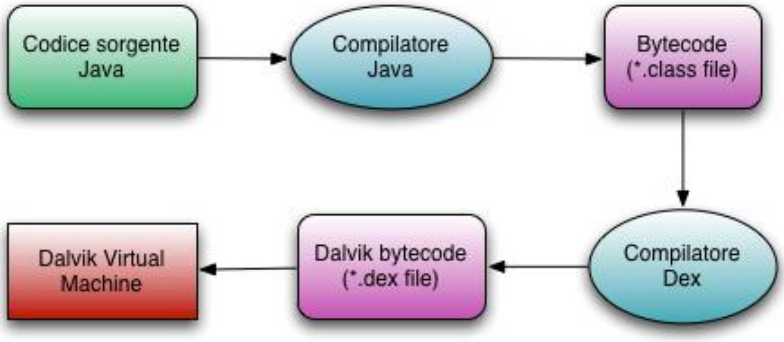

Figura 2.2: processo di compilazione di un file java in Android. *Sorgente : [4]*

Java è il linguaggio di riferimento adoperato per costruire le applicazioni sui dispositivi con sistema operativo Android. Il file (file.java), una volta compilato in formato bytecode (file.class), viene poi ricompilato in Dalvik Bytecode (file.dex), quest'ultimo permette alla macchina virtuale Dalvik di riconoscere tale linguaggio ed eseguirlo.

2.1.3 Componenti di Android

In Android vengono utilizzate quattro componenti principali, queste permettono di integrare alla perfezione un'applicazione nel sistema operativo durante il suo ciclo di vita:

1)le applicazioni in Android possono essere formate da uno o più "pagine", permettendo di inserire input o leggere dati che l'applicazione può manipolare. Questo tipo di interfaccia prende il nome di *Activity*;

2)le applicazioni possono svolgere calcoli anche in *background*, ovvero senza la necessità che l'utente interagisca. Questi componenti che prendono il nome di Service, vengono utilizzati per alleggerire il carico di lavoro che un'applicazione deve supportare;

3)Android dà la possibilità di condividere dei dati fra le applicazioni del sistema.

Questi dati possono essere di diversa natura: dati custoditi in un database, su di un file ecc. Questo tipo di componente prende il nome di Content Provider;

4)una componente molto importante è il BroadcastReceiver che permette all'applicazione di "rilevare" i messaggi che arrivano a livello di sistema, in modo che possa notificarli all'applicazione e gestire tali eventi (come ad esempio la ricezione di un SMS).

Inoltre, Android fa utilizzo di uno strumento di messaggistica per invocare azioni su dei componenti da altri componenti, questi messaggi sono chiamati *Intent*.

2.1.4 Ciclo di vita di un'activity

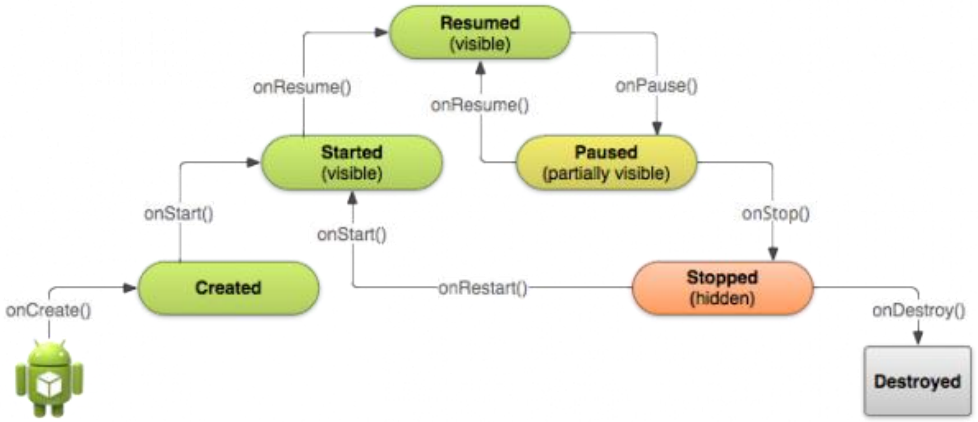

Figura 2.3: il ciclo di vita di un'activity in Android. *Sorgente : [5]*

Comprendere il ciclo di vita di un'activity è di fondamentale importanza per il corretto funzionamento dell'applicazione; essa è composta da differenti stati, ognuno dei quali richiama un metodo di call back da parte del sistema.

Android permette di implementare tali metodi, in modo da poter gestire l'esecuzione di quel metodo in base alle scelte del programmatore, ovviamente allineandosi il più possibile allo scopo dello specifico metodo.

Inizialmente, quando un'activity sta per essere visualizzata vengono eseguiti obbligatoriamente tre metodi:

-`onCreate`*:* a questo metodo si devono assegnare le configurazioni iniziali, come ad esempio il layout dell'activity;

-`onStart`: da questo metodo in poi, l'activity risulta visibile; è possibile avviare

le funzionalità previste dall'applicazione e/o offrire informazioni all'utente;
`onResume`: l'utente dopo la chiamata di questo metodo può interagire con l'applicazione, inserendo ad esempio input.
Durante l'esecuzione di un'activity, sono numerose le situazioni in cui l'activity attualmente visualizzata potrebbe essere sostituita con un'altra, ad esempio una chiamata o semplicemente l'apertura di un'ulteriore activity da parte dell'activity attualmente eseguita. Anche in questo caso, l'attuale activity visualizzata dall'utente deve effettuare una serie di metodi per dare "spazio" a un'ulteriore activity:
-`onPause`: appena viene chiamata, l'interazione dell'utente con l'activity viene cessata, in alcuni casi quest'ultima potrebbe essere anche parzialmente visibile dopo l'esecuzione di questo metodo;
-`onStop`: anche la visibilità dell'activity cessa di esistere.
-`onDestroy`: chiamato in casi in cui si decide di "distruggere" l'activity e terminarla per sempre. Questo metodo può essere eseguito in casi in cui il sistema ha bisogno di memoria, o anche per volere del programmatore.

2.1.5 Il cuore dell'applicazione

Un'applicazione per Android è composta da un insieme di file ognuno dei quali si occupa di uno specifico compito. Tra questi, esiste un file che si occupa della definizione dei contenuti e dei permessi di cui l'applicazione ha bisogno per funzionare, chiamato AndroidManifest.xml; nel file vengono riportate le componenti di cui l'applicazione necessita (activity, service ecc.) e i permessi che il processo deve possedere per eseguire l'applicazione.

2.2 Wi-Fi Direct su Android

Lo standard Wi-Fi Direct,meglio conosciuto come Wi-Fi P2P, è disponibile dalle versioni di Android 4.0 (livello API 14) in poi e, inoltre, il dispositivo deve avere un hardware appropriato per l'interazione diretta tra i dispositivi senza nodi

intermedi (Access Point).

2.2.1 Principi generali

Il Wi-Fi P2P su Android a livello implementativo, è composto da un insieme di funzioni disponibili al programmatore, chiamate API e che sono divise in tre parti principali:

-tutti i metodi definiti nella classe WifiP2pManager che consentono la scansione, richiesta e connessione con i peer;

-i listener, "oggetti" che permettono di notificare stati di "successo" o "fallimento" quando si eseguono metodi della classe WifiP2pManager;

-Intent che notificano per ogni specifico evento rilevato dal device, come ad esempio se la connessione cade, se un peer è stato appena scoperto ecc.

Qui di seguito, sono esposti 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 2.4: metodi della classe Wi-Fi P2pManger. *Sorgente : [6]*

Ogni metodo della classe WifiP2pManager è collegato a un listener, il quale a seconda dell'esito della chiamata del metodo, si occupa di avvisare con un messaggio di successo o fallimento.

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 2.5: lista dei listener associati ai vari metodi della classe WifiP2pManager. *Sorgente : [6]*

Inoltre le APi di Wi-Fi P2P definiscono degli intent che, registrati su un BroadcastReceiver, permettono all'applicazione di rilevare gli eventi che accadono in un preciso istante.

2.2.2 Un esempio pratico

Inizialmente bisogna settare il manifest con gli opportuni privilegi, in modo che l'applicazione possa accedere alla componente hardware Wi-Fi e che soprattutto sia assicurato il supporto del protocollo Wi-Fi P2P sul device.

```
<uses-sdk android:minSdkVersion="14" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 2.6: elenco di permessi necessari per il funzionamento dell'applicazione nel file manifest.xml. *Sorgente : [6]*

Dopo i privilegi, bisogna ottenere l'istanza di WifiP2pManager attraverso il metodo `getSystemService` e successivamente, con la stessa istanza, eseguire il metodo `initialize` che ritorna un channel il quale permette all'applicazione di connettersi al Wi-Fi P2P framework. Dopodiché è necessario salvare nel BroadcastReceiver i due valori inizialmente presi (ovvero l'istanza di WifiP2pManager e il channel) e l'elenco degli intent che s'intende registrare. Dopo la chiamata del metodo `registerReceiver`, il BroadcastReceiver è in

ascolto di tutti i possibili eventi, specificati nell'elenco degli intent, che si scatenano all'interno dell'applicazione.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // add necessary intent values to be matched.

    intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);

    manager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    channel = manager.initialize(this, getMainLooper(), null);
}

/** register the BroadcastReceiver with the intent values to be matched */
@Override
public void onResume() {
    super.onResume();
    receiver = new WiFiDirectBroadcastReceiver(manager, channel, this);
    registerReceiver(receiver, intentFilter);
}
```

Figura 2.7: configurazione iniziale di Wi-Fi Direct. *Sorgente : [6]*

Nel metodo `onReceive` del BroadcastReceiver bisogna gestire i vari intent che vengono rilevati, apportando le relative modifiche all'interfaccia grafica dell'activity principale.

```
@Override
public void onReceive(Context context, Intent intent) {
    ...
    String action = intent.getAction();
    if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
        int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
        if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
            // Wifi P2P is enabled
        } else {
            // Wi-Fi P2P is not enabled
        }
    }
    ...
}
```

Figura 2.8: ritaglio di codice della gestione dei vari intent rilevati dal BroadcastReceiver. *Sorgente : [6]*

Adesso è possibile implementare nell'applicazione le varie funzionalità di Wi-Fi Direct come ad esempio la fase di Discovery e la connessione con i peer:
Per elencare i peer nelle vicinanze al momento disponibili a connettersi, si deve eseguire il metodo `discoverPeers()`. Il listener associato al metodo `discoverPeers()` (ovvero ActionListener), avverte l'applicazione con una notifica di fallimento o successo.

```
mManager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
    @Override
    public void onSuccess() {
        ...
    }

    @Override
    public void onFailure(int reasonCode) {
        ...
    }
});
```

Figura 2.9: implementazione della notifica di successo o fallimento del listener in ascolto sul metodo discoverPeers(). *Sorgente : [6]*

La notifica di successo da parte della fase di Discovery non dà nessuna informazione sui rispettivi device scoperti, avverte solamente che il processo di Discovery è andato a buon fine.
Successivamente alla notifica di successo, il sistema si occupa di trasmettere un intent associato al processo di Discovery, il quale permette al BroadcastReceiver di rilevarlo e eseguire le seguenti operazioni: viene chiamato il metodo `requestPeers()` per richiedere la lista dei peer trovati;

```
PeerListListener myPeerListListener;
...
if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

    // request available peers from the wifi p2p manager. This is an
    // asynchronous call and the calling activity is notified with a
    // callback on PeerListListener.onPeersAvailable()
    if (mManager != null) {
        mManager.requestPeers(mChannel, myPeerListListener);
    }
}
```

Figura 2.10: richiesta della lista di peer col metodo requestPeers. *Sorgente : [6]*

subito dopo, viene eseguito in modo asincrono il metodo `onPeersAvailable()` che si occupa di modificare l'interfaccia grafica dell'activity inserendo la lista dei device trovati.

Una volta scelto un determinato dispositivo dalla lista, bisogna chiamare il metodo `connect()` per connettersi ad esso. Tale metodo richiede come parametro formale un oggetto Wi-Fi P2pConfig che contiene le informazioni necessarie per connettersi col dispositivo scelto. Il metodo `connect()` è associato allo stesso listener del metodo `discoverPeers()` e pertanto anche qui è possibile implementare nel medesimo modo la notifica di successo o fallimento.

```
//obtain a peer from the WifiP2pDeviceList
WifiP2pDevice device;
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
mManager.connect(mChannel, config, new ActionListener() {

    @Override
    public void onSuccess() {
        //success logic
    }

    @Override
    public void onFailure(int reason) {
        //failure logic
    }
});
```

Figura 2.11: implementazione della notifica di successo o fallimento del listener in ascolto sul metodo connect(). *Sorgente : [6]*

Subito dopo la notifica di successo, sul BroadcastReceiver viene eseguito il metodo `requestConnectionInfo()` che si occupa di richiedere le informazioni relative alla connessione appena instaurata.

Appena le informazioni sono disponibili, viene eseguito il metodo `onConnectionInfoAvailable()`. Con questo metodo è possibile sapere l'esito della Go Negotiation fra i due device e, in base al risultato, vengono apportate le seguenti operazioni:

nel caso il device risulti essere il P2P GO, viene eseguito un AsyncTask (che corrisponde in Android ad un comune thread utilizzato per svolgere operazioni in

background), il quale permette al P2P GO di restare in attesa delle connessioni dei P2P Client e gestire l'instradamento dei pacchetti. Queste operazioni sono gestite grazie ai Socket, che rappresentano l'interfaccia usata per la trasmissione e la ricezione dei dati all'interno di una rete.
Il P2P client, invece, deve salvarsi l'indirizzo e la porta del Socket P2P GO e, mediante un service, può contattare il P2P GO per trasmettere i propri dati.

2.2.3 Dettagli aggiuntivi

L'applicazione utilizza un file locale,chiamato p2p_supplicant.conf, dove è possibile salvare tutte le credenziali di accesso e le informazioni sui possibili gruppi persistent in cui si è partecipato. Inoltre è utilizzato in background un demone, chiamato wpa_supplicant, che si occupa della gestione delle connessioni wireless su piattaforma Android/Linux OS .

2.3 Scenario fra due dispositivi

Si considerano due device, A e B, entrambi eseguono il metodo `WifiP2pManager.DiscoverPeers()` che richiede al wpa_supplicant di iniziare una fase di Scan e Find. La fase di Discovery rimane attiva fino ad un totale di 120 secondi a meno che non si instauri una connessione con un peer trovato, oppure non si formi un gruppo "autonomamente" con la primitiva `WifiP2pManager.createGroup()`.
Ogni volta che un nodo viene scoperto, wpa_supplicant notifica l'applicazione con un evento chiamato "P2P Device Found", cosicché l'applicazione possa invocare delle determinate funzioni per estrapolare le informazioni del nodo appena trovato, per successivamente connettersi.

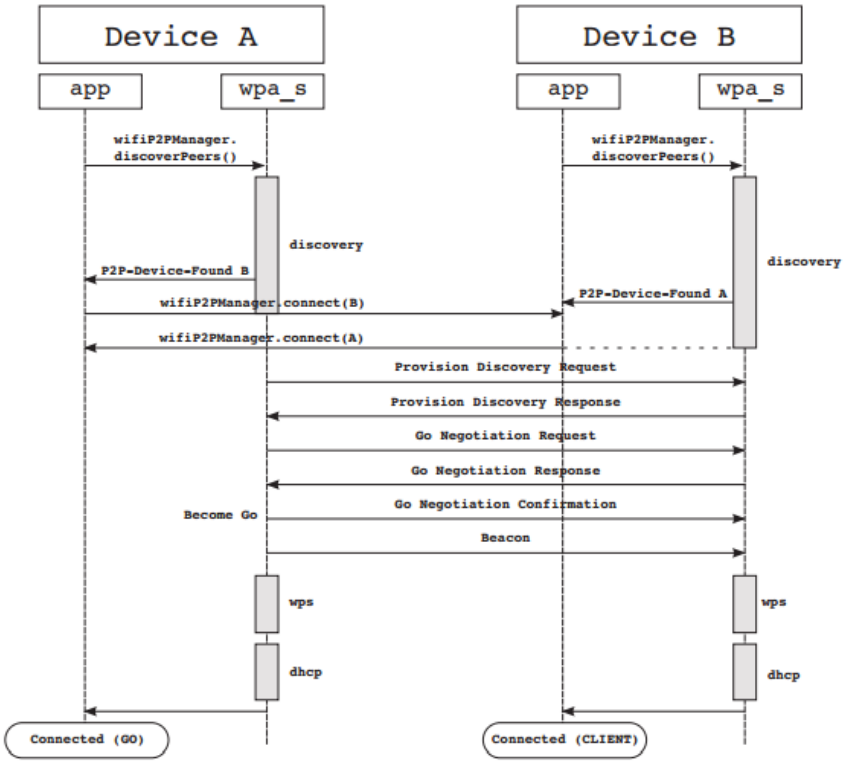

Figura 2.12: diagramma di sequenza per la fase di Group Formation. *Sorgente : [8]*

Capitolo 3

Definizione delle metriche e degli scenari

In questo capitolo sono state affrontate dapprima tre metriche le quali sono state necessarie per analizzare globalmente il protocollo nella sua efficacia e, in particolare, nella trasmissione dei dati tra il P2P Client e il P2P GO, una volta formato un gruppo.
In seguito, viene mostrato il funzionamento dell'applicativo costruito per effettuare le stime mediante le metriche di valutazione. Inoltre, la tesi si è occupata di un'ulteriore analisi e, in particolare, della quantità di tempo necessaria per completare la fase di Discovery e la Group Formation, mostrando infine, l'applicativo che ha permesso di ricavare i dati.

3.1 Metriche di valutazione

Per eseguire i test in modo completo sul Wi-Fi Direct, sono stati utilizzati due smartphone: il primo ha il compito di creare un gruppo P2P (in modo da agire da P2P GO), mentre il secondo deve solamente accedere al gruppo appena creato e, in questo modo, fungere da P2P Client. Da questo momento in poi, il P2P Client inizia ad inviare dati al P2P GO, cosicché quest'ultimo possa misurare le tre metriche. Con lo scopo di valutare in modo opportuno Wi-Fi Direct ed evitando di prelevare dati poco probabili, sono stati calcolati i valori delle tre metriche per cinquanta secondi.

3.1.1 Packet Delivery Ratio

Il Packet Delivery Ratio è il rapporto fra il numero di pacchetti ricevuti a destinazione e il numero di pacchetti inviati dalla sorgente.

$$Packet\ Delivery\ Ratio = \frac{\sum_{i=0}^{y} \text{i}}{\sum_{j=0}^{x} \text{j}}$$

Dove *x* rappresenta il numero di pacchetti inviati dalla sorgente e *y* il numero di quelli ricevuti a destinazione. Il risultato finale è compreso fra 0 e 1.

3.1.2 Throughput

Il Throughput rappresenta l'effettiva quantità di byte trasmessi in una determinata quantità di tempo.

$$Throughput\ (bit/s) = \frac{(numero\_di\_pkt) * 8 * dimensione\_pkt}{quantità\ di\ tempo}$$

Nei test, la quantità di tempo è stata di 50 secondi e la dimensione di ogni pacchetto è stata di 1000 byte.

3.1.3 Round Trip Time

Il Round Trip Time indica la quantità di tempo impiegata da un pacchetto per raggiungere la destinazione e poi tornare indietro.

$$Round\ Trip\ Time = \frac{\sum_{i=0}^{n} (\text{pkt}_{\text{i}}\ \text{timeB}) - (\text{pkt}_{\text{i}}\ \text{timeA})}{\text{n}}$$

Dove, *n* indica il numero di pacchetti che raggiungono la destinazione e ritornano alla sorgente, *timeB e timeA* indicano rispettivamente il tempo di ricezione e quello di invio.

3.2 Panoramica del software per il calcolo delle metriche

Lo smartphone che agisce da P2P GO si occupa di calcolare il Packet Delivery Ratio e il Throughput; invece il P2P Client affronta il calcolo del Round Trip Time dato che, per quest'ultima metrica, è necessario che il P2P Client riceva i pacchetti che ha inviato in precedenza al P2P GO.
Il test è composto da 6 cicli (ciascuno della durata di 50 secondi), ognuno dei quali valuta le tre metriche con un numero diverso di pacchetti.

3.2.1 Server side

Sono stati effettuati i test mediante l'ausilio di un gruppo *autonomo*. Lo smartphone che funge da server (P2P GO), forma il gruppo P2P attraverso la primitiva `createGroup()` e, con il listener associato (ActionListener), è possibile verificare l'esito di questa operazione.

```
public void crea_gruppo(){
    manager.createGroup(channel, new WifiP2pManager.ActionListener() {

        @Override
        public void onSuccess() {
            Toast.makeText(WiFiDirectActivity.this, "Success!",
                    Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onFailure(int reasonCode) {
            Toast.makeText(WiFiDirectActivity.this, "Failure!" + reasonCode,
                    Toast.LENGTH_SHORT).show();
        }
    });
}
```

Figura 3.1: formazione di un gruppo attraverso la primitiva createGroup() contenuta all'interno di una funzione convenzionalmente chiamata crea_gruppo().

Se l'operazione termina con successo, il BroadcastReceiver dell'applicazione rileva l'intent WI-FI _P2P_CONNECTION_CHANGED_ACTION, il quale viene trasmesso dal sistema ogni volta che lo stato della connessione Wi-Fi del dispositivo cambia. A questo punto, dopo aver rilevato l'intent, il BroadcastReceiver esegue il metodo `requestConnectionInfo()`, per richiedere le informazioni necessarie al dispositivo per la formazione del gruppo P2P. Le informazioni sono reperibili nel metodo `onConnectionInfoAvailable()` che permette, come si è visto nel precedente capitolo, di sapere l'esito della Go Negotiation e, in base al risultato, instanziare delle classi opportune per comunicare con i dispositivi del gruppo. Ma, data la natura del gruppo *autonomo*, la Go Negotiation non viene eseguita, il dispositivo diventa immediatamente P2P GO e instanzia un AsyncTask, che d'ora in avanti per semplicità viene chiamato thread1. Questo thread a sua volta crea un

nuovo thread (thread2) ed insieme eseguono le seguenti operazioni:

-il thread1 subito dopo aver creato il thread2, si mette in attesa per 50 secondi e, una volta terminati, attraverso delle variabili condivise col thread2, calcola le due metriche (Packet Delivery Ratio e Throughput). Dopo questa operazione, il thread1 ritorna in stato sleep per poi calcolare le metriche dei restanti cicli;

```
//thread1
synchronized(this){
    //notify sbloccherà il thread2
    this.notify();
}
```

```
//thread2
synchronized(_thread1) {
    try {
        _thread1.wait(); //in attesa del thread1
    } catch (InterruptedException e) {
        //
    }
}
```

Figura 3.2: esempio di messaggistica utilizzata per svegliare il thread2 tramite il thread1.

-il thread2 viene chiaramente creato una volta sola e, dal secondo ciclo in poi, entrambi i thread comunicano attraverso dei messaggi. Mentre il thread1 è in stato sleep per 50 secondi, il thread2 è in attesa di pacchetti (dati) provenienti dal client. Per ogni pacchetto ricevuto, vengono utilizzate due variabili condivise fra i due thread: una prende la dimensione di ogni pacchetto per ricavare il Throughput totale per 50 secondi, un'altra è un contatore per ricavare il numero di pacchetti ricevuti (per calcolare il Packet Delivery Ratio). Inoltre, per favorire al client il calcolo del Round Trip Time, il thread2 del server restituisce ogni pacchetto ricevuto dal client (senza alterarne il contenuto).

Figura 3.3: interfaccia grafica dell'applicazione del dispositivo che funge da P2P GO.

Una volta che sono trascorsi 50 secondi, il thread1 avverte il thread2; quest'ultimo si blocca (attraverso il meccanismo di messaggistica discusso poco prima) per consentire al thread1 il calcolo delle grandezze. Subito dopo questa operazione, il thread1 sblocca il thread2 per procedere con i restanti cicli.

```
while(true){
    try {
        WiFiDirectActivity.serverSocket.receive(WiFiDirectActivity.receivePacket);    //ricevi pacchetti
        String sentence = new String( WiFiDirectActivity.receivePacket.getData());    //preleva valore
        String[] pa = sentence.split(";"); //funzione per prelevare i dati contenuti tra i ";"
        if(pa[1].equals(part3)){ //blocco di codice dedicato all'esaminamento del pacchetto appena ricevuto.
            InetAddress IPAddress = WiFiDirectActivity.receivePacket.getAddress();
            int port = WiFiDirectActivity.receivePacket.getPort();
            byte [] sum = sentence.getBytes(); //invio con lo stesso contenuto
            DatagramPacket sendPacket = new DatagramPacket(sum, sum.length, IPAddress, port);
            WiFiDirectActivity.serverSocket.send(sendPacket); // Restuisci pacchetto al client (Round trip Time)
            WiFiDirectActivity.i=WiFiDirectActivity.i+1; // Calcolo del Packet Delivery Ratio
            WiFiDirectActivity.cont = WiFiDirectActivity.cont+ sum.length; // Calcolo del Throughput
        }
    } catch (SocketTimeoutException e) {
        //
    } catch (IOException e) {
        e.printStackTrace();
    }
    if(ferma==1){ //il thread 1 mi avverte di terminare la lettura
        break;
    }
} //fine while
```

Figura 3.4: ciclo eseguito dal thread2 per ricevere pacchetti dal P2P Client. Nella porzione di codice cerchiata in rosso è possibile notare il calcolo delle metriche.

3.2.2 Client side

Lo smartphone che agisce da client (P2P Client), si connette al gruppo appena formato dal P2P GO attraverso una semplice scansione Wi-Fi (il gruppo creato tramite la primitiva `createGroup()`, forma una rete P2P visibile da qualsiasi dispositivo che supporti la tecnologia Wi-Fi). A differenza del P2P GO, il P2P Client non utilizza nella sua applicazione i metodi della classe WifiP2pManager, BroadcastReceiver o altre componenti normalmente utilizzate in un'applicazione, ma dal momento che il dispositivo si connette al gruppo in modo manuale attraverso il protocollo Wi-Fi , è necessario utilizzare solamente due thread che permettano di inviare dati al P2P GO attraverso l'uso delle socket. Il primo thread, che per semplicità è stato chiamato thread1, crea un nuovo thread (thread2). Il primo thread si occupa di inviare pacchetti al server e il secondo di riceverli. Entrambi i thread condividono tra di loro due array, indispensabili per gestire

l’instradamento dei pacchetti, chiamati rispettivamente matrix e timex.

```
String[] matrix = new String[6];
matrix[0]= "100;a";  //numero di pkt/s , nome ciclo
matrix[1]= "500;b";
matrix[2]= "1000;c";
matrix[3]= "5000;d";
matrix[4]= "9000;e";
matrix[5]= "15000;f";
```

Figura 3.5: array matrix.

I dati contenuti nell’array matrix sono indispensabili al thread1 del client per capire quanti pacchetti deve inviare al server per ogni ciclo. Inoltre c’è bisogno di un ulteriore array, che permette di calcolare il Round Trip Time (timex).

Le due entità lavorano in parallelo nel seguente modo:

-il thread1 inizialmente, a partire dalla prima cella dell’array (ossia al primo ciclo), invia 100 pacchetti al secondo, per un totale di 50 secondi (5000 pacchetti). Ogni volta che invia un pacchetto *i*, salva l’attuale tempo alla posizione *i*-esima dell’array timex. Inoltre, prima dell’invio del pacchetto, il thread1 si occupa di inserire alcuni dati in esso, che servono poi al thread2 per distinguere i pacchetti che riceve dal server:

`lettera;i`, dove `lettera` indica il nome del ciclo che si è preso in considerazione e `i` corrisponde all’indice del pacchetto, in modo da poter ripescare nella matrice timex il tempo d’invio del pacchetto;

-nello stesso instante, il thread2, come già detto prima, è in attesa di pacchetti proveniente dal server e, appena ne riceve uno, preleva il suo indice. Una volta prelevato il tempo di invio del pacchetto dall’array timex, basta eseguire una semplice sottrazione col tempo attuale per sapere il Round Trip Time del pacchetto appena ricevuto.

```
while(true) {
    try {
        MainActivity.clientSocket2.receive(receivePacket); //ricevi pacchetto
        long after  = (System.currentTimeMillis()); //tempo di ricezione
        String se = new String(receivePacket.getData());
        String[] dati_pkt = se.split(";"); //preleva dati dal pkt
        domanda2 = dati_pkt[1];  //preleva il nome del ciclo
        if(domanda2.equals(ciclo_attuale) && (MainActivity.ferma !=1)){ //
            int indice_pkt = Integer.parseInt(dati_pkt[2]);
            long start = (Long.valueOf(MainActivity.timex[indice_pkt]));
            xx=xx+1; //pacchetti ricevuti
            total = total + (after-start);  //rtt di tutti i pkt ricevuti
            //media finale del Round Trip Time = (total/xx)
```

Figura 3.6: frammento di codice eseguito dal thread2 per la ricezione dei pacchetti e il calcolo del Round Trip Time.

Dopo la conclusione di ogni ciclo (50 secondi), vengono azzerate tutte le variabili utilizzate (compresa la matrice timex) e i due thread procedono a iterare nel medesimo modo le operazioni appena descritte per i restanti cicli.

3.3 Funzione di ripartizione del tempo di Discovery e Group Formation

In statistica, la funzione di ripartizione, è utilizzata per ricavare la probabilità che una variabile casuale *X* assuma valori minori o uguali a un dato valore *x* e può essere così definita:

$$F_X : R \to [0,1] \qquad F_X(x) := P(X \leq x)$$

Dove P(X ≤ x) rappresenta la probabilità descritta prima, che è appunto, uguale alla funzione di ripartizione.

La funzione di ripartizione è stata necessaria per tracciare la probabilità che un determinato evento,ossia il completamento della fase di Discovery e della Group Formation, si verifichi entro un dato valore t di tempo (espresso in secondi).

3.3.1 Panoramica del software

Per ricavare il piano cartesiano con l'andamento della funzione di ripartizione per le due fasi (Discovery e Group Formation), è stato fondamentale costruire un ulteriore applicativo.

Anche per questo test, è stato fondamentale l'uso di due smartphone con il supporto al Wi-Fi Direct: il primo utilizza un'applicazione che permette di ricavare i dati necessari per le stime, per il secondo, invece, non è necessaria l'installazione dello stesso applicativo e, pertanto, si occupa semplicemente di rendersi visibile dall'altro smartphone.
Il dispositivo che si serve dell'applicazione, effettua inizialmente una Discovery che permette di elencare l'altro dispositivo e, una volta trovato, si connette creando il gruppo P2P standard e concludendo il test.
In particolare:
-nell'applicazione, al momento della `DiscoverPeers()`, subito dopo la notifica di successo, viene inizializzata una variabile contenente il tempo attuale (in millisecondi).

Figura 3.7: alla sinistra viene mostrata la schermata dell'applicazione durante la ricerca dei dispositivi, alla destra quella visualizzata subito dopo averne trovato uno.

Una volta che il BroadcastReceiver rileva l'intent, fa la richiesta dei peer trovati. Quando viene eseguita in modo asincrono `onPeersAvailable()`, viene prelevato il tempo attuale attraverso l'uso di un'ulteriore variabile, ottenendo così, tramite una semplice sottrazione, il tempo di Discovery fra i due smartphone. Anche per calcolare il tempo richiesto per formare il gruppo, si procede nel medesimo modo della Discovery, utilizzando però, le primitive interessate alla connessione con il peer.

Figura 3.8: schermata dell'applicazione che mostra i risultati ottenuti.

Capitolo 4
Risultati sperimentali

Dopo aver ottenuto i dati sperimentali, ci si è occupati di rappresentarli graficamente.

4.1 Tipologie di valutazione

Qui di seguito, sono mostrati i risultati dei test effettuati durante lo studio dello standard Wi-Fi Direct. Lo standard è stato comparato, in alcuni casi, con altri protocolli, per permettere di valutare in modo completo le sue prestazioni.
I grafici sono stati suddivisi in due categorie: quelli relativi al calcolo delle metriche e quelli che riguardano Discovery e Group Formation.

4.1.1 Confronto tra Wi-Fi ed altre tecnologie

Il calcolo delle metriche è stato inizialmente effettuato in *Localhost* (sia il client che il server risiedono nella stessa macchina), in Wi-Fi (il client e il server sono su macchine differenti e comunicano fra loro attraverso un punto di accesso) e in *Ethernet* (il client e il server risiedono anch'essi su macchine differenti, ma comunicano mediante un cavo Ethernet).
Nella prima metrica (Packet Delivery Ratio) , si nota come il test effettuato in localhost risulti essere il più efficiente fra i tre, seguito poi dal protocollo Ethernet e infine dal Wi-Fi .

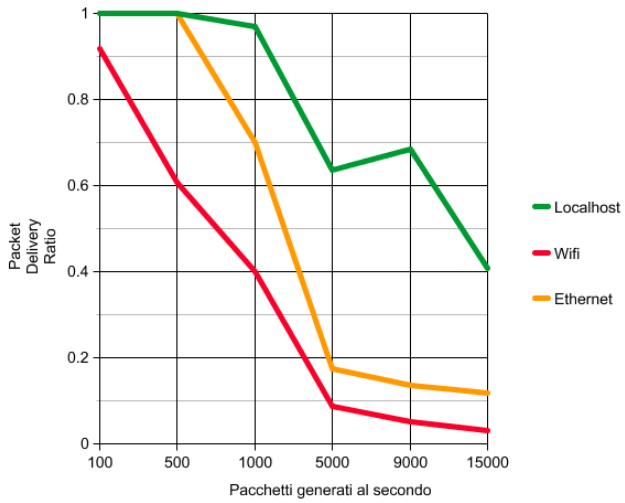

Figura 4.1: grafico PDR per Localhost,Wi-Fi ed Ethernet.

Per quanto concerne la metrica del Throughput, risulta maggiore la quantità di dati nel test in Localhost, specialmente quando si arriva ai 5000 pacchetti al secondo dove, sia il protocollo Wi-Fi che quello Ethernet, perdono una quantità maggiore di dati rispetto al Localhost. Inoltre si può notare che, anche in questa metrica, il Wi-Fi risulta essere il protocollo meno efficiente e inizialmente quasi alla pari delle prestazioni del protocollo Ethernet.

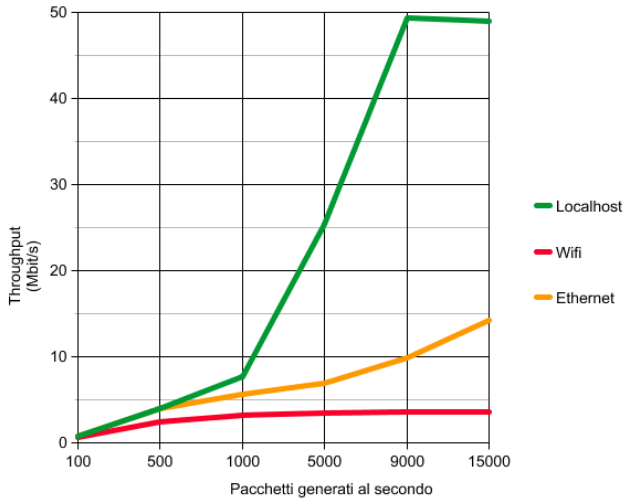

Figura 4.2: grafico Throughput per Localhost,Wi-Fi ed Ethernet.

Il grafico del Round Trip Time mostra un ritardo maggiore nel caso del protocollo Wi-Fi , che si prolunga in modo elevato all'aumentare dei pacchetti trasmessi al

secondo. Anche nel test eseguito in Localhost, si è riscontrato un aumento del ritardo in seguito ad una maggiore quantità di pacchetti trasmessi, ma non si superano i 120 ms rispetto ai 530 ms nel caso del Wi-Fi .

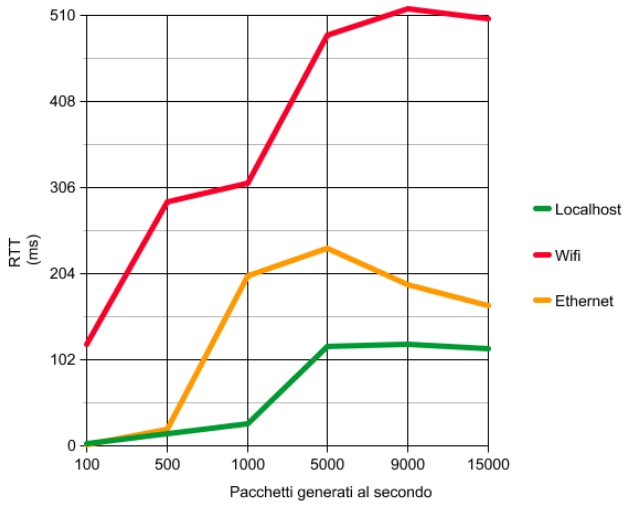

Figura 4.3: grafico RTT per Localhost,Wi-Fi ed Ethernet.

4.1.2 Valutazione del Wi-Fi Direct

Successivamente, ci si è occupati di eseguire i test delle metriche sul Wi-Fi Direct, valutando lo standard sia in ambiente outdoor che in quello indoor. Nei test eseguiti, è stato riscontrato che nel caso outdoor, i due smartphone hanno una portata di ricezione massima fino a 65 metri e mentre nei test indoor non si superano i 20 metri.

Inoltre, per quanto riguarda la prima metrica (Packet Delivery Ratio), nel caso outdoor la soglia massima dei 60-65 metri perde all'inizio leggermente più pacchetti rispetto alla soglia massima dei 20 metri nel caso indoor, poi, con l'aumentare dei pacchetti, le soglie convergono nei medesimi punti.

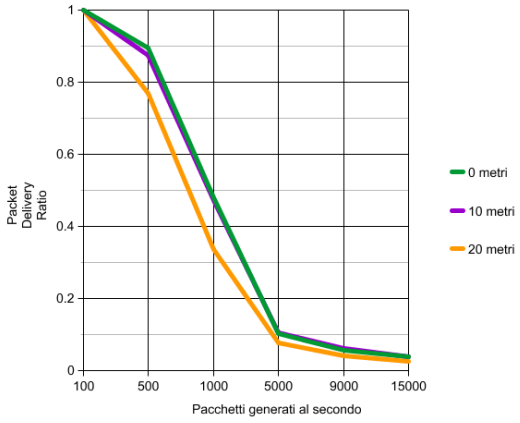

Figura 4.4*:* PDR indoor.

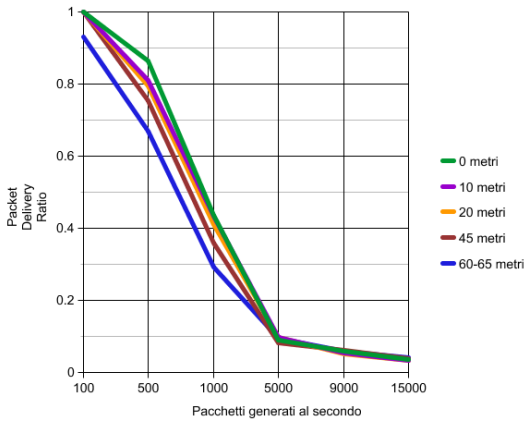

Figura 4.5: PDR outdoor.

Nel Throughput non si superano i 5 Mbit/s di dati, sia nel caso outdoor che indoor. La soglia massima nel caso indoor perde una quantità maggiore di byte rispetto al caso outdoor, in aggiunta, nel caso outdoor, l'ultima soglia di distanza, inizialmente perde una quantità maggiore di byte rispetto alle altre soglie, ma dai 1000 pacchetti in poi la quantità di byte sembra aumentare considerevolmente.

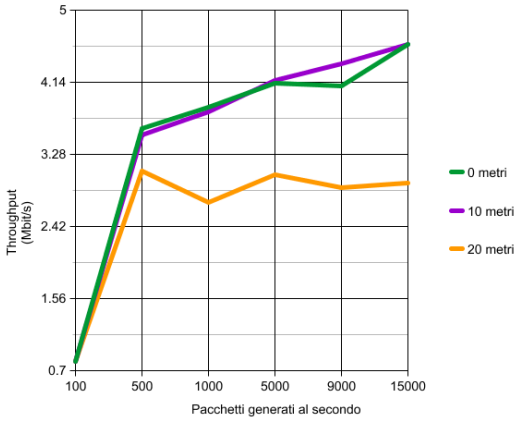

Figura 4.6: Throughput indoor

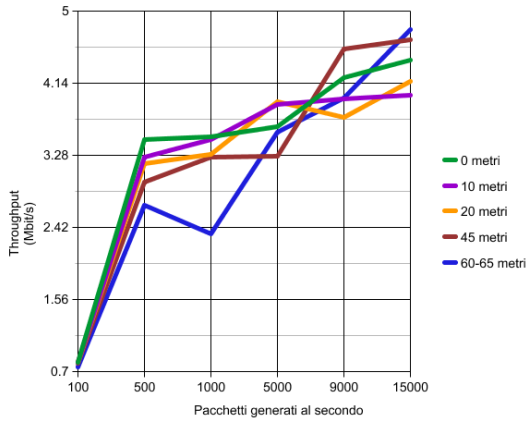

Figura 4.7*:* Throughput outdoor

Nell'ultima metrica nel caso outdoor, la soglia dei 60-65 metri è stata omessa perché, dai valori ricavati dai test, si evince che la metrica calcolata supera considerevolmente i valori segnati dai grafici, mentre nel caso indoor, la soglia dei 20 metri mostra un vertiginoso ritardo iniziale che poi diventa quasi costante verso la fine del test.

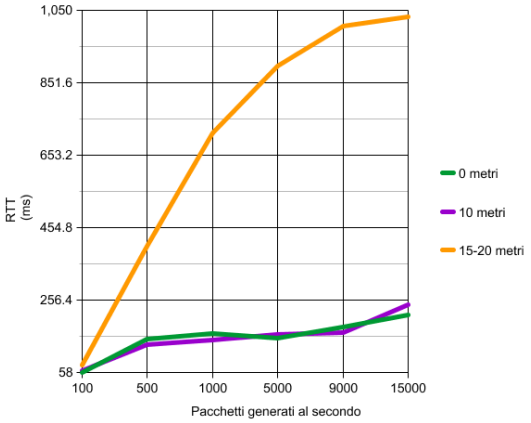

Figura 4.8: RTT outdoor.

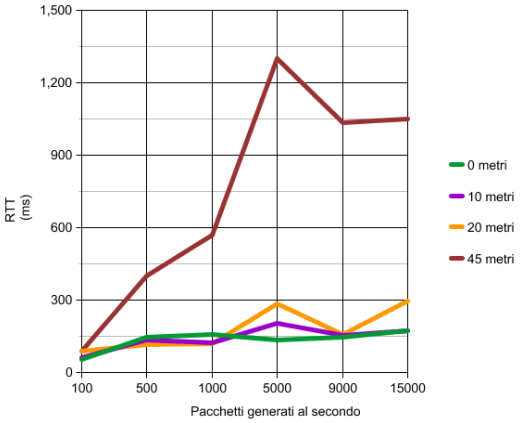

Figura 4.9: RTT indoor.

4.1.3 Calcolo della Discovery e Group Formation

Oltre ai calcoli riguardanti il trasferimento dei dati tra i due smartphone, ci si è interessati anche dei tempi richiesti dai dispositivi per eseguire le fasi di Discovery e Group Formation. Sono stati ripetuti per 22 volte gli esperimenti relativi alle due fasi e, dopo aver ricavato i dati tramite un applicativo installato su uno dei due smartphone, è stato possibile tracciare a livello statistico il completamento delle fasi, attraverso la funzione di ripartizione descritta nel precedente capitolo.

I grafici indicano sull'asse delle ascisse il tempo espresso in secondi, sull'asse delle ordinate la probabilità dell'evento.

Nel grafico della funzione di ripartizione della fase di Discovery, si può notare che nei primi 5 secondi, c'è una probabilità dell'80% di concluderla e, pertanto, pochi esperimenti hanno riportato una durata maggiore ai 10 secondi.

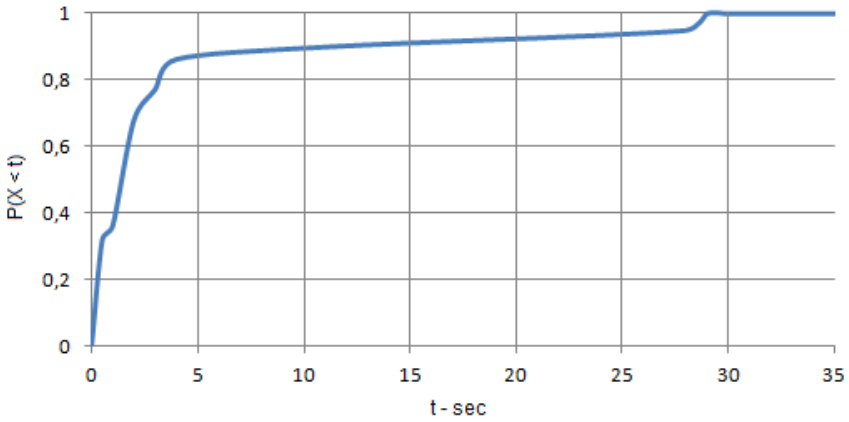

Figura 4.10: grafico della funzione di ripartizione della fase di Discovery.

Per quanto riguarda la fase della Group Formation, si può notare che vi è un 80% di probabilità di terminarla tra i 5 e i 10 secondi; a seconda della distanza, questo tempo potrebbe variare, fino ad un massimo di 25-30 secondi.

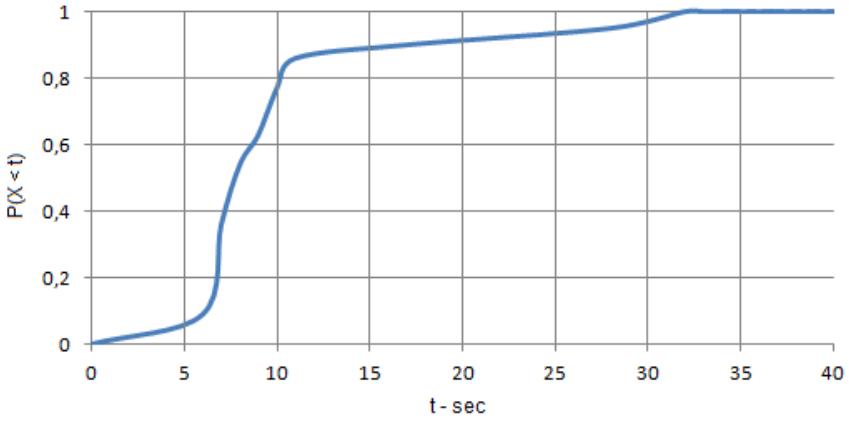

Figura 4.11: grafico della funzione di ripartizione della fase di Group Formation.

Per una valutazione generale del tempo richiesto per effettuare sia la fase di Discovery che la Group Formation, sono stati sommati entrambi i tempi delle fasi e inseriti in un grafico che disegna la funzione di ripartizione.
Si può notare che tra i 9 e i 12 secondi parecchi test terminano entrambe le fasi; solo dopo i 20 secondi c'è una probabilità dell'80 % di terminare entrambe le fasi, soltanto pochi test hanno riportato un tempo superiore ai 25 secondi, con un massimo di 36 secondi.

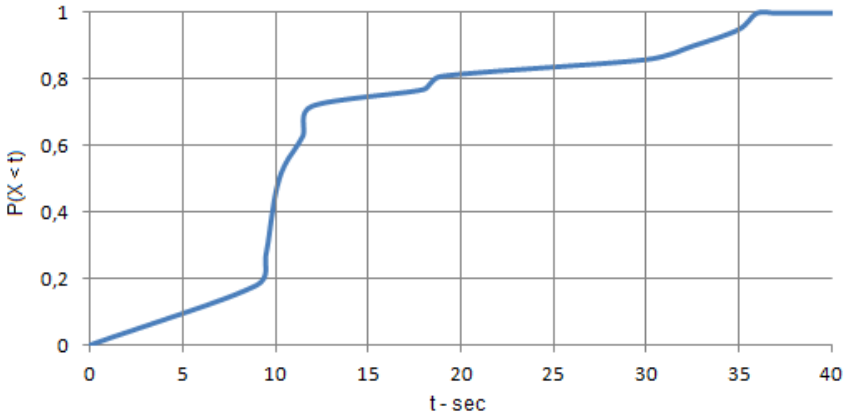

Figura 4.12: grafico della funzione di ripartizione del completamento di entrambe le fasi.

Capitolo 5

Sviluppo di un simulatore per la tecnologia Wi-Fi Direct

Dopo aver ottenuto i dati sulle prestazioni del protocollo Wi-Fi Direct con i corrispondenti grafici, si è giunti alla costruzione di un simulatore. Esso utilizza i dati ricavati nei test per simulare uno scenario di viabilità urbana (secondo il modello *Random waypoint*), formato da "entità" (o più propriamente delle persone) che, attraverso l'uso del Wi-Fi Direct supportato dai loro device, provvedono a ricercare e a trasmettere informazioni ai dispositivi limitrofi.
L'obiettivo principale per la costruzione di questo ulteriore applicativo, è riuscire a comprendere se il protocollo risulti essere funzionale e in grado di sostenere la costante mobilità urbana. Nella prima parte di questo capitolo è stato illustrato il suo funzionamento e i corrispettivi problemi affrontati durante la sua costruzione. In seguito vengono mostrati i grafici relativi ai risultati ottenuti dal Simulatore e le rispettive considerazioni al riguardo.

5.1 Panoramica del software

Il Simulatore è composto da 4 input:
-tempo di durata del test, espresso in secondi;
-numero di entità (che da ora in poi chiameremo nodi) che "navigano" all'interno dell'area;
-velocità dei nodi, espressa in m/s (metri al secondo);
-quantità di pacchetti che i nodi limitrofi devono inviarsi durante il test;
Nel Simulatore è stata utilizzata una scala che riprendesse un'area di 22.000 m^2 (110 x 200 metri) e, per comodità, si è pensato di fare riferimento ad un piano cartesiano, in cui per ogni nodo si decide un determinato percorso che deve seguire in modo indipendente dagli altri e, più precisamente, attraverso i seguenti

quattro *step*:

1)subito dopo l'avvio del Simulatore ogni nodo viene posizionato su un punto del piano cartesiano scelto in modo casuale e, prima che il nodo parta, bisogna decidere il punto finale da raggiungere (anch'esso deciso in modo casuale);

2)una volta che si è a conoscenza di quest'ulteriore punto, vengono calcolati i punti intermedi da i due punti iniziali;

3)dopo che si ricavano anche i punti intermedi (che sono tra loro equidistanti in relazione alla velocità dei nodi), il nodo inizia a percorrere la sua "rotta" di punti;

4)quando il nodo raggiunge il punto finale del suo percorso, bisogna decidere (sempre in modo casuale) un nuovo punto da raggiungere (ritornando allo step 2).

I nodi si muovono da un punto ad un altro in modo simultaneo fra loro, ogni loro "passo" corrisponde ad un secondo nel mondo reale.

5.1.1 Architettura

Il Simulatore è stato scritto interamente in java ed è composto da un unico Thread (Main Thread), che gestisce gli spostamenti dei nodi e le operazioni di interazione fra loro.

In base all'immissione dei 4 input previsti prima dell'avvio della simulazione, il thread popola l'area di nodi, impostando successivamente la loro velocità e la quantità di dati che devono spedirsi e le varie strutture dati utilizzate per organizzare i vari nodi. Inoltre, ad ogni nodo vengono salvate le coordinate di partenza e i rispettivi indici associati ad essi, per poi riconoscerli in un secondo momento.

Dopo questa fase di inizializzazione, si procede con un nuovo blocco di istruzioni dedicate sostanzialmente allo spostamento e alle interazioni dei nodi all'interno dell'area. Tale blocco di codice viene eseguito in modo ciclico per un numero di volte pari a quello indicato nel primo input. All'interno di questo ciclo, si annida un altro ciclo dedicato alle operazioni da svolgere ad ogni passo per tutti i nodi che popolano l'area:

-solo all'avvio della simulazione o se il nodo raggiunge il punto finale della sua rotta, attraverso una funzione che prende in input le coordinate attuali del nodo,

viene restituito il nuovo percorso, elencando tutti i punti che il nodo deve percorre ad ogni passo fino al raggiungimento del punto finale;

-per ogni nodo, viene indicata la sua nuova posizione ad ogni passo, tenendo conto del loro percorso salvato in opportune strutture dati;

-inoltre, sempre per ogni nodo, viene misurata la sua distanza con tutti gli altri nodi che, se minore di 60 metri (distanza massima consentita nel caso outdoor), si procede con l'inizializzazione della fase di Discovery (find e search) tra i nodi interessati e con la successiva fase di trasferimento (fase che segue subito dopo la fase di Group Formation tra i device).

Il programma dispone anche di un interfaccia grafica per tener conto degli spostamenti dei nodi ad ogni passo; per identificarli, sono stati associati a dei numeri. Se i nodi confinano ad una distanza minore ai 60 metri, parte la fase di Discovery (segmento nero); appena questa fase termina e viene formato il gruppo, i due dispositivi iniziano a trasferirsi dati (segmento blu).

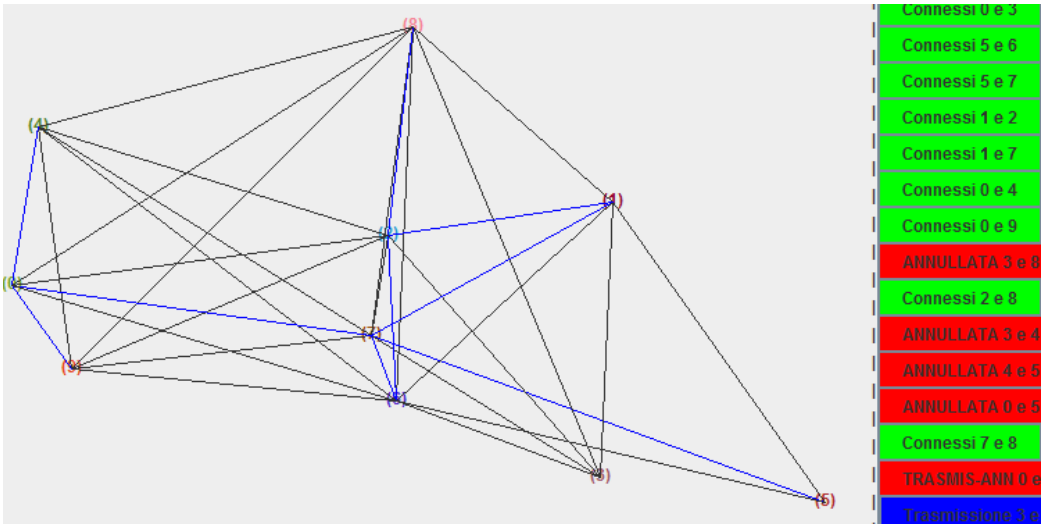

Figura 5.1: piccola rete P2P simulata tra i nodi. Solo i nodi 3 e 6 hanno terminato la fase di trasferimento.

5.1.2 Problemi affrontati

Ogni nodo è stato associato ad un intero positivo (≥ 1) e, ogni struttura dati del simulatore, conserva le informazioni di ogni singolo nodo nella posizione associata al numero intero del nodo. Per l' immagazzinamento di tutte le coordinate che ogni nodo deve percorrere per la sua rotta, è stato utilizzato

un’array di stringhe (chiamato nodes) grande quanto il numero dei nodi presenti nell’area, in cui ogni cella contiene l’intero percorso di un singolo nodo. Per tracciare gli avanzamenti dei nodi nei loro percorsi, si è utilizzato un altro array (anch’esso grande quanto il numero dei nodi, chiamato c_nodes) che, tramite un intero aggiornato ad ogni passo, permette di capire in quale parte del percorso si trova un nodo nell’altro array.

```
while(n<nodi){ //posiziona tutti i nodi nell'area in modo random
    cooX1=(int)(200*Math.random())+1; //coordinata x  200 metri
    cooY1=(int)(110*Math.random())+1; //coordinata y  110 metri
    nodes[n]=""+cooX1+";"+cooY1+";fine;"; //coordinate iniziali
    c_nodes[n]="0"; //azzera posizioni array c_nodes
    n=n+1;
}
```

Figura 5.2: inizializzazione dei nodi.

Ogni nodo che raggiunge il punto finale del suo percorso (è da notare che tutti i nodi inizialmente raggiungono il punto finale, dato che l’array nodes contiene al suo interno solamente le coordinate iniziali x e y e per cui il loro percorso è formato soltanto da un punto), esegue una funzione che permette di conoscere il prossimo punto finale da raggiungere e i relativi punti intermedi; in particolare, la funzione è composta dalle seguenti parti:

-nella fase iniziale si conosce il punto finale che il nodo deve raggiungere e, attraverso un’equazione che interessa le quattro coordinate dei due punti (punto iniziale e punto finale), è possibile individuare la retta che il nodo deve percorre.

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

Mediante l’equazione della retta passante per due punti, è possibile ricavare l’equazione in forma esplicita:

$$y = mx + q$$

dove *m* corrisponde al coefficiente angolare e *q* l’ordinata all’origine della retta.

Per questioni di semplicità, si è suddivisa l’ultima equazione in 4 classi differenti:

$$x = x_1 \qquad y = y_1 \qquad y = mx + q \qquad y = \frac{mx}{c} + \frac{q}{c}$$

Salvo le due equazioni iniziali, a seconda della pendenza della retta, si calcolano le incognite x o y tramite il metodo di sostituzione col sistema.

```
while((it) > min){ //trova coordinate centrali !
    if(tipo_f==1){ // tipo y = c (costante)
    //trovare la x
    // solo nelle equazione di tipo 1 e 2 non è necessario
        salva=salva+it+";"+cooY1+";";
    }else if(tipo_f==2){ //tipo x = c
    //trovare le y
        salva=salva+cooX1+";"+it+";";
    }else if((tipo_f==3) && (se.equals("y"))){ //tipo y="+c
        float wow= (den_y)*(it)+(somma);
        salva=salva+it+";"+wow+";";
    }else if((tipo_f==3) && (se.equals("x"))){
        float wow1= -(it);
        float wow2= (wow1) + (somma);
        float den = -(den_y);
        if(den < 0){
```

Figura 5.3: porzione di codice dell'implementazione del metodo di sostituzione per i 4 tipi di equazione.

Nella fig. 5.3 è possibile notare la variabile *it:*
essa rappresenta il numero relativo sull'asse delle ascisse o su quello delle ordinate (in relazione alle incognite x o y da trovare) da dove partire per trovare i punti intermedi della retta (per inciso, `it` corrisponde alla prima coordinata del punto da trovare tramite il metodo di sostituzione). La velocità dei nodi è in stretta relazione con il decremento della variabile `it`: maggiore è il suo decremento ad ogni iterazione e maggiore è lo spostamento di ogni singolo nodo nel suo percorso. Dopo il metodo di sostituzione si possiedono tutte le coordinate del percorso e, dopo averle inserite volta per volta in una stringa ad ogni iterazione, vengono poi restituite dalla funzione.

5.2 Risultati ottenuti

Sono stati compiuti diversi test tramite il simulatore, che hanno permesso di osservare in modo abbastanza dettagliato le molteplici possibilità del protocollo in uno scenario urbano.
Di seguito è stato affrontato lo studio della probabilità che hanno i nodi di effettuare la fase di Discovery e la fase di trasferimento al variare della velocità

dei nodi:

per questo test è stata impostata una durata massima di 100 secondi e sono stati inseriti nell'aria 7 nodi e la quantità di pacchetti da inviare è stata settata a 9000 (circa 10 secondi per terminare la fase di trasferimento).

Il test è stato ripetuto variando solamente la velocità dei nodi e, come si può notare, solamente alla velocità di 1 m/s è possibile avere un'alta probabilità di terminare entrambe le fasi e, pertanto, dai 2 m/s in poi è possibile scorgere un decremento sostanziale fino ad arrivare alla soglia dei 20 m/s, in cui nessun dispositivo riesce a terminare entrambe le fasi.

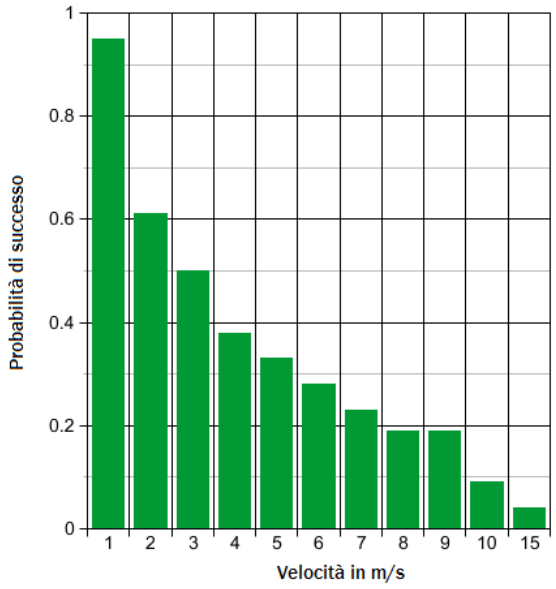

Figura 5.4: probabilità del completamento delle fasi al variare della velocità dei nodi nell'area.

Un'altra importante stima è stata quella di calcolare la probabilità di entrambe le fasi al variare della quantità di pacchetti da inviare tra i nodi:

anche per questo test sono stati adoperati 7 nodi e la loro velocità è stata settata a 3 m/s. Per realizzare il test, si è calcolato il tempo di trasferimento dei pacchetti tramite i dati ricavati per il calcolo del Throughput, mettendo in rapporto, per ogni classe di pacchetti, la dimensione dei dati (tenendo conto che ogni pacchetto ha la grandezza di 1000 byte) e la sua velocità trasmissiva. A partire dalla 3° classe di pacchetti (1000) in poi, si può notare una leggera diminuzione di probabilità, che non diventa nulla neppure con la massima quantità di pacchetti da inviare (15000,

circa 30 secondi per terminare la fase di trasferimento).

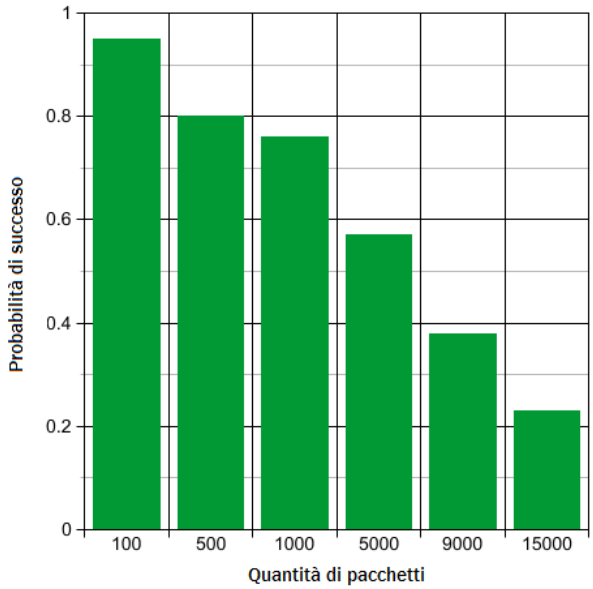

Figura 5.5: probabilità del completamento delle fasi al variare della quantità di pacchetti da inviare.

Conclusioni

In questa tesi inizialmente è stato descritto lo standard Wi-Fi Direct ed è stato presentato in seguito il sistema operativo per smartphone, Android.
Successivamente è stato analizzato lo schema implementativo in Android sia di una generica applicazione sia di un'applicazione che utilizza lo standard Wi-Fi Direct.
Dopo questa parte introduttiva, si è passati alla fase sperimentale del protocollo, introducendo le metriche di valutazione; esse hanno permesso di stimare, mediante un applicativo supportato da Android, l'efficienza dello standard durante il trasferimento di informazioni fra due smartphone al variare della loro distanza.
Inoltre sono stati esaminati i tempi necessari ai due smartphone per attuare le fasi fondamentali per la loro interazione, quali la Discovery e la Group Formation.
Dopo aver ottenuto i dati nella fase sperimentale, questi sono stati applicati in uno scenario composto da entità mobili che fanno uso dello standard Wi-Fi Direct (secondo il modello di mobilità Random waypoint): per tale scopo è stato costruito un simulatore che riprendesse uno scenario urbano composto da individui disposti in un'area di 22.000 m^2.
Tramite il simulatore è stato possibile fare le seguenti considerazioni finali: secondo i dati raccolti, si evince che lo standard Wi-Fi Direct consente di sostenere solamente gli scenari caratterizzati da una bassa mobilità relativa dei nodi: è necessario cioè che i nodi rimangano pressoché fermi gli uni rispetto agli altri. Considerando queste limitazioni, si deduce che questa tecnologia sia applicabile in contesti assolutamente statici (quali ad es. uffici o appartamenti) o relativamente statici (trasporti pubblici).

Sviluppi futuri

Durante lo studio è stato possibile determinare le caratteristiche dello standard solo in maniera sommaria; esistono ulteriori indagini che possono essere eseguite per acquisire informazioni più precise sulle possibilità del protocollo, in particolare:

-confrontare le stime del Wi-Fi Direct con diversi protocolli, quali ad esempio il Bluetooth;

-quantificare il risparmio energetico del P2P GO in differenti scenari mediante l'utilizzo dei protocolli Notice of Absence e Opportunistic Power Save;

-effettuare analisi di sicurezza sulle vulnerabilità del protocollo (*Wireless DoS* e *key cracking*) [9];

-osservare le politiche di interazione tra più dispositivi che richiedono simultaneamente la formazione di un gruppo;

-considerare un ulteriore modello di mobilità, più adatto ad uno scenario urbano: *Manhattan mobility*.

Appendice A

Qui di seguito sono elencati i link ai codici sorgenti degli applicativi utilizzati nella tesi:
Calcolo delle tre metriche, calcolo dei tempi di Discovery e Group Formation: http://urly.it/2w2q

Simulatore di uno scenario urbano:
http://pastebin.com/h5D2Ccd2

Bibliografia

[1] Device-to-device communications with Wi-Fi Direct: overview and experimentation (Daniel Camps-Mur, Andres Garcia-Saavedra and Pablo Serrano)

[2] Wi-Fi Direct™ (Hughes Systique Corporation, 15245 Shady Grove Road, Suite 330, Rockville, MD 20850.Copyright © 2006 Hughes Systique Coporation)

[3] http://android.devapp.it/larchitettura-di-android

[4] http://emanueldinardo.com/wp/android-dalvik-virtual-machine/

[5] http://www.html.it/pag/48652/il-ciclo-di-vita-di-unactivity/

[6] http://developer.android.com/guide/topics/connectivity/Wi-Fi p2p.html

[7] Experimenting opportunistic networks with Wi-Fi Direct (Marco Conti, Franca Delmastro, Giovanni Minutiello, Roberta Paris)

[8] Device-to-Device Communications with Wi-Fi Direct: Overview and experimentation (Daniel Camps-Mur, Nec Network Laboratories, Andreas Garcia-Saavedra and Pablo Serrano, Universidad Carlos III De Madrid)

[9] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6418681&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6418681