Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Magistrale

CUSTOMIZZAZIONE DI ANDROID

Tesi di Laurea in Sistemi Mobili

Relatore:
Dott.
VITTORIO GHINI

Presentata da:
GIUSEPPE PIO RINALDI

Sessione II
Anno Accademico 2012/2013

Parole chiave:
Wireless, VoIP, Android, Trasmission Error Detector, Kernel

A Francesco
che mi ha indirizzato sempre sulla retta via

Introduzione

Negli ultimi anni abbiamo assistito ad un'evoluzione delle tecnologie di comunicazione senza precedenti. Internet ha rivoluzionato una moltitudine di aspetti della nostra societá. Oggi é possibile trovare qualsiasi tipo di notizia, articolo o aneddoto digitando una manciata di parole chiave su un motore di ricerca, il singolo individuo per la prima volta puo pubblicare i propri contenuti attraverso blog, pagine web e servizi di microblogging piuttosto che essere uno spettatore passivo dei vecchi media ed ogni giorno persone di tutto il mondo si mantengono in contatto indipendentemente dalla loro posizione geografica grazie ad email, chat e siti di social networking. Internet ha introdotto scenari e classi di servizi completamente nuovi, stravolgendo le vecchie abitudini: si pensi alla posta elettronica piuttosto che alle lettere cartacee e ai servizi di banking on line comparate alle code agli sportelli, tutto questo con l'aspetto non certo secondario di un marcato abbattimento dei costi.

Una tecnologia che promette ulteriori vantaggi economici attirando in modo crescente l'attenzione su di sé é sicuramente il Voice over Internet Protocol (VoIP). Questa consente di trasportare una conversazione voce (cioé una conversazione telefonica) su di una qualunque rete basata su protocollo IP. L'adozione del VoIP é spinta quasi sempre da due motivazioni fondamentali: l'abbattimento dei costi di un classico contratto telefonico e la possibilitá di usufruire di servizi avanzati non sempre disponibili sulla rete telefonica tradizionale, ad esempio conversazione video o teleconferenza audio.

Allo stesso tempo, l'esigenza di essere connessi ad Internet sempre e co-

munque ha portato allo sviluppo di dispositivi portatili come smartphone, laptop e tablet ed a tecnologie per l'accesso alla rete senza fili. La tecnologia Wi-Fi con lo standard IEEE 802.11 si é imposta sopra le altre per quanto riguarda prestazioni e diffusione. Oggi, sempre piú luoghi come universitá, ospedali, uffici e abitazioni sono dotati da infrastrutture basate su tale tecnologia che consentono alle persone di poter accedere alla rete in libertá di movimento e senza vincoli fisici.

Per questo, il VoIP usato sopra tecnologie wireless é sempre piú frequente. Tale associazione peró presenta sfide non banali che devono essere indirizzate a dovere prima che tali servizi diventino di uso comune: il problema principale riguarda la qualitá del servizio, necessaria per una conversazione soddisfacente, che invece degrada a fronte di interferenze sul canale radio, la mobilitá del terminale e congestioni negli access point.

Il punto di partenza di questa tesi, é stato quello di capire quali erano i procedimenti necessari per poter arrivare ad un kernel personalizzato, partendo dai file sorgenti, e come alla fine poter rendere disponibile questa soluzione per diverse tipologie di smartphone. Successivamente viene installato sul sistema operativo Android, personalizzato, un modulo e un'applicazione, il primo denominato Transmission Error Detector (TED), che estende il funzionamento della tecnologia WiFi e la seconda denominata Wvdial che estende invece il funzionamento della tecnologia 3G(o UMTS). Entrambi fanno parte di una architettura per il supporto alla mobilitá in contesti eterogenei. Il contesto é eterogeneo poiché piú sistemi e protocolli convivono e vengono usati per il fine comune di migliorare l'usabilitá e l'esperienza dell'utente nel contesto wireless. Questa architettura si basa sul principio di usare piú interfacce di rete contemporanemanete per raggiungere l'obiettivo della qualitá del servizio. Sempre piú spesso i nuovi dispositivi in commercio sono dotati di piú interfacce wireless come Wi-Fi e tecnologie 3G. Ogni interfaccia é solitamente usata singolarmente e puó incorrere nei problemi sopra descritti; usate insieme, invece, esse possono migliorare le prestazioni di applicazioni multimediali, interattive e real-time che hanno bisogno di tutta la velocitá e

prontezza disponibili.

Nell'affrontare queste problematiche, un occhio di riguardo deve necessariamente essere rivolto al futuro. IP é il protocollo di Internet e, nella sua versione 4, é usato in modo ubiquo da praticamente tutti i dispositivi che si affacciano alla rete. IPv4 peró ha una limitazione che pesa sempre piú pressantemente: il suo spazio di indirizzamento é troppo piccolo per supportare la crescita esplosiva di Internet che, nel febbraio di questo anno, ha visto lo IANA consegnare gli ultimi cinque blocchi di indirizzi disponibili. La soluzione a questo problema esiste ed é il passaggio alla versione successiva, IPv6. Nel discutere l'architettura che si andrá a descrivere questo aspetto sará di importanza centrale.

Il **capitolo 1** sará incentrato sul kernel: dopo una prima panoramica sui vari tipi di kernel, il capitolo si incentrerá sul kernel Linux e sull'importanza dell'installazione di un kernel ricompilato sulla propria macchina.

Il **capitolo 2** sará incentrato sull'ambiente di sviluppo: verranno presentati i vari strumenti forniti dall'Android SDK, il plugin ADT per Eclipse e l'Android NDK. Infine saranno illustrati i due smartphone utilizzati per i test di "personalizzazione" di Android.

Nel **capitolo 3** sará presentata la guida per la customizzazione di Android, partendo dalla compilazione dei suoi file sorgente, per poi passare all'installazione del kernel sull'emulatore e sul Samsung Galaxy S, e ai vari test fatti su di essi. Verranno inoltre discussi i vari problemi riscontrati.

Nel **capitolo 4** si fará una panoramica dello scenario in cui si agisce il TED, ci si soffermerá sulle tecnologie wireless e sui protocolli che a tutti i livelli sono coinvolti nelle conversazioni veicolate sull'Internet Protocol.

Nel **capitolo 5** verrá descritta l'architettura proposta per mitigare i problemi del VoIP su reti wireless, l'uso contemporaneo di piú interfacce e i vari moduli di cui é composto il sistema.

Nel **capitolo 6** si passerá a mostrare nel dettaglio la progettazione e l'implementazione del TED.

Nel **capitolo 7** verrá descritto nel dettaglio come effettuare il porting dell'applicazione Wvdial

Nel **capitolo 8** si discuteranno i possibili sviluppi futuri e le aggiunte che potrebbero essere applicate per estendere e migliorare il lavoro svolto.

Infine l'appendice A, tratterá rispettivamente nel dettaglio: l'Android Virtual Device, il tool android, l'emulatore di Android.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Cos'é un kernel

Il kernel costituisce il nucleo di un sistema operativo e fornisce tutte le funzioni essenziali per il sistema, in particolare la gestione della memoria, delle risorse del sistema e delle periferiche, assegnandole di volta in volta ai processi in esecuzione. La controparte del kernel é la shell, ovvero l'interfaccia utente del sistema, la parte piú esterna. I programmi chiedono le risorse al kernel attraverso delle chiamate (system call) e non possono accedere direttamente all'hardware. Il kernel si occupa quindi di gestire il tempo processore, le comunicazioni e la memoria distribuendole ai processi in corso a seconda delle prioritá (scheduling) realizzando cosí il multitasking. L'accesso diretto all'hardware puó essere anche molto complesso, quindi i kernel usualmente implementano uno o piú tipi di astrazione dall'hardware, il cosiddetto Hardware Abstraction Layer. Queste astrazioni servono a "nascondere" la complessitá e a fornire un'interfaccia pulita ed uniforme all'hardware sottostante, in modo da semplificare il lavoro degli sviluppatori.

I kernel si possono classificare in base al grado di astrazione dell'hardware in quattro categorie:

- kernel monolitici, che implementano direttamente una completa astrazione dell'hardware sottostante;
- microkernel, che forniscono un insieme ristretto e semplice di astrazione dell'hardware e usano software (chiamati device driver o server) per

fornire maggiori funzionalitá;

- kernel ibridi (o microkernel modificati), che si differenziano dai microkernel puri per l 'implementazione di alcune funzioni aggiuntive al fine di incrementare le prestazioni;

- esokernel, che rimuovono tutte le limitazioni legate all'astrazione dell'hardware e si limitano a garantire l'accesso concorrente allo stesso, permettendo alle singole applicazioni di implementare autonomamente le tradizionali astrazioni del sistema operativo per mezzo di speciali librerie.

1.1 Kernel Linux

Il kernel Linux é un software libero distribuito con licenza GNU General Public License; é stato creato nel 1991 da Linus Torvalds. Integrato con il Sistema GNU, sviluppato da Richard Stallman, ha dato vita al sistema operativo GNU/Linux (chiamato comunemente con il solo nome Linux). Il kernel Linux ha una struttura monolitica; é considerata da alcuni obsoleta a differenza della piú moderna architettura a microkernel. Sebbene oggi il kernel possa essere compilato in modo da avere un'immagine binaria ridotta al minimo e i driver caricabili da moduli esterni, l'architettura originaria é chiaramente visibile: tutti i driver infatti devono avere una parte eseguita in kernel mode, anche quelli per cui ció non sarebbe affatto necessario (ad esempio i driver dei file system). Linux é un kernel che supporta il multitasking ed é multi utente. Ció permette che diversi utenti (con privilegi differenziati) possano eseguire sullo stesso sistema diversi processi software in simultanea. Attualmente Linux supporta gran parte dell'hardware disponibile per PC e supporta un numero enorme di architetture (tra cui SPARC, PowerPC e le piú moderne CPU a 64bit). Dato che il codice sorgente di Linux é disponibile a tutti, é ampiamente personalizzabile, al punto da rendere possibile, in fase di compilazione, l'esclusione di codice non strettamente indispensabile. La

flessibilitá di questo kernel lo rende adatto a tutte quelle tecnologie embedded emergenti e anche nei centri di calcolo distribuito (cluster Beowulf) fino ad essere incorporato in alcuni videoregistratori digitali e nei telefoni cellulari.

1.2 Android Kernel

Il kernel Android é sostanzialmente un kernel Linux nella versione 2.6. La scelta di una simile configurazione é nata dalla necessitá di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante attraverso l'utilizzo di diversi driver. L'installazione di un kernel "personalizzato" puó portare a numerosi vantaggi in termini di prestazioni e funzionalitá.

1.3 Custom Kernel

Per custom kernel si intende un kernel modificato per adattarlo alla esigenze del singolo utente. Spesso risulta vantaggioso installare sulla propria macchina un kernel "personalizzato", un kernel cioé ricompilato, anziché uno di quelli precompilati forniti dalla distribuzione GNU/Linux installata. Ció puó portare alcuni vantaggi, come:

- avere una configurazione diversa e piú adatta alla propria macchina;
- gestire hardware particolare o gestire conflitti hardware con kernel preconfezionati;
- usare opzioni del kernel che non sono supportate dai kernel preconfezionati (per esempio il supporto per la memoria alta);
- ottimizzare il kernel rimuovendo i driver inutili in modo da velocizzare l'avvio del sistema;
- avere un sistema operativo leggermente piú reattivo grazie all'ottimizzazione basata sul tipo di processore;

- usare un kernel aggiornato o di sviluppo;
- creare un kernel monolitico al posto di uno modulare;

Non é necessario compilare un kernel nei casi in cui l'hardware non funzioni alla perfezione o le periferiche non vengano completamente riconosciute. A volte per far riconoscere correttamente al sistema una data periferica basta caricare i moduli necessari con le dovute opzioni. É utile ricompilare il kernel solo se tali moduli non sono presenti o se si é certi che i driver della periferica sono presenti solo in una versione diversa da quella attualmente in uso. Inoltre, l'aumento di prestazioni tende a essere irrilevante, soprattutto su computer giá veloci. É bene tenere presente che compilare un nuovo kernel significa, nella sostanza, cambiare sistema operativo, in quanto esso ne costituisce il motor e inoltre é richiesta un buona conoscenza del proprio hardware.

Capitolo 2

Ambiente di sviluppo

Uno dei grandi punti di forza di Android é la sua piattaforma di sviluppo open source, a cui vanno attribuiti buona parte dei meriti del successo raggiunto fin'ora. I membri della Open Handset Alliance[1] sono convinti che il miglior modo per offrire software di qualitá ai consumatori é rendere semplice per gli sviluppatori scriverlo. L'idea su cui é basata la piattaforma di sviluppo di Android é quella di non limitare in nessun modo le potenzialitá degli sviluppatori offrendo loro gli stessi mezzi usati dai creatori di Android stesso. Infatti le applicazioni di terze parti vengono considerate allo stesso livello di quelle native. Inoltre Android permette di utilizzare parti di applicazioni all'interno di altre non solo attraverso il classico riutilizzo di codice ma soprattutto grazie ad un efficace meccanismo basato sugli Intent Filters. Gli Intent Filters sono un metodo per esporre al resto del sistema le azioni che un'applicazione puó compiere in modo da poter essere sfruttate da qualunque applicazione. Essi offrono numerosi benefici tra cui:

- massimizzare il riutilizzo e la modularitá dei componenti. Le applicazioni possono specializzarsi su servizi singoli e in caso di necessitá interagire tra loro per fornire un servizio migliore;

[1] L'Open Handset Alliance (abbreviato OHA) é un accordo di differenti compagnie con Google come capofila, ASUS, HTC, Intel, Motorola, Qualcomm, T-Mobile, e NVIDIA il cui obiettivo é sviluppare "standard aperti" per dispositivi mobili.

- adattare al meglio le applicazioni alle esigenze dell'utente. Ad esempio l'applicazione relativa alla fotocamera, che ha una funzione per permettere la condivisione delle immagini, puó chiedere all'utente di selezionare l'applicazione da usare per portare a termine tale operazione. Il software che richiede il servizio puó tranquillamente non essere a conoscenza delle applicazioni correntemente installate sul dispositivo, sará il meccanismo degli Intent Filters a individuare i programmi che potranno accogliere la richiesta.

2.1 Android Software Development Kit (SDK)

Il Software Development Kit[2] di Android mette a disposizione numerosi strumenti considerati nel seguito.

- **Application Programming Interface (API)**[7]: Il nucleo della SDK é costituito dalle API che sono librerie messe a servizio degli sviluppatori e che sono esattamente le stesse usate da Google per creare le applicazioni native.
- **Android Virtual Device Manager**: Un Android Virtual Device (AVD) consiste in un vero e proprio emulatore di dispositivi Android, utile per poter sviluppare applicazioni anche senza essere in possesso di uno smartphone oppure per simulare una configurazione hardware neutrale. É possibile creare dispositivi virtuali scegliendo la grandezza e densitá del display, la versione del sistema operativo installata e altri parametri.
- **Dalvik Debug Monitor Server (DDMS)**: É uno strumento essenziale per il debugging che include una serie di funzionalitá per monitorare l'esecuzione delle applicazioni su dispositivi reali o emulatori.

[2]Software Development Kit é un termine che in italiano si puó tradurre come "pacchetto di sviluppo per applicazioni" e sta ad indicare un insieme di strumenti per lo sviluppo e la documentazione di software [19].

- **Documentazione**: La documentazione inclusa nella SDK é molto dettagliata e offre informazioni precise su ogni singolo package e classe.
- **Tutorial**: Sono guide per la realizzazione di applicazioni basilari, utili per gli sviluppatori alle prime armi.
- **Codice di esempio/dimostrativo**: I tutorial sono accompagnati da applicazioni giá pronte che possono venire usate come modello per costruirne altre.

2.2 Gestione Dispositivi Virtuali

Un Android Virtual Device(AVD)[6] é una configurazione per un emulatore che permette di plasmare un vero e proprio dispositivo attraverso la definizione di opzioni hardware e software per essere emulate dall'emulatore Android. Un AVD é composto da:

- un profilo hardware: definisce le caratteristiche hardware del dispositivo virtuale. Ad esempio, é possibile definire la quantitá di memoria che ha, se il dispositivo ha una fotocamera, se utilizza una tastiera fisica QWERTY o un pad alfanumerico, e cosí via ;
- una mappatura di un'immagine del sistema: é possibile definire quale versione della piattaforma Android verrá eseguita sul dispositivo virtuale. É possibile scegliere una versione della piattaforma standard di Android o l'immagine del sistema fornita con un add-on dell'SDK;
- altre opzioni: é possibile specificare la skin (aspetto grafico) dell'emulatore che si desidera utilizzare con l'AVD, consentendo di controllare le dimensioni dello schermo, l'aspetto, e cosí via. É inoltre possibile specificare la scheda SD emulata da utilizzare con l'AVD;
- un'area di memorizzazione ad hoc sulla macchina di sviluppo: i dati utenti del dispositivo (applicazioni installate, impostazioni, e cosí via) e la scheda SD emulata vengono memorizzate in questa area.

É possibile creare tanti AVD quanti se ne ha bisogno, in base ai tipi di dispositivi che si desidera plasmare. Per testare a fondo un'applicazione, occorre creare un AVD per ogni configurazione del dispositivo (ad esempio, differenti dimensioni dello schermo e diverse versioni della piattaforma) con la quale l'applicazione é compatibile, e testare l'applicazione su ciascun o di essi. Il modo piú semplice per creare un AVD é quello di utilizzare l'interfaccia grafica del AVD Manager[15]. É possibile avviare questo strumento sia da Eclipse cliccando su Window → AVD Manager, sia dalla riga di comando richiamando il tool android con l'opzione avd. In alternativa é possibile creare un AVD dalla riga di comando attraverso l'uso del tool android.

2.2.1 Il tool android

Il tool android[2] é un importante strumento di sviluppo che consente di:

- creare, eliminare e visualizzare Android Virtual Device;
- creare e aggiornare progetti Android;
- aggiornare l'Android SDK con nuove piattaforme, add-on e documentazione.

É possibile trovare il tool android in <sdk-directory>/tools/. Nell'appendice A vengono descritti in maniera dettagliata i vari comandi android e il loro significato e utilizzo.

2.2.2 L'emulatore di Android

L'Android SDK include un emulatore di un dispositivo mobile, un dispositivo mobile virtuale che viene eseguito sulla propria macchina. L'emulatore consente di sviluppare e testare applicazioni Android senza l'utilizzo di un dispositivo fisico. L'emulatore di Android[22] simula tutte le funzioni hardware e software di un dispositivo mobile tipico, con la differenza che non puó

effettuare chiamate telefoniche reali. Esso fornisce una serie di pulsanti di navigazione e di controllo, che si possono "premere" con il mouse o la tastiera per generare eventi per le applicazioni. Esso prevede, inoltre, uno schermo in cui vengono visualizzate le applicazioni Android attive.

Per testare le applicazioni piú facilmente, l'emulatore utilizza configurazioni Android Virtual Device (AVD). Gli AVD consentono di definire alcuni aspetti hardware del dispositivo emulato e permettono di creare diverse configurazioni per testare molte piattaforme Android e permutazioni hardware. Una volta che un'applicazione é in esecuzione sull'emulatore, é possibile utilizzare i servizi della piattaforma Android per richiamare altre applicazioni, accedere alla rete, riprodurre file audio e video, archiviare e recuperare dati, mandare notifiche all'utente, ed eseguire il rendering delle transizioni grafiche e dei temi. L'emulatore include anche una varietá di funzionalitá di debug, come una console da cui é possibile registrare l'output del kernel, simulare gli interrupt di un'applicazione (come la ricezione di SMS o di telefonate), e simulare gli effetti di latenza e caduta di segnale sulla rete dati. Le immagini del sistema Android disponibili tramite l'Android SDK Manager contengono il codice per il kernel Linux di Android, le librerie native, la Dalvik VM, e i vari pacchetti Android (come il framework di Android e le applicazioni preinstallate). L'emulatore fornisce la traduzione binaria dinamica del codice macchina del dispositivo al sistema operativo e all'architettura del processore della propria macchina. L'emulatore di Android supporta molte caratteristiche hardware che possono essere presenti sui dispositivi mobili, tra cui:

- una CPU ARMv5 e la corrispondente unitá di gestione della memoria (MMU);
- un display LCD a 16-bit;
- una o piú tastiere (un a tastiera Qwerty e i pulsanti del telefono associati);
- un chip audio con funzioni di ingresso e di uscita;

- un modem GSM, tra cui una scheda SIM simulata;
- partizioni di memoria flash (emulate attraverso i file di immagine del disco sulla propria macchina);
- una fotocamera, utilizzando una webcam collegata alla propria macchina;
- sensori come l'accelerometro, utilizzando i dati da un dispositivo Android collegato tramite USB.

Sia quando si avvia l'emulatore, che in fase di esecuzione, é possibile utilizzare una varietá di comandi e opzioni per controllare il suo comportamento. Nell'appendice A vengono descritti in maniera dettagliata i vari comandi dell'emulatore[4] e della sua console, e il loro significato e utilizzo.

2.2.3 Android Debug Bridge

Android Debug Bridge (adb)[1] é un versatile strumento a riga di comando che permette di comunicare con un'istanza dell'emulatore o con un dispositivo Android collegato. Si tratta di un programma client-server che comprende tre componenti:

- un client, che viene eseguito sulla macchina. É possibile richiamare un client da una shell mediante l'emissione di un comando adb. Altri strumenti di Android come il plugin ADT e DDMS possono creare client adb;
- un server, che viene eseguito come un processo in background sulla macchina. Il server gestisce la comunicazione tra il client e il demone adb in esecuzione su un emulatore o un dispositivo;
- un demone, che viene eseguito come un processo in background su ogni istanza dell'emulatore o del dispositivo.

Quando si avvia un client adb, il client controlla in primo luogo se vi é un processo server adb giá in esecuzione. Se non c'é, avvia il processo server. Quando il server si avvia, si lega alla porta TCP locale 5037 e attende i comandi inviati dai client adb; tutti i client adb utilizzano la porta 5037 per comunicare con il server adb. Il server imposta quindi le connessioni a tutte le istanze dell'emulatore/dispositivo in esecuzione. Localizza le istanze dell'emulatore/dispositivo scandendo le porte dispari in un range da 5555 a 5587 (che é l'intervallo usato dagli emulatori/dispositivi). Se il server trova un demone adb, imposta una connessione a quella porta. Si noti che ogni istanza dell'emulatore/dispositivo acquisisce una paio di porte in sequenza: una porta con numero pari per le connessioni di console e una porta con numero dispari per le connessioni adb. Per esempio:

- Emulatore 1, console: 5554
- Emulatore 1, adb: 5555
- Emulatore 2, console: 5556
- Emulatore 2, adb: 5557

Come mostrato, l'istanza dell'emulatore collegata ad adb sulla porta 5555, é la stessa istanza la cui console comunica con la porta 5554. Una volta che il server ha impostato le connessioni a tutte le istanze dell'emulatore, é possibile utilizzare i comandi adb per controllare e accedere a tali istanze. Poiché il server gestisce le connessioni a istanze dell'emulatore/dispositivo e gestisce i comandi da piú client adb, é possibile controllare qualsiasi istanza dell'emulatore/dispositivo da qualsiasi client (o da uno script). É possibile trovare lo strumento adb in <sdk-directory>/platform-tools/.

2.3 Eclipse e Android Development Tools

Android Development Tools (ADT)[3] é un plugin per Eclipse[3] che é stato progettato per fornire un potente ambiente integrato in cui costruire le applicazioni Android. ADT estende le capacitá di Eclipse e consente di configurare rapidamente nuovi progetti Android, creare un'interfaccia utente dell'applicazione, aggiungere i pacchetti basati sulle Framework API di Android, eseguire il debug delle applicazioni utilizzando gli strumenti dell'Android SDK, e persino esportare file ".apk" al fine di distribuire l'applicazione. Molti degli strumenti dell'Android SDK che é possibile avviare o eseguire dalla riga di comando sono integrati nei menú e nelle prospettive di Eclipse, o come parte dei processi in background eseguiti da ADT. Tra questi abbiamo:

- Traceview: consente di profilare l'esecuzione del programma;
- Hierarchy Viewer: consente di visualizzare la vista gerarchica dell'applicazione per trovare le inefficienze;
- Perfect Pixel: consente di esaminare da vicino l'interfaccia utente per aiutare con la progettazione e la realizzazione;
- android: fornisce l'accesso all'Android SDK Manager e all'AVD Manager;
- DDMS: fornisce funzionalitá di debug, tra cui cattura dello schermo, informazioni su thread e heap, e logcat;
- adb: fornisce l'accesso a un dispositivo dalla propria macchina;
- ProGuard: permette l'offuscamento, la riduzione e l'ottimizzazione del codice.

[3]Eclipse é un ambiente di sviluppo integrato (IDE) multi-linguaggio e multipiattaforma [8]

2.4 Android Native Development Kit

L'Android Native Development Kit (NDK)[5] é un set di strumenti che consente di incorporare i componenti che fanno uso di codice nativo nelle applicazioni Android. Le applicazioni Android vengono eseguite nella macchina virtuale Dalvik. L'NDK consente di implementare parti delle applicazioni utilizzando linguaggi a codice nativo quali C e C++. Questo puó fornire benefici per certe classi di applicazioni, in forma di riutilizzo di codice esistente e in alcuni casi di una maggiore velocitá. L'NDK fornisce:

- un insieme di strumenti e file utilizzati per generare librerie di codice nativo da sorgenti C e C++;
- un modo per incorporare le relative librerie native in un pacchetto applicazione (.apk) che puó essere distribuito su dispositivi Android;
- un insieme di librerie native che saranno supportate in tutte le versioni future della piattaforma Android, a partire da Android 1.5;
- documentazione, esempi e tutorial.

2.5 Samsung Galaxy S

Il primo dispositivo utilizzato per eseguire i test di customizzazione di Android é un Samsung Galaxy S (GT-i9000). Si tratta di uno smartphone prodotto da Samsung annunciato nel mese di marzo 2010. Esteticamente si presenta piuttosto simile al rivale Apple iPhone, ma il dispositivo coreano ha un'identitá tutta sua, a partire dal sistema operativo Android 2.1 (aggiornabile alla verisone 2.3.3 o anche al nuovo arrivato Android 4.0 Ice Cream Sandwich se si ricorre a firmware modificati) ed al display full-touch capacitivo da 4" Super AMO-LED con una risoluzione di 800x480 pixels, 233ppi e 16 milioni di colori. Lo smartphone é inoltre dotato di un processore ARM Cortex A8 da 1GHz, 512MB di RAM, memoria interna da 8GB espandibile

con microSD fino a 32GB, e fotocamera da 5Mpx con autofocus che permette di girare video in alta definizione fino a 720p.

Figura 2.1: Samsung Galaxy S e Samsung Galaxy Ace.

2.6 Samsung Galaxy Ace

Il secondo dispositivo utilizzato per eseguire i test di customizzazione di Android é un Samsung Galaxy Ace (GT-S5830), chiamato piú comunemente Galaxy Ace, é uno smartphone prodotto da Samsung, facente parte della serie Samsung Galaxy e basato sul sistema operativo Android e annunciato e venduto a partire da febbraio 2011. É dotato di connettivitá completa con reti HSDPA 7.2Mbps 900/2100, EDGE/GPRS 850/900/1800/1900, Wi-Fi a/b/g/n, bluetooth 2.1. Monta un processore single core da 800Mhz e 278MB di RAM. Il display é un touchscreen 3,5″ LCD capacitivo TFT con risoluzione HVGA (480x320 pixels). Ha una videocamera da 5-megapixel con flash LED, capace di registrare video alla risoluzione VGA (640x480). La versione di

Android montata nativamente é 2.2 e utilizza l'interfacia grafica proprietaria TouchWiz di Samsung. Ace ha anche un router, GPS, accelerometro, sensore di prossimitá ma non ha il sensore di luminositá.

2.7 Samsung Kies e Odin

Samsung Kies[17] é un software scaricabile dal sito ufficiale della Samsung che permette di fare molteplici cose:

- creare playlist musicali;
- sincronizzare musica, foto e video;
- sincronizzare i contatti con quelli di Outlook, Google o Yahoo;
- acquistare e sincronizzare applicazioni;
- salvare video e foto scattate;
- aggiornare il firmware del dispositivo;
- eseguire il backup e il ripristino dei dati.

Kies permette solamente l'aggiornamento dei firmware ufficiali. Per poter installare firmware e kernel personalizzati (ma anche ufficiali), é necessari o l'utilizzo di un software differente: Odin. Si tratta di un programma che permette di flashare[4] firmware, kernel, modem, bootloader ed altro ancora su dispositivi Samsung.

[4]É il processo si sovrascrittura dei dati esistenti sui moduli ROM presenti in dispositivi elettronici con dei nuovi dati.

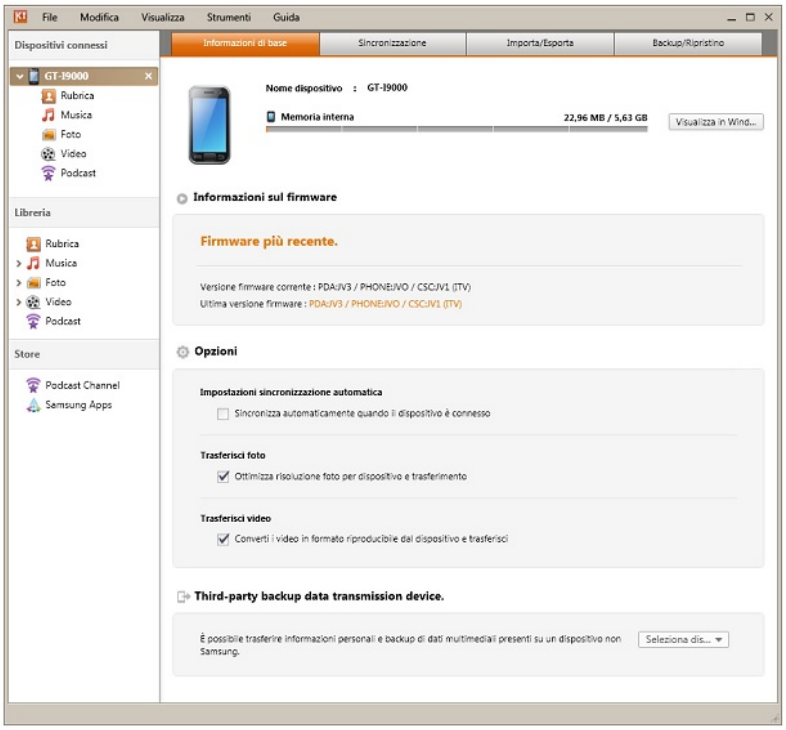

Figura 2.2: Samsung Kies.

Capitolo 3

Customizzazione di Android

In questo capitolo verrá descritto come customizzare (personalizzare) Android, partendo dalla compilazione dei suoi file sorgente, per poi passare all'installazione del kernel sull'emulatore e sul Samsung Galaxy S, e ai vari test fatti su di essi.

Come prima cosa si parlerá degli strumenti che serviranno in corso d'opera per poter utilizzare al meglio queste tecnologie e poterle sviluppare. Innanzi tutto si parlerá dei sistemi operativi utilizzati come ambiente di sviluppo: si é scelto di utilizzare windows 7 e windows XP come sistemi operativi di riferimento per poter fare determinate azioni, quali installazioni di software utile per i driver degli smartphone, "flash" di kernel sui cellulari ecc..., inoltre si é lavorato su sistemi operativi Unix, quali Xubuntu e Ubuntu 12.04 per la manipolazioni e compilazioni dei sorgenti dei kernel. I primi due sistemi operativi erano la base di partenza su cui si é virtualizzato, attraverso l'uso di Oracle VirtualBox[23], i rispettivi sistemi Unix.

3.1 Installazione Software

Altri software richiesti per proseguire nel lavoro verranno elencati qui di seguito riportando quanto piú fedelmente possibile tutti i passi per l'instal-

lazione e l'utilizzo.

JDK Java, versione 6 o superiore Si é scelto la versione 7 di java poiché si é riscontrato che alcuni repository in cui doveva essere presente la versione di java 6 non era disponibile. Ecco i comandi eseguiti da shell:

NOTA: é possibile anche installare openjava-6-sdk.

- sudo add-apt-repository ″deb http://archive.canonical.com/ lucid partner″
- sudo add-apt-repository ppa:webupd8team/java
- sudo apt-get update
- sudo apt-get install oracle-jdk7-installer

Un volta installato il tutto é possibile verificare che é andato tutto bene eseguendo il comando :

- java -version

In caso in cui non si voglia installare il tutto da riga di comando, é possibile collegarsi al sito della oracle e installare manualmente la versione di java JDK[10] desiderata.

NOTA: nel caso ci fossero problemi con il comando "sudo add-apt-repository" é possibile aggiungere la repository attraverso il seguente percorso: Ubuntu Software → Modifica Sorgenti Software → Altro Software e infine segnare "Partner di Canonical" e "Partner di Canonical (Codice Sorgente)".

Eclipse SDK Installare un versione adeguata di Eclipse SDK dal sito ufficiale:

- http://www.eclipse.org/

Si tratta di uno strumento utile per poter emulare velocemente uno smartphone senza averlo a disposizione realmente. La versione installata utilizzata é stata la Classic 4.2.

Installazione pacchetti necessari Sono librerie e pacchetti linux necessari per la compilazione di kernel android. In particolare bisogna far attenzione al fatto che ogni versione di ubuntu ha i suoi pacchetti e che in caso di problemi bisogna cercarli manualmente in rete. Ecco tipo i pacchetti per la versione di Ubuntu 10:

- sudo apt-get install git-core gnupg flex bison gperf build-essential zip curl zlib1g-dev libc6-dev lib32ncurses5-dev ia32-libs x11proto-core-dev libx11-dev lib32readline5-dev lib32z-dev libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown libxml2-utils xsltproc valgrind libsdl-dev libesd0-dev libwxgtk2.6-dev libglade2-dev

Mentre per la versione di Ubuntu 12.04 :

- sudo apt-get install git-core gnupg flex bison gperf build-essential zip curl zlib1g-dev libc6-dev libncurses5-dev:i386 ia32-libs-multiarch x11proto-core-dev libx11-dev libreadline6-dev zlib1g-dev libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown libxml2-utils xsltproc valgrind libsdl-dev libesd0-dev libwxgtk2.6-dev libglade2-dev

NOTA: in caso di problemi con alcune librerie, scaricarle e installarle manualmente dai siti relativi: lib32ncurses5-dev[12],lib32readline5-dev[13],lib32z-dev[14].

In aggiunta é possibile installare il pacchetto “libqt3-mt-dev” per poter configurare il kernel da interfaccia grafica, utilizzando il comando “make xconfig”.

NOTA: per installare le librerie manualmente lanciare i seguenti comandi in sequenza :

- *./configure*
- *make*
- *make install*

Compilatore toolchain per l'emulatore: Android NDK Si tratta del compilatore di kernel android che é disponibile su sito di Android Developers e vá scompattato nella home directory:

- http://developer.android.com/tools/sdk/ndk/index.html

NOTA: all'interno ci sono varie versioni di cross-compiler[21] (o toolchain) per i kernel, infatti una serve per compilare i kernel per l'emulatore, un'altra per gli smartphone.

Compilatore toolchain per smartphone: Sourcery G++ Lite Si tratta del compilatore del kernel android utilizzato per lo smartphone, disponibile al seguente indirizzo :

- http://www.codesourcery.com/sgpp/lite/arm/portal/package5355/public/arm-none-eabi/arm-2009q3-68-arm-none-eabi.bin

Per installare il file seguire le seguenti istruzioni:

- chmod +x arm-2009q3-68-arm-none-eabi.bin
- ./ arm-2009q3-68-arm-none-eabi.bin

NOTA: non provare a scaricare versione del cross-compiler con versione differente alla 2009q3-68. Si hanno errori in fase di compilazione.

Plugin ADT di Eclipse e Android SDK Si tratta di alcuni pacchetti di plugin per Eclipse che permettono di emulare un dispositivo telefonico sul computer e insieme a questi verrá richiesto anche di installare i pacchetti di Android SDK che verranno utilizzati servono. Ecco i passi da seguire:

- dal menú a tendina di Eclipse selezionare "Help" → "Install New Software";
- cliccare su "Add" nell'angolo in alto a destra;
- nella finestra di dialogo Add Repository, inserire "ADT Plugin" nel campo Name e l'URL "https://dl-ssl.google.com/android/eclipse/" nel campo Location;
- cliccare su OK.

NOTA: in caso di problemi nell'acquisizione del plugin, provare ad usare "http" invece di "https" nell'URL del campo Location;

- nella finestra di dialogo Available Software, selezionare il checkbox accanto a Developer Tools e cliccare su Next;
- nella finestra successiva verrá visualizzato un elenco degli strumenti da scaricare. Cliccare su Next;
- leggere e accettare gli accordi di licenza, quindi cliccare su Finish.

NOTA: se si ricevesse un avviso di sicurezza dicendo che l'autenticitá e la validitá del software non puó essere stabilita, cliccare su OK;

- al termine dell'installazione, riavviare Eclipse.

Dopo aver installato ADT e riavviato Eclipse, é necessario specificare il percorso della directory dell'SDK di Android:

- dal menú a tendina selezionare Window → Preferences;

- selezionare Android dal pannello di sinistra;
- cliccare su Browse dal campo SDK Location e individuare la directory relativa all'SDK;
- cliccare su Apply, quindi OK.

Infine bisogna installare alcuni pacchetti accessori per poter avviare l'emulatore:

- dal menú a tendina selezionare Window → Android SDK Manager;
- dal menú controllare che siano installare tutte le voci Tools, altrimenti installarle;
- controllare che siano installati dalla cartella Android 4.1(API 16) i pacchetti SDK Platform e ARM EABI v7a System Image, altrimenti farlo.

Samsung Kies Si tratta di un programma di gestione dei driver, aggiornamento e backup dei dati di telefoni samsung. É utilizzato per installare i driver di riconoscimento dei telefoni a disposizione e per fare backup dei dati presenti sui nostri terminali. É disponibile solo per windows e MacOs:

- http://www.samsung.com/it/support/usefulsoftware/KIES/JSP

Oltre a questo c'é anche disponibile una guida dettagliata sull'utilizzo:

- http://www.samsung.com/it/support/usefulsoftware/KIESTUTORIAL/JSP

Odin Si tratta di un programma utile a "flashare" o per meglio dire installare kernel android originali e non all'interno di un cellulare Samsung. Le versioni di Odin[16] sono molteplici e ognuna ha un elenco dettagliato dei cellulari samsung che riesce a riconoscere o meno. Per evitare ogni problema con le versione, scaricare la 3.04 (o 1.85) che riconosce la maggior parte dei cellulari samsung.

3.2 Compilazione Kernel

Una volta installati i software si passa alla fase successiva, ovvero come compilare un kernel personalizzato e utilizzarlo per l'emulatore e i cellulari. Innanzitutto preparare un kernel per l'emulatore e un kernel per un cellulare sono due cose diverse, infatti richiedono giá in partenza sorgenti differenti e passi diversi. Andiamo quindi ad elencare nel dettaglio prima i passi per un kernel funzionante sull'emulatore e poi per il cellulare.

3.2.1 Kernel per l'emulatore

Come prima cosa bisogna scaricare i sorgenti del kernel android:

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

Una volta scaricati i sorgenti si potrá entrare nella cartella goldfish presente nella home e al suo interno controllare personalmente quali configurazioni del kernel esistono giá, in particolare il percorso é:

- goldfish/arch/arm/configs

In questa cartella sono situate un elenco delle piú disparate configurazioni utilizzate e tra queste, quella che é stata testata e funzionante é la "goldfish_armv7_defconfig". Ecco il comando da eseguire per poter lasciare la configurazione desiderata:

- make ARCH=arm goldfish_armv7_defconfig

In caso si voglia una configurazione manuale, il comando da eseguire é il seguente:

- make ARCH=arm gconfig (oppure xconfig o menuconfig per l'interfaccia grafica)

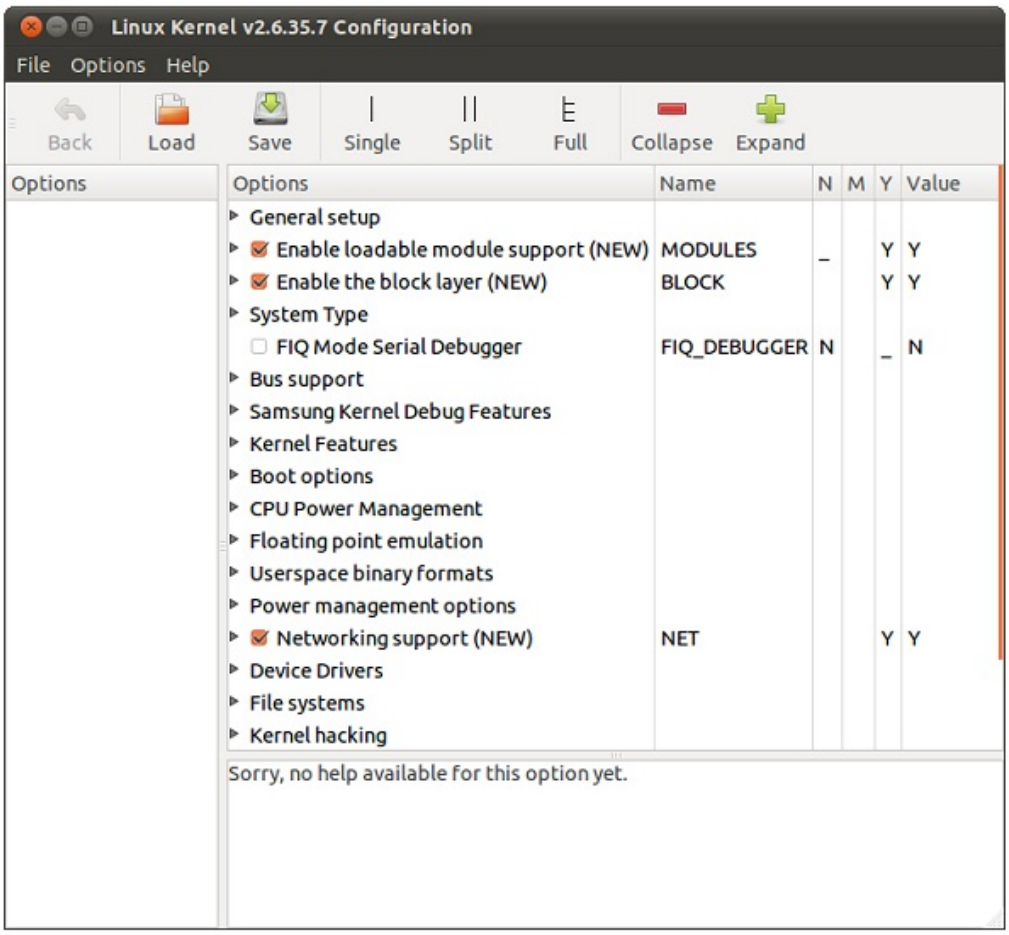

Figura 3.1: Schermata di Configurazione del Kernel.

NOTA: Ricordarsi di copiare la configurazione che si é scelto da goldfish/arch/arm/configs in goldfish/arch/x86/configs altrimenti non compila.

Una volta scelta la configurazione si danno i comandi per compilare il kernel:

- make ARCH=arm SUBARCH=arm CROSS_COMPILE= /<PATH Android NDK>/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86/bin/arm-linux-androideabi- -j4

L'ultimo argomento -j4 indica il numero di thread che puó invocare simultaneamente la compilazione, in ogni caso il sistema gestisce da solo quel numero in caso di numeri che non rispecchiano la realtá. Una volta lanciato il comando ci vorrá circa una mezz'ora di compilazione con un computer dual core, con computer piú potenti il tempo diminuisce drasticamente. In ogni caso, alla fine di tutto, si avrá un messaggio come questo ad indicare che la compilazione é andata a buon fine:

- arch/arm/boot/zImage is ready

In tutti gli altri casi qualcosa non é andato per il verso giusto e nei vari test era sempre a causa di una configurazione sbagliata, vedasi configurazione goldfish_defconfig di quelle giá presenti, oppure di una configurazione manuale errata. Una volta compilato nel modo giusto, nella cartella seguente sará presente un file di nome zImage che é il nostro kernel da utilizzare nell'emulatore:

- goldfish/arch/arm/boot/

Fatto questo abbiamo un kernel da poter lanciare con un istanza dell'emulatore (AVD).

NOTA: prima di ogni ricompilazione eseguire il comando "make mrproper" per pulire tutto.

3.2.2 Creazione AVD (Android Virtual Device)

Per poter testare il funzionamento di un kernel é necessario prima creare un AVD. Questo AVD é una configurazione del dispositivo per l'emulatore di Android che consente di modellare diverse configurazioni di dispositivi Android. Per poter creare un AVD ci sono due strade da poter seguire, ovvero utilizzare l'AVD Manager installato su Eclipse, oppure lanciarlo da riga di comando attraverso il tool android.

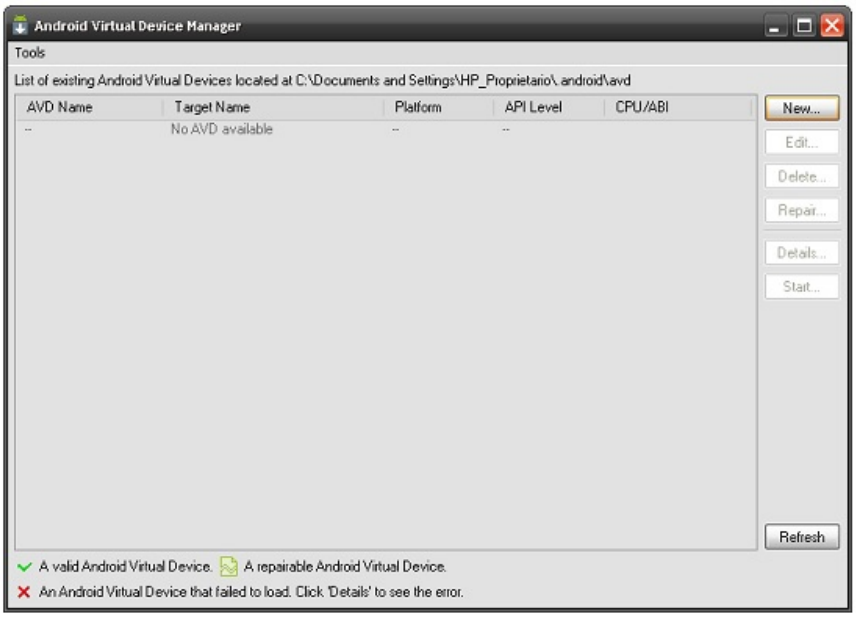

Figura 3.2: Schermata di Odin durante il flashing del kernel.

Per creare un AVD:

1. avviare l'AVD Manager:

   - in Eclipse: selezionare Window → AVD Manager, oppure cliccare sull'icona di AVD Manager nella barra degli strumenti di Eclipse;
   - in altri IDE: navigare fino alla directory tools/ dell'SDK ed eseguire il tool android con argomento avd;

2. nel pannello Virtual Devices, é possibile vedere un elenco degli AVD esistenti. Cliccare su New per creare un nuovo AVD. Apparirá la finestra di dialogo Create New AVD

3. compilare i dettagli per l'AVD:

   - inserire un nome;
   - target = Android 4.1 - API Level 16;
   - CPU/ABI = ARM (Armeabi-v7a);
   - una dimensione per la scheda SD;
   - una skin (HVGA é di default).

4. cliccare su Create AVD.

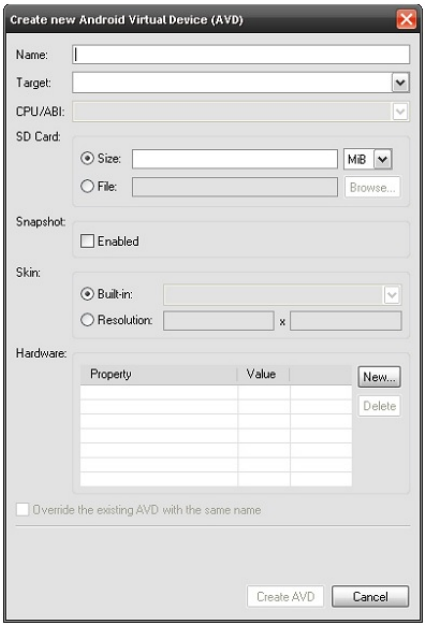

Figura 3.3: Schermata di Odin durante il flashing del kernel.

É inoltre possibile aggiungere funzionalitá hardware specifiche del dispositivo emulato facendo clic sul pulsante "New" e selezionando la funzionalitá.

L'AVD é ora pronto ed é possibile chiudere l'AVD Manager, creare altri AVD, o lanciare un emulatore con l'AVD appena creato, selezionando un dispositivo e cliccando su Start. Per lanciare invece AVD da linea di comando basta andare nella cartella dell'android SDK → tools e lanciare l'AVD in questo modo:

- ./emulator -avd <NOME AVD> -kernel <PATH dello zImage>

In entrambi i casi a video apparirá una finestra che rappresenta lo smartphone precedente creato.

NOTA : In caso di utilizzo di VirtualBox per emulare il sistema Unix é obbligatorio disabilitare l'accelerazione 3D affinché l'AVD parta, altrimenti questo va in loop infinito

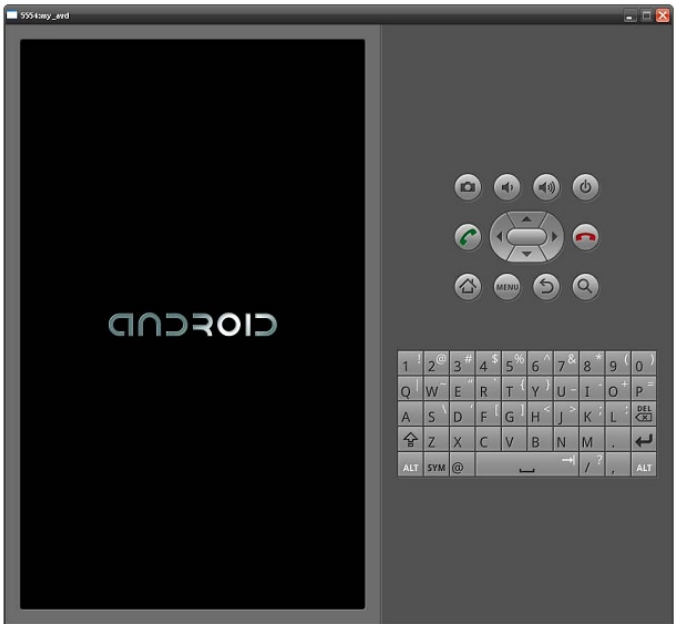

Figura 3.4: Schermata di Caricamento Di Android.

3.2.3 Kernel per lo smartphone

Come prima cosa bisogna scaricare i sorgenti del kernel GingerBread dal sito della Samsung Open Source Center[18]. Nella barra di ricerca inserire la stringa del dispositivo che ci interessa, in questo caso il GT-I9000 per il Samsung Galaxy S (GT-S5830I per il Samsung Ace). Scegliere il seguente pacchetto:

- GT-I9000_Opensource_GB_Update2.zip

Scompattare l'archivio in una qualsiasi directory ed eseguire i seguenti passi:

- Scompattare l'archivio contenente i sorgenti del kernel
- Scompattare l'archivio contenente l'initramfs[9] in /usr rinominando la cartella in initrd_files
- Copiare l'archivio prebuilt_ko.zip in /usr
- Copiare buildgalaxy.sh e build.sh nella directory root del kernel
- Modificare in arm/arch/configs, aries_eur_defconfig aggiungendo:
  - CONFIG_ INITRAMFS_SOURCE = usr/initrd_file
  - CONFIG_ INITRAMFS_ROOT_UID=500
  - CONFIG_ INITRAMFS_ROOT_GID=500
- Modificare il flag in build.sh inserendo il path del cross-compiler ,come "CROSS_COMPILE ?= <Path CodeSourcery> /Sourcery_G++_Lite/bin/arm-none-eabi-"
- Lanciare ./buildgalaxy.sh

Anche qui le tempistiche variano a seconda della velocitá del computer, in ogni caso, una volta terminata la compilazione, se é andato tutto bene, apparirá il seguente messaggio:

- arch/arm/boot/zImage is ready

L'immagine risultante sará circa 5.7 MB. La grandezza puó variare a seconda della configurazione scelta in quanto produce altri moduli ".ko" che vengono inseriti nell'initramfs.

NOTA 1: Ricordarsi che in caso si debba ricompilare, prima di tutto, bisogna eseguire il comando "make mrproper" per pulire tutto.

NOTA 2: qualora si scegliesse un initramfs diverso da quello fornito assicurarsi che sia compatibile con il kernel gingerbread.
Alcuni esempi di versioni compatibili:

- *XXJVP*
- *XXJVO*
- *XXJVK*
- *XWJVH*
- *XWJVB*

Anche se dopo diversi test,la scelta dell'initramfs é risultata irrilevante. La dimensione degli initramfs puó variare da 2.4 MB a 4.1 MB circa,a seconda dei moduli precompilati all'interno. Cosa importante é assicurarsi che il file default.prop all'interno sia composto nel seguente modo:

- ro.secure=0
- ro.allow.mock.location=0
- ro.debuggable=0
- persist.service.adb.enable=1

e il file recovery.rc abbia le sequenti stringhe non commentate:

- on property:persist.service.adb.enable=1
- start adbd
- on property:persist.service.adb.enable=0
- stop adbd

Fatto questo, per poterlo flashare sul nostro Samsung, bisogna spostarsi su un sistema Windows e utilizzare Odin.

Flash di un kernel con Odin Prima di lasciare il nostro sistema Unix peró, dobbiamo fare un ultimo passaggio, ovvero trasformare il file zImage in un archivio tar poiché Odin riconosce kernel solo in quel tipo di formato. Quindi da terminale:

- cd <kernel-directory>/arch/arm/boot/
- tar -cvf custom-kernel.tar zImage

Copiare quindi il file tar sul sistema windows e avviare Odin.
Una volta avviato Odin seguire questi passi:

- spegnere lo smartphone.
- accenderlo in modalitá Download attraverso la pressione dei tasti in sequenza : Volume Giú + tasto Home + Accensione.
- collegarlo tramite cavo usb al computer.
- premere sul bottone PDA e si aprirá una finestra da dove prendere il nostro kernel.
- a questo punto premere sul bottone start, aspettare qualche secondo che l'installazione del kernel termini.

- Staccare il terminale sono quando verrá visualizzata il messaggio “removed”. A quel punto staccare il cavo usb e aspettare qualche secondo l’accensione del Samsung S.

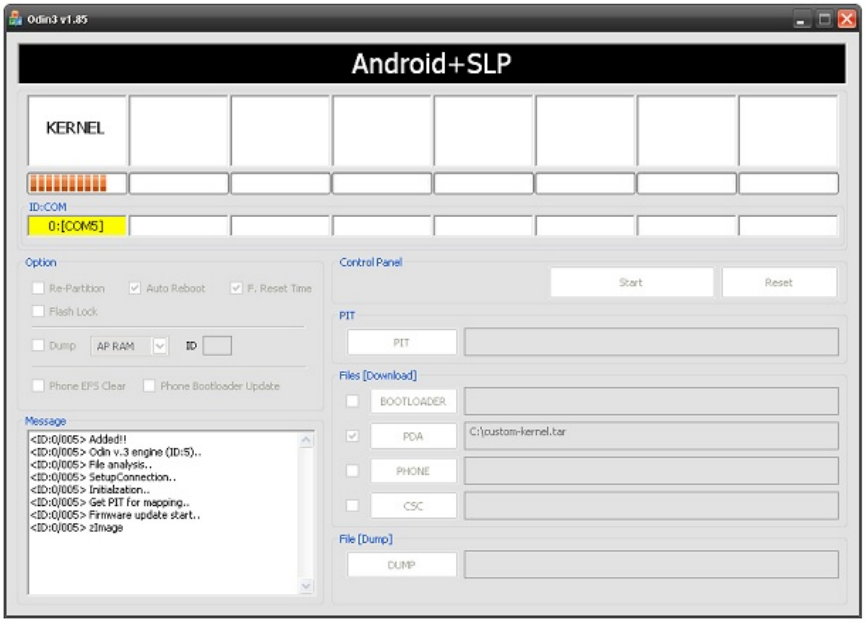

Figura 3.5: Schermata di Odin durante il flashing del kernel.

3.3 Test e Problemi

Una volta seguiti questi passi si ha a disposizione un cellulare funzionante come da obiettivo iniziale. Chiaramente per arrivare a questo risultato siamo dovuti passare per situazioni spiacevoli e problemi vari, ed ecco quindi un sunto delle strade esplorate e le fasi di sviluppo attraversate per arrivare ad un risultato:

- Guida iniziale fallace

- Impossibile estrarre la configurazione del dispositivo attraverso il comando “adb pull /proc/config.gz” in quanto non esiste nessuna directory “proc” e nessun file “config.gz”.
- Compilazione del kernel con il metodo standard da Reame dei sorgenti, risultato é stato un telefono in loop sulla schermata Samsung iniziale.
- Diverse versioni di GingerBread dal Samsung Open Source Center
- Diverse configurazioni di kernel tra quelle messe a disposizione dai sorgenti.
- Diverse configurazioni manuali, anche le piú assurde.
- Installazione kernel funzionante su emulatore, ma si ottiene lo stesso risultato.
- Installazione di kernel sull’emulatore per fare test e debug.
- Ricerca di kernel funzionanti online e paragone con i nostri per capire cosa cambiava tra i due.
- Modifica della configurazione per compilare i nostri sorgenti con initramfs di kernel funzionanti.
- Utilizzo di Initramfs diversi compilati con il nostro kernel.
- Modifica dei file di default contenuti nell’initramfs (default.prop,recovery.rc).
- Utilizzo di due file bash per la compilazione dei sorgenti.

Dopo i primi test effettuati ci si é accorti che il problema non era nelle configurazioni diverse, ma che il problema del loop poteva essere causa di un kernel che non sapeva come proseguire la sua fase di boot una volta accesso. Quindi ci si é concentrati su una parte particolare del kernel android, ovvero l’initramfs. Il passo successivo é stato capire di cosa si trattasse, come modificarlo e come utilizzarlo in fase di compilazione. Il risultato é stato che il

dispositivo, invece di andare in loop, si accendeva e restava fermo alla scritta samsung iniziale. A quel punto, anche provando a compilare i sorgenti con initramfs scompattati da kernel funzionanti non ci davano nessuna nuova prospettiva, fino a che non abbiamo provato con un kernel funzionante, il cui nome é Talon[20], una versione del kernel android GingerBread modificata. Questo kernel é stata la svolta, poiché esaminandolo si é capito che il problema grosso era nella compilazione dei sorgenti, ma soprattutto nella mancanza di alcuni moduli di partenza. Durante la compilazione di questo Talon occorrevano ben 40-45 minuti rispetto agli usuali 30 minuti per la compilazione e durante questa fase utilizzava moduli e file sconosciuti. In ogni caso si é aspettato la fine della compilazione e la successiva fase di flashatura sul dispositivo, il risultato é stato che comunque restava bloccato sulla scritta "Talon" iniziale. Quindi si é andati nel dettaglio. Ecco cosa variava rispetto alla procedura normale:

- Una configurazione personalizzata.
- In fase di compilazione inseriva nell'initramfs ulteriori moduli precompilati contenuti in prebuilt_ko.zip.
- Esecuzione del makefile in drivers/misc/samsung_modelctl.
- Con uno smartphone Samsung galaxy Ace di nuova generazione non é possibile flashare kernel poiché Odin ritorna sempre un errore FAIL!

Provando a compilare i file sorgenti con i moduli precompilati di Talon, il suo initramfs e la sua configurazione, il risultato é stato che Odin dava il risultato FAIL! durante il flash. Lo stesso risultato é stato ottenuto quando l'initramfs era troppo grande (circa 4.5 MB) o incorretto. Cambiando quindi ancora una volta,e compilando i file sorgenti, l'initramfs, la configurazione di Talon,i moduli precompilati e utilizzando i due file bash modificati, finalmente si é arrivati al risultato sperato avendo a disposizione una zImage funzionante di 6.2 MB. Non contenti di questo risultato, per non utilizzare la configurazione di Talon, si é modificata la configurazione standard dei

sorgenti sorgenti cioé la “aries_eur_defconfig” arrivando finalmente ad una zImage piú snella di 5.7 MB.

NOTA: la differenza tra le due configurazione,quindi nelle zImage risultanti, risiede nelle quantitá di moduli del kernel compilati e inseriti nell'initramfs.

3.3.1 Script Bash utilizzati per la compilazione

I file bash utilizzati sono stati estratti dal kernel Talon e in seguito modificati. Successivamente sono stati effettuati dei test per eliminare “codice spazzatura” ma soprattutto scoprire i punti cardini nel file build.sh:

- É possibile modificare il flag KBUILD_BUILD_VERSION a piacere;
- É possibile modificare il flag LOCAL_VERSION solo se si utilizza una versione differente dei sorgenti;
- L'esecuzione del kbuilt makefile[11] in drivers/misc/samsung_modelctl é essenziale e non puó essere rimosso;
- La copia dei moduli precompilati nell'initramfs é essenziale e non puó essere rimossa.

Il file buildgalaxy.sh richiama a sua volta il file build.sh dando come parametro la configurazione da utilizzare, nel nostro caso la “aries_eur_defconfig”. Non sono stati effettuati test su altre configurazioni.

Capitolo 4

Scenario

In questo capitolo si vuole presentare un'architettura per il supporto alla mobilitá in sistemi multihomed eterogeni. Per *sistemi multihomed eterogenei* si intende dispositivi dotati di piú interfacce di rete, come ad esempio Wi-Fi, WiMAX o UMTS, tecnologie che al giorno d'oggi sono sempre piú spesso integrate in un unico dispositivo come uno smartphone o un portatile. Queste tecnologie vengono usate nelle comunicazioni Internet per accedere ai piú disparati servizi, ad esempio applicazioni multimediali, navigazione web e comunicazione vocale e video. Internet é una rete decentralizzata che si basa su una multitudine di meccanismi e protocolli, questi devono essere standardizzati e regolati in modo che ogni nodo nella rete possa comunicare l'un l'altro senza le limitazioni dovute all'eterogeneitá dei dispositivi presenti. Qui di seguito viene presentato il modello ISO/OSI [33]: uno standard che stabilisce un modello di architettura composto da livelli. Ogni livello si occupa di uno specifico aspetto delle comunicazioni, fornisce delle funzionalitá al livello superiore e usa le astrazioni del livello inferiore. In questo modo é possibile ridurre la complessitá non banale delle comunicazioni di rete. Verranno mostrati i protocolli che riguardano piú da vicino il lavoro di questa tesi ed introdotti i sistemi che sono direttamente coinvolti nell'architettura proposta.

4.1 Livello fisico e data-link

4.1.1 Wi-Fi

Oggi, in Italia e nel mondo, girando per le cittá é possibile avere accesso a reti wireless protette e non. Grazie a questo e mediante alcuni accorgimenti é possibile rimanere sempre connessi alla rete. Una rete LAN wireless (WLAN - wireless local area network) consiste in una rete formata da due o piú dispositivi che, grazie a tecnologie di trasmissione radio, possono comunicare tra loro senza l'utilizzo di cavi. La tecnologia permette comunicazioni in un'area limitata, consentendo mobilitá all'interno di essa senza perdere l'accesso alla rete. I vantaggi sono ampi e nei piú disparati contesti. In ambito privato, le reti senza fili hanno avuto un massivo utilizzo grazie alla facilitá di installazione e alla crescente diffusione di portatili e palmari. Alcuni tipi di esercizi come coffee-shop o centri commerciali hanno iniziato ad offrire questo tipo di servizio ai clienti. Nelle cittá sono nati progetti per piccole reti pubbliche, per esempio quelle di biblioteche e universitá, oppure grandi reti civiche che coprono le vie del centro. La tecnologia di gran lunga piú diffusa per la creazione di reti wireless é senza dubbio quella Wi-Fi.

4.1.2 IEEE 802.11

Essa é basata sulla famiglia di standard IEEE 802.11, che specifica un insieme di regole da seguire per il livello fisico e data link del modello ISO/OSI per permettere l'interoperabilitá tra i dispositivi dei diversi produttori esistenti. Come indicato dal numero, IEEE 802.11 si adegua perfettamente agli altri standard 802.x per reti locali wired e le applicazioni che lo utilizzano non dovrebbero notare nessuna differenza logica, una degradazione delle performace invece é possibile. All'interno della famiglia, i protocolli dedicati alla trasmissione delle informazioni sono a, b, g e n. Gli altri standard riguardano estensioni dei servizi base e miglioramenti di servizi giá disponibili. Il primo protocollo largamente diffuso é stato lo b; in seguito si sono diffusi il

protocollo a e soprattutto il protocollo g. Recentemente, con il protocollo n, le performace teoriche sono state notevolmente aumentate da 54 Mb/s a 600 Mb/s [27] grazie all'uso della teconologia MIMO (Multiple-Input Multiple-Output). Gli altri standard della famiglia riguardano estensioni dei servizi base e miglioramenti di servizi giá disponibili.

4.1.3 Architettura

Le reti wireless possono usare due diverse architetture: ad-hoc, nella quale i nodi mobili (Mobile Node, MN) comunicano direttamente tra loro, o basata su infrastruttura, dove ogni nodo comunica con una stazione radio base (Base Station, BS), detta comunemente access point (AP), questa stazione si occupa di instradare i dati verso il destinatario finale direttamente se é associato alla stessa Base Station, oppure ad altri Access Point nel caso di una rete cablata o mesh. Quando un nodo mobile cambia posizione allontanantosi dalla compertura dalla propria Base Station e avvicinandosi ad un'altra, avviene il cosiddetto roaming, con la quale il nodo mobile viene associato alla nuova stazione. Poiché questo avviene a livello 2 del modello ISO/OSI, l'indirizzo IP rimane lo stesso cosí che le connessioni presenti in quel nodo restano intatte.

4.1.4 Livello fisico

Lo standard 802.11 utilizza le bande di frequenza 2.4, 3.6 e 5 GHz, scelte per la caratteristica di essere liberamente disponibili nella maggior parte del del mondo. Come tecniche di modulazione, sono usate la Orthogonal Frequency-Division Multiplexing, che utilizza un numero elevato di sottoportanti ortogonali tra di loro, o quella chiamata Direct Sequence Spread Spectrum, che é una tecnologia di trasmissione a banda larga nella quale ogni bit viene trasmesso come una sequenza ridondante di valori, detti chip, tali per cui una ricevente puó ricostruire il segnale al quale é interessata.

4.1.5 Livello MAC

I compiti principali del livello MAC (Media Access Control) [26] sono regolamentare l'accesso al medium, frammentare i dati ed applicare la crittografia, altri incarichi sono controllare i meccanismi di accesso, la sincronizzazione degli access point e gestire le modalitá di risparmio di energia.

Accesso al canale Solo un nodo alla volta puó trasmettere all'interno del proprio raggio d'azione. Una collisione avviene se un dispositivo riceve contemporanemanete trasmissioni (detti frame) da due nodi diversi, in questo caso i dati inviati sono confusi tra loro e il ricevente non riesce ad utilizzarli. L'accesso al canale é regolato dal protocollo CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), una modifica del CSMA/CD (CSMA/Collision Detection), che non puó essere usato nelle reti senza fili a causa della difficoltá di realizzazione di un apparato di ricetrasmissione che possa contemporaneamente trasmettere ed ascoltare sullo stesso canale radio. Prima di trasmettere, uno nodo si mette in ascolto, se il canale é libero per almeno un certo periodo (detto DIFS), il nodo puó trasmettere. Se il canale é occupato, il nodo sceglie un intervallo di tempo casuale (detto random backoff) per il quale attendere e poi riprovare. Il nodo ricevente ascolta il dato, se il dato viene ricevuto senza errori attende un intervallo fisso di tempo (SIFS, piú piccolo del DIFS, quindi a maggiore prioritá) per rispedire al mittente un ACK di consegna correttamente avvenuta. Se nessun ACK é stato ricevuto, il dato viene ritrasmesso per un certo numero di volte, il mittente comunque deve competere di nuovo per l'accesso al medium come tutti gli altri. Il meccanismo, cosí descritto, non consente ad ogni nodo di accedere al canale in modo equo, visto che un nodo che sta trasmettendo in un dato momento ha la stessa probabilitá degli altri di accedere al canale subito dopo, IEEE 802.11 supera questo problema in quanto un nodo sceglie un random backoff soltanto al primo tentativo di accesso al canale per poi decrementare il tempo di attesa finché gli é consentito di trasmettere.

Problema dei terminali nascosti Una modalitá di accesso al canale cosí descritta é soggetta al problema dei terminali nascosti: due nodi, disposti in modo che abbiano raggio di ricezione non sovrapposto e raggio di trasmissione intersecante, cercano di trasmettere contemporaneamente ad un terzo nodo che si trova proprio nell'area di trasmissione comune ai precedenti due. Con questa disposizione, un nodo mittente non riesce a capire se il canale é occupato dall'altro cosí che nel nodo ricevente, che é nel mezzo, si verificano un gran numero di collisioni. Per superare ció, lo standard definisce dei meccanismi opzionali che usano due pacchetti di controllo, RTS e CTS. Invece che inviare direttamente un frame dati, un nodo trasmette un RTS (Request To Send) che contiene il mittente, il ricevente della trasmissione e la durata della trasmissione che seguirá. Se il ricevente riceve l'RTS, risponde con un CTS (Clear To Send) che contiene gli stessi dati, il canale viene prenotato e i dati della prenotazione sono propagati anche alla stazione radio nascosta al mittente, che si mette in attesa per la durata della trasmissione come specificato nei frame di controllo. Le collisioni possono avvenire ancora, ad esempio quando i due mittenti inviano un RTS contemporaneamente, ma essendo questo pacchetto molto piccolo (da 0 a 2347 ottetti), la probabilitá é piú abbassa. Se il frame dati da inviare é piú piccolo dell'RTS, un nodo puó decidere di inviare direttamente il pacchetto senza utilizzare questo meccanismo di prenotazione.

Struttura del frame 802.11 Lo standard 802.11 definisce tre differenti tipi di frame: dati (DATA), controllo (CTRL), manutenzione (MGMT). Ognuno di essi é formato da svariati campi utilizzati dal livello MAC per svolgere le sue funzioni.

Un campo degno di nota é il Frame Control, composto da undici sottocampi. Esso contiene informazioni riguardanti il protocollo, il tipo (uno dei tre precedentemente citati, DATA, CTRL, MGMT), il sottotipo del frame (nel caso sia di controllo, per esempio un ACK), se vi é frammentazione e altro ancora. Un'altra informazione sensibile riguarda la crittografia e cioé

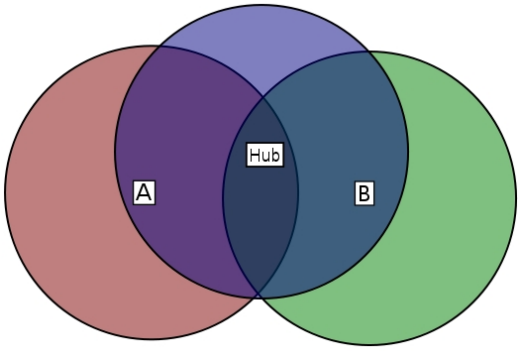

Figura 4.1: I due nodi A e B hanno raggio di ricezione separato, quando inviano dati all'hub contemporaneamente, avviene una collisione.

Figura 4.2: Struttura generale del frame 802.11.

specifica se il payload é crittato oppure no.

Per completezza qui di seguito viene riportata la struttura di un ACK frame. Come si puó notare, non é presente nessun campo esplicito che permetta di ricondurre l'acknowledgement ricevuto col relativo pacchetto trasmesso, infatti, poiché la trasmissione é di tipo stop-and-wait, si puó capire che l'ACK ricevuto é associato a quello precedentemente inviato.

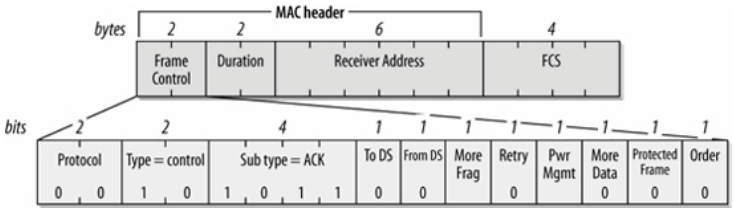

Figura 4.3: Frame ACK 802.11.

Risparmio energetico I nodi mobili non hanno una fonte stabile di energia ma utilizzano batterie, un aspetto importante é quindi risparmiare il piú possibile energia per allungare i tempi di operativitá tra una ricarica e l'altra.

IEEE 802.11 include una modalitá dedicata a questo aspetto. Un nodo puó essere in due stati: sleep, quando la sua ricetrasmittente é spenta, o awake, completamente funzionante. Gli access point sono generalemente sempre awake. Quando un access point deve inoltrare dati ad un nodo mobile in stato sleep, li bufferizza finché il nodo mobile é di nuovo attivo. Ad intervalli determinati, i nodi sleep si svegliano e gli access point possono annunciare le destinazioni dei frame bufferizzati, se un nodo mobile é tra essi, rimane awake fino alla corretta ricezione.

Sicurezza Il mezzo di trasmissione del Wi-Fi é percepibile da tutti senza restrizioni fisiche, la sicurezza quindi é un aspetto da non trascurare. Diversi standard basati su crittografia sono stati sviluppati per superare il problema. Le versioni originali dei protocolli 802.11 erano basati sulla crittografia WEP (Wired Equivalent Privacy): essa si basa su un cifrario a flusso con 40 bit per le chiavi e 24 bit per il vettore di inizializzazione che non deve mai ripetersi per la corretta implementazione dell'algoritmo. Essi peró non sono abbastanza e in una rete con molto traffico potrebbero essere usati piú volte, questo ed altri problemi come la mediocre gestione delle chiavi rendono WEP molto insicuro, tale da poter essere superato in pochi minuti [31]. WPA (Wi-Fi protected access) [24] cerca di superare i problemi di WEP allungando il numero di bit usati per la chiave ed includendo un meccanismo per cambiare dinamicamente il vettore di inizializzazione. Ciononostante, nel 2008 é stato dimostrato che anche WPA puó essere vulnerabile ad alcuni tipi di attacchi in particolare sul metodo WPA-PSK (TKIP) di cifratura, mentre WPA2-AES rimane immune a questa vulnerabilitá.

4.2 Livello Rete

Il livello Rete é il terzo livello nella pila ISO/OSI, esso si occupa dello scambio logico di pacchetti tra due nodi arbitrari, che in generale non sono direttamente connessi, ovvero non hanno un collegamento diretto tra di

loro e si possono trovare in reti diverse, spesso anche basate su tecnologie differenti. In sostanza il livello rete si occupa di indirizzamento e instradamento (routing) verso la giusta destinazione attraverso il percorso di rete piú appropriato. É quindi importante per questo livello conoscere la topologia di reti e sottoreti per potere scegliere il giusto percorso attraverso di esse, soprattutto nel caso sorgente e destinazione si trovino in reti differenti. Sono stati creati diversi protocolli per questo livello, come IPX e AppleTalk, ma quello di gran lunga piú usato é il protocollo IP, declinato nella versione 4 e 6.

4.2.1 IPv4

IPv4, come descritto nell'RFC 791 [37] dell'IETF del 1981, é un protocollo senza connessione per l'uso su reti a commutazione di pacchetto, come ad esempio Ethernet. Opera con un modello di tipo best effort, non garantisce cioé la consegna né assicura che i pacchetti inviati, detti datagrammi, siano recapitati in ordine o duplicati, questi aspetti infatti sono presi incarico dal livello superiore nella pila di protocolli.

Indirizzamento IPv4 usa indirizzi a 32 bit per identificare un host, ció significa che lo spazio di indirizzamento é fissato a 4.294.967.296 possibili indirizzi. In realtá, molti di questi sono riservati per scopi speciali come reti private e indirizzi multicast, questo riduce il numero di indirizzi che possono essere potenzialmente allocati per il routing nella rete pubblica.

Il numero di indirizzi IP pubblici si sta esaurendo ad una velocitá che non era stata anticipata quando il protocollo é stato ideato: proprio mentre si sta scrivendo questo documento, nel 3 febbraio 2011, lo IANA (Internet Assigned Numbers Authority, l'organismo che ha responsabilitá nell'assegnazione degli indirizzi IP) ha consegnato gli ultimi cinque blocchi di indirizzi /8 ai RiR (Regional Internet Registry) dei cinque continenti [40]. L'esaurimento é dovuto a vari fattori: il rapido incremento del numero di utenti di internet, gli apparati sempre connessi, come i modem ADSL e i nuovi dispositivi

mobili come smartphone e PDA che possono accedere alla rete pubblica. Alcuni stratagemmi sono stati utili a mitigare il problema: il Network Address Translation (NAT), l'uso di reti private, l'uso del DHCP, un controllo piú stretto da parte dei RiR nell'assegnare blocchi di indirizzi, talvolta il reclamo di blocchi assegnati ma non usati, ma l'unica soluzione resta il passaggio ad IPv6, che ha allargato lo spazio di indirizzamento a 128 bit e che verrá ampiamente descritto piú in basso.

Un indirizzo IPv4 é di solito rappresentato con gruppi di quattro numeri separati da punti (es. 127.0.0.1) ed é diviso in due parti: una, quella definita nei bit piú significativi, indica la rete alla quale esso appartiene, l'altra invece identifica l'host. Originariamente le reti avevano un numero di bit fisso, ad esempio una rete di classe A é identificata da 8 bit e puó indirizzare 24 bit per gli host; oggi, per avere maggiore flessibilitá, una rete é definita con il numero di bit dedicato ad essa, ad esempio la rete 192.168.0.0/16 ha 16 bit usati per l'indirizzamento degli host. Un indirizzo IP non é valido ai livelli inferiori. Per associare un indirizzo ip ad una interfaccia di rete, identificata a livello data link da un indirizzo MAC, si usa il protocollo ARP.

Reti private e NAT Dei circa quattro miliardi di indirizzi disponibili in IPv4, tre gruppi di indrizzi sono riservati per reti private: 172.16.0.0/12 192.168.0.0/16 10.0.0.0/8, queste reti non sono instradabili da indirizzi al di fuori di esse né i nodi all'interno possono comunicare con l'esterno se non con il meccanismo NAT. NAT é un metodo semplice per permettere a tutti i computer di una sottorete di dialogare con tutta la rete pur non possedendo un indirizzo visibile dall'esterno. Il metodo consiste nell'avere un solo gateway con ip pubblico accessibile dall'esterno che presta il suo indirizzo per le comunicazioni di tutte le macchine della sottorete traducendo e tenendo traccia dei pacchetti in transito per poter recapitare le risposte ai legittimi destinatari. In questo modo si migliora la sicurezza perché le macchine interne alla rete non sono direttamente accessibili dall'esterno e si migliora l'utilizzo dello spazio di indirizzamento possibile grazie al fatto che

un'intera rete privata utilizza un solo ip. Questa tecnica é utilizzata dalla maggior parte delle reti casalinghe.

DHCP Il Dynamic Host Configuration Protocol é un protocollo con il quale é possibile autoconfigurare i nodi di una rete eliminando il bisogno dell'intervento di un amministratore di rete. DHCP permette anche di avere un database centrale per tener traccia degli host che sono connessi alla rete, prevenendo che due nodi abbiano configurato lo stesso indirizzo. Oltre all'IP, DHCP permette l'assegnazione del server NTP (Network Time Protocol), quello DNS (Domain Name System) ed altri particolari.

ARP Il protocollo ARP (Address Resolution Protocol) é usato per determinare, dato un indirizzo IP, l'indirizzo hardware del livello data-link. Quando un host vuole spedire un datagramma ad un altro nodo conoscendo il suo indirizzo IP ma non quello fisico, fa una richiesta ARP tramite un broadcast sulla rete di appartenenza, chi ha l'indirizzo richiesto risponde con il proprio indirizzo MAC, il richiedente adesso puó creare il frame da inviare.

Formato del pacchetto Un paccehtto IP consiste di un header e di una sezione dati. Nella Figura 4.4 é riportata la sua struttura.

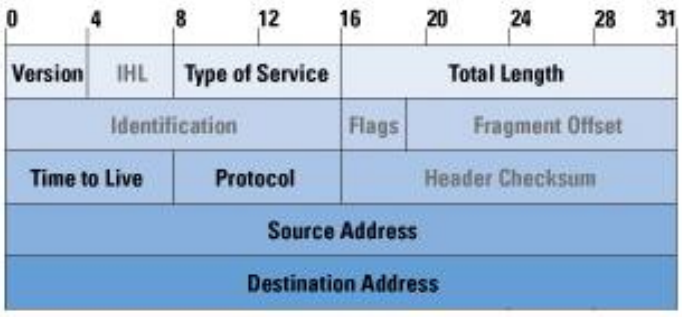

Figura 4.4: Formato del pacchetto IPv4.

Nell'header ci sono quattordici campi:

- **Version** indica la versione del datagramma IP.

- **Internet Header Length** indica la lunghezza (in word da quattro byte) dell'header del paccehtto IP, l'header infatti non ha sempre dimensione fissa ma varia in base alla presenza dell'ultimo campo, opzionale.

- **Type of Service** specifica la precedenza con cui l'host deve trattare il datagramma. Questo tipo di servizio é caduto in disuso, comunque, recentemente questi 8 bit sono stati ridefiniti ed hanno la funzione di Differentiated services (DiffServ nell'IETF e Explicit Congestion Notification (ECN) codepoints (vedi RFC 3168 [39]), necessari per le nuove tecnologie basate sullo streaming dei dati in tempo reale, come ad esempio per il Voice over IP (VoIP) usato per lo scambio interattivo dei dati vocali.

- **Total Length** indica la dimensione (in byte) dell'intero datagramma, comprendendo header e dati; tale lunghezza puó variare da un minimo di 20 byte (header minimo e campo dati vuoto) ad un massimo di 65535 byte. In ogni momento, ad ogni host é richiesto di essere in grado di gestire datagrammi aventi una dimensione minima di 576 byte mentre sono autorizzati, se necessario, a frammentare datagrammi di dimensione maggiore.

- **Identification** é usato per identificare in modo univoco i vari frammenti di un datagramma IP originale.

- **Flags** sono bit utilizzati per il controllo del protocollo e della frammentazione dei datagrammi.

- **Fragment Offset** indica l'offset (misurato in blocchi di 8 byte) di un particolare frammento relativamente all'inizio del datagramma IP originale: il primo frammento ha offset 0. L'offset massimo risulta pertanto pari a 65528 byte che, includendo l'header, potrebbe eccedere la dimensione massima di 65535 byte di un datagramma IP.

- **Time To Live** indica il numero massimo di hop al quale il datagramma é concesso fare, necessario per evitarne la persistenza indefinita sulla

rete nel caso in cui non si riesca a recapitarlo al destinatario. Quando questo arriva a zero il datagramma non viene piú inoltrato ma scartato. Tipicamente, quando un datagramma viene scartato per esaurimento del TTL, viene automaticamente inviato un messaggio ICMP al mittente del datagramma, specificando il codice di richiesta scaduta; la ricezione di questo messaggio ICMP é alla base del meccanismo del traceroute.

- **Protocol** indica il tipo di protocollo di livello trasporto nella porzione dati del pacchetto IP.
- **Header Checksum** é un campo usato per il controllo degli errori dell'header. Ad ogni hop, il checksum viene ricalcolato e confrontato con il valore di questo campo: se non corrisponde il pacchetto viene scartato.
- **Source address** indica l'indirizzo IP associato al mittente del datagramma.
- **Destination address** indica l'indirizzo IP associato al destinatario del datagramma.
- **Options + padding** sono opzioni (facoltative e non molto usate) per usi piú specifici del protocollo. Devono essere necessariamente di dimensione multipla di 32 bit, nel caso contrario, dei bit senza significato, il padding, sono aggiunti.

Frammentazione Per rendere il protocollo tollerante alle differenze delle varie sottoreti si é inserito il meccanismo della frammentazione. Grazie a questo ogni device ha la possibilitá di spezzare i dati in piú pacchetti. Questo é necessario nel caso la MTU (maximum transmission unit) della rete sia minore della dimensione del pacchetto. Per esempio la dimensione massima di un pacchetto IP é di 65.535 byte ma la dimensione tipica del MTU di una rete Ethernet é di 1500. Questo comporta che per inviare tutto il payload del

pacchetto IP sono necessati 45 frame ethernet. Se la frammentazione fosse implementata a livelli inferiori (Ethernet, ATM, etc.) risulterebbe necessario frammentare e riassemblare ad ogni hop, questo implicherebbe una aumento dei costi. Quando un device riceve un pacchetto IP esamina la destinazione e sceglie su quale interfaccia instradarlo. Ogni interfaccia ha una MTU specifica, quindi si controlla se é neccessaria una frammentazione. In caso affermativo, i dati vengono suddivisi in diversi pacchetti, ognuno ha un proprio header con impostate la nuova dimensione, l'offset dei dati dall'originale e il bit MF (more fragment) in caso seguano altri frammenti. Una volta che tutti i frammenti sono stati ricevuti dalla destinazione, l'host puó riassemblarli e passarli ai livelli superiori.

Routing Il routing é il meccanismo per il quale un nodo decide su quale interfaccia inviare un pacchetto che deve essere racapitato alla destinazione desiderata. Generalmente questa decisione viene fatta consultando una tabella che, per ogni destinazione, indica quale interfaccia usare. La costruzione di questa tabella puó essere fatta manualmente per piccole reti, reti piú complicate presentano topologie complesse e collegamenti ridondanti che possono cambiare rapidamente per esempio a causa di guasti, la configurazione manuale quindi non puó essere un'opzione attuabile. Gli algoritmi di routing, come il vecchio RIP o il piú usato Open Shortest Path First (OSPF), utilizzano le informazioni scambiate tra i nodi per generare automaticamente queste tabelle. Gli algoritmi di routing si possono classificare in due principali famiglie: Distance-vector e Link-state. Nell'algoritmo Distance-vector, ogni router misura la distanza che lo separa dai nodi adiacenti e riceve informazioni dai router vicini con le quale aggiorna la propria tabella. Ad ogni nuovo aggiornamento, la tabella viene ricalcolata ed inviata ad i nodi adiacenti; questo é il metodo usato da RIP; nel link-state invece ogni nodo invia a tutti gli altri le informazioni locali di sua conoscenza in modo che alla fine dello scambio ogni router abbia dati sull'intera topologia della rete; viene successivamente applicato l'algoritmo di Dijkstra per determinare il cammi-

no minimo verso ogni nodo e quindi il prossimo hop per ogni destinazione; questo é il metodo usato da OSPF. Per interconnettere reti non é possibile usare queste famiglie di algoritmi poiché genererebbero troppo traffico e tabelle di routing enormi, in tali casi si ricorre ad altri metodi come il BGP, boarder gateway protocol.

4.2.2 IPv6

Internet Protocol versione 6 (IPv6) é la versione del protocollo IP designata per succedere alla versione 4. IPv6 é stato sviluppato dalla Internet Engineering Task Force (IETF) ed é descritto dall'RFC 2460 [32] pubblicato nel dicembre 1998. A causa dell'esplosiva crescita di Internet, il bisogno di indirizzi é aumentato a tal punto da esaurire quelli messi a disposizione da IPv4. IPv6 risolve il problema aumentando lo spazio di indirizzamento a 128 bit, pari a circa $3,4 \times 10^{38}$ possibili indirizzi, giudicato abbastanza grande per qualsiasi espansione futura della rete globale. Oltre al nuovo spazio di indirizzamento, al protocollo sono state aggiunte nuove caratteristiche come il supporto nativo alla sicurezza (IPsec) e meccanismi di autoconfigurazione. IPv6 non é direttamente compatibile con IPv4, ció crea effettivamente una rete indipendente da quella esistente, ciononostante sono stati sviluppati diversi meccanismi per agevolare la transazione tra i due protocolli.

Indirizzamento La caratteristica piú importante di IPv6 é il piú ampio spazio di indirizzamento, portato da 32 a 128 bit. Ció é stato fatto anche per semplificare l'allocazione di indirizzi, permettere una piú efficente aggregazione delle rotte ed implementare speciali metodi di indirizzamento come il multicast e l'anycast.

Gli indirizzi sono scritti in forma di otto gruppi di quattro cifre esadecimali separati da due punti, ad esempio 2001:1234:5678:9abc:0000:0000:0000:1234, gli zeri possono essere abbreviati utilizzando i caratteri ::, per l'esempio precendete, la forma abbreviata é 2001:1234:5678:9abc::1234, un gruppo di tutti zeri inoltre puó essere sostituito da un solo zero. Gli indirizzi sono divisi in

due parti, i primi 64 bit rappresentano il prefisso di rete, i restanti 64 bit l'identificativo di rete. Indirizzi unicast identificano una interfaccia di un host, indirizzi anycast un gruppo di interfacce, di solito in diversi host cosí che il piú vicino é automaticamente selezionato, e indirizzi multicast per inviare dati ad un gruppo di nodi differenti. Il broadcast cosí come é conosciuto in IPv4 é implementato attraverso il multicast all'indirizzo ff02::1. Gli indirizzi inoltre hanno uno scope che specifica in quale parte della rete sono validi, ad esempio solo nel collegamento diretto con un'altra interfaccia (link-local), nella propria sottorete (site-local, deprecato), o globale.

Formato del pacchetto IPv6 specifica un nuovo formato per i pacchetti ideato per minimizzare il lavoro che i ruoter devono compiere. Il nuovo formato rassomiglia a quello della versione 4 ma é stato semplificato, sono stati eliminanti molti campi poco usati e spostati in estensioni degli header separate. L'header ha dimensione fissa di quaranta byte, solo circa il doppio piú grande della precedente versione a fronte di un aumento di quattro volte dello spazio di indirizzamento.

Le opzioni sono implementate come estensioni addizionali dell'header e sono inviate subito dopo l'intestazione fisso. Queste hanno un campo, chiamato next header, che indica il tipo dell'header successivo, se presente, o il protocollo di quarto livello trasportato nel payload. Quindi, piú di un estensione puó essere presente per ogni datagramma. Questo meccanismo permette flessibilitá e supporto per futuri servizi per la sicurezza, la mobilitá ed altro ancora, senza bisogno di riprogettare il protocollo, e, non meno importante, una maggiore efficienza dei router nel processare i pacchetti classici: nella maggior parte dei casi, infatti, i router intermedi non devono fare il parsing di queste estensioni. Di seguito é riportata una lista delle estensioni piú comuni:

- **Hop-by-hop options** Definisce un insieme arbitrario di opzioni intese per ogni hop attraversato.

- **Routing packet** Definisce un metodo per permettere al mittente di specificare la rotta da segure per un datagramma.

- **Fragment packet** É usato quando un datagramma viene frammentato.

- **No next header** Indica che i dati nel payload del datagramma non sono incapsulati in nessun altro protocollo.

- **Destination options** Definisce un insieme arbitrario di opzioni di interesse del solo destinatario.

- **Mobility options** Usato per Mobile IPv6.

- **Altri protocolli (TCP, UDP, ...)** Indica il protocollo di livello 4 del pacchetto.

L'efficienza é ulteriormente aumentata grazie alla rimozione del checksum (si assume che l'integritá dei datagrammi sia garantita dagli alti livelli come il TCP e l'UDP e da quelli piú bassi come quello data-link) e all'eliminazione della frammentazione nei router intermedi: il mittente deve inviare pacchetti piccoli quanto la minore MTU della rotta, per fare ció viene usato il Path MTU Discovery, o, quando la MTU di un link é piú piccola del datagramma, un messaggio ICMPv6 é inviato dal nodo che riscontra il problema verso il primo nodo mittente, che quindi é avvisato di ridurre la quantitá di dati inviata per pacchetto.

Figura 4.5: Formato del pacchetto IPv6.

Autoconfigurazione statica Gli host possono autoconfigurarsi automaticamente quando connessi a una rete IPv6 usando messaggi ICMPv6 di roter discovery. Appena connesso, un host spedisce una richiesta multicast per avere i suoi parametri di configurazione, i router rispondono a questa con un messaggio di router advertisement che contiene i parametri necessari, come ad esempio il prefisso della rete. Questo meccanismo puó non essere indicato per tutti i tipi di applicazioni, se necessario una rete puó comunque usare il DHCPv6 o essere configurata manuamente.

Supporto obbligato alla sicurezza IPsec (Internet Protocol Security) é stato originariamente sviluppato per IPv6 ma ha trovato una larga diffusione anche per IPv4, per il quale é stato integrato successivamente. IPsec é parte integrante dello standard IPv6 e il suo supporto é obbligatorio mentre per IPv4 questo non é vero.

Mobilitá A differenza di mobile IPv4, mobile IPv6 evita il routing triangolare e quindi é efficente come l'IPv6 nativo. I router IPv6 possono inoltre supportare la mobilitá di rete che permette ad intere sottoreti di essere spostate senza cambiare prefisso.

Meccanismi di transizione da IPv4 a IPv6 IPv4 e IPv6 sono effettivamente due diversi protocolli indipendenti e non compatibili. Nonostante il bisogno del passaggio a IPv6 sia necessario, non é possibile richiedere a tutti i produttori e utilizzatori del mondo di fissare una data precisa per passare da un sistema all'altro. Diversi protocolli sono stati sviluppati per agevolare il graduale passaggio, qui di seguito sono riportati i principali meccanismi:

Dual Stack L'implementazione del doppio stack IP in un sistema operativo permette di interpretare entrambe le versioni del protocollo ed é una necessitá fondamentale nelle tecnologie di transizione da IPv4 a IPv6. Questo puó essere implementato sviluppando due stack completamente indipendenti o in modo ibrido, cioé con un unico modulo software che

é in grado di trattare le due forme. La seconda soluzione é piú comunemente scelta nei moderni sistemi operativi ed é descritta nel RFC 4213 [36].

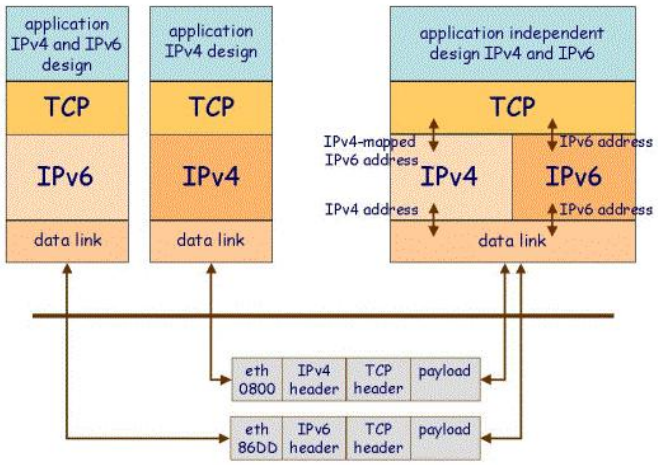

Figura 4.6: Applicazione indipendente dal protocollo su un host con doppio stack.

Il supporto dual stack permette agli sviluppatori di applicazioni di lavorare trasparentemente con IPv4 o IPv6 sullo stesso socket: quando si vogliono usare indirizzi IPv4, essi vengono rappresentati in un indirizzo IPv6 con i primi ottanta bit a zero, i successivi sedici ad uno e i restanti trentadue con l'effettivo indirizzo IPv4, questa mappatura viene scritta con la forma ::FFFF:indirizzo-v4, ad esempio ::FFFF:192.0.2.128. La mappatura, inoltre, puó essere usata per permettere ad host IPv6 che non hanno un indirizzo IPv4 assegnato di comunicare con altri nodi IPv4. I sistemi operativi con stack separati, come Windows XP e Server 2003, non possono avvalersi di questa funzionalitá, altri, come OpenBSD, la disabilitano per questioni di sicurezza. Se si vuole avere maggiore controllo sui tipi di pacchetti gestiti da un socket é possibile comunque usare delle opzioni per limitare il socket ad un solo protocollo.

Tunneling Per raggiungere la rete Internet IPv6, un host che ha accesso solo ad una rete IPv4 puó usare la tecnica del tunnelling. Essa consiste nell'incapsulare pacchetti IPv6 in pacchetti IPv4. Questo é comunemente indicato come protocollo 41. IPv6 puó inoltre essere incapsulato in pacchetti UDP per superare una rotta con una rete nattata o con firewall che blocca il traffico del protocollo 41. Allo stesso modo, il problema si pone se da una rete IPv6 si vuole raggiungere un nodo che supporta soltanto IPv4, ció é risolto usando altri tipi di tunnel basati sullo stesso principio.

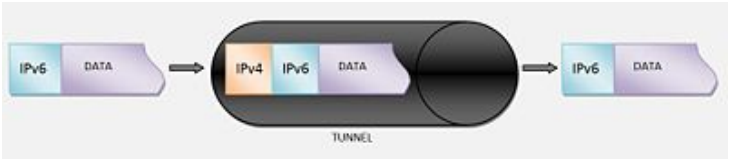

Figura 4.7: Per attraversare una rete IPv4, un pacchetto IPv6 viene incapsulato in uno IPv4, una tecnica chiamata Tunnelling.

Un tunnel puó essere costituito automaticamente grazie al protocollo 6to4, descritto nel RFC 3056 [30]. Gli endpoint del tunnel sono determinati usando uno speciale indirizzo anycast IPv4 dal lato remoto e aggiungendo informazioni sull'indirizzo IPv4 all'interno di quello IPv6 nel lato locale. Le entitá che forniscono questo servizio si chiamano tunnel broker. La configurazione automatica di un tunnel si puó raggiungere anche grazie al tunnelling Teredo. Esso é utilizzato principalmente per superare reti con NAT incapsulando il pacchetto IPv6 in uno UDP che gestibile dal network address translation protocol. ISATAP (Intra-Site Automatic Tunnel Addressing Protocol), invece, tratta la rete IPv4 come un link locale IPv6, mappando cioé ogni indirizzo IPv4 con un indirizzo link-local IPv6, ed é indicato solo per la connettivitá all'interno di una singola organizzazione.

Proxying e traduzione per host IPv6-only Il NAT che traduce da IPv4 a IPv6 e viceversa é possibile con i protocolli NAT-PT e NAPT-PT

definiti nel RFC 2766 [41]. Questa tecnica é stata molto discussa e considerata da taluni controversa fino al RFC 4966 [28] che la depreca definitivamente.

4.3 Livello Trasporto

Il livello trasporto permette il dialogo strutturato tra mittente e destinatario. I protocolli definiti permettono la comunicazione end-to-end, senza considerare i passaggi intermedi, tra processi residenti in host diversi. Uno di questi protocolli é il TCP (Transmission Control Protocol). Esso é connection-oriented e permette al flusso di dati di essere recapitato senza errori. I frammenti dello stream vengono impacchettati e passati ai layer inferiori, sará poi il ricevente che provvederá a riassemblarli e in caso di errore a chiederne il rinvio. Implementa anche un meccanismo per il flow control, non permette cioé ad una sorgente veloce di congestionare un ricevente lento, ed il congestion control, per evitare di congestionare i router che sono tra i due end-point.

Il protocollo piú interessante dal punto di vista di questa tesi é l'UDP (User Datagram Protocol). Questo é di tipo inaffidabile, non assicura cioé che che i pacchetti arrivino a destinazione, e senza connessione. Tali caratteristiche lo rendono proficuamente utilizzabile laddove le applicazioni necessitano di interattivitá e velocitá. Per esempio audio e video vengono quasi sempre inviati attraverso UDP, questo perché piuttosto che la ricezione affidabile di tutti i pacchetti, é molto piú importante la regolaritá e la continuitá del flusso di dati.

Protocollo UDP L'User Datagram Protocol é uno dei protocolli fondamentali di Internet. Le applicazioni possono creare e inviarsi dei messaggi brevi (conosciuti anche come datagram) attraverso degli appositi socket. Il protocollo é stato definito nel 1980 da David P. Reed (RFC 768 [38]). UDP non garantisce l'affdabilitá della connessione o che i pacchetti vengano rice-

vuti nell'ordine di invio. Infatti, essi possono anche essere duplicati o non arrivare affatto senza che questo venga segnalato in alcun modo. L'eliminazione di questo controllo rende il protocollo veloce ed efficiente per le applicazioni che non necessitano di queste garanzie. Spesso applicazioni audio e video lo utilizzano data la preferenza ad avere pacchetti mancanti invece che ritardi elevati dovuti alla ritrasmissione, oppure in caso sia necessario una sincronia tra un server e dei client (per esempio i giochi in rete) o anche se si debbano gestire messaggi in broadcast o multicast. Un esempio di applicazioni che utilizzano UDP sono il Domain Name System (DNS), streaming media applications come IPTV, Voice over IP (VoIP) e per esempio Trivial File Transfer Protocol (TFTP).

UDP header Il protocollo UDP impone un header dei pacchetti con la struttura riportata nella Figura 4.8.

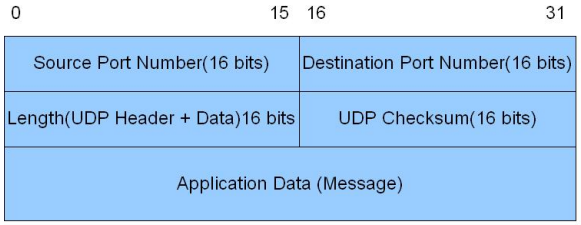

Figura 4.8: Formato del pacchetto UPD.

É da notare come l'overhead aggiunto ad IP sia limitato ai campi che specificano le porte usate delle applicazioni coinvolte nella conversazione fra i due host (le porte) ed un checksum; il calcolo di quest'ultimo é facoltativo se si usa IPv4, mentre nel caso di IPv6 é obligatorio (si ricorda che IPv6 non ha nessun controllo di integritá a differenza di IPv4).

Caratteristiche del protocollo UDP é un semplice protocollo basato sul trasferimento di messaggi e privo di connessione (non si crea una connessione end-to-end dedicata). Semplicemente lo stream di pacchetti

viene inviato alla destinazione senza nessun controllo di sorta e ognuno dei pacchetti é completamente indipendente dall'altro.

Inaffidabilitá Quando un messaggio viene inviato, non é possibile sapere nulla della sua sorte.

Mancanza di ordinamento Il flusso di messaggi inviati puó essere ricevuto in ordine differente.

Leggero Data la mancanza di controlli é leggero in termini di dimensione dell'header, efficiente e veloce.

Datagram I pacchetti vengono inviati individualmente e nel caso arrivino a destinazione essi sono sicuramente integri. Hanno dimensione predefinita e non vengono spezzettati o riassemblati a livello trasporto.

4.4 Livello Applicazioni

Il livello applicazioni si riferisce alla parte piú alta della pila ISO/OSI che contiene tutti i protocolli che si occupano di fornire servizi per i processi delle applicazioni usate dagli utenti finali. Un programma applicativo interagisce con uno dei protocolli di livello di trasporto per ricevere dati o inviarli passandoli nella forma richiesta. Nel modello OSI la definizione del Livello Applicazioni é piú ristretta rispetto a quello TCP/IP, esso divide le funzioni in tre ulteriori livelli:

- **Livello Sessione** offre servizi che consentono ad utenti operanti su macchine differenti di colloquiare tra loro attraverso la rete di comunicazione;
- **Livello Presentazione** ha il compito di di trasformare i dati forniti dal Livello applicazioni in un formato standard e offrire servizi di comunicazione comuni, quali la crittografia, la compressione del testo e la riformattazione;

- **Livello Applicazioni** il livello ultimo dedicato al vero e proprio scambio di dati per le applicazioni.

Nell'uso comune, spesso questi livelli sono accorpati.

A questo livello appartengono miriade di protocolli diversi per i piú disparati bisogni, da quelli per i file system condivisi a BitTorrent, da FTP (File Transfer Protocol) ad HTTP (HyperText Transfer Protocol), che é alla base del World Wide Web, da IMAP per i servizi di email a SSH per il controllo di terminali remoti. In questa sezione saranno descritti i protocolli che toccano piú vicino il lavoro di questa tesi, in particolare i protocolli utilizzati nel VoIP.

4.4.1 VoIP

Voice over Internet Protocol (Voice over IP, VoIP) é una famiglia di tecnologie internet e protocolli di comunicazione progettata per distribuire comunicazioni vocali e sessioni multimediali attraverso il protocollo IP, invece della rete tradizionale PSTN (public switched telephone network). Il vantaggio principale di questa tecnologia sta nel fatto che essa elimina l'obbligo di riservare una quota di banda fissa per ogni telefonata (il concetto di commutazione di circuito), sfruttando l'allocazione dinamica delle risorse, caratteristica del protocollo IP, che é a commutazione di pacchetto.

Protocolli Per effettuare una chiamata VoIP, generalmente, si digitalizza il segnale fisico della voce, si codifica il segnale analogico in digitale, eventualmente comprimendolo, si pacchettizzano i dati risultanti dal passo precedete e si trasmettono questi pacchetti attraverso il protocollo IP; similmente, dal lato ricezione vengono effettuate le corrispettive operazioni inverse.

Il VoIP richiede due tipologie di protocolli di comunicazione in parallelo, una per il trasporto dei dati (pacchetti voce su IP), ed una per la segnalazione della conversazione, come la ricostruzione del frame audio, la sincronizzazione e l'identificazione del chiamante. Per il trasporto dei dati, nella grande

maggioranza delle implementazioni VoIP viene adottato il protocollo RTP (Real-time Transport Protocol, a sua volta basato su UDP).

Sono stati sviluppati diversi protocolli di segnalazione. H. 323, della ITU (International Telecommunications Union), é stato uno tra i primi protocolli VoIP: nasce in ambito telefonico e delinea un'architettura completa per lo svolgimento di conferenze multimediali, comprendente la definizione dei formati di codifica a livello applicativo, quelli per la segnalazione e il controllo, per il trasporto dei flussi audio, video e dati e per la gestione degli aspetti di sicurezza. Ognuno di essi é stato studiato avendo come riferimento architetture di rete locali. SIP, della IETF (Internet Engineering Task Force), é piú recente e sta riscontrando un maggiore successo di H. 323: esso ha funzionalitá di instaurazione e terminazione della sessione, operazioni di segnalazione, tono di chiamata, chiamata in attesa, trasferimento, identificazione del chiamante ed altro ancora. Altri protocolli utilizzati per la codifica della segnalazione della conversazione (ricostruzione del frame audio, sincronizzazione, ecc.) sono: Skinny Client Control Protocol, protocollo proprietario della Cisco, Megaco (conosciuto anche come H.248), MGCP, MiNET, protocollo proprietario della Mitel, Inter Asterisk Xchange, (soppiantato da IAX2) usato dai server Asterisk open source PBX e dai relativi software client, e XMPP, usato da Google Talk, inizialmente pensato per l'IM ed ora esteso a funzioni Voip grazie al modulo Jingle. É da notare anche che uno dei protocolli piú usati per le utenze domestiche é quello di Skype, proprietario e non rilasciato pubblicamente, che si basa sui principi del Peer-to-Peer.

Benefici I benefici dell'utilizzo di una tecnologia innovativa quale il VoIP sono molteplici:

Diminuzione dei costi Il VoIP Consente un abbattimento dei costi notevole al singolo utente. Sia per chiamate nazionali che internazionali o intercontinentali, il costo della chiamata praticamente non varia sia che l'interlocutore sia nello stesso paese che dalla parte opposta del mondo: utilizzando una linea ADSL si puó chiamare qualsiasi altro PC gratui-

tamente, per contattare telefoni PSTN, invece, le compagnie fornitrici di VoIP hanno bisogno di pagare solo l'ultimo tratto, cioé l'azienda telefonica del ricevente.

Agevolazioni in campo lavorativo Il VoIP puó essere utilizzato sia all'interno di una stessa azienda, sia per telefonate piú complesse, come teleconferenze in multi-point. Inoltre si puó utilizzare una sola rete integrata per voce e dati, che garantisce una riduzione dei bisogni totali di infrastruttura e, in ambito aziendale uno scambio agevolato delle informazioni.

Flessibilitá Il VoIP puó fornire servizi che possono essere molto difficili da implementare usando PSTN, come ad esempio la possibilitá di trasmettere piú di una telefonata contemporaneamente con una sola linea, indipendenza del numero dalla locazione fisica, integrazione con altri servizi Internet, ad esempio conversazioni video, scambio di messaggi e dati, conferenze audio e gestione di rubriche telefoniche.

Sicurezza Rispetto al PSTN, il VoIP consente con relativa facilitá di eseguire conversazioni non intercettabili grazie all'uso della crittografia e protocolli standardizzati come il Secure Real-time Transport Protocol, quando i dati voce entrano in una fibra ottica, inoltre, sono praticamente impossibili da spiare.

Numeri di emergenza Dal 2006 la maggior parte dei provider di servizi VoIP ha abilitato le chiamate ai numeri di emergenza, localizzando in tempo reale l'utente.

Problematiche Come tutte le tecnologie in via di sviluppo, e quindi da perfezionare, anche questa del VoIP presenta alcune sfide da indirizzare.

Qualitá del servizio Le comunicazioni basate su IP sono inerentemente meno affidabili rispetto alla commutazione di circuito del PSTN perché

per definizione IP non fornisce un meccanismo di riservazione del canale, non assicura che i pacchetti dati non siano persi o recapitati in ordine, in altre parole, non esistono garanzie QoS (Quality of Service). Le conversazioni VoIP possono presentare due fondamentali problemi: latenza e jitter. Normalmente, i router IP gestiscono il traffico in modo FIFO, il primo pacchetto ad entrare é anche il primo ad essere processato. Router su una linea ad alto volume di traffico potrebbero introdurre una latenza che eccede quella massima per avere una conversazione di qualitá, inoltre, é da ricordare che la voce viaggia sulla stessa linea dei dati. Di solito i protocolli sono basati su UDP che ammette la perdita di pacchetti, questo non é necessariamente un male perché, fino ad una certa soglia, una perdita é tollerabile e piú sensata dell'attendere una ritrasmissione. Con il termine jitter si intende la variazione della velocitá con la quale i pacchetti voce arrivano, se questo varia troppo la coversazione é degradata con momenti di qualitá ed attimi di interruzione; il jitter é trattato introducendo un buffer per i pacchetti in entrata in modo da mitigare l'arrivo fuori ordine e da regolare la velocitá con la quale i dati in ingresso sono trasformati in audio al costo di un incremento della latenza.

Necessitá della rete elettrica Una linea telefonica classica é direttamente alimentata dal cavo telefonica: in caso di rotture della rete elettrica, il telefono é comunque funzionante. A causa dei differenti tipi di dispositivi, un telefono VoIP non puó contare su questa peculiaritá. Alcuni produttori, comuqnue, hanno iniziato ad aggiungere delle batterie ai propri device in modo da mantenere il servizio per un ceto periodo anche in assenza di alimentazione diretta.

Numerazioni speciali Un ulteriore svantaggio é dato dalla impossibilitá di alcuni operatori di chiamare i numeri di emergenza e alcuni numeri speciali.

Capitolo 5

Architettura

Qui di seguito si descrivono le problematiche che vengono poste nell'usare il VoIP su reti 802.11: vengono descritti gli standard e i meccanismi fino ad adesso sviluppati per il supporto all'interattivitá in reti mobili, se ne evidenziano i punti positivi e quelli negativi. Si passa poi alla definizione di un architettura che intende superare alcune di queste limitazioni.

5.1 VoWIFI

Il VoIP é un'applicazione con requisiti molto specifici che, se non rispettati, impediscono una corretta fruizione del servizio stesso. Le caratteristiche tecniche di un canale Wi-Fi dovrebbero teoricamente permettere di soddisfare tali requisiti ma in realtá questo non sempre é possibile.

L'attuale standard 802.11b/g infatti é stato realizzato senza porre la dovuta attenzione alle problematiche delle applicazioni real-time. Il protocollo MAC definito da questo permette di coordinare l'accesso al canale da parte dei vari trasmettitori ma non discrimina sulle tipologie di traffico generate da questi. Di conseguenza non vi é modo di garantire che un'applicazione real-time possa spedire i propri dati entro determinati limiti di tempo, ovvero non sono previste politiche di QoS.

L'accesso al canale avviene mediante contesa dello stesso da parte dei vari trasmettitori; il vincitore ottiene il permesso di trasmettere e non sono previsti limiti temporali entro i quali liberare il canale stesso. Un'applicazione real-time ha quindi le stesse probabilitá di una qualunque altra di accedere al canale, ma, sebbene altre applicazioni come ad esempio navigazione web, trasferimento file o mail sono tipicamente meno sensibili a ritardi sulla trasmissione questo non vale per un'applicazione real-time. Un altro fattore limitante per un'applicazione real-time é la procedura di roaming. Tale procedura viene eseguita dal terminale quando questo si sposta eccessivamente dalla area di copertura/ricezione dell'access point a cui é connesso. Quando il collegamento con l'AP viene perso il terminale avvia tale procedura per ripristinare il collegamento con un AP diverso. Il roaming viene eseguito solo per connettersi ad AP facenti parte dello stesso ESS (Extended Service Set) e richiede anche svariati secondi per il suo completamento se risulta necessario riconfigurare anche il livello network. Infine i pacchetti VoIP, le cui dimensioni del payload sono molto ridotte e sono inferiori ai 200 Byte, subiscono un forte overhead dovuto ai vari header RTP/UDP/IP/MAC. In generale si osserva che l'attuale standard 802.11 ha una efficienza molto ridotta nella trasmissione di pacchetti di piccole dimensioni.

Nonostante una banda teorica di 54 Mb/s per l'802.11g e 600 Mb/s per il nuovo draft n, rispetto alle esigenze di un'applicazione VoIP, si osserva che un singolo AP puó gestire non piú di dieci sessioni VoIP simultanee, pena un degrado netto della qualitá di conversazione. Le problematiche precedentemente illustrate sono state oggetto di studio ed hanno portato a due estensioni dello standard: 802.11e per fornire il QoS, 802.11r per velocizzare i tempi di roaming.

5.1.1 802.11e

IEEE 802.11e [25] é un nuova sezione dello standard IEEE 802.11 che definisce un insieme di miglioramenti riguardanti il Quality of Service per le

reti wireless lan. Questo standard é considerato di cruciale importanza per le applicazioni sensibili alla latenza come quelle multimediali e il VoIP.

Nello standard originale, IEEE 802.11 introduce il concetto di coordinazione (PCF, point coordination function): gli access point spediscono frame di tipo beacon ad intervalli regolari, di solito 0,1 secondi; vengono definiti due periodi, un periodo con possibilitá di contesa (CP, Contention Period) ed uno senza (CFP, Contention Free Period), dove é possibile il meccanismo distributed coordination function (DCF) che consiste nell'assegnare turni fissi ai nodi che vogliono inviare traffico sensibile al ritardo. Lo standard 802.11e estende tale meccanismo introducendo il concetto di classi/tipologie di traffico: ad esempio il VoIP potrebbe essere in una classe a maggiore prioritá delle email. La nuova funzione di coordinazione é chiamata hybrid coordination function (HCF), in essa ci sono due metodi di accesso al canale, HCF Controlled Channel Access (HCCA) e Enhanced Distributed Channel Access (EDCA), entrambe dotate di classi di traffico.

Con l'EDCA, il traffico ad alta prioritá ha maggiori possibilitá di essere spedito: viene usata una finestra di contesa ridotta e viene fornito al nodo che la utilizza un periodo limitato di acesso al canale senza contesa (TXOP, Transmit Opportunity) durante il quale un nodo mobile puó inviare tanti frame quanti disponibili, se un frame é troppo grande per essere spedito in questo periodo, il frame dovrebbe essere frammentato in modo da rientrare nell'intervallo.

Nell'HCCA, viene mantenuta l'impostazione di base della PCF aggiungendo ad essa i concetti di classi di traffico e TXOP. HCCA é considerato il piú avanzato e complesso meccanismo di coordianzione, in esso il QoS puó essere configurato con gran precisione anche nei parametri quali il data rate e il jitter. questa modalitá tuttavia é definita come opzionale dallo standard ed é scarsamente diffusa e poco supportata dai dispositivi hardware attualmente in commercio.

Con lo standard 802.11e, inoltre, sono introdotte funzionalitá di:

- **Admission Control** Permette all'AP di impedire l'accesso al cana-

le ad un terminale con una determinata tipologia di traffico. Un AP tipicamente ha una bassa capacitá in termini di gestione di applicazioni real-time. Nel caso del VoIP non é possibile supportare piú di 10 sessioni VoIP contemporanee, pena un degrado netto della qualitá del servizio. Una undicesima sessione VoIP causerebbe tale degrado senza inoltre poter essere servita correttamente, in tali casi l'Admission Control permette all'AP di proteggere le attuali sessioni da questo fenomeno, rifiutando l'undicesima sessione.

- **QosNoAck** Questo parametro, specificabile per ogni frame che un terminale desidera spedire all'AP, indica di non generare un ACK di risposta in seguito alla corretta ricezione del frame. Questo serve ad evitare tentativi potenzialmente inutili di ritrasmissione, da parte del terminale, di dati real-time.

- **DLS (Direct Link Setup)** Permette la comunicazione diretta tra terminali all'interno dello stesso BSS.

- **Block acknowledgments** Permette di ricevere un solo ACK per tutti i frame in un TXOP.

- **Automatic power save delivery** Permette un risparmio di energia consentendo all'AP di inviare dati al MN solo subito dopo aver ricevuto un frame da esso, successivamente il MN entra in uno stato di sleep finché non ha un altro frame da inviare.

Basato sullo standard 802.11e, la certificazione Wireless Multimedia Extensions (WME) fornisce funzionalita QoS di base grazie all'introduzione di quattro categorie, voce, video, best effort e background, ma non dá garanzie per quanto riguarda il throughput.

5.1.2 802.11r

Lo standard 802.11r [34] ha come obiettivo quello di ridurre il tempo necessario a completare la procedura di roaming del nodo mobile da un access point all'altro appartenenti allo stesso extended service set.

Il roaming di un un nodo all'interno di un ESS é supportato sin dalla prima versione dello standard 802.11. Quando un terminale sta uscendo dal range di un access point ed entrando in un altro, é possibile disconnettersi dall'AP originale per associarsi a quello nuovo, una procedura chiamata handoff. In principio, l'handoff in reti Wi-Fi base é molto semplice e necessita soltanto dello scambio di quattro messaggi, tuttavia, con l'introduzione degli standard per la sicurezza ed altre estensioni, i tempi di associazione si sono dilatati fino ad arrivare ad impiegare tra due e dieci secondi. Questo mina il QoS di cui le applicazioni multimediali hanno bisogno, da qui l'introduzione dell'802.11r.

Invece di eseguire la procedura di autenticazione ad ogni handoff, essa viene fatta solo la prima volta che un terminale si autentica presso un AP del ESS, l'autenticazione viene mantenuta nella rete wireless attraverso una chiave e, al passaggio di AP in AP, solo questa chiave viene richiesta, senza necessitá di ulteriori negoziazioni per l'accesso.

5.1.3 Considerazioni

Le soluzioni sopra presentate sono un passo avanti verso la costruzione di una solida infrastruttura per usufruire di applicazioni multimediali e VoIP su reti wireless.

Tuttavia, il supporto offerto da questi al VoIP ed ai servizi real-time non é completo ma solo parziale. Mentre le funzionalitá di QoS offerte da 802.11e possono sembrare sufficienti ad una gestione opportuna del traffico real-time, la gestione della mobilitá dovrebbe essere ulteriormente potenziata.

Sebbene si sia ridotto significativamente la durata della procedura di roaming va evidenziato che in tale conteggio non si considera il ritardo introdotto

da un eventuale riconfigurazione del livello rete (eg: indirizzo IP); se l'AP verso cui avviene il roaming é attestato su di una sub-net diversa da quella di provenienza allora sará necessario eseguire una riconfigurazione di livello rete ed applicazione. L'uso di MobileIP contribuisce a mitigare il problema, in ogni caso, dei ritardi vengono inevitabilmente aggiunti: sia durante l'handoff che nella normale attivitá se si é in una rete IPv4 a causa del routing triangolare (un problema superato con MobileIPv6). Inoltre, nonostante le novitá introdotte da 802.11r, la procedura di roaming é supportata solo tra AP appartenenti allo stesso ESS. Ancora, il QoS non puó essere garantito solo da una prioritá sul canale Wi-Fi: esso potrebbe essere disturbato non solo da eventuali collisioni ma anche da fenomeni fisici.

Tali considerazioni fanno emergere la necessitá di un monitoraggio costante del collegamento Wi-Fi, utile a rilevare tempestivamente questo tipo di problematiche in modo da poter eseguire azioni correttive o di recupero prima che causino un degrado nel QoS percepito dall'applicazione VoIP. Si ritiene quindi che le attuali estensioni allo standard 802.11 possano essere affiancate da ulteriori soluzioni che aumentino ulteriormente l'affidabilitá e la qualitá di uno scenario VoWIFI.

5.2 Always Best Packet Switching

La soluzione che sará in seguito analizzata é chiamata Always Best Packet Switching (ABPS) [34] ed é stata sviluppata dal Dipartimento di Informatica dell'universitá di Bologna. Essa permette alle applicazioni di usare simultaneamente tutte le interfacce di rete disponibili ed inviare pacchetti UDP attraverso quella al momento reputata piú adeguata. Essa garantisce il rispetto dei requisiti di QoS dettati da ITU per un servizio VoIP assumendo che:

- il terminale é equipaggiato con due o piú interfacce wireless
- la probabilitá di perdita di un pacchetto sul collegamento Wi-Fi non sia superiore al 10% dei pacchetti trasmessi

- il numero di pacchetti non persi sul tratto wireless ma ricevuti dal destinatario dopo 150 millisecondi non sia superiore all'1%

in particolare, é possibile ottenere un one-way delay medio inferiore ai 150 millisecondi ed un packet loss rate inferiore al 3%, a fonte del 10% sperimentato senza tale sistema, soddisfando quindi i requisiti ITU di un servizio VoIP.

Il sistema proposto si basa su di un approccio cross-layer, utile a far risalire dal livello data-link e fino a livello applicazione opportune informazioni sullo stato del collegamento Wi-Fi. Tali informazioni saranno gestite da un'attivitá di monitoraggio eseguita a livello applicazione, il cui scopo é quello di rilevare eventuali violazioni dei requisiti di QoS del servizio VoIP.

In caso di violazioni di tali requisiti, il sistema, sfruttando la presenza multipla di interfacce wireless, si occupa di ritrasmettere eventuali pacchetti persi. Compito del sistema é anche quello di gestire e configurare opportunamente le interfacce wireless in modo che siano sempre pronte all'utilizzo in base alle informazioni derivanti dalla fase di monitoraggio.

5.3 Architettura

Si immagini l'esistenza di una comunicazione VoIP tra due terminali, denominati A e B nella Figura 3.1. Il terminale A é di tipo mobile, equipaggiato con due o piú interfacce wireless; é posizionato in una tipica area metropolitana, supportata da connettivitá Wi-Fi, dove si trovano anche altri terminali che usano la medesima rete per le loro trasmissioni. Tale rete é composta di molteplici AP connessi ad internet mediante infrastruttura wired. Il terminale B, per semplicitá, é collegato direttamente ad internet mediante collegamento wired.

Tra i due end point é frapposta una terza componente, un server esterno alle reti wireless. Questo server fa da proxy per il nodo mobile, il suo compito é quello di mantenere la comunicazione con tutte le interfacce del MN comunicando con il corrispettivo proxy lato client nel MN. Il proxy server é

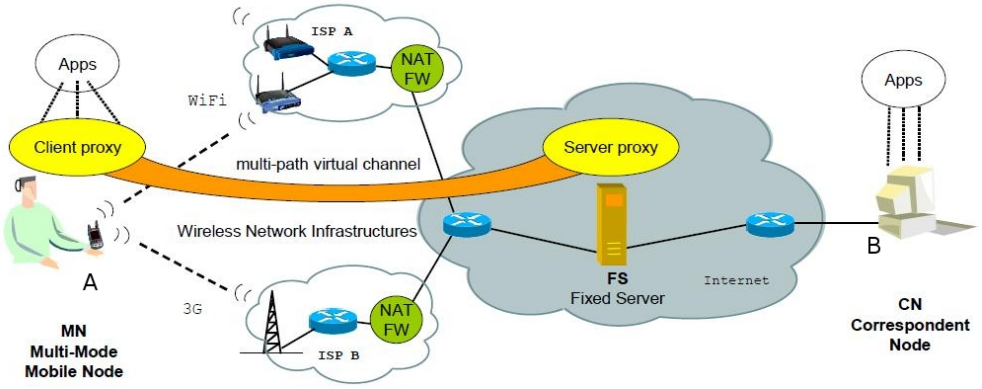

Figura 5.1: L'architettura descritta per la gestione della mobilitá ed il mantenimento del QoS.

un punto fisso della rete, quando le interfacce del MN effettuano il roaming e vengono riconfigurate, esso mantiene la connessione VoIP in corso e nasconde i cambiamenti di indirizzo al nodo B (Corresponded Node, CN). Il nodo mobile, inoltre, potrebbe essere coperto da firewall e NAT, grazie al proxy server questa limitazione puó essere superata.

É da notare che l'infrastruttura di rete esistente non deve essere alterata in alcun modo, soltanto il nodo mobile avrá bisogno di software aggiuntivo per sfruttare il meccanismo ABPS, e che il meccanismo proposto é sufficentemente generale da poter essere applicato in un contesto completamente wireless dove i due end system della comunicazione VoIP sono entrambi nodi mobili connessi a differenti reti wireless ed entrambi equipaggiati con piú di un'interfaccia.

Un aspetto fondamentale del sistema é il monitoraggio del QoS e la gestione delle molteplici interfacce wireless presenti nel terminale. La fase di monitoraggio serve a rilevare tempestivamente, e possibilmente prevenire, eventuali violazioni del QoS. Lo stato del collegamento Wi-Fi in uso dal servizio VoIP, nello specifico, i pacchetti persi, viene dedotto mediante informazioni fatte risalire dal livello data-link con un approccio cross-layer. Poiché la grande maggioranza di perdite di pacchetti avviene nel link wireless, ta-

le approccio é notevolmente piú vantaggioso ad esempio di un ACK livello applicazione che arriva dopo un intero round trip tra i due nodi in comunicazione. Una volta che il monitoraggio sospetta una perdita di dati, un'azione correttiva puó essere intrapresa: il pacchetto puó essere ritrasmesso da un'altra interfaccia in un lasso di tempo che é sufficente a rispettare i requisiti di interattivitá precedentemente menzionati. Le interfacce wireless non utilizzate per la trasmissione vengono mantenute possibilmente attive e configurate, pronte quindi ad essere usate nel caso l'interfaccia in uso introduca violazioni di QoS.

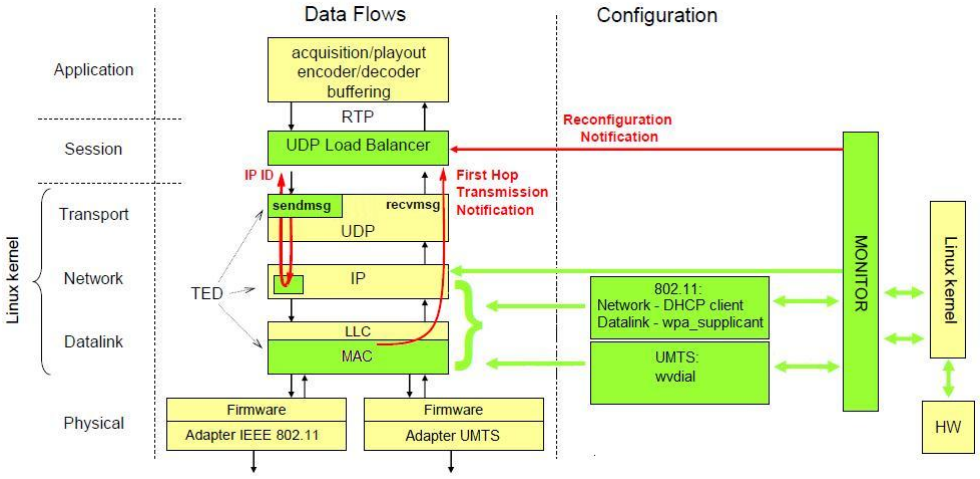

Figura 5.2: Struttura dell'architettura ABPS sul MN.

L'architettura del sistema proposto, come mostrata nella Figura 5.2, prevede le seguenti 4 componenti, ma in questa tesi verranno utilizzate su sistema operativo Android, solo le prime due componente, il TED (Trasmission Error Detector) e il Wvdial :

- **Transmission Error Detector (TED)** Considerato il componente piú importante del sistema, ha il compito di monitorare la trasmissione dei dati provenienti dall'applicazione VoIP. In particolare si assicura che tali dati siano stati ricevuti correttamente dall'AP, per ogni pacchetto spedito verifica la ricezione del relativo ACK proveniente dall'AP

stesso. Ha il compito ulteriore di notificare all'UDP Load Balancer (descritto in seguito) eventuali errori nella spedizione di uno specifico pacchetto.

- **Wvdial (TED per UMTS)** é un piccolo programma di utilitá che assiste nello stabilire connessioni ad Internet basate su modem. Wvdial comanda di selezionare il numero di un modem remoto e avvia pppd (Point-to-Point Protocol Daemon) per stabilire il collegamento a Internet.

- **UDP Load Balancer (ULB)** Questo componente, in base alle notifiche provenienti dal TED, gestisce una eventuale ritrasmissione dei pacchetti non ricevuti dall'AP. Inoltre stabilisce, sempre in base alle suddette notifiche, quale interfaccia Wi-Fi tra quelle disponibili debba essere usata per le future trasmissioni e/o ritrasmissioni di pacchetti.

- **Monitor** Gestisce e configura le interfacce Wireless disponibili sul MN, notificando al componente ULB quali interfacce sono attive e configurate.

Il Monitor, eseguito come applicazione separata, configura dinamicamente le interfacce wireless e relative regole di routing; comunica con il kernel Linux e con ULB, informando quest'ultimo quando una interfaccia é stata correttamente configurata oppure disabilitata in seguito ad un errore di trasmissione (Reconfiguration Notification, in Figura 3.3).

Il TED opera invece a livello data-link, piú precisamente nel protocollo MAC previsto dallo standard 802.11; per ogni pacchetto spedito verso l'AP tiene traccia della sua corretta ricezione (in caso di ricezione del realtivo ACK) o di un eventuale errore sulla trasmissione. L'esito di tale trasmissione viene comunicato ad ULB tramite la "First-hop Transmission Notification" in figura.

ULB riceve i pacchetti generati dall'applicazione VoIP e si occupa della loro trasmissione; usa un socket UDP per ogni interfaccia wireless attiva

e configurata dal Monitor. Tali socket vengono "legatí' alle loro interfacce mediante la primitiva bind in modo da assicurarsi che l'effettiva trasmissione del pacchetto avvenga attraverso l'interfaccia scelta.

ULB inoltre puó ricevere tre tipologie di notifica (indicate con linee non trattegiate in Figura 5.2):

- **Reconfiguration Notification** Proveniente dal Monitor, informa ULB che una interfaccia wireless é stata configurata ed attivata o che, viceversa, é stata disabilitata. Per ogni interfaccia attivata viene creato un socket UDP che viene legato a tale interfaccia mediante la primitiva bind. Per ogni interfaccia disabilitata viene chiuso il rispettivo socket ed ogni pacchetto spedito tramite questa interfaccia, per il quale non é stata ricevuta alcuna First-hop Transmission Notification dal TED, viene considerato perso.

- **ICMP Notification** Proviene dal protocollo ICMP e notifica l'eventuale perdita di pacchetti avvenuta nel tratto di comunicazione tra A e B.

- **First-hop Transmission Notification** Proviene dal TED, notifica per ogni pacchetto spedito se é stato correttamente ricevuto dall'AP oppure definitivamente scartato dal livello MAC.

Queste tre tipologie di notifiche permettono ad ULB di stabilire se un determinato pacchetto debba essere ritrasmesso (mediante un'altra interfaccia wireless) o definitivamente scartato. Permettono inoltre di realizzare un opportuno algoritmo per selezionare dinamicamente l'interfaccia wireless che offre maggiori garanzie in trasmissione.

Questo appena descritto rappresenta il middleware presente sul terminale mobile, il server proxy presenta invece una architettura molto piú semplice, composta solo dall'ULB. Quest'ultimo, in versione semplificata, controlla una sola interfaccia wired sul terminale fisso; assume che i pacchetti trasmessi non

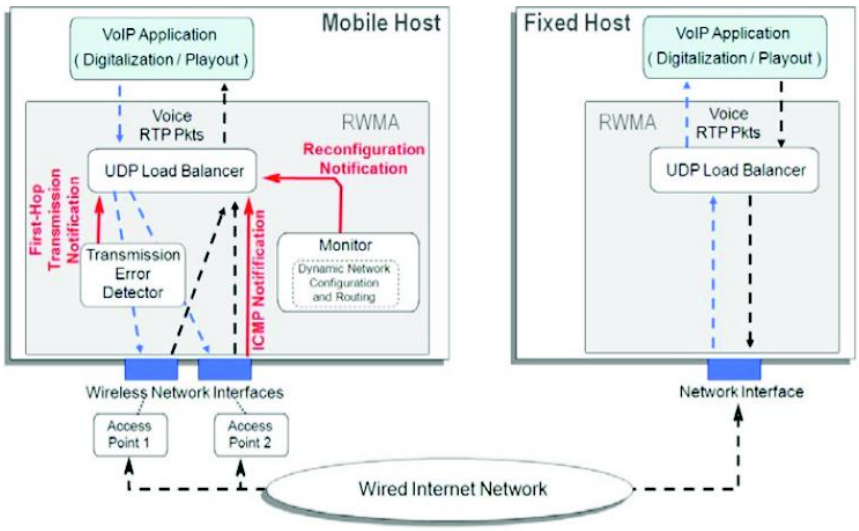

Figura 5.3: L'architettura descritta per la gestione della mobilitá ed il mantenimento del QoS.

vengano scartati dal first-hop wired. In sintesi il suo unico e principale compito é quello di tenere traccia degli indirizzi IP da cui provengono i pacchetti trasmessi dal nodo mobile, informazione essenziale per poter trasmettere correttamente i propri dati verso il MN.

5.3.1 Algoritmo di selezione delle interfacce Wi-Fi

ULB sfrutta le notifiche ricevute dalle altre componenti del sistema per realizzare un algoritmo di gestione e selezione delle interfacce Wi-Fi presenti nel terminale.

Tale algoritmo assume che un pacchetto trasmesso sia stato perso se riceve una opportuna notifica dal TED oppure se non riceve alcuna notifica entro 30 millisecondi. Il timeout é necessario perché il firmware di alcune interfacce Wi-Fi attualmente in commercio non informa il protocollo MAC se un determinato pacchetto é stato scartato, tuttavia quasi tutte informano quando un pacchetto é stato ricevuto dall'AP.

I pacchetti vengono spediti da ULB nello stesso ordine in cui sono stati ricevuti dall'applicazione VoIP, per la spedizione viene scelta l'interfaccia che puó fornire le maggiori garanzie di trasmissione. Una interfaccia configurata ed attiva si trova inizialmente nello stato funzionante; se viene ricevuto un errore ICMP o un Reconfiguration Notification l'interfaccia passa allo stato disabilitata. Se invece entro il timeout non viene ricevuta alcuna segnalazione per un pacchetto, o viene ricevuta una First-hop Transmission Notification che segnala la perdita di un pacchetto, l'interfaccia passa allo stato sospetta. Se l'interfaccia si trova nello stato sospetta e viene ricevuto un pacchetto dal destinatario B, o viene ricevuta una First-hop Transmission Notification riguardante la corretta trasmissione di un pacchetto, torna allo stato funzionante. Di conseguenza quando ULB deve trasmettere un pacchetto seleziona una interfaccia che sia nello stato funzionante, se non ve ne sono di disponibili allora seleziona una interfaccia nello stato sospetta, altrimenti scarta il pacchetto. Se é stato possibile selezionare una interfaccia per la trasmissione allora viene avviato il timeout; al suo scadere, se non sono state ricevute notifiche di avvenuta trasmissione, il pacchetto viene rispedito selezionando una interfaccia diversa. Anche in questo caso se non sono disponibili interfacce nello stato funzionante o sospetta il pacchetto viene scartato.

5.3.2 Considerazioni

Recentemente è stato realizzato un prototipo dell'architettura descritta operante su sistema operativo Linux con indirizzamento IPv4 e sono stati fatti esperimenti in uno scenario reale, essi hanno confermato la bontá del modello.

La presenza di una fase di monitoraggio continua permette una gestione tempestiva del QoS, mentre lo sfruttamento del multi-homing permette una gestione del roaming strettamente correlata alle effettive condizioni del canale wireless e concede il tempo necessario alle interfacce con segnale degradato di negoziare nuove connessioni con access point migliori. A differenza dello standard 802.11r, non tutti gli access point devono far parte dello stesso

extended service set e, a differenza del MobileIP, la connessione non subisce sostanziali rallentamenti a causa del roaming dell'unica interfaccia sfruttabile. Infine, tale soluzione é integrabile e compatibile con i servizi offerti dallo standard 802.11e.

Tali pregi tuttavia comportano alcuni difetti. L'assunzione che il terminale mobile sia dotato di connettivitá Wi-Fi multipla risulta ad oggi irrealistica, specie considerando dispositivi come smartphone e palmari, orientati ad avere sí piú interfacce wireless ma di diverso tipo, come UMTS, Wi-Fi, BlueTooth e Near field Communication.

Questo risulta essere il principale limite della soluzione proposta. Ulteriori problemi, seppur di minore impatto, riguardano il supporto richiesto nel terminale fisso, a fronte peró dell'innegabile vantaggio di poter lasciare inalterata l'infrastruttura wireless esistente, a differenza ad esempio di IEEE 802.11r che deve essere supportato da tutti i device. Inoltre, in uno scenario in cui entrambi i terminali sono mobili, si ritiene realisticamente necessaria l'introduzione di un sistema intermedio che funzioni da punto d'incontro tra le trasmissioni di entrambi i terminali.

Nel seguente capitolo verrá descritta piú in profonditá l'implementazione del transmission error detector con indirizzamento IPv6 su sistema operativo Android. Questo ha comportato la necessità di modificare il kernel dei sistemi Android per adattarli alle esigenze.

Capitolo 6

Progettazione TED

6.1 Obiettivo

L'obiettivo é lo studio e l'installazione, sul sistema operativo Android, di un modulo, denominato Transmission Error Detector (TED), che estende il funzionamento della tecnologia WiFi e che fa parte di una architettura per il supporto alla mobilità per applicazioni multimediali e VoIP. L'idea di base del modello descritto nel capitolo precedente é quella di usare piú interfacce per raggiungere la qualitá del servizio necessario al tipo di applicazioni che vogliamo supportare. Grazie ad un meccanismo cross layer, é possibile monitorare lo stato di un link wireless in anticipo rispetto ad una sua definitiva disconnessione; avendo l'accortezza di tenere preparate tutte le interfacce di rete disponibili si puó reagire tempestivamente alla degradazione del segnale su un collegamento e scegliendo il link da usare pacchetto per pacchetto la comunicazione tra nodo mobile e nodo fisso risulta piú robusta e reattivá ad eventuali cambiamenti di stato del canale radio. Precedentemente a questa tesi, un prototipo dell'architettura é giá stato sviluppato. Attualmente, il modello sfrutta il protocollo IPv4. Il passaggio a IPv6, benché graduale, é inevitabile: é bene quindi volgere l'attenzione ad esso, non trascurando peró il supporto alla versione 4 che attualmente é usata dalla stragrande maggioranza dei dispositivi in produzione. Il prototipo del modulo TED

dell'architettura é esteso al supporto di IPv6 su Android. Come precedentemente detto, il Trasmission Error Detector si occupa di monitorare lo stato delle connessioni con gli Access Point.

6.2 Trasmission Error Detector

Le applicazioni, per comunicare con il mondo, usano le astrazioni al proprio livello del modello ISO/OSI. Le astrazioni sono senza dubbio utili a nascondere la complessitá soverchiante dei meccanismi che permettono di inviare e ricevere dati attraverso la rete: con esse, in poche istruzioni e una quantitá ragionevole di tempo é possibile creare servizi e architetture che altrimenti bisognerebbero di capacitá, esperienze ed un impegno tali da rendere improponibile l'impresa.

Nel mondo wireless, peró, questo significa anche mascherare tutta una serie di problematiche legate alle limitazioni fisiche e tecniche dei meccanismi e protocolli utilizzati. Lo Wi-Fi, benché logicamente identico allo standard Ethernet, si differenzia in modo marcato da esso per quanto riguarda la velocitá e l'affidabilitá dei servizi offerti. La semplificazione data dai protocolli di livello tre e superiori si scontra con la necessitá di avere informazioni chiare e responsive sullo stato delle connessioni in modo da riparare tempestivamente a possibili cali di prestazioni.

In particolare, il segmento wireless, di solito il primo nel tragitto per i dati tra i due end system, é l'anello debole della connessione: in esso si possono avere una notevole perdita di pacchetti e quindi di tempo nel quale l'interfaccia di un nodo mobile cerca di ritrasmettere datagrammi o associarsi ad un nuovo access point. Invece che ostinarsi a fare ció, la decisione migliore potrebbe essere quella di utilizzare un'altra interfaccia prontamente disponibile ma che non viene sfruttata poiché non prevista dall'astrazione del modello OSI.

É sentito quindi il bisogno di penetrare queste astrazioni in modo da rendere disponibile ai livelli superiori informazioni riguardanti lo stato attuale

della linea. L'unico modo per fare ció é attraverso un meccanismo cross-layer, dove diversi livelli del modello ISO/OSI in concerto collaborino all'obiettivo comune del raggiungimento del QoS.

Il Trasmission Error Detector ha un ruolo di primaria importanza nella nostra architettura per il supporto alla mobilitá. Lo standard IEEE 802.11 invia un frame e attende un ACK, se tale conferma di avvenuto recapito non é ricevuta, il frame viene rispedito per un certo numero di volte. Il TED si propone di esaminare ogni pacchetto inviato per ogni interfaccia wireless e carpire nel layer 2 IEEE 802.11 la ricezione dell'ACK, e quindi del frame associato, queste notifiche poi verranno valutate dagli altri componenti, come l'UDP load balancer, che le utilizzerá per decidere se ritrasmettere il pacchetto e in tal caso con quale interfaccia wireless. La notifica dell'ACK IEEE 802.11 é nascosta ai layer superiori e non é direttamente conoscibile dalle applicazioni, quindi, per implementare il meccanismo, va modificato lo stack di rete.

6.2.1 Stack rete di Android

Il sistema operativo Android condivide ed amplia lo stack di rete di Linux. Linux é il piú famoso sistema operativo open source. Esso é stato inizialmente concepito e creato da Linus Torvalds nel 1991 come hobby; in quel periodo non esistevano sistemi operativi libereramente accessibili: BSD era ancora legato ad una licenza proprietaria, MINIX, il sistema operativo creato da Andrew S. Tanenbaum per scopi educativi, era a pagamento, seppure ad un costo molto accessibile, e GNU era, ed é tuttora, ad uno stato non maturo per l'uso giornaliero. Negli anni successivi, Linux ha generato sempre piú interesse, all'inizio da appassionati poi sempre di piú dal realtá professionali fino a diventare al giorno d'oggi uno dei sistemi operativi di maggiore successo. Anche se non originariamente pensato per essere portabile, Linux puó essere utlizzato in un vasto numero di architetture e contesti: dai supercomputer (é usato nel 91.8% dei sistemi nella lista TOP500) ai server di piccole societá, si ricorda che Linux é la prima lettera nell'acronimo LAMP

(Linux, Apache, MySQL, PHP), e massivamente nei data center di grosse aziende come Google e Facebook, dall'embedded, notevole ad esempio l'uso nei cellulari basati su MeeGo, webOS e il successo sorprendente di Android, ad ambienti classici desktop.

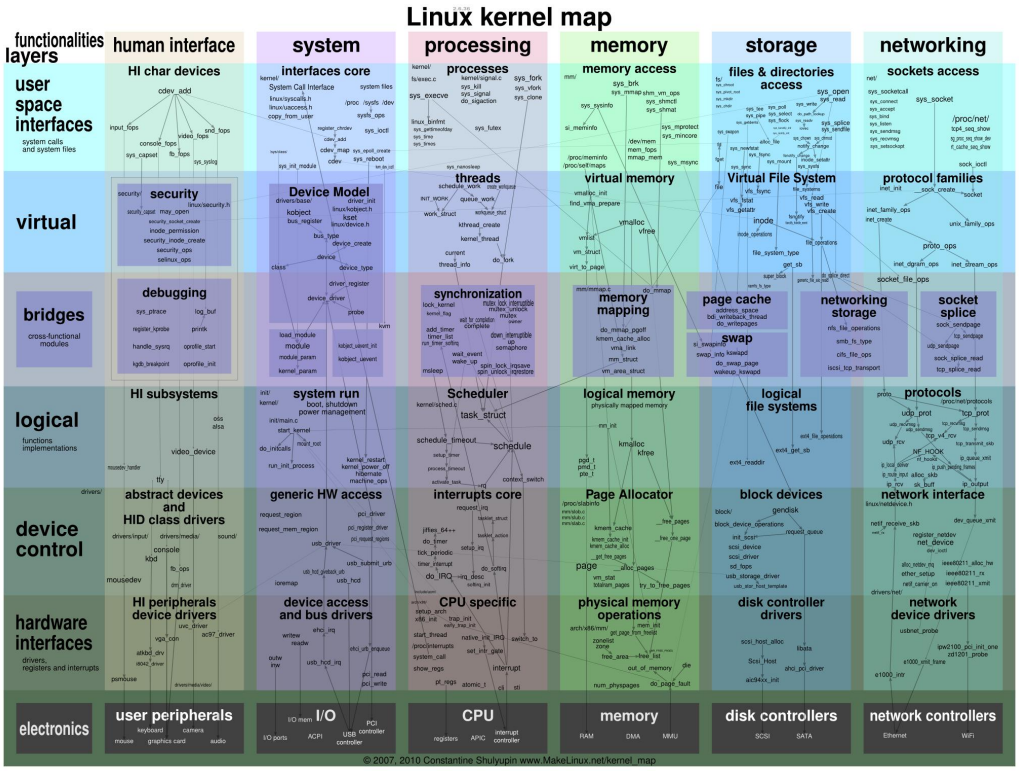

Figura 6.1: Panoramica dei sottosistemi del kernel Linux 2.6 e delle loro interazioni.

Linux é un kernel monolitico, supporta il multitasking di tipi preemptive, la memoria virtuale con paginazione e segmentazione, librerie condivise, il threading, diversi protocolli di rete ed altro ancora. I driver e le estensioni del kernel lavorano nel kernel space (ring 0 nella maggior parte delle CPU), con pieno accesso all'hardware, sebbene alcune eccezioni lavorino in user space. Il sistema grafico che la maggior parte delle persone utilizza con Linux non lavora nel kernel space.

Essendo Linux open source con licenza GPL versione 2, si ha liberamente accesso ai suo sorgenti ed é possibile modificare ogni sua parte. Il suo uso quindi é una scelta naturale per l'implementazione del meccanismo cross-layer sopra descritto. Qui di seguito viene descritto lo stack di rete di Linux e le modifiche apportate ad esso per realizzare il Transmission Error Detector. Quanto viene descritto si riferisce alla versione 2.6.36 di Linux ma é largamente applicabile a tutte le revisioni 2.6, dove lo stack é notevolmente cambiato rispetto alla precedente versione, la 2.4. La Figura 6.1 offre una panoramica di tutti i sottosistemi in cui é diviso il kernel 2.6 di Linux. La parte piú di interesse per questa tesi é il sottosistema Networking [29].

Prima di andare a descrivere nel dettaglio lo stack di rete, é bene concentrarsi sulle strutture dati utilizzate. Una struttura dati critica, forse la piú importante nello stack di rete di Linux, é `sk_buff` (socket buffer): essa si occupa di mantenere tutte le informazioni riguardanti un pacchetto che é stato ricevuto o sta per essere spedito cosí come il payload e gli header del pacchetto stesso.

Uno dei problemi di avere diversi livelli nei protocolli di rete che usano ognuno i servizi dell'altro é che ogni protocollo ha bisogno di modificare un pacchetto in modo da aggiungere o eliminare gli header relativi al suo livello. Una soluzione a ció é ogni volta copiare il buffer dal livello precedente e manipolarlo secondo le esigenze del livello corrente. Questa operazione peró avrebbe un grosso impatto sulle prestazioni che sono critiche per uno stack di rete che deve gestire interfacce con velocitá anche dell'ordine di Gigabit al secondo. Linux risolve in maniera pragmatica il problema usando lo stesso buffer, `sk_buff` appunto, per ogni livello, dal piú alto fino ai device driver. Ogni livello riconosce la posizione del proprio header tramite i campi `h`, `nh` e `mac` rispettivamente per i livelli trasporto, rete, e data-link, il campo `data` invece punta alla posizione dove é memorizzato il payload del pacchetto.

Poiché il payload ha diversi significati a seconda di quale protocollo si sta trattando correntemente, il campo `data` viene spostato a seconda del livello a cui si riferisce al momento. Quando si riceve un pacchetto, la funzione

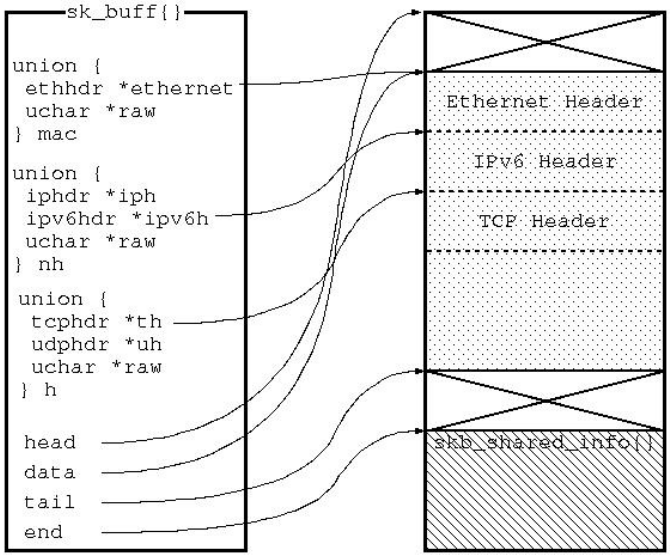

Figura 6.2: La struttura dati sb_buff e come vengono rappresentati gli header in essa.

responsabile di trattare il livello n riceve un buffer dal livello n-1 con il campo `data` che punta all'inizio dell'header del livello n. La funzione che gestisce il layer n inizializza il proprio puntatore all'header per il suo livello (ad esempio il campo `nh` per l'L3) per preservare il capo `data`, esso infatti sará perduto durante la gestione del livello successivo, dove `data` sará inizializzato a un differente offset nel buffer. Nello spedire un pacchetto, il processo viene invertito con la complessitá ulteriore di aggiungere un nuovo header ad ogni livello.

Appendere un header di un livello ad un buffer esistente é molto piú efficente che copiarlo. Poiché man mano che si scende di livello da quello trasporto a quello data-link un header va aggiunto in testa, il kernel lascia la possibilitá di riservare spazio in testa ad un socket buffer tramite la funzione `skb_reserve`.

Il socket buffer mantiene diversi altri campi generici: il time stamp che indica quando un pacchetto é stato ricevuto, il device, virtuale e non, dove

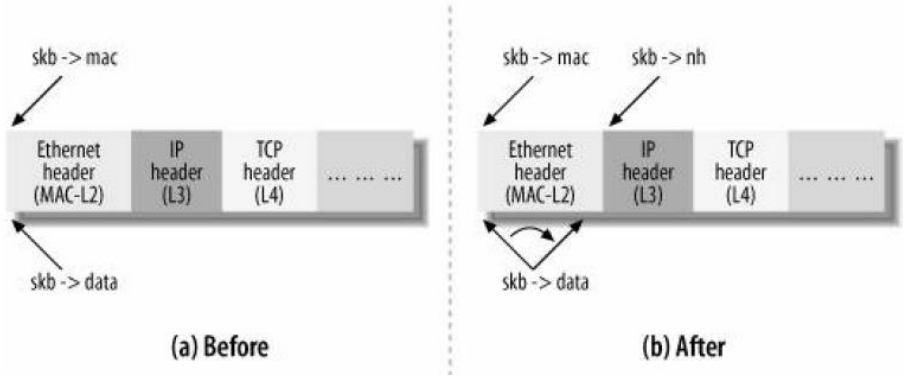

Figura 6.3: Inizializzazione del puntatore agli header e dati quando ci si muove da un layer ad un altro.

é stato ricevuto, il socket a cui esso appartiene, la sua destinazione, l'SKB control block, usato per inserire informazioni private per ogni livello, la sua prioritá, informazioni per il QoS e per le funzionalitá del firewall (Netfilter) e dati per il management della memoria, come flag che indicano se il buffer é stato clonato e quanti puntatori sono associati ad esso. Altri campi sono utili a facilitare la ricerca e l'organizzazione della struttura dati stessa, questo é il caso dei campi per il mantenimento nelle liste. Ogni socket buffer, infatti, é un elemento di una lista circolare doppiamente lincata, con i classici campi `prev` e `next` e con l'aggiunta del puntatore `list` che indica il primo elemento della lista. Il primo elemento, la sentinella, é di tipo `skb_head` ed ha i primi due elementi identici alla normale `sk_buff`, ció rende possibile la loro coesistenza nella stessa lista.

Quando un buffer deve essere processato indipendentemente da diversi consumatori, essi potrebbero aver bisogno di cambiare il suo contenuto della struttura `sk_buff`. In generale non si ha bisogno di eseguire anche una copia completa del buffer dati associato, viene incrementato quindi un campo, `users`, che fa da reference count e indica quante clonazioni sono state fatte, quando si vuole deallocare il buffer si usa il metodo `free_skb` che controlla che il campo `users` sia uguale a uno e solo in tal caso lo spazio viene liberato.

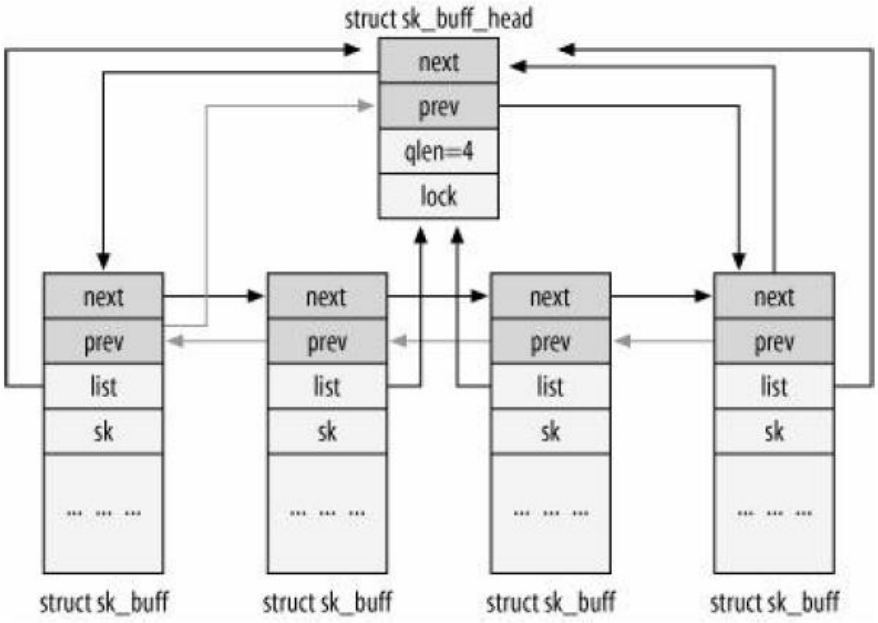

Figura 6.4: Esempio di una lista di socket buffer.

Qui di seguito sará descritto il percorso che fa un pacchetto UDP da quando un'applicazione vuole spedire dei dati fino ad arrivare al livello data-link, dove viene preso in carico dal sottosistema 802.11.

Un'applicazione usa l'interfaccia socket per spedire dati. Questa API fornisce un ricco insieme di opzioni per interagire con la rete. Alcune delle funzioni piú comuni sono `send`, `sendto`, `sendmsg`, `write` e `writev`. Ad esempio, la funzione `write` prende tre argomenti, il file descriptor del socket, il buffer dei dati da inviare e la sua lunghezza; la funzione `writev` ha lo stesso scopo della `write` ma permette di usare piú buffer non contigui tra loro attraverso il meccanismo scatter-gather, e la funzione `sendmsg` permette anche di inviare dati *ancillary*, cioé ulteriori informazioni di controllo. In ogni caso, quando una di queste funzioni viene chiamata, il controllo raggiunge la system call `sock_sendmsg` che fa parte del socket interface layer.

Il socket interface layer si occupa essenzialmente di interfacciarsi con il livello applicazioni e di trasporto per creare (`socket`) e gestire socket (`bind`, `listen`, `accept`, ecc.) e redirigere le chiamate di sistema per l'invio e rice-

zione di dati verso il giusto protocollo di livello inferiore. Quindi, se si tratta di TCP, viene chiamata la funzione `tcp_sendmsg`, se si tratta di UDP invece, il controllo passerá a `udp_sendmsg`. Si entra quindi nel livello di trasporto.

Nel caso di interesse per questa tesi, UDP[1], il livello di trasporto ad esso associato si occupa di ricevere i dati in user space e convertirli in kernel space, in particolare, la funzione `udp_sendmsg` esegue diversi controlli, sulla destinazione e sulle opzioni, per poi passare momentaneamente il controllo a `ip_append_data`, che prepara il buffer user space in tanti `sk_buff` dipendentemente dalla MTU e li inserisce nella coda write del socket, e successivamente a `udp_push_pending_frames`. `udp_push_pending_frames` genera l'header UDP per il pacchetto da inviare e passa il controllo al livello rete.

Il livello IP riceve il pacchetto e costruisce l'header per esso. Questo livello su prende cura del routing, sia per i pacchetti generati dallo stesso host che da quelli in transito: se il pacchetto esaminato é destinato ad un indirizzo esterno allora é consegnato al livello data-link, altrimenti se deve essere recapitato all'host stesso é ridirezionato ai livelli superiori.

Quando la funzione `ip_push_pending_frames` é richiamata, ad esempio dal livello superiore UDP in `udp_push_pending_frames`, viene preparato l'header IP per ogni pacchetto in coda del socket, se necessario frammenta il pacchetto tramite il metodo `ip_fragment`, dopo che diversi controlli sono stati eseguiti, é chiamata `ip_route_output_flow` che é la funzione principale che si prende cura del routing. `ip_route_output_flow` richiama `ip_route_output_key`, essa cerca prima nella route cache (un'area dove sono memorizzate le rotte recentemente accedute), se una rotta non é ancora stata trovata allora si continua cercando nella Forwarding Information Base (FIB). `ip_route_output_key` é pensata per il ritrovo veloce della rotta, se tutti i suoi tentativi falliscono, allora il controllo passa ad una funzione piú esaustiva, `ip_route_output_slow`. Brevemente, alla fine del routing puó succedere una tra queste tre possibilitá:

[1]Qui di seguito viene descritto il livello trasporto e rete per IPv4, la controparte v6 si avvale di funzioni analoghe.

- se il pacchetto deve essere inoltrato allora verrá richiamata la funzione `ip_forward`;
- se la rotta é ancora non risolta anche dopo che il processo é concluso, verrá richiamata la funzione `ip_output`;
- se il routing é stato risolto dopo questo stadio, il controllo passerá al livello data-link e verrá richiamata la funzione `dev_queue_xmit`.

Il layer data link é responsabile per una serie di operazione che vanno oltre il solo inoltro del pacchetto al device con il quale sará spedito. Questo livello é a volte detto queueing layer in quanto la maggior parte delle discipline di accodamento vengono implementate in questa sezione, cosí come il traffic shaping. La funzione `dev_queue_xmit` é chiamata da ogni livello che intende consegnare dati a una destinazione esterna: essa registra il socket buffer nella coda del device, viene dunque chiamata la routine `queue_disc`, a questo punto, se essa conferma che lo stato dell'interfaccia é up, cerca di ottenere il controllo esclusivo del device e passa l'esecuzione alla funzione specifica `dev->hard_start_xmit` che nel caso di interfaccie wireless che usano il softmac, si appoggia al sottosistema mac80211.

6.2.2 Sottosistema mac80211

Prima del 2005, l'implementazione del protocollo IEEE 802.11 in Linux era lasciata ai device driver e firmware che gestivano completamente tutti i suoi meccanismi. Ogni scheda wireless quindi aveva un modo proprio di gestire il Wi-Fi che non era condivisibile con le altre, ció rendeva impegnativa lo sviluppo di driver per le schede in commercio senza supporto diretto dei vendor. A partire dal 2005, Jhoannes Berg ha rilasciato il sottosistema mac80211 [35], il Media Access Control (MAC) Sublayer Management Entity che sposta i compiti di autenticazione, associazione, e management Wi-Fi al di fuori dei device driver che ne fanno uso. Oggi la maggior parte dei device supporta questo sistema e in molti dei vecchi dispositivi i device dirver sono stati riscritti tenendo conto di esso.

Tornando a descrivere il percorso che un pacchetto fa nel kernel per essere spedito con una scheda wireless, ad un certo punto la funzione di livello data link `hard_start_xmit` passa il controllo al sottosistema mac80211 che ha il punto d'ingresso in `ieee80211_xmit`, qui viene calcolato l'header 802.11 ed sono eseguite una serie di operazioni tramite degli handler:

- **ieee80211_tx_h_dynamic_ps**, **check_assoc** e **ps_buf**: si occupuano di gestire il power saving;
- **ieee80211_tx_h_select_key**: é usata per selezionare una chiave per la crittografia;
- **ieee80211_tx_h_rate_ctrl**: seleziona il bit rate per il quale il frame deve essere spedito;
- **ieee80211_tx_h_michael_mic_add**: é usata in caso di crittografia basata su TKIP;
- **ieee80211_tx_h_sequence**: calcola il sequence number del frame;
- **ieee80211_tx_h_fragment**: se necessario frammenta il frame alla MTU gestita dalla scheda wireless e dall'access point associato;
- **ieee80211_tx_h_stats**: compila delle variabili usate per tenere la contabilitá di alcune statistiche;
- **ieee80211_tx_h_encrypt**: cripta il frame se necessario con l'algoritmo desiderato (WEP, TKIP, CCMP, AES);
- **ieee80211_tx_h_calculate_duration**: calcola il campo duration del frame, cioé per quanto tempo sará occupato il canale durante la trasmissione.

Un frame a questo punto viene passato al device driver di competenza ed effettivamente spedito. Il buffer peró non viene eliminato: se l'ACK non viene ricevuto, mac80211 ritenta l'invio per un numero di volte dipendente

dal device e dalle condizioni del canale, comunque, alla fine del processo la funzione `iieee80211_tx_status` viene chiamata per raccogliere l'esito della trasmissione, in particolare statistiche come quante volte il frame é stato ristrasmesso (il retry count) e se é stato effettivamente consegnato all'access point. Dopo di ció, sia che il frame sia stato scartato o che sia stato consegnato, il socket buffer associato puó venire deallocato.

6.2.3 Implementazione di TED

Il trasmission Error Detector é stato implementato estendendo le opzioni dei socket. Nessuna nuova system call é stata introdotta: piuttosto, un programma che vuole sapere se i suoi datagrammi UDP sono stati inviati con successo usa la classica funzione:

```
ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags)
```

Il parametro `msg` é una struttura usata per specificare i buffer che devono essere inviati (questa funzione infatti supporta lo scatter-gather I/O) e dei dati ausiliari (detti *ancillary*): in essi possono essere fissate un gran numero di opzioni come il supporto alla sicurezza, routing, timestamp, traceroute ed altre ancora. Alle altre opzioni specificabili, ne é stata aggiunta un'altra, `SO_EE_ORIGIN_LOCAL_NOTIFY`, per indicare che l'applicazione é interessata ad essere informata sulla consegna al primo hop. Insieme ad essa viene anche specificato un puntatore ad un intero che sará inizializzato dal kernel nel processare la chiamata `sendmsg` all'id che identifica univocamente il pacchetto.

Per ricevere la notifica vera e propria, ci si mette in ascolto della coda degli errori per il socket con la funzione:

```
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags)
```

specificando come `flags` il valore `MSG_ERRQUEUE`, tutti gli errori sia di tipo ICMP che locali verrano recapitati all'applicazione, in particolare, anche il messaggio di avvenuta consegna all'access point. Questo messaggio

comprende: l'id del datagramma a cui la notifica é associata, informazioni per riconoscere il frammento a cui si riferisce (il datagramma UDP infatti potrebbe essere stato frammentato, l'applicazione quindi riceve un messaggio per frammento che é riconosciuto tramite l'offset nel pacchetto UDP e la sua lunghezza) e naturalmente la notifica se il pacchetto é stato effettivamente recapitato o no.

Sul lato kernel, sono apportate meno modifiche possibili allo stack di rete in modo da rendere piú semplice l'integrazione con futuri aggiornamenti. Il problema di far risalire l'ACK livello 2 dell'IEEE 802.11 fino al livello applicazione deve essere affrontato in modo cross-layer, andando ad esaminare cioé ogni livello e inserendo i cambiamenti necessari; il meccanismo del Trasmission Error Detector quindi non puó essere attuato con un modulo del kernel, bensí con una patch.

Il primo problema da affrontare é l'identificazione del datagramma UDP: poiché la chiamata `sendmsg` non é bloccante fino all'effettiva uscita del pacchetto dall'interfaccia wireless ma al massimo fino alla consegna alle code in kernel space, la notifica dell'avvenuto invio all'access point avviene necessariamente in un secondo momento (nella ricezione dalla coda degli errori del socket). É necessario dunque che la funzione `sendmsg` restituisca all'applicazione un identificativo del pacchetto che si intende spedire, tale id sará successivamente usato per riconoscere l'informazione associata nella coda degli errori. Nel caso particolare di IPv4, un candidato per essere usato come id é sicuramente il campo Identification dell'header IP. Malauguratamente, questo campo é stato eliminato nella versione 6 dell'Internet Protocol: se si vuole un meccanismo che valga per entrambi i potocolli questa strada va quindi scartata. Come detto in precedenza peró, si puó notare che Linux usa la struttura `sk_buff` in ogni parte dello stack di rete. Un socket buffer é associato ad un pacchetto: dopo la sua creazione, per questioni di efficenza, ogni livello lavora sullo stesso socket buffer (salvo casi di frammentazione). Si é deciso quindi di aggiungere un campo ad hoc alla struttura `sk_buff`,

skbID, ed usare esso per superare questa prima difficoltá.

Quando un socket buffer arriva alle funzioni udp_push_pending_frames o udp_v6_push_pending_frames, si controlla se la struttura msghdr con cui é stata chiamata la funzione sendmsg contiene dei dati ancillary con l'opzione SO_EE_ORIGIN_LOCAL_NOTIFY: in tal caso viene generato un nuovo id ed assegnato al socket buffer appena creato. Questo identificativo viene anche salvato nei dati ancillary in user space cosí che l'applicazione possa memorizzarlo al ritorno della chiamata. Il socket buffer quindi attraversa ogni livello dello stack di rete. Nel livello IP é possibile che avvenga una sua frammentazione: in pratica il datagramma é troppo grande per essere spedito in un'unica trasmissione, cosí viene diviso in diversi sk_buff separati tra loro. Per evitare problemi, ognuna di queste situazioni é stata identificata e, al momento della frammentazione, ogni buffer generato viene marcato con lo stesso skbID di quello originale. Alla fine, l'applicazione potrá ricevere piú di una notifica per singolo datagramma, comunque, ció non porterá confusione in quanto essa sará in grado di riconoscere a quale frammento ogni notifica si riferisce grazie alle informazioni di corredo come l'offset e la lunghezza del frammento. Nel percorso verso i livelli inferiori, il socket buffer arriva alla funzione ieee80211_tx del sottosistema mac80211. In essa, gli handler precedentemente descritti vengono chiamati ad apportare le ultime modifiche al socket. In particolare, l'handler ieee80211_tx_h_sequence calcola e aggiunge il sequence number del frame livello 2. É a questo punto che le informazioni sul pacchetto UDP sono salvate in una coda: é importante infatti avere il sequence number del frame associato cosí che esse possono essere identificate univocamente quando si riceverá l'ACK. Nella coda vengono posti:

- l'offset dal pacchetto IP,
- la lunghezza del payload del frame da spedire,
- un flag che indica se seguono altri frammenti,
- l'identificativo del socket buffer associato,

- il timestamp del momento di invio del frame,
- il sequence number che identifica univocamente il frame.

La funzione `ieee80211_tx_status` é chiamata in un momento successivo quando é ricevuto un acknowledgement o quando dopo diversi tentativi il sottosistema mac80211 ha deciso di scartare il frame. Essa contiene tutte le informazioni che vogliamo sullo stato della trasmissione di un frame: ACK ricevuto o no, numero di ritrasmissioni eseguite ed altri dati di corredo. Il sequence number del frame per questi dati viene ricercato nella coda sopra menzionata, le informazioni salvate vengono ritrovate, tolte dalla coda, associate ai nuovi dati e, tramite l'id del socket buffer che era stato conservato nell'elemento della coda, la notifica della trasmissione puó essere fatta arrivare tramite un messaggio ancillary all'applicazione.

Capitolo 7

TED per UMTS : Wvdial

7.1 Obiettivo

L'obiettivo é lo studio e l'installazione, sul sistema operativo Android, di un programma, denominato Wvdial (o WaveDial), che estende il funzionamento della tecnologia UMTS e che fa parte di una architettura per il supporto alla mobilità per applicazioni multimediali e VoIP. L'idea di base del modello descritto precedetemente é quella di usare piú interfacce per raggiungere la qualitá del servizio necessario al tipo di applicazioni che vogliamo supportare.

7.2 WvDial

Wvdial é un piccolo programma di utilitá che assiste nello stabilire connessioni ad Internet basate su modem che é compreso in alcune importanti distribuzioni. Wvdial seleziona il numero di un modem remoto e avvia pppd (Point-to-Point Protocol Daemon) per stabilire il collegamento a Internet. Questo software é l'unico riconosciuto per la sua capacitá di configurare qualsiasi tipo o quasi di modem 3G, ed é quindi di fondamentale importanza per il supporto alla mobilitá. Di base su Android non sono presenti strumenti per l'utilizzo di interfacce 3G per la connessione ad internet. Per superare questo

problema si é reso necessario fare il porting di wvdial, uno dei programmi piú diffusi in ambiente linux per la gestione di connessioni ad internet basate su modem, come per l'appunto le tecnologie 3G. Un tipico sistema Android non supporta il set completo standard di librerie GNU, e nel caso del C++ vi é solo una implementazione parziale delle STL. Tutto ció rende difficile il porting di applicazioni Linux o librerie per Android.

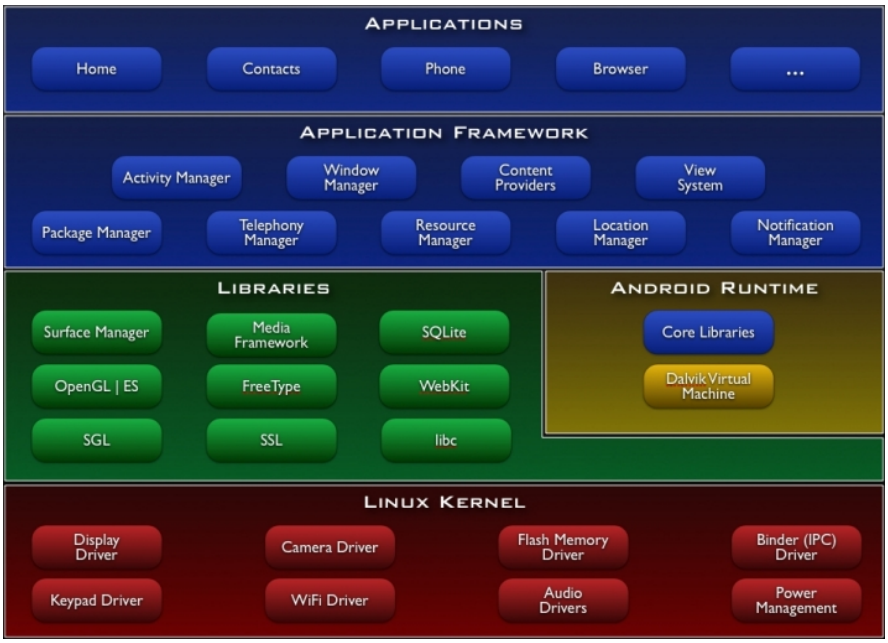

Figura 7.1: Architettura di Android.

I motivi che hanno portato gli sviluppatori di Android a fare questa scelta sono principalmente 3:

- licenza: si è voluto lasciare fuori dall'user space tutto il codice con licenza GPL.
- dimensioni: queste librerie devono essere caricate in ogni processo, perció devono essere di dimensione ridotta, soprattutto quando si utilizza dispositivi come cellulari.
- velocità: Android nasce per cellulari, quindi per dispositivi che non hanno una elevata potenza di calcolo (messi a confronto con dei PC),

perció é necessario avere delle librerie che siano il piú leggere possibili per quello che riguarda la velocitá di esecuzione.

Wvdial si appoggia sulle librerie libuniconf, le quali si basano su un'altra serie di librerie ed eseguibili, tra cui libwvstreams e pppd. L'albero completo delle sue dipendenze é molto vasto, e quindi un primo passo é stato quello di capire quali erano opzionali e quali invece obbligatorie. Non era chiaro come si dovessero generare questi file, e soprattutto come generarli per l'architettura ARM.

7.2.1 Cross-compile

La cross-compilazione é la tecnica mediante la quale si compila un codice sorgente con un cross-compilatore, ottenendo cosí un file binario eseguibile su di un elaboratore con architettura diversa da quella della macchina su cui si é lanciato il cross-compilatore (o toolchain) stesso. La cross-compilazione viene usata anche quando é necessario compilare un programma per un sistema operativo differente dalla macchina su cui si trova il compilatore e linker. Questa tecnica é usata solitamente per compilare applicazioni per sistemi embedded dove le risorse sono generalmente molto limitate e non sarebbe quindi possibile effettuare la normale compilazione.

Il cross-compilatore utilizzato per compilare programma e librerie é:

- gcc-4.4-arm-linux-gnueabi

E' possibile scaricare il seguente pacchetto in ambiente linux eseguendo le seguenti istruzioni:

1. Modificare il file sources.list attraverso il comando, da terminale:
    - sudo /etc/apt/sources.list.
2. Aggiungere le seguenti righe, alla fine del file:

   (a) deb http://ubuntu.mirror.cambrium.nl/ubuntu/ precise main universe

   (b) deb-src http://ubuntu.mirror.cambrium.nl/ubuntu/ precise main universe

3. Lanciare in seguito il comando:

   (a) sudo apt-get update

4. Installare il cross-compiler:

   - sudo apt-get install g++-4.4-arm-linux-gnueabi binutils-arm-linux-gnueabi

NOTA : qualora ci fossero problemi di "Cache-Limit",da terminale posizionarsi nella directory /etc/apt/apt.conf.d ed eseguire "sudo echo "APT::Cache-Limit 33554432;" >> 20archive".

Nel caso si vogliano compilare applicazioni piú complesse, come wvdial o che necessitano comunque di particolari librerie extra, é necessario aggiungere alla propria toolchain anche queste librerie preventivamente cross-compilate.

7.3 Cross-Compilazione Sorgenti

Il sistema operativo utilizzato come ambiente di sviluppo é windows XP inoltre si é lavorato su sistemi operativi Unix, quali Xubuntu10.04 e Xubuntu 12.04 per la manipolazioni e compilazioni dei sorgenti. Il sistema operativo windows é stata la base di partenza su cui si é virtualizzato, attraverso l'uso di Oracle VirtualBox, i rispettivi sistemi Unix.

7.3.1 Software Necessari

Come detto in precedenza, Android non contiene alcuni pacchetti e alcune librerie che invece si trovano normalmente di base in un sistema Linux.

Per questo il porting di wvdial su Android comporta l'inclusione di alcune librerie non presenti (libwvstreams), oltre che la modifica di alcune definizioni del Makefile.

I pacchetti aggiunti per poter utilizzare wvdial sono i seguenti:

- libwvstreams
- libuniconf
- pppd
- openssl
- zlib
- crypto

Di seguito saranno riportati quanto piú fedelmente possibile tutti i passi.

Zlib

Zlib é una libreria software usata per la compressione dei Dati. Zlib é stata scritta da Jean-Loup Gailly e Mark Adler ed é una astrazione della algoritmo di compressione DEFLATE utilizzano nel loro programma di compressione dei file gzip. Zlib é anche una componente fondamentale di molte piattaforme software, tra cui Linux, Mac OS X e iOS. É stato anche usato in console di gioco come la PlayStation 3, Wii e Xbox 360.

La versione di zlib utilizzata é la 1.2.8 ed é possibile scaricarla attraverso il seguente link:

- http://zlib.net/zlib-1.2.8.tar.gz

A questo punto si puó scompattare zlib e aprire un terminale nella medesima cartella. La cross-compilazione avverrá seguendo le prossime istruzioni:

- ./configure
- export LDSHARED=″arm-linux-gcc -shared -Wl,-soname,libz.so.1″
- make CC=arm-linux-gnueabi-gcc-4.4 AR=arm-linux-gnueabi-ar RANLIB=arm-linux-gnueabi-ranlib
- sudo cp libz.a /usr/arm-linux-gnueabi/lib

OpenSSL

OpenSSL é un'implementazione open source dei protocolli SSL e TLS. Le librerie di base (scritte in linguaggio C) eseguono le funzioni crittografiche principali. Nei diversi linguaggi di programmazione sono disponibili procedure che permettono di accedere alle funzioni della libreria OpenSSL. É disponibile per la maggior parte dei sistemi operativi unix-like, inclusi GNU/Linux e Mac OS X, ed anche per Microsoft Windows. OpenSSL é originariamente basato sulle librerie SSLeay di Eric Young e Tim Hudson. La libreria OpenSSL crypto implementa implementa una vasta gamma di algoritmi crittografici usati in vari standard di Internet. I servizi forniti da questa libreria sono usati nelle implementazioni OpenSSL di SSL, TLS e S/MIME, e sono stati anche utilizzati per implementare SSH, OpenPGP, e altri standard crittografici.

La versione di openssl utilizzata é la 1.0.1e ed é possibile scaricarla attraverso il seguente link:

- http://www.openssl.org/source/openssl-1.0.1e.tar.gz

A questo punto si puó scompattare openssl e aprire un terminale nella medesima cartella. La cross-compilazione avverrá seguendo le prossime istruzioni:

- ./Configure dist -fPIC
- export cross=arm-linux-gnueabi-

- make CC=$crossgcc-4.4 AR=$crossar r RANLIB=$crossranlib
- sudo cp libssl.a libcrypto.a /usr/arm-linux-gnueabi/lib
- sudo apt-get install libssl-dev
- sudo cp -r /usr/include/openssl /usr/armlinuxgnueabi/lib

NOTA: l'istruzione "sudo apt-get install libssl-dev" é necessaria per evitare di inserire i file con estensione ".h" uno alla volta nella directory "/usr/arm-linux-gnueabi/lib". La directory che avrebbe dovuto contenere i file libreria, "openssl-1.0.1e/include/openssl", é composta esclusivamente da collegamenti. Quindi é stato utilizzato questo escamotage per questione di tempo.

Wvstreams

WvStreams é un insieme di librerie composto da un numero di parti. Tra loro ci sono WvIPStreams (WvTCPStream e WvUDPStream), WvStrings, WvCrypto Streams (tra cui un modo semplice per aggiungere il supporto SSL per le applicazioni), UniConf (un sistema di configurazione universale), WvLog (semplice gestione di log file), e tutta una serie di altre classi. Queste sono le classi di base utilizzate per costruire programmi come WvDial, TunnelVision, FastForward, KWvDial e retchmail.

La versione utilizzata é la 4.6.1 ed possibile scaricare il codice sorgente di wvstreams attraverso il sito ufficiale:

- https://wvstreams.googlecode.com/files/wvstreams-4.6.1.tar.gz

A questo punto si puó scompattare wvstreams e aprire un terminale nella medesima cartella eseguendo il comando:

- ./configure

La libreria presenta diversi errori nei codici sorgenti. Si andranno ad effettuare opportune modifiche al software:

Nel file wvstreams-4.6.1/crypto/wvx509.cc riga 1160:

```
if (ext)
    {
-         X509V3_EXT_METHOD *method = X509V3_EXT_get(ext);
+         X509V3_EXT_METHOD *method = (X509V3_EXT_METHOD *) X509V3_EXT_get(ext);
          if (!method)
          {
              WvDynBuf buf;
```

Il segno "-" indica la riga da togliere,mentre il segno "+" quella da aggiungere.

Nel file wvstreams-4.6.1/ipstreams/wvunixdgsocket.cc riga 1:

```
#include "wvunixdgsocket.h"
-#ifdef MACOS
+#if defined(MACOS) || defined(__GNUC__)
 #include <sys/types.h>
 #include <sys/stat.h>
 #endif
```

Nel file wvstreams-4.6.1/streams/wvatomicfile.cc riga 4:

```
#include "wvfileutils.h"
#include "wvstrutils.h"

-#ifdef MACOS
+#if defined(MACOS) || defined(__GNUC__)
+#include <sys/types.h>
 #include <sys/stat.h>
 #endif
```

Nel file wvstreams-4.6.1/Makefile riga 133:

```
# libwvstreams: stream/event handling library
 #
 TARGETS += libwvstreams.so
-TARGETS += crypto/tests/ssltest ipstreams/tests/unixtest
+TARGETS += crypto/tests/ssltest
+crypto/tests/ssltest: $(LIBWVSTREAMS)
+
+TARGETS += ipstreams/tests/unixtest
+ipstreams/tests/unixtest: $(LIBWVSTREAMS)
+
 TARGETS += crypto/tests/printcert
+crypto/tests/printcert: $(LIBWVSTREAMS)

 ifndef _MACOS
   ifneq ("$(with_readline)", "no")
     TARGETS += ipstreams/tests/wsd
+    ipstreams/tests/wsd: $(LIBWVSTREAMS)
     ipstreams/tests/wsd-LIBS += -lreadline
   else
     TEST_SKIP_OBJS += ipstreams/tests/wsd


 ifneq ("$(with_dbus)", "no")
   TARGETS += libwvdbus.so
-  TARGETS += dbus/tests/wvdbus dbus/tests/wvdbusd
+  TARGETS += dbus/tests/wvdbus
+  dbus/tests/wvdbus: $(LIBWVDBUS)
+
+  TARGETS += dbus/tests/wvdbusd
```

```
+  dbus/tests/wvdbusd: $(LIBWVDBUS)
   TESTS += $(call tests\_cc,dbus/tests)
   libwvdbus_OBJS += $(call objects,dbus)
   libwvdbus.so: $(libwvdbus_OBJS) $(LIBWVSTREAMS)
```

Nel file wvstreams-4.6.1/wvrules-posix.mk riga 88:

```
-CC: FORCE
+CC:
  @CC="$(CC)" CFLAGS="$(CFLAGS)" CPPFLAGS="$(CPPFLAGS)"
    $(WVSTREAMS)/gen-cc CC c

-CXX: FORCE
+CXX:
  @CC="$(CXX)" CFLAGS="$(CXXFLAGS)" CPPFLAGS="$(CPPFLAGS)"
    $(WVSTREAMS)/gen-cc CXX cc

+#All files must depend on the above two rules. This is a godawful hack.

+$(shell find -type f '(' -name '*.c' -o -name '*.cc' ')' ): CC CXX

 wvlink=$(LINK_MSG)$(WVLINK_CC) $(LDFLAGS) $($1-LDFLAGS) -o
 $1 $(filter %.o %.a %.so, $2) $($1-LIBS) $(XX_LIBS)
 $(LDLIBS) $(PRELIBS) $(LIBS)
```

La cross-compilazione puó proseguire seguendo le prossime istruzioni:

- make CC=arm-linux-gnueabi-gcc-4.4 AR=arm-linux-gnueabi-ar RANLIB=arm-linux-gnueabi-ranlib CXX=arm-linux-gnueabi-g++-4.4
- sudo cp -r include/ /usr/arm-linux-gnueabi/include/wvstreams
- sudo cp lib*.* /usr/arm-linux-gnueabi/lib

Wvdial

Il porting su android del pacchetto wvdial 1.61 é stato effettuato partendo dai sorgenti scaricati attraverso:

- http://wvstreams.googlecode.com/files/wvdial-1.61.tar.gz

A questo punto si puó scompattare wvdial e aprire un terminale nella medesima cartella. La cross-compilazione avverrá seguendo le prossime istruzioni:

- sudo apt-get install libwvstreams-dev pkg-config
- ./configure –target=arm-linux –host=arm-linux
- editare makefile alla riga 37 con LDFLAGS+=-luniconf -lwvstreams -lwvutils -lwvbase -lz
- make CC=arm-linux-gnueabi-gcc-4.4 CXX=arm-linux-gnueabi-g++-4.4 AR=arm-linux-gnueabi-ar

NOTA 1: come si puó notare la prima istruzione é un "sudo apt-get install libwvstreams-dev pkg-config", viene richiesta l'installazione della libreria wvstreams per evitare ad altre modifiche nel makefile. Alla riga 12 e 18 del makefile viene verificata l'installazione della libreria nel sistema operativo,ma di fatto non verrá mai utilizzata nella fase di cross-compilazione. Ovviamente per chi volesse cimentarsi nella modifica puó sostituire la riga 12 con "PC_CFLAGS=-I/usr/arm-linux-gnueabi/include/wvstreams" e la riga 18 con "PC_LIBS=-lwvstreams -lwvutils -lwvbase".

NOTA 2: Per chi volesse provare ad installare Wvdial in ambiente Unix sono richieste le seguenti librerie

- *zlib : zlib-bin zlibc zlib1g zlib1g-dev libxpm-dev libxpm4*
- *openssl : libssl-dev*
- *wvstreams : libwvstreams-dev*

7.4 Problemi

Come precedentemente detto, é stato abbastanza semplice trovare giá una implementazione funzionante di wvdial per macchine Linux, non c'é quindi da stupirsi che la cross-compilazione e il porting sia stato effettuato con successo, poiché anche Android é basato sul medesimo sistema operativo.
Uno dei problemi principali con cui si é scontrati é che Android non possiede una implementazione completa delle librerie C, in quanto la disponibilitá di capacitá di calcolo e memoria di un dispositivo mobile sono nettamente inferiori ad un sistema standard. Questo ha portato a confrontarsi piú volte con il problema delle funzioni mancanti.

Altri problemi affrontati, sono di seguito elencati :

- trovare le librerie necessarie all'intero funzionamento dell'applicazione
- come cross-compilare i sorgenti
- risolvere errori generici durante la compilazione
- modifica Makefile dei sorgenti
- modifica codici sorgenti wvstreams
- modifica flag per adattarli alla cross-compilazione
- provare a compilare i sorgenti su diversi toolchain (es. CodeSourcery o Android-ndk)

In particolare, gli errori riscontrati in Wvdial durante la cross-compilazione avvenivano in fase di linking, cioé un processo che accetta in input i file oggetto e le librerie per prudurre infine un file eseguibile.

NOTA: Si consiglia di seguire tutti i passi alla perfezione e non utilizzare librerie pre-compilate per l'architettura ARM, in modo da non incorrere in problemi di compatibilitá tra toolchain e quindi in errori del tipo:

- *arm-linux-gnueabi/lib/libwvbase.a: has EABI Version 0, but target wvdial has EABI Version 5*

Capitolo 8

Conclusioni e sviluppi futuri

La sempre piú crescente popolaritá di Android e il fatto che sia un sistema operativo open source, rende questo sistema operativo un campo fertile per numerosi e sempre piú avanzati sviluppi futuri. In rete esistono molti tutorial per ottenere un kernel personalizzato, che peró sono molto superficiali, o si riferiscono a kernel per specifici device. Trovare una guida generale per tutti i tipi di kernel, senza cadere nel vago, é difficile se non impossibile. Uno dei punti piú delicati e difficili nella creazione della propria versione del kernel Android é stato quello di determinare esattamente quali driver e quali opzioni di configurazione sono richiesti per il corretto funzionamento dalla macchina su cui viene installato. Lo scopo di questa tesi é guidare il lettore attraverso questo processo di selezione e scelta dei passi corretti riuscendo cosí ad ottenere un kernel Android personalizzato, partendo dai file sorgenti, fornendo cosí una piattaforma per installare un'architettura per il supporto alla mobilitá in sistemi multihomed eterogenei. In particolare, lo stato attuale delle comunicazioni wireless consente solo limitatamente di sfruttare appieno le potenzialitá del Voice Over IP: a fronte dell'indubbio vantaggio di veicolare le comunicazioni vocali attraverso la rete Internet, sorgono delle problematiche date dall'intrinseca inaffidabilitá delle trasmissioni senza fili. Il modello ABPS propone di superare le limitazioni utilizzando tutte le interfacce a disposizione in un nodo mobile e scegliere pacchetto per pacchetto

quale interfaccia utilizzare. Cosi' facendo, viene migliorata l'interattivitá per le applicazioni multimediali anche in persenza di interferenze, perdite di dati e handoff tra diverse reti. Il sistema offre un incremento delle prestazioni a fronte dello svantaggio di un consumo energetico piu' elevato dovuto al contemporanea attivazione di piu' dispositivi wireless.

In questa tesi, inoltre, é stato introdotta il problema della scarsezza di indirizzi IPv4 disponibili e la necessitá del passaggio ad IPv6. L'architettura descritta, quindi, é stata ripensata per essere pronta per questa transizione. Il passaggio dalla versione 4 alla versione 6 dell'Internet Protocol non sará immediato, non esiste una data per la quale tutti i nodi di Internet saranno pronti allo switch, piuttosto, la transizione sará graduale, entrambi i protocolli conviveranno per un periodo piu' o meno lungo. Per questo, é importante non scartare la retrocompativilitá con IPv4 ed é necessario il supporto a tutti e due i protocolli.

Si é proceduto ad esaminare i diversi componenti del modello ABPS. Il Trasmission Error Detector, che si occupa di notificare all'applicazione l'avvenuta consegna di un pacchetto all'accesso point, é indipendente dal protocollo di rete utilizzato, che ora puo' essere sia IPv4 che IPv6. Il codice é stabile e indipendente dai driver degli specifici dispositivi montati sul nodo mobile, questo permette di usare il sistema in tutti gli host che utilizzano un kernel Linux 2.6.36.

Wvdial assiste nello stabilire connessioni ad Internet basate su modem 3G ed é compreso in alcune importanti distribuzioni Unix. Questo software é l'unico riconosciuto per la sua capacitá di configurare qualsiasi tipo o quasi di modem 3G, ed é quindi di fondamentale importanza per il supporto alla mobilitá.

Il Monitor, il modulo che si occupa di mantenere sempre attive e pronte all'uso tutte le interfacce presenti nel nodo mobile, é esteso ed é in grado di lavorare con le reti dei due diversi protocolli. Il Monitor é integrato col MiD^3, un sistema per il supporto all'interattivitá con il DNS.
Anch'esso era stato pensato per lavorare solo con IPv4: si é proceduto quindi

al suo porting in IPv6.

Appendice A

Android Virtual Device

A.1 Creazione di un AVD da riga di comando

Il tool android consente di gestire gli AVD da lla riga di comando. Per creare un AVD occorre prima di tutto aprire un terminale e posizionarsi nella directory <sdk-directory>/ tools/. Si esegue quindi il comando android create avd, con opzioni che specificano un nome per il nuovo AVD e l'immagine del sistema che si desidera eseguire sull'emulatore quando viene invocato l'AVD. É possibile specificare altre opzioni dalla riga di comando, come la dimensione della scheda SD emulata, la skin dell'emulatore, o un percorso personalizzato per i file dei dati utente. Ecco la riga di comando di utilizzo per la creazione di un AVD:

android create avd -n <name> -t <targetID> [-<option> <value>]...

É possibile utilizzare qualsiasi nome che si desidera per l'AVD, ma poich é probabile che si creino molteplici AVD, si dovrebbe scegliere un nome che permetta di riconoscere le caratteristiche generali offerti dal AVD. Il target ID é un numero intero assegnato dal tool android. Il target ID non é derivato dal nome dell'immagine del sistema, dalla versione, o dal livello API, o da un altro attributo, quindi é necessario eseguire il comando android list targets

per elencare il target ID di ogni immagine del sistema. Si dovrebbe fare questo prima di eseguire il comando android create avd. Per generare un elenco dei target delle immagini di sistema, si utilizza questo comando:

android list targets

Il tool android esegue la scansione delle cartelle <sdk-directory>/platforms/ e <sdk-directory>/add-ons/ alla ricerca di immagini di sistema valide e genera quindi l'elenco dei target. Ecco un esempio di output del comando:

```
Available Android targets:
id: 1 or ‘‘android-3’’
  Name: Android 1.5
  Type: Platform
  API level: 3
  Revision: 4
  Skins: QVGA-L, HVGA-L, HVGA (default), HVGA -P, QVGA-P
id: 2 or ‘‘android-4’’
  Name: Android 1.6
  Type: Platform
  API level: 4
  Revision: 3
  Skins: QVGA, HVGA (default), WVGA800, WVGA854
id: 3 or ‘‘android-7’’
  Name: Android 2.1-update1
  Type: Platform
  API livel: 7
  Revision: 2
  Skins: QVGA, WQVGA400, HVGA (default), WVGA854,
  WQVGA432, WVGA800
id: 4 or ‘‘android-8’’
  Name: Android 2.2
  Type: Platform
```

```
    API livel: 8
    Revision: 2
    Skins: WQVGA400, QVGA, WVGA854, HVGA (default),
    WVGA800, WQVGA432
  id: 5 or ‘‘android-9’’
    Name: Android 2.3
    Type: Platform
    API livel: 9
    Revision: 1
    Skins: HVGA (default), WVGA800, WQVGA432, QVGA,
    WVGA854, WQVGA400}
```

Dopo aver selezionato il target che si desidera utilizzare e preso nota del suo ID, utilizzare il comando android create avd per creare l'AVD, fornendo il target ID come argomento -t. Ecco un esempio che crea un AVD con nome "my_android1.6" e target ID "2" (l'immagine del sistema di Android 1.6 nella lista sopra):

android create avd -n my_android1.6 -t 2

Se il target selezionato é un' immagine del sistema Android (Type: Platform), il tool android chiederá se si desidera creare un profilo hardware personalizzato.

Android 1.6 is a basic Android platform.
Do you wish to create a custom hardware profile [no]

Se si desidera impostare opzioni di emulazione hardware personalizzate per l'AVD, digitare "yes" e impostare i valori in base alle esigenze. Se si desidera utilizzare le opzioni di emulazione hardware predefinite per l'AVD, basta premere il tasto invio (il valore di default é "no"). Il tool android creerá l'AVD con il nome e l'immagine del sistema richiesto, con le opzioni specificate. Durante il test di un'applicazione, é consigliabile testare un'applicazione in diversi AVD, utilizzando diverse configurazioni dello schermo

(diverse combinazioni di dimensioni e densitá). Inoltre, é necessario impostare gli AVD in modo da eseguirli ad una dimensione fisica che si avvicina al dispositivo reale.

Per impostare gli AVD ad una specifica risoluzione o densitá, occorre:

1. utilizzare il comando *create avd* per creare un nuovo AVD, specificando l'opzione –**skin** con un valore che faccia riferimento al nome di una skin di default (ad esempio " WVGA800") o ad una risoluzione della skin personalizzata (ad esempio 240x432). Ecco un esempio:
   android create avd -n <nome> -t <targetID> –skin WVGA800

2. per specificare una densitá personalizzato per la skin, rispondere "yes" quando viene chiesto se si desidera creare un profilo hardware personalizzato per il nuovo AVD;

3. continuare con le varie impostazioni del profilo finché il tool non chiede di specificare "Abstracted LCD density" (hw.lcd.density). Inserire un valore appropriato, come "120" per uno schermo a bassa densitá, "160" per uno a media densitá, o "240" uno ad alta densitá ;

4. impostare le altre opzioni hardware e completare la creazione dell' AVD.

L'AVD é ora pronto ed é possibile lanciare, tramite la riga di comando, un emulatore con l'AVD appena creato (vedi paragrafo D.2 dell'appendice D).

A.2 Opzioni di emulazioni hardware

La seguente tabella elenca le opzioni hardware che si possono impostare al momento della creazione di un nuovo AVD e che vengono memorizzate nel file di configurazione dell'AVD (il file config.ini nella directory locale dell'AVD).

Caratteristica	Descrizione	Proprietá
Device ram size	La quantitá di RAM fisica sul dispositivo, in megabyte. Il valore di default é “96”.	hw.ramSize
Touch-screen support	Se c’é un touch screen nel dispositivo. Il valore di default é “yes”.	hw.touchScreen
Trackball support	Se c’é un trackball nel dispositivo. Il valore di default é “yes”.	hw.trackBall
Keyboard support	Se il dispositivo ha una tastiera QWERTY. Il valore di default é “yes”.	hw.keyboard
DPad support	Se il dispositivo ha pulsanti DPad. Il valore di default é “ yes”.	hw.dPad
GSM modem support	Se c’é un modem GSM nel dispositivo. Il valore di default é “yes”.	hw.gsmModem
Camera support	Se il dispositivo ha una fotocamera. Il valore di default é “yes”.	hw.camera
Maximum horizontal camera pixels	Il valore di default é “640”.	hw.camera.maxHorizontalPixels

Caratteristica	**Descrizione**	**Proprietá**
Maximum vertical camera pixels	Il valore di default é “480 ”.	hw.camera.maxVerticalPixels
GPS support	Se c’é GPS nel dispositivo. Il valore di default é “yes”.	hw.gps
Audio recording support	Se il dispositivo puó registrare audio. Il valore di default é “yes”.	hw.audioInput
Audio playback support	Se il dispositivo puó riprodurre audio. Il valore di default é “yes”.	hw.audioOutput
Battery support	Se il dispositivo puó essere eseguito su una batteria. Il valore di default é “yes”.	hw.battery
Accelerometer	Se c’é un accelerometro nel dispositivo. Il valore di default é “yes”.	hw.accelerometer
SD Card support	Se il dispositivo supporta l’inserimento/rimozione di schede SD virtuali. Il valore di default é “yes”.	hw.sdCard
Cache partition support	Se si usa una partizione cache sul dispositivo. Il valore di default é “ yes”.	disk.cachePartition
Cache partition size	Il valore di default é “66MB ”.	disk.cachePartition.size
Abstracted LCD density	Imposta la densitá utilizzata dallo schermo dell’AVD. Il valore di default é “160”.	hw.lcd.density

A.3 Il tool Android

L'utilizzo del tool android dalla riga di comando é il seguente:

```
android [global options] action [action options]
```

Le opzioni globali possibili sono:

- -s: silent mode (vengono stampati solo gli errori)
- -h: usage help
- -v: verbose mode (vengono stampati gli errori, gli avvisi e i messaggi informativi)

La seguente tabella elenca le azioni (action) e le opzioni (action option) del tool android, e spiega il loro significato e utilizzo.

Azione	**Opzione**	**Descrizione**
avd	Nessuna	Avvia l'AVD Manager.
sdk	Nessuna	Avvia l'SDK Manger.
create avd	-n <name>	Il nome dell'AVD.
	-t <name>	Target ID del l'immagine del sistema da utilizzare con il nuovo AVD. Per ottenere un elenco dei target disponibili, utilizzare il comando: *a*ndroid list targets

Azione	Opzione	Descrizione
	-c <path>|<size>[K|M]	Il percorso all'immagine della scheda SD da utilizzare con questo AVD o la dimensione di una nuova immagine della scheda SD da creare per questo AVD. Ad esempio: -c path/to/sdcard oppure -c 1000M
	-f	Forza la creazione dell'AVD.
	-p <path>	Percorso alla locazione in cui creare la directory per i file di questo AVD.

Azione	Opzione	Descrizione
	-s <name>|<width>-<height>	La skin da utilizzare per questo AVD, identificato da un nome o dalle dimensioni. Il tool android analizza la skin corrispondente in base al nome o alla dimensione nella directory skins/ del target riferito nell'argomento -t <targetID>. Per esempio: -s HVGA-L
	delede avd -n <name>	Il nome dell'AVD da eliminare.
	move avd -n <name>	Il nome dell'AVD da spostare.
	-p <path>	Percorso alla locazione in cui creare la directory per i file di questo AVD.
	-r <new-name>	Il nuovo nome dell'AVD se lo si desidera rinominare.
	update avd -n <name>	Il nome dell'AVD da aggiornare.

A.4 L'emulatore di Android

Per utilizzare l'emulatore, é necessario creare una o piú configurazioni AVD. In ogni configurazione, é necessario specificare una piattaforma Android da eseguire nell'emulatore, una serie di opzioni hardware e la skin dell'emulatore che si desidera utilizzare. Poi, quando si avvia l'emulatore, é necessario specificare la configurazione AVD che si desidera caricare. Ogni AVD funziona come un dispositivo indipendente, con la sua area di memorizzazione privata per i dati utente, per la scheda SD, e cosí via. Quando si avvia l'emulatore con una configurazione AVD, esso carica automaticamente i dati utente e i dati della scheda SD dalla directory AVD. Per default, l'emulatore memorizza i dati utente, i dati delle schede SD e la cache nella directory di AVD.

A.4.1 Elenco dei comandi emulator

Come descritto nell'appendice B, il modo piú semplice per avviare l'emulatore é quello di utilizzare l'Android Virtual Device Manager. É possibile specificare una serie di opzioni per controllare l'aspetto e il comportamento dell'emulatore al momento del suo lancio. Tuttavia, queste opzioni possono essere specificate solo dalla riga di comando attraverso l'uso del comando emulator. Ecco la sintassi del comando:

emulator -avd <**avd_name**> [-<**option**> [<**value**>]]...[-<**qemu args**>]

La seguente tabella elenca tutti i comandi emulator e spiega il loro significato e utilizzo.

É possibile eseguire un'applicazione su una singola istanza dell'emulatore o, a seconda delle esigenze, é possibile avviare istanze multiple dell' emulatore (ognuno con la propria configurazione AVD e area di memorizzazione per i dati utente, per la scheda SD, e cosí via) ed eseguire un'applicazione in piú

Categoria	Comando	Descrizione
AVD	-avd <avd_name> o <avd_name>	Obbligatorio. Specifica l'AVD da caricare per questa istanza dell'emulatore. É necessario creare una configurazione AVD prima di lanciare l'emulatore.
Disk Image	-cache <filepath>	Utilizza <filepath> come immagine della partizione di cache di lavoro. Un percorso assoluto o relativo alla directory di lavoro corrente. Se non é specificato nessun file di cache, il comportamento predefinito dell'emulatore é quello di utilizzare un file temporaneo.
	-data <filepath>	Utilizza <filepath> come immagine del disco dei dati utente di lavoro. Opzionalmente, é possibile specificare un percorso relativo alla directory di lavoro corrente. Se -data non viene utilizzato, l'emulatore cerca un file chiamato userdata -qemu.img nella zona di memoria dell'AVD utilizzato.

Categoria	Comando	Descrizione
	-initdata <filepath>	Quando si azzera l'immagine dei dati utente (attraverso -wipe-data), copia il contenuto di questo file nella nuova immagine del disco dei dati utente. Di default, l'emulator e copia <system>/userdata.img. Opzionalmente, é possibile specificare un percorso relativo alla directory di lavoro corrente.
	-wipe-data	Ripristina l'immagine del disco corrente dei dati utenti (cioé il file specificato da -datadir e -data, o il file di default). L'emulatore cancella tutti i dati dal file immagine dei dati utente, quindi copia il contenuto del file del dato -inidata nel file dell'immagine prima dei iniziare.
	-nocache	Avvia l'emulatore senza una partizione di cache.
	-ramdisk <filepath>	Utilizza <filepath> come immagine della ramdisk. Il valore di default é <system>/ramdisk.img. Opzionalmente, é possibile specificare un percorso relativo alla directory di lavoro corrente.
	-sdcard <filepath>	Utilizza <filepath> come immagine della scheda SD. Il valore di default é <system>/sdcard.img. Opzionalmente, é possibile specificare un percorso relativo alla directory di lavoro corrente.

Categoria	Comando	Descrizione
Debug	-verbose	Abilita un output dettagliato. Equivalente a -debug -init. É possibile definire le opzioni di output predefinite, utilizzate dalle istanze dell'emulatore, nella variabile di ambiente Android ANDROID_VERBOSE. Per fare ció, si definiscono le opzioni che si vogliono utilizzare in un elenco separato da virgole, specificando solo la radice di ogni opzione: -debug -<tags>.Ecco un esempio che mostra ANDROID_VERBOSE definita con le opzioni -debug -init e -debug-modem: ANDROID_VERBOSE=init,modem
	-debug <tags>	Abilita/disabilita i messaggi di debug per il tag di debug specificato. <tags> é un elenco separato da spazi/virgole/colonne di nomi di componenti di debug.

Categoria	Comando	Descrizione
	-debug -<tag>	Abilita/disabilita i messaggi di debug per il tag di debug specificato.
	-debug -no-<tag>	Disattiva i messaggi di debug per il tag di debug specificato.
	-logcat <logtags>	Abilita l'output di logcat con i tag dati. Se la variabile di ambiente ANDROID_LOG_TAGS é definita e non vuota, il suo valore sará utilizzato per abilitare l'output di logcat per impostazione di default.
	-shell	Crea una console shell di root sul terminale corrente. É possibile utilizzare anche se il demone adb nel sistema emulato é rotto. Premendo CTRL+C dalla shell, viene fermato l'emulatore invece che la shell.

Categoria	Comando	Descrizione
	-shell -serial <device>	Abilita la shell di root (come in -shell) e specifica il dispositivo QEMU da utilizzare per la comunicazione con la shell. <device> deve essere un dispositivo di tipo QEMU device type. Vedi la documentazione [49] per "-serial dev" per un elenco dei tipi di dispositivo. Ecco alcuni esempi: • -shell -serial stdio é identico a -shell • -shell -serial tcp::4444, server, nowait consente di comunicare con una shell sulla porta TCP 4444 • -shell -serial fdpair:3:6 consente ad un processo padre di comunicare con una shell utilizzando fds 3 (in) e 6 (out) • -shell -serial fdpair:0:1 utilizza il normale stdin e stdout fds.
	-show-kernel <name>	Visualizza i messaggi del kernel.
	-trace <name>	Abilita il profiling del codice (premere F9 per iniziare), scritto in un file specificato.
	Media -audio <backend>	Utilizza il backend audio specificato.
	-audio -in <backend>	Utilizza il backend audio-input specificato.
	-audio -out <backend>	Utilizza il backend audio-output specificato.
	-noaudio	Disabilita il supporto audio nell'istanza dell'emulatore corrente.

Categoria	Comando	Descrizione
	-radio <device>	Reindirizza l'interfaccia radio modem a un dispositivo host.
	-useaudio	Abilita il supporto audio nell'istanza dell'emulatore corrente. Abilitato per default.
Network	-dns -server <servers>	Utilizza i server DNS specificati. Il valore di <servers> deve essere un elenco separato da virgole di un massimo di 4 nomi di server DNS o indirizzi IP.
	-http-proxy <proxy>	Effettua tutte le connessioni TCP attraverso un proxy HTTP/HTTPS specifico. Il valere di <proxy> puó essere uno dei seguenti: http://<server>:<port>, http://<username>:<password>@-<server>:<port>. Il prefisso http:// puó essere omesso. Se il comando -http-proxy <proxy> non viene fornito, l'emulatore cerca la variabile di ambiente http_proxy e automaticamente utilizza qualsiasi valore corrispondente al formato <proxy> descritto in precedenza.
	-netdelay <delay>	Imposta l'emulazione della latenza di rete a <delay>. Il valore di default é none.
	-netspeed <speed>	Imposta l'emulazione della velocitá di rete a <speed>. Il valore di default é full.
	-netfast	Scorcaitoia per -netspeed full -netdelay none

Categoria	Comando	Descrizione
	-port <port>	Imposta il numero di porta della console per questa istanza dell'emulatore a <port>. Il numero di porta della console deve essere un numero intero pari tra 5554 e 5584, inclusi. <port>+1 deve essere libero e sará riservato ad ADB.
	-report-console <socket>	Riporta la porta della console assegnata per questa istanza dell'emulatore ad una terza parte remota prima di avviare l'emulazione. <socket> deve utilizzare uno di questi formati: tcp:<port>[,server][,max=<seconds>]unix:-<port>[,server][,max=<seconds>]
Sistema	-cpu -delay <delay>	Rallenta la velocitá della CPU emulata di <delay>. I valori supportati per <delay> sono numeri interi tra 0 e 1000.
	-gps <device>	Reindirizza il GPS NMEA al dispositivo. Utilizzare questo commando per emulare un GPS compatibile con NMEA collegato ad un dispositivo esterno o a un socket. Il formato di <device> deve essere una specificazione QEMU del dispositivo seriale. Vedi la documentazione [49] per "-serial dev".
	-qemu	Passa argomenti al software emulatore qemu. Quando si utilizza questa opzione, assicurarsi che sia l'ultima opzione specificata, dal momento che tutte le opzioni successive sono interpretate come opzioni specifiche qemu.

Categoria	Comando	Descrizione
	-qemu -enable-kvm	Abilita l'accelerazione KVM della macchina virtuale dell'emulatore. Questa opzione é efficace solo quando il sistema é impostato per utilizzare l'accelerazione KVM della macchina virtuale. É possibile specificare una dimensione della memoria (-m <size>) per la macchina virtuale, che deve corrispondere alla dimensione della memoria dell'emulatore: -qemu -m 512 -enable-kvm, -qemu -m 1024 -enable-kvm
	-qemu -h	Visualizza l'help di qemu.
	-nojni	Disabilita i controlli JNI nella Dalvik runtime.
	-gpu on	Attiva l'accelerazione grafica per l'emulatore. Questa opzione é disponibile solo per gli emulatori che utilizzano un'immagine di sistema con API Level 15, revision 3 e superiori.
	-radio <device>	Reindirizza la modalitá radio al dispositivo specificato. Il format di <device> deve essere una specificazione QEMU del dispositivo seriale. Vedi la documentazione [49] per "-serial dev".
	-timezone <timezone>	Imposta il fuso orario per il dispositivo emulato a <timezone>, invece che al fuso orario dell'host. <timezone> deve essere specificato nel formato zoneinfo. Per esempio: "America/Los_Angeles", "Europe/Paris"
	-version	Visualizza il numero di versione dell'emulatore.

Categoria	Comando	Descrizione
UI	-dpi -device <dpi>	Ridimensiona la risoluzione dell'emulatore in modo che corrisponda alle dimensioni dello schermo di un dispositivo fisico. Il valore di default é 165.
	-no-boot-anim	Disabilita l'animazione di boot durante l'avvio dell'emulatore. Disabilitare l'animazione di boot puó velocizzare il tempo di avvio per l'emulatore.
	-no-window	Disabilita la visualizzazione della finestra grafica dell'emulatore.
	-scale <scale>	Ridimensiona la finestra dell'emulatore. <scale> é un numero tra 0.1 and 3 che rappresenta il fattore di scala desiderato. É inoltre possibile specificare la scala come un valore DPI se si aggiunge il suffisso "dpi" al valore della scala. Un valore "auto" dice all'emulatore di selezionare la migliore dimensione della finestra.
	-raw -keys	Disabilita la tastiera Unicode reverse-mapping.
	-noskin	Non viene utilizzata alcuna skin dell'emulatore.
	-keyset <file>	Utilizza il file keyset specificato invece di quello predefinito. Il file keyset definisce l'elenco delle combinazioni di tasti tra l'emulatore e la tastiera dell'host.
	-onion <image>	Utilizza l'immagine overlay sullo schermo. Nessun supporto per JPEG. Solo PNG é support ao.
	-onion -alpha <percent>	Specifica il valore della traslucenza della skin (in percentuale). Il valore di default é 50.

Categoria	Comando	Descrizione
	-onion -rotation <position>	Specifica la rotazione della skin. <position> deve essere uno dei valori 0, 1, 2, 3.
	-skin <skinID>	Queste opzioni dell'emulatore sono deprecate.
	-skindir <dir>	Utilizzare gli AVD per impostare le opzioni della skin, piuttosto che utilizzare questa opzione dell'emulatore. L'utilizzo di questa opzione puó provocare inaspettati e in alcuni casi fuorvianti risultati, poiché la densitá con cui si rende la skin puó non essere definita. Gli AVD consentono di associare ogni skin con una densitá predefinita e non tener conto di quella predefinita se necessario.
Help	-help	Stampa un elenco di tutte le opzioni dell'emulatore.
	-help-all	Stampa l'help per tutte le opzioni di avvio.
	-help-<option>	Stampa l'help per una specifica opzione di avvio.
	-help-debug-tags	Stampa un elenco di tutti i tag per -debug <tags>.
	-help-disk-images	Stampa l'help per l'utilizzo delle immagini del disco dell'emulatore.
	-help-environment	Stampa l'help per le variabili di ambiente dell'emulatore.
	-help-keys	Stampa la mappatura corrente delle tasti.
	-help-keyset-file	Stampa l'help per definire un file di mappatura delle chiavi personalizzato.
	-help-virtual-device	Stampa l'help per l'utilizzo dell'Android Virtual Device.

di un dispositivo emulato. Per avviare un'istanza dell'emulatore dalla riga di comando, navigare fino alla directory tools/ dell'SDK e digitare il seguente comando:

emulator -avd <avd_name> [<option>].

Questo inizializza l'emulatore, carica una configurazione AVD e visualizza la finestra dell'emulatore. Per interrompere un'istanza dell'emulatore, é sufficiente chiudere la finestra dell'emulatore.

A.4.2 La console dell'emulatore

Ogni istanza dell'emulatore in esecuzione fornisce una console che permette di interrogare e controllare l'ambiente del dispositivo emulato. Ad esempio, é possibile utilizzare la console per gestire il reindirizzamento delle porte, le caratteristiche della rete, ed gli eventi di telefonia mentre un'applicazione é in esecuzione sull'emulatore. Per connettersi in qualsiasi momento alla console di una qualsiasi istanza dell'emulatore in esecuzione, utilizzare questo comando:

telnet localhost <console-port>

Un'istanza dell'emulatore occupa una coppia di porte adiacenti: una porta di console ed una porta adb. I numeri di porta differiscono di 1, con la porta adb avente il numero piú alto di porta. La console della prima istanza dell'emulatore in esecuzione su una determinata macchina utilizza la porta di console 5554 e la porta adb 5555. Istanze successive utilizzano numeri di porta incrementati di due: ad esempio, 5556/5557, 5558/5559, e cosí via. É possibile eseguire fino a 16 istanze simultanee dell'emulatore. Per connettersi alla console dell'emulatore, é necessario specificare una porta di console valida. Se piú istanze dell'emulatore sono in esecuzione, é necessario determinare la porta di console dell'istanza dell'emulatore che si desidera connettersi. É possibile trovare la porta di console dell'istanza elencata nel titolo della finestra dell'istanza. In alternativa, é possibile utilizzare il comando adb devices, che stampa un elenco delle istanze dell'emulatore in esecuzione e i loro numeri di porta di console (vedi L'emulatore di Android 83 paragrafo E.2

dell'appendice E). Si noti che l'emulatore rimane in ascolto per connessioni su porte in un range da 5554 a 5587 e accetta connessioni solo da localhost. Una volta collegati alla console, é possibile digitare il comando help [command] per visualizzare un elenco dei comandi della console. Per uscire dalla sessione della console, utilizzare il comando quit o exit.

A.4.3 Reindirizzamento delle porte

É possibile utilizzare la console per aggiungere e rimuovere il reindirizzamento delle porte, mentre l'emulatore é in esecuzione. Dopo essersi connessi alla console, é possibile gestire il reindirizzamento di porta inserendo il seguente comando: **redir <list|add|del>**

Il comando redir supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	**Descrizione**
list	Elenca il reindirizzamento di porta corrente.
add <protocol>:<host-port>:<guest-port>	Aggiunge un reindirizzamento di porta. • <protocol> deve essere "tcp" o "udp" • <host-port> é il numero di porta da aprire sull'host • <guest-port> é il numero di porta per instradare i dati sull'emulatore/dispositivo

Sottocomando	Descrizione
del <protocol>:<host-port>	Elimina un reindirizzamento di porta. Il significato di <protocol> e <protocol> sono elencati nella riga precedente.

A.4.4 Emulazione della geolocalizzazione

É possibile utilizzare la console per impostare la posizione geografica segnalata alle applicazioni in esecuzione all'interno di un emulatore. Utilizzare il comando geo per inviare un semplice fix GPS all'emulatore, con o senza formattazione NMEA 1083:

geo <fix|nmea>

É possibile eseguire il comando geo non appena un'istanza dell'emulatore é in esecuzione. Il comando geo supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	Descrizione
fix <longitude> <latitude> [<altitude>]	Invia un semplice fix GPS all'istanza dell'emulatore. Specificare longitudine e latitudine in gradi decimali. Specificare l'altitudine in metri.
nmea <sentence>	Invia una frase NMEA 0183 al dispositivo emulato, come se fosse inviata da un modem GPS emulato. <sentence> deve essere con "$GP tcp". Solo le frasi "$GPGGA" e "$GPRCM" sono attualmente supportate.

A.4.5 Caratteristiche dell'alimentazione del dispositivo

Il comando power controlla lo stato di alimentazione riportato dall'emulatore per le applicazioni. La sintassi per questo comando é la seguente:

power <display|ac|status|present|health|capacity>

Il comando power supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	**Descrizione**
display	Visualizza lo stato della batteria e di carica.
ac <on|off>	Imposta lo stato di carica AC a on o off.
status <unknown|charging|-discharging|not -charging|full>	Cambia lo stato della batteria come specificato.
present <true|false>	Imposta lo stato di presenza della batteria.
health <unknown|good|overheat|-dead|overvoltage|failure>	Imposta lo stato di salute della batteria.
power capacity <percent>	Imposta lo stato di capacitá rimanente della batteria (0-100).

A.4.6 Emulazione degli eventi hardware

Il comando event invia eventi hardware per l'emulatore. La sintassi per questo comando é la seguente:

event <send|types|codes|text>

Il comando event supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	Descrizione
send <type>:<code>:<value> [...]	Invia uno o piú eventi al kernel di Android. É possibile utilizzare nomi testuali o numeri interi per <type> e <value>.
send <type>:<code>:<value> [...]	Invia uno o piú eventi al kernel di Android. É possibile utilizzare nomi testuali o numeri interi per <type> e <value>.
types	Elenca tutti gli alias della stringa <type> supportati dai sottocomandi di event
codes <type>	Elenca tutti gli alias della stringa <codes> supportati dai sottocomandi di event per lo specificato <type>.
event text <message>	Simula i tasti premuti da inviare la specificata stringa di caratteri come un messaggio. Il messaggio deve essere una stringa UTF -8.

A.4.7 Emulazione della rete

É possibile utilizzare la console per controllare lo stato della rete, il ritardo corrente e le caratteristiche della velocitá. Per fare ció, occorre utilizzare il comando network status. L'emulatore permette di simulare diversi livelli di latenza della rete, in modo da poter testare l'applicazione in un ambiente piú tipico delle condizioni concrete in cui verrá eseguito. É possibile impostare un livello o un intervallo di latenza all'avvio dell'emulatore o é possibile utilizzare la console per modificare la latenza, mentre l'applicazione é in esecuzione nell'emulatore. Per impostare la latenza all'avvio dell'emulatore, occorre utilizzare il comando emulator con l'opzione -netdelay e con un valore <delay> supportato, come indicato nella tabella sottostante. Ecco alcuni esempi:

- emulator -netdelay gprs
- emulator -netdelay 40 100

Per apportare modifiche al ritardo della rete mentre l'emulatore é in esecuzione, occorre invece collegarsi alla console e utilizzare il comando netdelay con un valore <delay> supportato: network delay gprs. Il formato di <delay> é uno dei seguenti (i numeri sono millisecondi):

Valore	**Descrizione**
gprs	GPRS (min 150, max 550)
edge	EDGE/EGPRS (min 85, max 400)
umts	UMTS/3G (min 35, max 200)
none	Nessuna latenza (min 0, max 0)
<num>	Emula un'esatta latenza (in millisecondi).
<min>:<max>	Emula un specifico intervallo di latenza (in millisecondi).

L'emulatore consente inoltre di simulare varie velocitá di trasferimento della rete. É possibile impostare una velocitá o un in tervallo di trasferi-

mento all'avvio dell'emulatore oppure é possibile utilizzare la console per modificare la velocitá, mentre un'applicazione é in esecuzione nell'emulatore. Per impostare la velocitá della rete all'avvio dell'emulatore, occorre utilizzare il comando emulator con l'opzione -netspeed e con un valore <speed> supportato, come indicato nella tabella sottostante. Ecco alcuni esempi:

- emulator -netspeed gsm
- emulator -netspeed 14.4 80

Per apportare modifiche alla velocitá della rete, mentre l'emulatore é in esecuzione, occorre invece collegarsi alla console e utilizzare il comando netspeed con un valore <speed> supportato: network speed 14.4 80.

Il formato di <speed> é uno dei seguenti (i numeri sono kilobit/sec):

Valore	**Descrizione**
gsm	GSM (up: 14.4, down: 14.4)
hscsd	HSCSD (up: 14.4, down: 43.2)
gprs	GPRS (up: 40.0, down: 80.0)
edge	EDGE/EGPRS (up: 118.4, down: 236.8)
umts	UMTS/3G (up: 128.0, down: 1920.0)
hsdpa	HSDPA (up: 348.0, down: 14400.0)
full	Nessun limite (up: 0.0, down: 0.0)
<num>	Imposta un'esatta velocitá utilizzata sia per l'upload che per il download.
<up>:<down>	Imposta esatte velocitá per l'upload e il download.

A.4.8 Stato della Virtual Machine

É possibile utilizzare il comando vm per controllare la Virtual Machine su un'istanza dell'emulatore. La sintassi per questo comando é la seguente: **vm <start|stop|status>**

Il comando vm supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	Descrizione
start	Avvia la Virtual Machine sull'istanza stop Arresta la Virtual Machine sull'istanza.
stop	Arresta la Virtual Machine sull'istanza.
status	Visualizza lo stato corrente della Virtual Machine (in esecuzione o ferma).

A.4.9 La finestra dell'emulatore

É possibile utilizzare il comando window per gestire la finestra dell'emulatore. La sintassi per questo comando é la seguente: window <scale>. Il comando window supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	Descrizione
scale <scale>	Ridimensiona la finestra dell'emulatore. <scale> é un numero tra 0.1 and 3 che rappresenta il fattore di scala desiderato. É inoltre possibile specificare la scala come un valore DPI se si aggiunge il suffisso "dpi" al valore della scala. Un valore "auto" dice all'emulatore di selezionare la migliore dimensione della finestra.

A.4.10 Chiusura di un'istanza dell'emulatore

É possibile interrompere un'istanza dell'emulatore tramite la console, utilizzando il comando kill.

A.4.11 Limitazioni dell'emulatore

Le limitazioni funzionali dell'emulatore comprendono:

- nessun supporto per effettuare o ricevere telefonate reali. É comunque possibile simulare chiamate (effettuate e ricevute) attraverso la console dell'emulatore;
- nessun supporto per le connessioni USB;
- nessun supporto per le cuffie collegate al dispositivo;
- nessun supporto per determinare lo stato di connessione della rete;
- nessun supporto per determinare il livello di carica della batteria e lo stato di carica AC;
- nessun supporto per determinare l'inserimento/espulsione di una scheda SD;
- nessun supporto per il Bluetooth.

Vorrei ringraziare innanzitutto tutta la mia famiglia,
che ha reso possibile tutto questo.
Un ringraziamento particolare al Dott. Vittorio Ghini,
che seguendomi per tutta la durata della tesi mi ha permesso di
realizzare una soluzione ad un problema reale.
Un grazie alle mie migliori amiche Marilena, Veronica e
Alessia. Quest'ultima mi é stata particolarmente
vicina in ogni momento del mio percorso universitario,
sopportandomi con estrema pazienza.
Un grazie a tutti agli amici d'infanzia, Francesco, Pietro,
Antonio, Michele, Raffaello e Sesto.
Un grazie ai vecchi amici, Pasquale, Giuseppe, Sonia, Stefania,
Federica, Angela, Rosa, Jasmin, Rosaria,
Mella, Michela, Elison e Rita.
Un grazie agli amici bolognesi, Nikolas, Marco M., Rita,
Davide, Manuela, Marco V., Massimo, Chiara, Andrea,
Giuseppe, Selena, Pasquale C., Laura, Fedo, Pasquale P. e
Concetta. Alcuni di loro sono stati piú che amici,
la mia famiglia bolognese direi, delle persone fantastiche con
cui ho condiviso parte della mia vita universitaria

e in parte quella privata, che hanno avuto sempre tempo per ascoltarmi e per consigliarmi.

In ultimo,e non per importanza, i nuovi amici, Nicola, Antonio P., Tiziana, Giuseppe, Antonio L., Lorenzo R., Federica, Mariapaola, Annapina, Tommaso, Valentina e Luisella che hanno reso la mia estate indimenticabile.

Grazie a tutti, quelli che indirettamente o direttamente hanno contribuito al conseguimento della mia laurea.

Bibliografia

[1] Android debug bridge: http://developer.android.com/tools/help/adb.html.

[2] Android developers: http://developer.android.com/tools/help/android.html.

[3] Android development tools: http://developer.android.com/tools/help/adt.html.

[4] Android emulator: http://developer.android.com/tools/help/emulator.html.

[5] Android native development kit: http://developer.android.com/tools/sdk/ndk/index.html.

[6] Android virtual device: http://developer.android.com/tools/devices/index.html.

[7] Api: http://it.wikipedia.org/wiki/application_programming_interface.

[8] Eclipse: http://www.eclipse.org/.

[9] Initramfs: http://tdknights.com/downloads/initramfs-xxjvq .tar.gz.

[10] Java jdk: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

[11] Kbuilt makefile: http://www.ravnborg.org/kbuild/ makefiles.html.

[12] Libreria lib32ncurses5-dev: http://packages.ubuntu.com/precise/lib32ncurses5-dev.

[13] Libreria lib32readline5-dev : http://packages.ubuntu.com/lucid/lib32readline5-dev.

[14] Libreria lib32z-dev: http://packages.debian.org/squeeze/lib32z1-dev.

[15] Managing avds with avd manager: http://developer.android.com/tools/devices/managing-avds.html.

[16] Odin: http://www.androidgalaxys.net/dwl/odin/.

[17] Samsung kies: http://www.samsung.com/it/support/ usefulsoftware/kies/jsp.

[18] Samsung open source center: http://opensource.samsung.com/.

[19] Software development kit: http://it.wikipedia.org/wiki/software_development_kit.

[20] Talon: https://github.com/existz/linux_gt-i9000-gb.

[21] Toolchain:http://it.wikipedia.org/wiki/cross-compilazione.

[22] Using the android emulator: http://developer.android .com/tools/devices/emulator.html.

[23] Virtualbox: https://www.virtualbox.org/.

[24] *IEEE 802.11i: Amendment 6: Medium Access Control (MAC) Security Enhancements.* 2004.

[25] *IEEE 802.11e: Wireless LAN Medium Access Control and Physical Layer Specification: Amendment for Quality of Service Enhancements.* 2005.

[26] *IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* 2007.

[27] *IEEE 802.11n Amendment 5: Enhancements for Higher Throughput.* 29 Ottobre 2009.

[28] C. Aoun and E. Davies. *RFC 4966 - Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status.* Luglio, 2007.

[29] Christian Benvenuti. *Understanding Linux Network Internals.* 2005.

[30] B. Carpenter and K. Moore. *RFC 3056 - Connection of IPv6 Domains via IPv4 Clouds.* Febbraio, 2001.

[31] Humphrey Cheung. The feds can own your wlan too. www.smallnetbuilder.com, Marzo 2005.

[32] S. Deering and R. Hinden. *RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification.* Dicembre, 1998.

[33] Stefano Ferretti, Vittorio Ghini, Fabio Panzieri, and Elisa Turrini. *Seamless Support of Multimedia Distributed Applications Through a Cloud.* Luglio, 2010.

[34] Vittorio Ghini, Giorgia Lodi, and Fabio Panzieri. *Always Best Packet Switching: the mobile VoIP case study.* 2009.

[35] johannes Berg. Linux wireless. wireless.kernel.org.

[36] E. Nordmark and R. Gilligan. *RFC 4213 - Basic Transition Mechanisms for IPv6 Hosts and Routers.* Ottobre, 2005.

[37] Information Sciences Institute University of Southern California. *RFC 791 - Internet Protocol.* 1981.

[38] J. Postel. *RFC 768 - User Datagram Protocol.* 28 Agosto 1980.

[39] K. Ramakrishnan, S. Floyd, and D. Black. *RFC 3168 - The Addition of Explicit Congestion Notification (ECN) to IP.* 1981.

[40] Fahmida Y. Rashid. Ipv4 address depletion adds momentum to ipv6 transition. www.eweek.com, Febbraio, 2011.

[41] G. Tsirtsis and P. Srisuresh. *RFC 2766 - Network Address Translation - Protocol Translation (NAT-PT).* Febbraio, 2000.