Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Porting monitor di rete per gestione interfacce su device Android

Tesi di Laurea in Reti

Relatore:
Dott.
Vittorio Ghini

Presentata da:
Antonello Antonacci

Sessione II
Anno Accademico 2012/2013

"Desidero innanzitutto ringraziare il Professor Vittorio Ghini per i preziosi insegnamenti ricevuti sia durante le lezioni da lui tenute che nel periodo di sviluppo del tirocinio e della tesi, e per la sua grande disponibilità mostrata nei miei confornti. Inoltre vorrei dire grazie anche ai miei genitori e alla mia famiglia in generale per avermi sempre sostenuto e aver creduto in me. Infine un abbraccio và anche ai miei amici che sono sempre stati al mio fianco aiutandomi sotto ogni punto di vista, non solo quello didattico. Se ho raggiunto questo obbiettivo è anche merito di tutti voi.

Grazie" ...

Indice

Capitolo 1

Introduzione

Ormai, grazie alla rapida evoluzione nel giro di pochi anni ed ad un sempre maggiore numero di servizi offerti, i dispositivi mobili, come *smartphone* e *tablet*, hanno invaso il mercato tanto che nei maggiori paesi il numero di questi dispositivi e addirittura superiore al numero degli abtanti stessi. È nato così da pochi anni un mercato che ha portato alla creazione di nuovi sistemi software e al riadattamento di applicazioni già esistenti ed in uso in altri ambienti per permettere la gestione di diverse operazioni tramite questi device, dalle più semplici, come ad esempio leggere un quotidiano, alle più complesse come il controllo totale di un' abitazione.
Tutti questi dispositivi sono in possesso di più **interfacce di rete**, alcune simili a quelle dei dispositivi Desktop come le schede **wireless**, utilizzate per interfacciarsi con gli **Access Point**, gestite esattamente come avviene nei sistemi desktop, altre invece ideate per questo tipo di tecnologia come il **3G** (Terza Generazione, si sta però già lavorando per l' introduzione del **4G** che è ovviamente un ulteriore aggiornamento rispetto alla terza generazione, in grado di offrire una connessione ancora più veloce e più stabile). Può essere quindi utile poter avere a disposizione un sistema che sia in grado di configurare tutte queste interfacce in maniera dinamica ed essere quindi in grado di riconfigurare la stessa interfaccia in qualsiasi momento, passando quindi da una connessione ad un' altra considerata in quel momento migliore in base

alla potenza del segnale radio ricevuto. Questo sistema è già implementato nei sistemi operativi desktop basati su Linux, anche se all' interno di un contesto più complesso, cioè l' **Always Best Packet Switching** (ABPS), sistema di cui parleremo più avanti, spiegandone i principi fondamentali ed il suo funzionamento.

Agli sviluppatori **Android** fornisce diversi pacchetti (**SDK,NDK**) che tra le altre cose mettono a disposizione un emulatore in grado di simulare sia gli *smartphone* che i *tablet*, inoltre essendo un progetto **Open Source** rende disponibile i sorgenti di tutte le versioni del sistema operativo rilasciate negli anni dalla prima **Apple Pie** (1.0) all' ultima **Jelly Bean** (4.3), compresa una versione chiamata **Goldfish** rilasciata appositamente per l' emulatore, difatti è una versione che non è possibile installare su un dispositivo fisico. Tutto questo ci offre la possibilità di apportare delle modifiche anche a livello **Kernel** del sistema operativo.

Lo scopo di questa tesi è quello di effettuare il **porting** del **Monitor** di rete già disponibile per Linux su Android, senza compromettere il funzionamento del sistema stesso.

Il lavoro è stato suddiviso in due parti la prima ha previsto il **porting** del **Monitor** su un emulatore Android, successivamente il lavoro si è spostato su un device reale, nel mio caso un Samsung Galaxy Next con versione **Android Gingerbrad** 2.3.6, ovviamente apportando le dovute modifiche è possibile lavorare con qualsiasi device che monti qualsiasi versione di tale sistema operativo.

Capitolo 2

Scenario

Il monitor per la configurazione delle interfacce di rete, che in ambito Linux lavora all' interno di un contesto più ampio che è quello dell' **ABPS**[1], ha il compito di gestire l' associazione tra una scheda di rete ed un **Access Point**[2].
Il sistema dell' **ABPS** fornisce un meccanismo che permette alle applicazioni di utilizzare simultaneamente tutte le interfacce disponibili, introducendo nuove funzionalità e la possibilità di creare nuove politiche sia per il bilanciamento del carico che per il controllo della comunicazione, mentre il ruolo del monitor svolto all' interno di questo sistema è quello di configurare dinamicamente le interfacce di rete, in modo tale da interfacciarsi sempre con la migliore delle reti disponibili, filtrando ovviamente sulle reti di cui si conoscono le credenziali per accedere.

2.1 Scheda di rete Wireless

Una Wireless Network Interface Controller (WNIC) è una scheda di rete capace di connettersi ad una rete su mezzo radio.

[1] Always Best Packet Switching

[2] Access Point (AP), dispositivo elettronico che permette la connessione di uno o più dispositivi sfruttando i segnali radio

Lavora sui livelli **OSI**[3] 1 e 2 ed è costruita secondo le regole stabilite dallo standard **IEEE 802.11**[4].
I tipici parametri di una scheda wireless sono:

- La banda, in Mb/s: da 2 Mbit/s a 54 Mbit/s
- La potenza di trasmissione, in dBm
- Gli standard supportati, ad esempio 802.11 b/g/n, ecc...

Inoltre una scheda wireless può lavorare in due diversi modi:

1. Infrastructure
2. Ad-hoc

Nel primo caso, la scheda ha bisogno di un AP con cui interfacciarsi per la trasmissione dei dati. Per connettersi alle SSID[5] protette, tutte le stazioni devono essere a conoscenza della chiave di identificazione (password).
Mentre nella modalità Ad-hoc gli AP non sono richiesti, dato che è la scheda wireless stessa a comunicare con le altre station, tutti i nodi devono comunque essere sullo stesso canale.
Per il lavoro svolto in questa tesi bisogna usare schede di rete wireless che lavorino nella prima modalità.

2.2 Wireless Access Point

Un **Wireless Access Point**, abbreviato in AP, è un dispositivo facente parte di una rete IEEE 802.11, che si presenta come punto di interconnessione tra una station ed una rete. In generale qualsiasi dispositivo wireless,

[3]Open System Interconnection reference model, è uno standard che stabilisce per l' architettura logica di rete una struttura a strati

[4]Institute of Electrical and Electronic Engineers, associazione di scienziati il cui scopo è quello di promuovere le scienze tecnologiche

[5]Service Set IDentifier, nome con cui una rete wifi si identifica agli utenti

se opportunamente configurato, può fungere da AP, ma ovviamente esistono dispositivi appositamente dedicati.
Un AP può funzionare in diverse modalità, anche se alcuni di quelli in commercio non supportanto tutte le possibili funzionalità, mentre altri ne svolgono più di una contemporaneamente, le diverse modalità di funzionamento sono le seguenti:

1. **Root Mode**: è la modalità standard di funzionamento di un AP, in cui esso è collegato alla rete e funziona da punto di accesso per tutti i nodi che vogliono connettersi a quella determinata rete;

2. **Bridge Mode**: crea un link wireless tra due (point-to-point) o più (point-to-multipoint) AP, ciascuno dei quali è collegato ad un segmento di rete cablata, in questo modo i diversi segmenti vengono interconnessi;

3. **Repeater Mode**: tipo di configurazione usata per aumentare il raggio di copertura di una rete wireless quando vi sono difficolta di raggiungimento tramite la rete cablata in una determinata zona, questo però comporta una drastica riduzione del throughput[6];

4. **Client Mode**: un AP si comporta come un client verso un altro AP in modalità Root.

2.3 Always Best Packet Switching

La progettazione dell' ABPS è basata su un approccio cross-layer[7], grazie al quale è possibile acquisire dati provenienti dai vari livelli dello stack ISO/OSI, dai quali è possibile ottenere informazioni sulla qualità del canale di comunicazione. Ed è proprio attraverso questi dati che l' ABPS esegue il monitoraggio dei link di comunicazione wireless usati per la trasmissione dei pacchetti di dati. Gli elementi che caratterizzano questo tipo di approccio sono:

[6]indica la capacità di trasmissione effettivamente utilizzata dalla rete.

[7]metodo che consiste nello scambio di feedback tra i livelli adiacenti dello stack ISO/OSI

- un meccanismo di monitoraggio rivela se ogni detagram UDP[8] è stato perso durante la trasmissione tra l' interfaccia del dispositivo e l' AP con il quale è associato, inoltre comunica queste informazioni al produttore del datagram perso;
- l' applicazione può rispedire ogni datagram UDP perso utilizzando interfacce diverse.

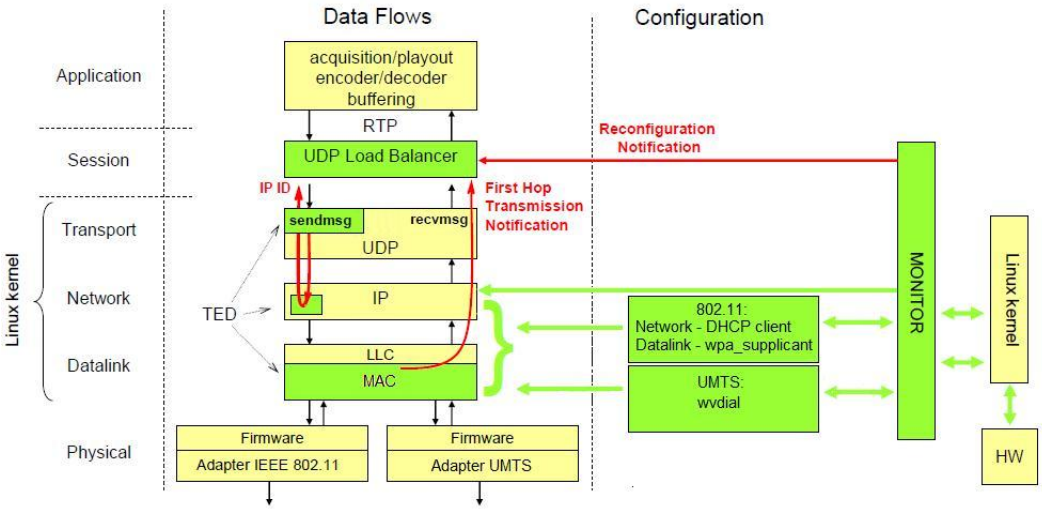

Figura 2.1: Schema e funzionamento dell' ABPS

Il modello ABPS è composto principalmente da tre elementi fondamentali:

- **TED**[9]: si occupa di rilevare se un frame inviato ha raggiunto oppure no l' AP;
- **Monitor**: parte del sistema del lavoro di questa tesi, ha il compito di gestire e configurare le varie interfacce di rete dinamicamente;
- **ULB**[10]: gestisce la trasmissione dei pacchetti sulle interfacce di rete disponibili.

[8]User Datagram Protocol, è uno dei principali protocolli di rete e solitamente viene utilizzato in combinazione con il protocollo di livello di rete IP.

[9]Transmission Error Detector

[10]UDP Load Balancer

2.3.1 TED: Transmission Error Detector

Il TED è un componente che lavora tra i livelli trasporto, network e datalink, ha il compito di individuare i frame per i quali il dispositivo non riceve l' acknowledgement, cioè individua quali pacchetti sono anadati persi nella rete durante la comunicazione.
Il TED implementa due strumenti:

- il primo è un sistema per l' estrazione degli identificativi dei frame inviati dalle interfacce di rete;
- il secondo è un sistema di notifiche implementato a livello MAC[11] dello stack wireless, che permette di ottenere informazioni sui pacchetti inviati, informazioni che poi vengono notificate all' ULB che le associa al pacchetto tramite l' id precedentemente ottenuto.

2.3.2 Monitor

Il **Monitor** si occupa di configurare dinamicamente le interfacce wireless e le regole di routing, notificando all' ULB quali interfacce sono state configurate e quindi pronte per l' uso.
A livello datalink vengono svolte le operazioni che permettono ad ogni scheda wireless di individuare la presenza di AP, le relative configurazioni per la comunicazione e le operazioni per la gestione delle interfacce quando un AP non è più raggiungibile. Per eseguire questi compiti il monitor utilizza il **Wpa_Supplicant**[12], il quale racchiude tutte le informazioni relative all' autenticazione e alla sicurezza delle reti all' interno del file di configurazione *wpa_supplicant.conf*. Il wpa_supplicant guida l' adattatore wireless nell' effettuare lo scan dei canali radio per individuare gli AP dei quali conosce le informazioni di autenticazione, se sono disponibili più AP allora seleziona

[11] codice univoco di 48 bit che ogni produttore associa ad una scheda di rete, tuttavia è modificabile a livello software.

[12] programma Open Source utilizzato per la gestione delle connessioni WPA e WPA2.

quello che ha il miglior segnale radio, dopodichè esegue la procedura di associazione.
Dopo che il wpa_supplicant ha configurato il livello datalink di un interfaccia wireless, associandola ad un dato AP, il kernel informa il monitor, il quale avvia il client DHCP[13], quest' ultimo richiede un indirizzo IP[14], una maschera di rete ed un gateway. Quando il client termina correttamente la configurazione del livello network dell' interfaccia, il Monitor crea una nuova regola IP ed una nuova tabella di ruoting (tramite i comandi **ip rule** ed **ip route**) al fine di avere un routing dinamico tramite quella interfaccia. Tramite queste regole, le tabelle di routing impongono ad un datagram IP avente lo stesso indirizzo IP sorgente dell' interfaccia di essere inviato tramite quest' ultima, indipendentemente dall' IP di destinazione. Una volta configurata correttamente, la scheda può essere usata dal componente ULB.
Infine l' ultimo compito svolto dal Monitor è quello di disabilitare l' interfaccia ed eliminare le regole di routing relative ad essa nel momento in cui viene avvisato dal kernel di aver perso l' associazione con quel determinato AP.

2.3.3 ULB: UDP Load Balancer

Il componente ULB è responsabile della trasmissione dei pacchetti di dati attraverso tutte le interfacce disponibili, oltre che del recupero dei possibili pacchetti persi. Riceve dal componente TED notifiche riguardanti errori nella comunicazione e informazioni a riguardo dello stato delle reti disponibili, quindi rispedisce i datagram andati persi durante la comunicazione, ed inoltre avendo informazioni sulle reti ha anche la possibilità di scegliere su quale interfaccia ritrasmettere quel pacchetto. L' ULB riceve informazioni su tutte le interfacce da parte del Monitor e crea un socket[15] UDP per ognuno di essi,

[13]Dynamic Host Configuration Protocol, protocollo di rete che permette ai dispositivi di ricevere una conifgurazione dell' indirizzo IP dinamica ad ogni richiesta di connessione verso la rete.

[14]Internet Protocol adress, etichetta numerica univoca che identifica un dispositivo collegato ad una rete.

[15]canale di comunicazione creato all' interno della rete.

usando la system call **bind**[16] per assegnare l' indirizzo IP dell' adattatore in questione al socket associato.
Grazie alle regole di routing create dal Monitor, tutti i datagram appartenenti ad un dato socket saranno instradati tramite la relativa interfaccia al quale il socket è stato associato.

[16]system call che associa un nome ad un socket

Capitolo 3

Obiettivo

Il lavoro svolto in questa tesi è quello di effettuare il **porting** del componente Monitor del sistema di Always Best Packet Switching su dispositivi mobili dotati di sistema operativo Android.
Il porting consiste nel riadattare un sistema o una applicazione, in questo caso il Monitor, apportando le giuste modifiche sia a livello dell' applicazione stessa che a livello kernel del nuovo ambiente in cui si và a lavorare, eseguendo inoltre le giuste configurazioni di tutti i parametri.
Come già detto in precedenza il Monitor è già implementeto in ambiente Linux, seppur in un contesto più complesso, e dato che Android è un sistema operativo il cui kernel è basato su Unix e quindi può essere visto come una versione semplificata ed ottimizzata per determinati dispositivi dei sistemi Linux, avremo allora la possibilità di effettuare il porting del Monitor senza dover partire da zero, proprio perchè andiamo a lovare con un sistema che non è del tutto estraneo da quello di partenza.

3.1 Android

Per poter comprendere meglio lo scopo e il lavoro svolto in questa tesi è bene fare due accenni alla storia e alla struttura del sistema operativo **Android**.

3.1.1 Evoluzione

Android è un sistema operativo per i dispositivi mobili organizzato in un' architettura software che include un sistema operativo di base, i middleware[1] per le comunicazioni e le applicazioni di base.
La Android Inc venne fondata nel 2003 e acquisita da Google nel 2005 che nel 2008 pubblicò la prima versione stabile del sistema operativo.
Andorid è un sistema Open Source e questo ci permette di modificarlo a qualsiasi livello e di redistribuirlo, ovviamente mantanendo la stessa licenza. La sua caratteristica è quella di poter essere integrato e personalizzato in base alle proprie esigenze grazie all' utilizzo delle app, scaricabili dallo Store ufficiale di Google, o da altri store.
Di Android sono disponibili quattro versioni principali, alcune divise in più varianti, più una quinta annunciata ma non ancora pubblicata,

1. **1.0 Apple Pie**: 23-settembre-2008;
2. **1.1 Petit Four**: 9-febbraio-2009, risolti vari bug rispetto alla 1.0;
3. **1.5 Cupcake**: 30-aprile-2009, introdotti i widget;
4. **1.6 Donut**: 15-settembre-2009;
5. **2.0.x Eclair**: 26-ottobre-2009;
6. **2.2.x Froyo**: 20-maggio-2010;
7. **2.3.x Gingerbread**: 6-dicembre-2010, attualmente ancora la più diffusa;
8. **3.x Honeycomb**: 22-febbraio-2011, versione ottimizzata per tablet;
9. **4.0.x Ice Cream Sandwich**: 21-ottobre-2011;
10. **4.1.x Jelly Bean**: 9-luglio-2012;

[1] Insieme di programmi che fungono da intermediari tra diverse applicazioni e componenti software

11. **4.4 Kit Kat**: annunciato inizialmente come quinta versione e con il nome di Key Lime Pie.

Una piccola curiosità, i nomi, ad esclusione della versione 1.1, seguono un ordine alfabetico e si riferiscono a dei dolci.

3.1.2 Architettura

Come detto Android è basato su kernel Linux, con middleware, librerie e API[2] scritte in C o C++ e software in esecuzione su framework di applicazioni che includono librerie Java.
Per la compilazione del codice, Android utilizza la Dalvik Virtual Machine[3] per l' esecuzione del Delvik dex-code.
Il kernel Linux Android ci consente di effettuare qualsiasi tipo di modifica, infatti solitamente questi sistemi non includono tutte le librerie standard GNU, questo proprio per renderlo più leggero in relazione alla potenza di calcolo di un dispositivo mobile comunque inferiore ai moderni calcolatori, è quindi possibile includere qualsiasi delle librerie di GNU in Android, ovviamente apportando le opportune modifiche, ma sempre per una questione di performance non conviene comunque esagerare con l' aggiunta di software.
L' Application Framework (nella figura 3.1) indica come viene suddiviso il lavoro all' interno del sistema:

- **Activity Manager**: mette in collegamento tutte le activity del dispositivo;
- **Package Manager**: classe che si occupa di recuperare diversi tipi di informazioni a riguardo delle applicazioni attualmente installate sul device;
- **Window Manager**: interfaccia con cui le applicazioni comunicano per la gestione delle finestre;

[2]Application Programming Interface, solitamente indica un set di strumenti che svolgono un lavoro preciso all' interno di un programma.
[3]Macchina virtuale utilizzata per eseguire il codice Java.

- **Telephony Manager**: classe che si occupa delle funzioni di telefonia del dispositivo;
- **Content Providers**: gestisce l' accesso alle strutture dati e fornisce i meccanismi per definire regole di protezione sui dati;
- **Resource Manager**: gestisce le risorse del dispositivo, ad esempio la RAM;
- **View System**: rappresenta la struttura dello User Interface;
- **Location Manager**: classe che gestice l' accesso ai servizi di sistema per la localizzazione;
- **Notification Manager**: gestore delle notifiche;
- **XMPP Service**: insieme di protocolli usati per la messaggistica istantanea.

Lo strato più alto rappresenta quello delle applicazioni, ed è l' insieme dei programmi che ogni utente scarica ed installa in base alle proprie esigenze, l' utilizzatore medio interagisce e personalizza il proprio dispositivo soltanto attraverso quest' ultimo livello.

3.2 Obiettivo tesi

Dopo aver dato una panoramica dell' architettura Android è possibile specificare più nel dettaglio qual è l' obiettivo di questa tesi.
Essenzialmente il lavoro si svolge ai livelli più bassi del sistema operativo, cioè sul kernel e sulle librerie.
Per quanto riguarda il kernel verrano apportate delle piccole modifiche in modo che quest' ultimo sia in grado di comunicare con il Monitor avvisandolo nel momento in cui non vengono ricevuti pacchetti durante la comunicazione.
Mentre per quanto riguarda le librerie avremo bisogno di modificare e riadattare delle librerie da Linux in modo da poterle importare,così da avere a

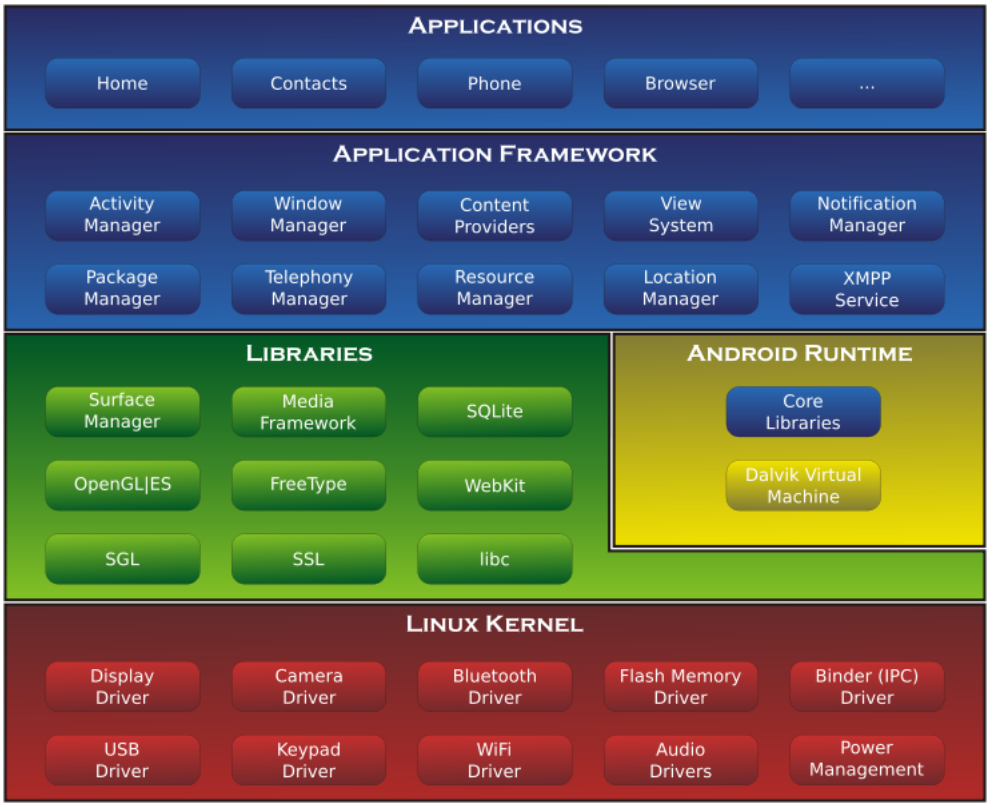

Figura 3.1: Architettura del sistema operativo Android

disposizione tool e funzioni che sono necessarie per l 'integrazione del Monitor, che Android non supporta nativamente.
Grazie a queste modifiche sarà possibile installare ed eseguire una versione anch' essa modificata ed adattata ai sistemi Android, del Monitor, cosicchè avremo a disposizione sul dispositivo un' applicazione in grado di configurare le interfacce di rete dinamicamente e di comunicare istantaneamente al kernel quali sono i parametri di configurazione di queste interfacce, come ad esempio indirizzo IP, porta, indirizzo del Gateway e tipo di interfaccia utilizzata e di deconfigurarle ed eliminare tutte le regole di connessione per esse create nel momento in cui quella determinata interfaccia non sia più disponibile.

Capitolo 4

Strumenti

Per poter svolgere tutto il lavoro ho utilizzato vari strumenti, che adesso vedremo nel dettaglio.
Tra quelli forniti da Android ho usufruito del **Software-Developement-Kit** (SDK) e del **Native-Developement-Kit** (NDK), tra le tante funzionalità l' SDK ci permette di creare un emulatore e di scrivere applicazioni ad alto livello implementate in linguaggio **Java**, mentre l' NDK offre un compilatore per il kernel Android e per compilare le applicazioni di basso livello, in questo modo è possibile implementare software a basso livello, come il Monitor, scritte in **C**. Altri strumenti utili ed indispensabili sono **BusyBox** ed il **Wpa_supplicant**, quest' ultimo installato di default sui device Android.

4.1 Android Software-Developement-Kit

L' Android SDK è un kit di sviluppo software attraverso il quale è possibile creare emulatori sia di smartphone che di tablet ovviamente Android, implementare nuove applicazioni oppure modificarne di già esistenti, e interagire con i dispositivi reali.
Questo pacchetto mette a disposizione diversi tool, tra i più importanti troviamo:

- emulatore;

- panoramica della gerarchia;
- ddms (Dalvik Debug Monitor Server);
- adb (Android Debug Bridge).

4.1.1 Emulatore Android

Il tool **emulator** ci permette di creare nuovi emulatori o di lanciarne di già esistenti.
Quando si crea un nuovo Android Virtual Device (AVD) usando il tool emulator da shell vanno specificati diversi parametri che servono per la configurazione del nuovo emulatore, bisogna indicare un nome univoco per questo emulatore e che versione del sistema operativo dovrà montare, dopodichè andranno indicati quanta memoria RAM vogliamo dedicare al device, la grandezza e il path nel quale memorizzare la SdCard, possiamo indicare persino che tipo di CPU vogliamo simulare per la nostra AVD e la dimensione e risoluzione dello schermo. Gli unici parametri indispensabili sono il nome e la versione del sistema operativo, gli altri se non vengono specificati allora il tool provvederà a fornire dei valori standard.

4.1.2 Panoramica della gerarchia

Lo **HierarchyViewer** è un tool grafico molto utile per analizzare la struttura di un' applicazione Android, grazie ad esso è possibile avere una panoramica delle **view** di cui è composta una determinata app, classificando ognuna di esse in base a tre colori, verde, giallo e rosso, che servono per indicare se il tempo di caricamento è rispettivamente, sotto la media, sopra la media oppure il peggiore.
Il tutto è riferito al contesto in cui si lavora, cioè ci sarà sempre una foglia classificata come rossa, ciò però non significa forzatamente che il processo di sviluppo di quella foglia sia stato fatto male, ma semplicemente che è oltre la media.

4.1.3 Dalvik Debug Monitor Server

Il Dalvik Debug Monitor Server (ddms) è un tool che fornisce una serie di servizi molto utili per testare le nuove applicazioni, ad esempio un ruolo molto importante che svolge il ddms è il port-forwarding, cioè permette di collegare un debugger ad un dispositivo oppure un emulatore usando le porte dalla 8000 alla 8019 della macchina su cui si lavora, in questo modo è possibile distinguere ed operare separatamente su due emulatori contemporaneamente.
Tra le altre funzionalità permette di catturare la schermata del dispositivo, di esplorare lo stato dei processi in esecuzione, provocare un Garbage Collection[1], gestire il logcat[2], simulare la ricezione di una chiamata o di un sms e specificare la posizione del gps passando i parametri di latitudine e longitudine.
Esistono tanti altri servizi offerti dal **ddms**, questi sono solo quelli più imprtanti ed utilizzati.
I problemi finora conosciuti del ddms sono essenzialmente due:
Il primo si ha quando si connette e disconnette un debugger, allora il ddms abbandona e riconnette il client, in modo che la AVD realizzi che il client non c'è più, questo può causare una perdita di dati durante la comunicazione con l' emulatore.
Mentre il secondo è l' impossibilità di collegare un ddms ad un' applicazione già in esecuzione, infatti se si prova a farlo l' AVD si blocca e irrimediabilmente crasha.

4.1.4 Android Debug Bridge

L' Android Debug Bridge (adb) è un tool da riga di comando, che ci permette di comunicare con le AVD oppure con i dispositivi collegati, è un programma client-server composto da tre elemnti:

[1]Metodo di gestione della memoria, solitamente è automatico e a discrezione del sistema, esistono però metodi per forzarne l' esecuzione

[2]file di log degli errori

- un client che esegue sulla macchina di sviluppo e che viene invocato digitando **adb** e i relativi comandi per le varie opzioni disponibili, da terminale. Ad esempio è possibile installare un' applicazione oppure aprire un terminale, con o senza i permessi di root, relativo al dispositivo;

- un server, che viene eseguito in background anch' esso sulla macchina di sviluppo e che ha il compito di gestire la comunicazione tra il client e il demone presente sul device o sull' emulatore di destinazione;

- infine vi è il demone che esegue in background sul dispositivo (o sull' emulatore) e serve per ricevere e gestire i comandi ricevuti da parte del client.

Il funzionamento è abbastanza semplice, quando si lancia l' adb bisogna innanzitutto specificare l' ID del device (o emulatore) sul quale andiamo a lavorare, se è disponibile uno solo allora non vi è il bisogno di indicarlo, dopodiche indichiamo il comando, ad esempio *shell, install, push, pull, logcat, ecc..* ed infine i parametri giusti in base al comando utilizzato.

4.2 Android Native-Developement-Kit

L' Android NDK è un pacchetto software che permette di implementare applicazioni in linguaggio nativo, cioe in **C** e **C++**, in questo modo possiamo interagire a basso livello con il sistema ed utilizzare tutte le sue librerie. È possibile inoltre non solo creare nuovi programmi ma anche modificare le librerie di sistema o il sistema stesso, quindi a differenza del SDK questo pacchetto permette di eseguire lavori a livello più basso, bypassando la Dalvik Virtual Machine.
La documentazione ufficiale di Android indica, però, l' NDK come uno strumento che estende il SDK, cioè la loro idea è che tramite questo pacchetto software si vada a modificare una parte del sistema piuttosto che una libreria

da adattare ad un' applicazione implementata tramite l' SDK.
I tool più utilizzati ed importanti sono:

- ndk-build;
- ndk-gdb;
- ndk-stack.

4.2.1 Ndk-build

L' ndk-build è uno script che ci permette di fare essenzialmente due cose, compilare applicazioni scritte in codice nativo per Android e compilare il sistema operativo stesso.
Le azioni principali che si possono eseguire tramite questo script sono: compilare un progetto o il sistema operativo, eliminare tutti i file binari generati per un determinato progetto e generare codice debuggabile.

4.2.2 Ndk-gdb

Questo script non è altro che un debug per questo tipo di applicazioni, per poterlo eseguire bisogna aver prima compilato il codice utilizzando l' ndk-build e indicando tra le opzioni del comando il parametro **NDK-DEBUG=1**, in modo da aver generato del codice che sia debuggabile ed infine che si stia lavorando utilizzando le librerie relative alla versione 2.2 o superiore di Android, poichè non è compatibile con quelle precedenti. Come ogni strumento di debug anche con l' ndk-gdb è possibile fornire dei breakpoint.

4.2.3 Ndk-stack

L' ndk-stack è un tool che permette di filtrare le tracce dello stack e mostrarle così come appaiono nell' output dell' *adb logcat*, sostituendo ogni

indirizzo interno ad una libreria condivisa con il corrispondente valore *file-di-origine*:*numero-riga*.
È molto comodo se utilizzato in aggiunta all' ndk-gdb per individuare più rapidamente gli errori all' interno del codice.

4.3 BusyBox

BusyBox è un software libero rilasciato con licenza *GNU General Public License*[3] che combina diverse applicazioni standard di Unix in un piccolo eseguibile. BusyBox fornisce quindi un metodo per rimpiazzare tutte le utility che solitamente si hanno a disposizione sui sistemi Linux, queste utility però sono ovviamente fornite in una versione minimalista, ma completa, rispetto alla loro controparte nei sistemi desktop, questo per rendere il più leggero possibile questo pacchetto. Cioè in sostanza si possono eseguire tutti i comandi che si trovano nei sistemi Unix, avendo però a dispozione meno opzioni per ognuno di essi, le opzioni che sono state scartate però non sono quelle fondamentali ma soltanto quelle che offrono dei servizi per rendere il lavoro più confortevole, questo significa quindi che non c'è perdità di funzionalità.
BusyBox è stato implementato e pensato per ottimizzare lo spazio e le risorse utilizzate, inoltre è estremamente modulare e questo permette a chiunque e abbastanza facilmente di includere o escludere un comando piuttosto che una semplice opzione, durante la compilazione. In questo modo è possibile personalizzare a proprio piacimento BusyBox e quindi il sistema stesso. Proprio per questo gli sviluppatori stessi definiscono BusyBox *"il coltellino svizzero del Linux embedded"*.
In definitiva senza questa libreria non potremmo eseguire alcuni comandi importanti per il successo del lavoro come ad esempio tra gli altri, **ip route** e **ip rule** che sono fondamentali per creare le regole di ruoting per interfacciarsi con la rete, senza questi sarebbe infatti impossibile configurare adeguata-

[3]Licenza che permette all' utente di utilizzare,copiare,modificare e redistribuire il software.

mente un' interfaccia e come conseguenza sarebbe impossibile accedere alla rete.
Per quanto riguarda Android esistono diversi modi per avere a disposizione BusyBox:

- scaricarlo dal Google Play Store, in questo modo si avrà a disposizione una vera e propria applicazione con tanto di interfaccia grafica per la gestione, però la libreria sarebbe statica e gestita dal sistema impedendo all' utente il pieno controllo;
- scaricando l' eseguibile ed importandolo nella cartella bin del device, in questo caso potremo gestire come meglio crediamo il file binario, ma come prima non è possibile aggiungere o rimuovere comandi e/o opzioni;
- l' ultima possibilità è quella di scaricarsi il codice sorgente, scrivere l' Android.mk[4] e compilarlo, in questo modo possiamo aggiungere o rimuovere qualsiasi comando e/o opzione.

4.4 Wpa_supplicant

L' ultimo, ma non meno importante, strumento indispensabile per il corretto funzionamento del monitor è il wpa_supplicant.
Il wpa_supplicant è un software libero, implementato seguendo gli standard dell' **IEEE 802.11**, disponibile per i maggiori sistemi operativi presenti sul mercato.Grazie a questo tool è possibile gestire le connesioni di tipo WPA e WPA2. L' elenco completo dei tipi di rete gestiti è il seguente:

1. WPA;
2. WPA-PSK e WPA2-PSK con chiave per-condivisa;
3. WPA con EAP

[4]makefile dei sistemi operativi Android

inoltre è in grado di gestire tutte le chiavi del tipo CCMP,TKIP e WEP. Esistono due versioni di questo software, una versione con interfaccia grafica ed una utilizzabile soltanto da linea di comando, offrono entrambe gli stessi servizi e funzionalita, ma ovviamente la prima è più user-friendly mentre la seconda è più rapida da utilizzare.
Su Android di default è già presente il wpa_supplicant, disponibile da linea di comando, quindi successivamente indicherò come utilizzarlo e configurarlo correttamente, ma non ci sarà nessun problema di portabilità o simili.
Il suo funzionamento è abbastanza semplice ed intuitivo, il wpa_supplicant grazie alle informazioni relative all' autenticazione e alla sicurezza delle reti, guida l' adattatore wireless nell' effettuare lo scan dei canali radio per individuare gli AP dei quali conosce le informazioni di autenticazione. Se più AP sono disponibili allora seleziona quello che ha il miglior segnale radio, dopodichè esegue la procedura di assegnazione. Affiancando al wpa_supplicant il tool **wireless tool** è possibile configurare altri parametri, come ad esempio la possibilità di connettere una sola interfaccia dello stesso dispositivo con un AP.
Ecco un esempio del file di configurazione di questo tool:

```
modules="iwconfig"

# Configuro le chiavi WPA per gli Access Point denominati ESSID1 e ESSID2
# È possibile configurare fino a 4 chiavi WPA,
# ma utilizzarne solamente 1 alla volta
#
# Prefissare la chiave con s: significa che è una chiave ASCII,
# altrimenti è una chiave esadecimale (HEX)
#
# enc open specifica una sicurezza aperta
# enc restricted specificata una sicurezza ristretta
key_ESSID1="[1] s:"chiave" key [1] enc open"
key_ESSID2="[1] aaaa-bbbb-cccc-dd key [1] enc restricted"
```

```
#
# Qualche volta è visibile più di un Access Point per cui si deve
# definire un ordine preferito per connettersi
preferred_aps="'ESSID1' 'ESSID2'"
#
# Altre volte invece non ci si vuole connettere ad alcuni access point
blacklist_aps="'ESSID3' 'ESSID4'"
#
# Se si possiede più di una scheda wireless, è possibile dare il
# permesso a ogni scheda di associarsi (o no) allo stesso Access Point
# Valori sono "yes" e "no"
# Predefinito è "yes"
unique_ap="yes"
```

Tornando al wpa_supplicant, durante l' associazione con un access point esegue questi passi:

- chiede all' apposito driver del kernel di eseguire la scansione delle reti;
- selezione una rete in base alla sua configurazione;
- richiede all' apposito driver di effettuare l' associazione con quella rete;
- quindi in base al protocollo di autenticazione si distinguono due casi:
  1. **WPA-EAP**: l' IEEE 802.1X supplicant integrato completa l' autenticazione EAP fornendo la chiave al server di autenticazione;
  2. **WPA-PSK**: il wpa_supplicant usa il PSK come chiave di sessione;
- configura le chiavi di crittografia unicast[5] e broadcast[6];
- a questo punto può partire la comunicazione e quindi lo scambio di pacchetti di dati.

[5]Comunicazione uno ad uno.

[6]Comunicazione uno a tutti gli elementi della rete

Il wpa_supplicant viene configurato tramite un file di testo, all' interno del quale vengono indicate tutte le connessioni di cui l' utente dispone le credenziali per l' accesso.
Per ogni connessione possono essere indicati più parametri, alcuni di essi sono essenziali (ad esempio: ssid, cifratura, password, ecc..), mentre altri sono opzionali (ad esempio: la priorità), altri ancora si utilizzano solo in alcuni casi, come ad esempio indicare l' utilizzo di un certificato di connessione che avviene soltanto quando la connessione ne fa uso, come avviene con l' AlmaWiFi.

Capitolo 5

Progettazione

Android, ovviamente, è nativamente in grado di gestire le interfacce di rete, ma in maniera statica, tramite questo lavoro invece si vuole fornire la possibilità di gestire le interfacce in modo dinamico, in modo da poter fare switch tra le reti conosciute, nel momento in cui è presente una rete con un segnale migliore rispetto all' attuale connessione. Lo scopo del lavoro è quello di fornire una versione adatta ai sistemi Android del Monitor per la configurazione dinamica delle interfacce di rete, partendo dalla versione disponibile in tutti i sistemi basati su kernel Unix.
Il lavoro svolto in questa tesi può essere suddiviso in tre sotto parti, la prima riguardante la compilazione del Monitor in modo da renderlo compatibile con Android, la seconda consiste nella configurazione e testing su un emulatore e l' ultima riguarda il porting del Monitor su un dispositivo reale.
In questo capitolo verrà tralasciato l' aspetto della compilazione del monitor,e comunque tutta la parte tecnica, che varranno ripresi nel successivo capitolo, quello relativo all' implementazione.

5.1 Porting Monitor sull' emulatore

L' obiettivo finale è quello di poter gestire il Monitor, lanciandolo da riga di comando, cosicchè possa configurare e deconfigurare le interfacce

dinamicamente.

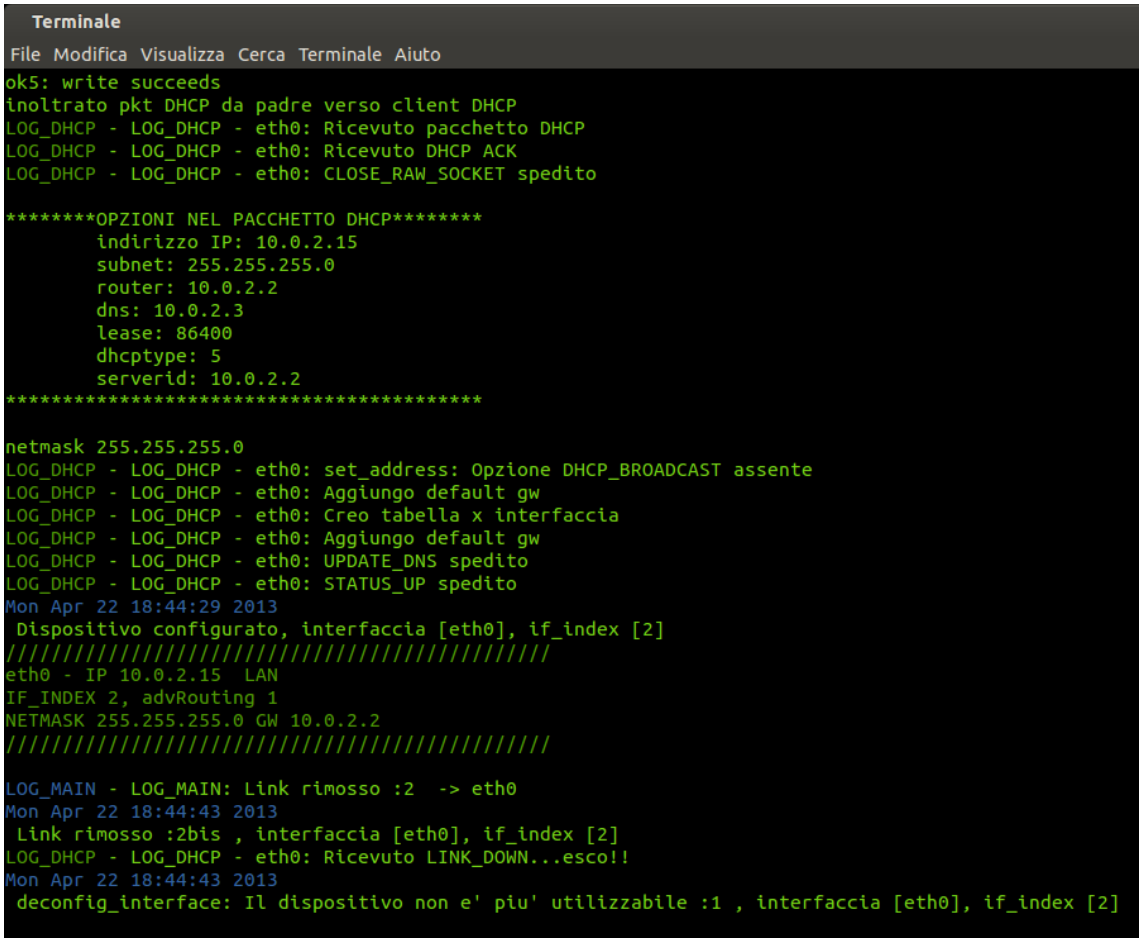

Figura 5.1: Shell dell' emulatore che mostra il monitor in esecuzione

5.1.1 Ambiente di lavoro

Questa parte di lavoro è stata svolta utilizzando un' emulatore che ci viene fornito dal pacchetto degli sviluppatori Android, SDK. Inoltre è necessario scaricare una versione del sistema operativo sviluppata appositamente per gli emulatori, chiamata **Goldfish**, altrimenti non sarebbe possibile apportare le modifiche necessarie al kernel per il corretto funzionamento del Monitor.
Infine per poter eseguire tutti i comandi necessari alla creazione delle regole

di routing[1], e quindi delle tabelle IP da parte del Monitor, bisogna importare BusyBox, che come detto in precedenza può essere installato direttamente dal Google Play Store, oppure è possibile scaricarlo dal sito ufficiale e compilarlo, in questo modo possiamo aggiungere o eliminare comandi per renderelo rispettivamente ancora più funzionale o più leggero.
Inoltre bisogna configurare opportunamente il wpa_supplicant tramite il file di configurazione *wpa_supplicant.conf*.

5.2 Porting Monitor su smartphone

In questo caso ho lavorato su un dispositivo Samsung Galaxy Next con versione Android 2.3.6 Gingerbread, con installato la patch per ottenere i permessi di root.

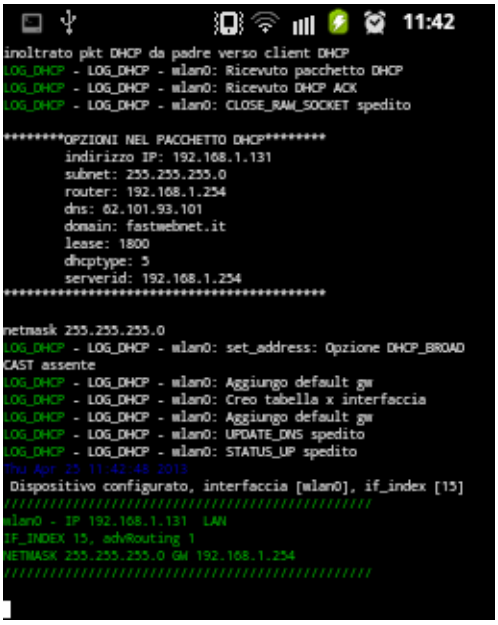

Figura 5.2: Shell dello smartphone che mostra il monitor in esecuzione

[1]Insieme di regole che permette la configurazione dell' interfaccia di rete

5.2.1 Ambiente di lavoro

Come detto bisogna quindi avere a disposizione un device sul quale si è in possesso dei permessi di amministartore, senza tali permessi non sarebbe possibile eseguire il Monitor, infatti Android di default non concede questi permessi all' utente al fine di proteggerlo, poichè in questo modo ci si espone, oltre alla possibilità di un errore umano che può compromettere il sistema, a maggiori possibilità di subire un attacco da parte di un malintenzionato con l' obiettivo di creare un malfunzionamento nel sistema o di appropriarsi di dati sensibili.
Come con l' emulatore anche in questo caso è necessario importare BusyBox, sempre per gli stessi motivi.
Inoltre è molto utile avere a disposizione sul proprio dispositivo un' applicazione che emuli il terminale ed una per la gestione del file system per gli amministratori.

Capitolo 6

Implementazione

L' implementazione ha richiesto una prima fase di lavoro e sviluppo su emulatore, e soltanto dopo aver raggiunto l' obiettivo preposto e cioè rendere funzionante il Monitor in un sistema Android si è passati ad un lavoro su un device reale, uno smartphone in questo caso.
Le differenze tra i due casi non sono poche, anche se la maggior parte di esse risiedono nelle configurazioni dei due diversi ambienti di lavoro piuttosto che nel codice stesso del software.

6.1 Implementazione su emulatore

Innanzitutto avremo bisogno di scaricare l' Android SDK per poter creare un nuovo emulatore sul quale andare a lavorare.

6.1.1 Creare emulatore

Dopo aver scaricato l' SDK tramite il tool **android** effettuiamo il download delle API in base alla versione del sistema operativo che vogliamo emulare, come già detto ho optato per la versione 2.3 perchè è ancora quella più in uso. A questo punto sarà possibile creare un nuovo emulatore da linea di comando oppure utilizzando un altro tool che il pacchetto ci mette a disposizione e cioè il **ManageAVDs**. Dopo aver definito tutte le caratteritiche

del nostro emulatore sarà possibile lanciarlo tramite lo stesso tool, oppure da linea di comando digitando **emulator** e il nome dell' AVD.
L'emulatore da me utilizzato per svolgere l'intero lavoro ha le seguenti caratteristiche:

- **Sistema Operativo**: Android Gingerbread 2.3.3 - API level 10;
- **CPU/ABI**: ARM (armeabi);
- **SdCard size**: 50 MB;
- **SdCard path**: /home/antonello/android;
- **Skin**: WVGA 800.

6.1.2 Download e configurazione Goldfish

Adesso che abbiamo il nostro emulatore dobbiamo scaricare il kernel Goldfish, che ci permetterà di apportare modifiche a basso livello senza le quali il Monitor non funzionerebbe. Per farlo ci basta digitare da terminale:

- git clone https://android.googlesource.com/kernel/goldfish.git
- cd goldfish
- git checkout -t origin/android-goldfish-2.6.29 -b goldfish

Al kernel sono stati modificati alcuni file per fare in modo che quest' ultimo possa comunicare con il monitor in modo da segnalare eventuali pacchetti persi.
L' ultimo accorgimento da prendere prima di compilare il kernel è modificarne la configurazione in modo che supporti la gestione multipla delle tabelle per gli indirizzi IP, per far sì bisogna modificare il file nascosto *.config* presente all' interno della cartella *goldfish* e cambiare attivando le seguenti due opzioni:

- CONFIG_IP_ADVANCED_ROUTER

- CONFIG_IP_MULTIPLE_TABLE

adesso si può compilare il kernel, utilizzando il compilatore adatto fornito dall' NDK, e lanciare l' emulatore sul kernel appena scaricato e compilato indicando l' immagine contenuta nella cartella *zImage* di **Goldfish** come parametro dell' opzione **-kernel** del comando di shell **emulator**.
Ecco un esempio del comando:

./emulator -avd EmulatoreProva -kernel /home/goldfish/arch/arm/boot/zImage

6.1.3 Android.mk

Adesso prima di poter compilare il progetto c'è bisogno di un makefile, Android però non utilizza gli stessi makefile di Linux, ma una sua versione molto simile nell'idea a quelli appunto per Linux, ma utilizzando un linguaggio leggermente diverso.
Ecco un esempio di come funzionano gli Android.mk

```
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)


LOCAL_MODULE := monitor
LOCAL_SRC_FILES := sorgente1.c,sorgente2.c,sorgente3.c
LOCAL_SHARED_LIBRARIES := libc
LOCAL_CFLAGS += -O1 -s -fomit-frame-pointer -W -Wall
include $(BUILD_EXECUTABLE)
```

All'inizio del file va indicata la variabile **LOCAL_PATH**, che indica dove sono collocati i file sorgenti. La macro **my-dir**, fornita dal buil-system, ritorna il path della directory corrente. La macro **include (CLEAR_VARS)**,

anch'essa fornita dal build-system, richiama un particolare script che ha il compito di inizializzare le variabili utilizzate.
Con la variabile **LOCAL_MODULE** si indica il nome associato al modulo all'interno dell'Android.mk, questo deve essere unico e privo di spazi e sarà il nome che il compilatore assegnerà all'eseguibile. La variabile **LOCAL_SRC_FILES** indica quali sono i file sorgenti da compilare, con **LOCAL_SHARED_LIBRARIES** si indica la lista delle librerie dinamiche da cui il modulo dipende a runtime.
La varibile **LOCAL_CFLAGS** serve per indicare i flag per la compilazione.Infine la macro **BUILD_EXECUTABLE** serve per richiamare un particolare script utilizzato per creare l'eseguibile in base alle informazioni espresse tramite le variabili precedentemente utilizzate.

6.1.4 Compilazione ed installazione monitor

Adesso si può passare alla compilazione ed all'installazione del monitor. Un progetto Android, seppur a basso livello, deve rispettare una certa struttura per poter essere compilato, cioè all' interno della cartella del progetto, bisogna creare delle sottocartelle per separare il codice sorgente, dalle librerie piuttosto che dalle immagini.
La struttura è la seguente:

- **assets**: questa cartella contiene quelle risorse che non vengono mappate nella classe auto-generata **R**, queste risorse non vengono compilate e tantomeno ottimizzate da Android. Per accedere a queste informazioni tramite un' Activity si utilizza la classe *android.content.res.AssetManager*;

- **bin**: questa è una cartella auto-generata e come intuibile contiene tutti i file binari che si ottengono dopo la compilazione;

- **gen**: anche questa è una cartella che si genera solo dopo la compilazione e contiene risorse che Android gestisce autonomamente, come il file **R.java**, che ad ogni compilazione viene generato, è quindi inutile modificarlo manualmente;

- **jni**: all'interno di questa cartella vanno inseriti tutti i file che compongono il codice sorgente dell'applicazione, compresi i file di compilazione;
- **libs**: contiene file di terze parti di cui l'applicazione ha bisogno (librerie) in formato Jar;
- **obj**: contiene i file oggetto relativi al codice sorgente presente all'interno della cartella jni ottenuti dopo la compilazione;
- **res**: questa cartella contiene le risorse e i file di configurazione esterni al codice, a sua volta anch'essa si divide in sottocartelle in base al tipo di risorsa, le cartelle *drawable*, che contengono le immagini nelle varie risoluzioni, la cartella *values*, che contiene file xml che definiscono variabili e valori e la cartella *layout*, che contiene file xml che definiscono lo stile del layout dell'applicazione, è inoltre possibile definirsi nuove cartelle per indicare altri tipi di risorse utili;
- **src**: questa cartella contiene i file sorgenti relativi alla parte ad alto livello dell'applicazione, per farla breve in jni inseriamo i file C mentre in src i file Java.

Quando si lavora escusivamente a basso livello le cartelle con cui si interagisce maggiormente sono assets e jni, mentre quelle meno utilizzate saranno ovviamente res e src.

Per compilare il Monitor, utilizziamo ancora una volta l' Android NDK, dall' interno della cartella del progetto richiamo il compilatore, che sarebbe il tool descritto in precedenza **ndk-build**.
Per utilizzare questo tool è sufficiente posizionarsi da shell all'interno della cartella del progetto, tramite il comando *cd* e digitare sempre da shell il seguente comando:

/home/android-ndk/ndk-build

Adesso per installare il Monitor sull' emulatore, mi basta copiare l' eseguibile appena generato all' interno della cartella */system/bin*, dopo aver montato il file system in lettura e scrittura tramite il comando **mount**.
Quindi configuro il wpa_supplicant aggiungendo le opzioni delle reti conosciute all' interno del file *wpa_supplicant.conf*, ecco un esempio di configurazione per una rete protetta da una chiave WPA-PSK e per un'altra aperta

```
network={
ssid="ESSID1"
psk="password"
key_mgmt=WPA-PSK
priority=1
}

network={
ssid="ESSID2"
key_mgmt=none
priority=2
}
```

Infine importo BusyBox copiando l'eseguibile all'interno della cartella */system/bin*, proprio come fatto per il binario del monitor, a questo punto posso esseguire il monitor digitando il seguente comando:

monitor [**NumeroPorta**]

ottenendo quindi una risposta simile a quella che si può vedere nella figura 5.1.

6.2 Implementazione su smartphone

Come già specificato in precedenza per l' implementazione su smartphone ho utilizzato un Samsung Galaxy Next con installata la versione Gingerbread 2.3.6 del sistema operativo Android.

6.2.1 Root dello smartphone

Come prima cosa bisogna effettuare il rooting dello smartphone, cioè installare una patch del sistema operativo, che ci consenta di ottenere i permessi di amministratore del dispositivo. Android non concede questi diritti di default perchè se utilizzati nel modo sbagliato possono portare ad un mal funzionamento o ad un completo blocco dello smartphone, inoltre ci si espone maggiormente agli attacchi da parte degli hacker.
Android è un mondo vasto, quindi non esiste un solo metodo per effettuare il rooting, nel mio caso ho semplicemente scaricato e copiato sulla sd card un pacchetto che poi ho installato tramite il **recovery mode**[1](Vedi Foto 6.1) del dispositivo.
Per accedere al recovery mode dello smartphone bisogna accenderlo premendo contemporaneamente più tasti, questa combinazione varia in base al dispositivo sul quale si lavora, nel mio caso si tratta dei seguenti tre tasti : *tasto accensione + tasto centrale + tasto del volume in giù.* A questo punto muovendoci nell'elenco con i tasti per aumentare e diminuire il volume, che permettono di spostarsi tra le varie voci disponibili, selezioniamo **apply update from sdcard**, dopodiché selezioniamo la patch sulla SdCard e lanciamo l'installazione della patch al termine della quale lo smartphone verrà riavviato.
A questo punto nello smartphone verrà installata un' applicazione tramite la quale si possono gestire i permessi da amministratore da fornire alle varie applicazioni che ne faranno richiesta.

[1]modalità di avvio dello smartphone utile per il ripristino, la formattazione o l' installazione di patch.

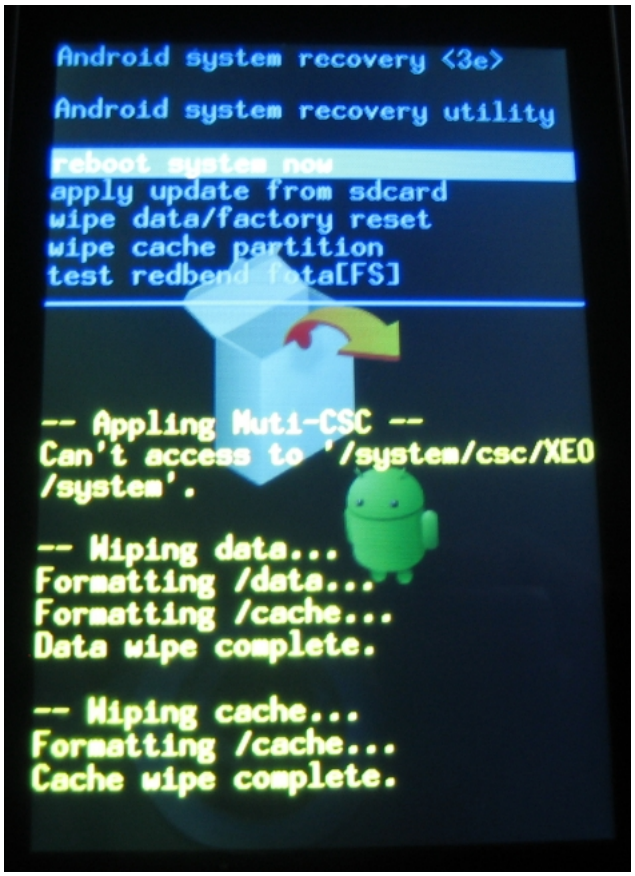

Figura 6.1: Recovery Mode dello smartphone

6.2.2 Configurazione del Monitor

A differenza dell' emulatore dove tutto è definito seguendo un certo standard, nel caso di un dispositivo reale non esiste un' uniformità tra i vari produttori o anche tra i vari prodotti della stessa azienda.
Ad esempio non tutti usano la stessa stringa per identificare la stessa interfaccia di rete, oppure lo stesso indirizzo per indicare il file di configurazione del wpa_supplicant.
Quindi come prima cosa bisogna controllare come il nostro dispositivo identifichi l'interfacce sulle quali lavora in modo da poterle specificare al Monitor, per ottenere l' elenco di tutte le interfacce di rete basta digitare da linea di comando **netcfg**, questo comando ci restituirà in output un elenco di inter-

facce, dove indica con i flag UP e DOWN quali sono attive e quali spente, per quelle attive mostra anche l' indirizzo IP.
A questo punto nel file *init.c* all'interno del main vi è un elenco con tutte le interfacce da non gestire, quindi troviamo le nostre interfacce e le commentiamo, in modo che il monitor le gestisca.
Il Monitor ha anche bisogno del file di configurazione del wpa_supplicant, il *wpa_supplicant.conf*, e dell' interfaccia di controllo per ogni interfaccia disponibile, ognuno di questi file è salvato come *nome_interfaccia.conf* (Es: *wlan0.conf*), per trovarli basta usare la funzione di ricerca di un gestore di file manager e a quel punto indicarne i vari path al Monitor.
Il path del *wpa_supplicant.conf* va indicato all'interno del file *config.h* andando a modificare la define **WPA_CONF_DEF_PATH**, mentre il path del file dell'interfaccia di controllo va indicato di nuovo nel file *init.c* all'interno della funzione **exec_wpa_supplicant** là dove vi è il commento della ricerca, appunto, del file di controllo.

6.2.3 Compilazione ed installazione Monitor

Il processo di compilazione è del tutto identico a quanto effettuato per l' emulatore, dunque utilizzo nuovamente il tool **ndk-build** fornito dall' Android-NDK, seguendo la stessa procedura descritta nel capitolo 6.1.4.
Anche per l' installazione sul device faccio la stessa cosa, cioè copio l' eseguibile appena generato del Monitor all' interno della cartella bin del sistema.
Quindi dopo aver configurato il wpa_supplicant tramite il file *wpa_supplicant.conf* l' ultima operazione da effettuare prima di poter lanciare il Monitor è l' installazione di BusyBox, entrambi questi passaggi sono del tutto simili a quanto descritto nel capitolo 6.1.4, dopodichè è possibile eseguire il Monitor da linea di comando digitando:

monitor [***numero_porta***]

otteneno un risultato simile a ciò che si può vedere nella figura 5.2.

Capitolo 7

Valutazioni

Con l' installazzione e l' esecuzione del Monitor sullo smartphone è possibile essere connessi sempre alla rete migliore disponibile, ovviamente tra quelle conosciute dal wpa_supplicant, questo perchè una volta lanciato il Monitor lavora sottotraccia, senza che l' utente debba fare altro, facendo lo switch tra un Access Point e l' altro qualora ne venga individuato uno che offre un segnale più potente e quindi potenzialmente una rete dalle prestazioni migliori.
L' operazione di switch tra i vari AP nel mio caso avviene orientativamente entro i quattro, cinque secondi, però qui entrano in gioco tanti fattori, inoltre bisogna valutare anche la potenza dei diversi segnali dei vari AP, e non ultimo il tipo di protezione della rete, una rete non protetta consente ovviamente ad un dispositivo di ottenere prima un idirizzo IP rispetto alle reti protette, ed inoltre anche i diversi tipi di autenticazione richiedono tempistiche differenti, ad esempio una connessione come quella dell' università richiede un tempo maggiore per l' autenticazione rispetto ad una rete con una semplice autenticazione WPA oppure WEP.
Nel mio test effettuato con due connessioni su due AP differenti, una con protezione WPA-PSK che chiamaeremo connessione A e l' altra senza alcun tipo di autenticazione che chiameremo connessione B ho potuto notare come il Monitor si comportasse in base alla mia posizione avvicinandomi ad un AP

e di conseguenza allontanandomi dall' altro.
Partendo dalla connessione A ed eseguendo il monitor dopo all' incirca cinque secondi il dispositivo si configura ed ottiene quindi l' indirizzo IP, quindi spostandomi verso l' AP della connessione B, raggiunto il punto x dove il segnale radio dell' AP della connessione B è maggiore di quello della connessione A, il Monitor deconfigura l' interfaccia cancellando le regole di routing relative alla connessione A e si configura con l' AP della connessione B creando una nuova tabella con delle nuove regole di routing relative alla nuova connessione, in questo caso trattandosi di una connessione senza protezione il tempo di configurazione è stato più breve, circa un paio di secondi, quindi come detto prima le tempistiche variano in base a diversi fattori, ma il tutto viene risolto sempre nell' arco di pochi secondi.

Capitolo 8

Conclusioni

Il codice del monitor è stato scritto e implementato per lavorare in ambienti Linux, quindi essendo Android un sistema operativo **Unix Based** non bisogna sorprendersi più di tanto se è stato possibile effettuare un porting "completo", cioè senza la minima perdità di funzionalità tra la versione Desktop e quella Mobile del Monitor.
Durante il mio lavoro ho notato come sia utile poter implementare applicazioni ad un livello così basso, poichè in questo modo è possibile sfruttare al meglio le potenzialità del device ed avere il "pieno" controllo di ciò che si sta implementando senza dover passare attraverso macchine virtuali come avviene per le applicazioni di alto livello, l' unico problema a mio parere è che questo tipo d' approccio rende la portabilità del software molto ridotta, piochè appunto si lavora a stretto contatto con il sistema che si ha a disposizione legandocisi ad esso e rendendo praticamente impossibile eseguire lo stesso software da un dispositivo ad un altro senza apportare neanche la minima modifica.
Un problema che porta via molto tempo nel porting di un qualsiasi software da Linux ad Android risiede nei file di compilazione, infatti il sistema operativo mobile di Google non usa i **Makefile**, bensì gli **Android.mk**, che altro non sono che un dialetto dei makefile utilizzati in Linux, ed essenzialmente il problema è che non esiste un software in grado di generare automaticamente

un Android.mk partendo da un Makefile, questo perchè si considera che un makefile possa mostrare strutture troppo complesse per essere gestite da uno script automatico.
Infine bisogna aggiungere che Android non possiede un' implementazione completa delle librerie di C, in quanto la disponibilità di memoria e capacità di calcolo di un dispositivo mobile sono inferiori ad un sistema Desktop, certo è possibile importare ciò di cui si ha bisogno, però bisogna trovare il giusto compromesso tra funzionalità e "peso" del software, altrimenti ci si ritrova ad avere un dispositivo molto lento con il quale diventa difficile eseguire anche le operazioni di base per cui essi sono stati ideati.Certo negli ultimi anni l' abbattimento dei costi dei materiali primi e di assemblaggio dell' hardware e le maggiori innovazioni nel mondo del software hanno portato in poco tempo a produrre smartphone con capacità di calcolo superiori alla media dei computer assemblati non più addietro di una decina di anni fà o forse anche meno, quindi magari tra qualche anno questo è un problema che non ci si porrà.
Quindi in conclusione posso affermare che il porting del **Monitor** è avvenuto con successo e che tutte le modifiche apportate al sistema per permette appunto al Monitor di funzionare non hanno creato nessun problema nè alle altre applicazioni, nè tantomeno al sistema stesso.

8.1 Sviluppi futuri

Per ora Android non consente di utilizzare più interfacce di rete in contemporanea, questo perchè lo reputano un consumo inutile di batteria, aspetto sicuramente vero e da tenere in considerazione, però può essere comodo avere attive più interfacce nello stesso momento consentendoci di effettuare in questo modo due operazioni separate tra di loro, ad esempio magari su di una posso avviare il download di un' applicazione dal Play Store e sull' altra fare una chiamata VoIp[1], cosicché le due azioni non si "rubino" banda a vicenda,

[1]Voice Over IP, protocollo utilizzato per il trasferimento della voce nella rete

finendo col non riuscire a completare nessuna delle due azioni.
Infine un ulteriore sviluppo futuro potrebbe essere il porting dell' intero sistema di ABPS (Always Best Packet Sitching) di cui il Monitor è un componente, anche perchè, a mio parere, se è stato possibile effettuare il porting di un elemento del sistema e considerando che tutti gli altri componenti lavorano allo stesso livello e sfruttando più o meno le stesse risorse, è ovvio pensare che questo sia un lavoro sicuramente fattibile.

Bibliografia

[1] V. Ghini, G. Lodi, F. Panzieri, "Always Best Packet Switching: the Mobile VoIP Case Study", Journal of Communications (Academy Publisher), Oct. 2009. http://www.academypublisher.com/ojs/index.php/jcm/article/view/0409700713/920.

[2] Information Sciences Institute University of Southern California, "RFC791 - Internet Protocol", Websiste, 1981, <http://www.faqs.org/rfcs/rfc791.html>

[3] IEEE Std. 802.11n, Website, 2009, <http://standards.ieee.org/announcements/ieee802.11n_2009amendment_ratified.html>

[4] IEEE Std. 802.11n, "Higher Throughput Improvements using MIMO", IEEE Standard for Information Technology, 2007

[5] IEEE Std. 802.11e, Website, 2008, <http://standards.ieee.org/getieee802/download/802.11e-2005.pdf>

[6] ANSI/IEEE Std 802.11, 1999 Edition

[7] Massimo CARLI "ANDROID: GUIDA PER LO SVILUPPATORE", Apogeo Editore, 2010

[8] ANDROID developer, Websiste, <http://developer.android.com/index.html>

[9] ANDROID Software Developemente Kit, Websiste, <http://developer.android.com/sdk/index.html>

[10] ANDROID Native Developemente Kit, Websiste, <http://developer.android.com/tools/sdk/ndk/index.html>

[11] BusyBox tool, Website, <http://www.busybox.net>

[12] Wpa_supplicant tool, Website, <http://hostap.epitest.fi/wpa_supplicant>

[13] Linux system call journal, Website <http://www.linuxjournal.com/article/4048>