Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Incapsulamento di applicazioni per il controllo del flusso dati : monitor

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Carlo Rimondi

Sessione II
Anno Accademico 2013/2014

dedicato ad Elena
Margherita e Lidia

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La grande diffusione dei dispositivi mobili verificatasi negli ultimi 10 anni ha fatto sì che internet abbia assunto un ruolo ancora più centrale nel panorama delle comunicazioni globali. Internet e dispositivi mobili hanno visto una crescita molto rapida ed in costante aumento: le tabelle 1.1 e 1.2 mostrano rispettivamente gli utenti che utilizzano internet e il numero di dispositivi mobili venduti negli ultimi anni.

A questo è seguita la diffusione delle connessioni domestiche con conseguente aumento del numero di access point disponibili sul territorio. Non è difficile andare in un centro abitato e scoprire la presenza di access point costantemente accesi e spesso utilizzati solo qualche ora al mese. In un panorama del genere si potrebbe ipotizzare una condivisione controllata delle risorse che consentirebbe a tutti gli individui della comunità di utilizzare le connessioni altrui. L'attuale diffusione dei dispositivi mobili lascerebbe pensare ad un interesse comune verso questo tipo di organizzazione, poichè fondamentalmente permetterebbe di condividere un servizio sottoutilizzato in cambio della costante connessione ad internet. Questo modello avrebbe sia una valenza etica che una economica dal momento che limita gli sprechi ottimizzando le risorse.

Viviamo in un epoca in cui l'approccio ad internt è cambiato radicalmente:

Anno	Utenti Internet	Popolazione mondiale	% popolazione con Internet
2014	2.925.249.355	7.243.784.121	40,4%
2013	2.712.239.573	7.162.119.430	37,9%
2012	2.511.615.523	7.080.072.420	35,5%
2011	2.272.463.038	6.997.998.760	32,5%
2010	2.034.259.368	6.916.183.480	29,4%
2009	1.752.333.178	6.834.721.930	25,6%
2008	1.562.067.594	6.753.649.230	23,1%
2007	1.373.040.542	6.673.105.940	20,6%
2006	1.157.500.065	6.593.227.980	17,6%
2005	1.029.717.906	6.514.094.610	15,8%
1995	44.838.900	5.741.822.410	0.8%

Tabella 1.1: Utilizzatori di Internet nel mondo. Le stime dell'anno in corso di riferiscono al mese di Giugno [1]

siamo passati dal modello dei motori di ricerca, in cui il web veniva percepito come un semplice contenitore di informazioni, al modello dei social network, in cui il livello di interazione ha coivolto ogni singolo utente, dandogli la possibilità di pubblicare i propri contenuti, di essere in costante contatto con una comunità in espansione e di essere sempre presente soprattutto grazie ai dispositivi mobili. Le nuove generazioni sono tra le più sensibili a questo genere di cambiamenti e sono le più inclini a sperimentare nuove tendenze non appena se ne presenta l'occasione. Ed è proprio l'occasione che molto spesso manca, soprattutto se pensiamo alle opportunità che ci potrebbero essere nel caso la disponibilità di banda fosse superiore a quella degli attuali livelli. Si potrebbe per esempio utilizzare internet per veicolare il traffico telefonico mediante applicazioni che realizzano VoIP[1].
Tralasciando tutti gli aspetti legali legati alle normative necessarie a regolare la condivisione controllata e sicura degli access point, ciò che abbiamo voluto

[1]Voice over IP

Anno	Dispositivi mobili	% popolazione con disp. mobili
2014	2.680.200.125	37%
2013	2.363.499.412	33%
2012	2.053.221.002	29%
2011	1.749.499.690	25%
2010	1.383.236.696	20%
2009	1.093.555.509	16%
2008	1.013.047.385	15%

Tabella 1.2: Dispositivi mobili venduti nel mondo. Le stime dell'anno in corso di riferiscono al mese di Giugno [2]

valutare è l'aspetto tecnico legato all'utilizzo delle reti wi-fi in mobilità. Ipotizzando di poter avere credenziali uniche per identificarsi presso gli access point della sottorete della comunità, il problema principale rimane la perdita dell'indirizzo IP ogni qualvolta si cambia hotspot [3] di riferimento. Essendo questo inevitabile, bisogna immaginare uno scenario in cui la variazione di indirizzo IP non venga percepito come tale dall'endpoint esterno a questa struttura.

Il presente lavoro ha permesso di valutare quali risorse hardware e software sono necessarie ad un calcolatore affiché si possa interfacciare con un'infrastruttura simile a quella discussa fino ad ora.

La tesi è stata svolta in collaborazione con il collega Raffaele Lovino. In particolare Raffaele si è occupato dello sviluppo del modulo da collegare ad umView, mentre la parte di mia competenza ha riguardato dapprima l'analisi del comportamento di umView nel gestire applicazioni che vanno a modificare la configurazione delle interfacce di rete mediante l'uso di socket netlink e successivamente lo sviluppo del *monitor* di sistema.

Capitolo 2 : verrà presentato uno scenario di riferimento attraverso un esempio concreto e verranno introdotti i primi concetti relativi ad

umView.

Capitolo 3 : verranno presentati gli strumenti software utilizzati nel progetto: si comincerà da umView, per poi passare agli strumenti correlati al *monitor* quali iproute2, wpa_supplicant e wvdial.

Capitolo 4 : verrà descritto l'obiettivo del progetto nella sua globalità per poi scendere nel dettaglio degli obiettivi a cui il *monitor* dovrà tendere.

Capitolo 5 : verrà spiegato il percorso che è stato fatto per giungere alla definizione dell'architettura generale e i motivi che ci hanno indotto a scartare le prime due soluzioni a favore di una terza ritenuta più equilibrata.

Capitolo 6 : verrà descritto nel dettaglio il modulo chiamato *monitor*: si partirà dall'architettura, per poi proseguire con considerazioni relative all'interazione che esso ha con applicazioni ed interfacce di rete e si concluderà con la descrizione sommaria dell'algoritmo sul quale il *monitor* si basa.

Capitolo 7 : verranno discussi i test eseguiti sul *monitor* al fine di dimostrarne il corretto funzionamento; i test riguarderanno sia il dialogo con le applicazioni che l'interazione con le periferiche.

Capitolo 8 : verranno infine descritti i possibili sviluppi futuri che costituiscono la differenza tra gli obiettivi presentati e ciò che è stato realizzato concretamente finora.

Capitolo 2

Scenario

Supponiamo di avere un nodo mobile (smartphone, tablet...) munito di due interfacce di rete:

- un' interfaccia di rete wireless collegata ad esempio ad una WLAN
- un' interfaccia di rete UMTS[1] collegata ad una rete cellulare

Generalmente, in presenza di un Access Point wireless, si tende a sfruttare l'interfaccia wireless per ovvi motivi di carattere tecnico (una maggiore banda in upload/download, migliore qualità del segnale...) o di carattere economico (costo del traffico dati). D'altra parte, le caratteristiche tecniche degli standard IEEE 802.11 [4] non permettono di avere un range del segnale superiore a qualche decina o centinaia di metri, molto limitato se si considera la copertura territoriale del segnale cellulare.

Il seguente esempio mostra un tipico scenario: supponiamo di voler usare il nostro nodo mobile per aprire uno stream di dati duraturo nel tempo (ad esempio il download di un file molto grande) e di trovarci inizialmente in un'area effettivamente coperta da segnale wifi (figura 2.1).

[1] Universal Mobile Telecommunications System

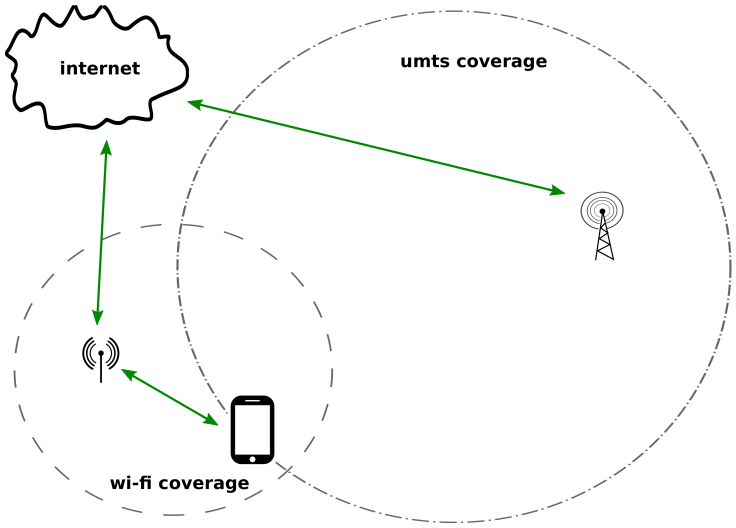

Figura 2.1: Nodo mobile all'interno di un' area coperta da segnale Wi-Fi

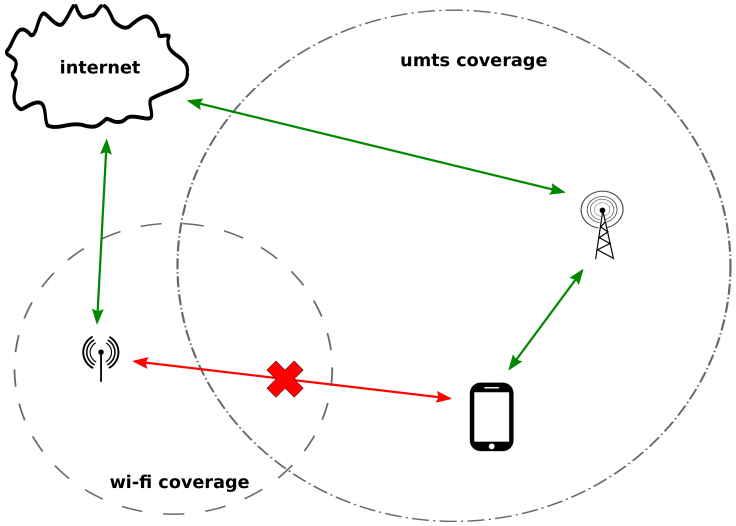

Figura 2.2: Nodo mobile all'interno di un' area coperta solo da segnale UMTS

Una connessione TCP resta aperta finché ci si muove all'interno dell'area, ma viene chiusa non appena si supera il range di copertura, provocando anche la chiusura del canale di comunicazione a livello più alto (figura 2.2).
A questo punto non resta che attivare l'interfaccia UMTS e ristabilire una nuova connessione a partire da un diverso indirizzo IP.

Il problema resta invariato anche nel caso di protocolli di livello applicativo che si basano su invio e ricezione di datagram UDP[2] (protocolli VoIP), in quanto ogni interfaccia di rete possiede un proprio indirizzo IP, quindi lo switch da interfaccia wifi a umts rende irraggiungibile l'indirizzo IP della prima.

Una soluzione può essere quella di incapsulare l'applicazione all'interno di una macchina virtuale parziale (umView), con lo scopo di fornire un'unica interfaccia di rete virtuale. Un particolare sottomodulo di questa macchina virtuale si farà carico di nascondere le due interfacce reali, di dialogare con esse e di mantenere aperte le eventuali connessioni.

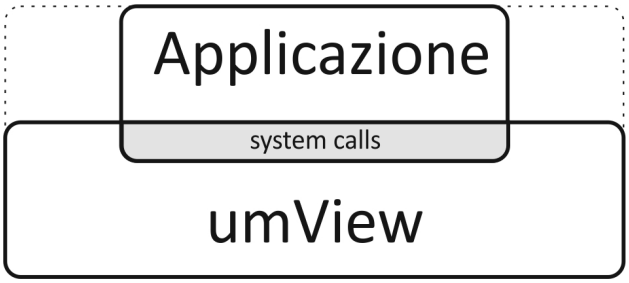

Figura 2.3: umView: incapsulamento applicazione

La figura 2.3 sintetizza due caratteristiche salienti di umView:

1. **incapsulamento**: la linea tratteggiata esterna indica che le applicazioni (possono essere una o più di una) sono processi figli di umView; questo significa che sono soggette alle regole impostate da unView stesso. Inoltre, trattandosi di processi figli, quando umView termina tutta la gerarchia termina di conseguenza.

[2]User Datagram Protocol

2. **virtualizzazione parziale**: umView non crea un ambiente completamente isolato da quello sottostante: limita la virtualizzazione alle sole system calls [5].

Capitolo 3

Strumenti

Come anticipato il progetto è stato sviluppato su un sistema GNU-Linux, distribuzione Debian, utilizzando il linguaggio ANSI C. Nel corso del capitolo verrà posta particolare attenzione all'applicativo umView utilizzato per virtualizzare le system call dell'applicazione che si vuole controllare.
Nel caso specifico del *monitor* sono stati utilizzati alcuni strumenti appositamente progettati per la configurazione delle interfacce di rete quali iproute2, wpa_supplicant e wvdial.

3.1 umView

umView fa parte del progetto Virtual Square [6] ed utilizza la system call ptrace per catturare le chiamate di sistema di un processo, al fine di modificarne il comportamento utilizzando se necessario moduli esterni. Viene pertanto realizzata una paravirtualizzazione del processo da controllare che permette a quest'ultimo di avere una visione personalizzata dell'ambiente nel quale viene eseguito ed una visione parziale dell'ambiente reale.
A fronte di questo, umView può essere definita una "macchina virtuale

per chiamate di sistema" perchè limita la virtualizzazione a questo aspetto, lasciando inalterato l'ambiente sottostante.

3.1.1 Struttura

La struttura di umView è tale da consentire l'aggiunta di moduli esterni, ognuno con specifiche funzionalità, allo scopo di definire l'ambiente adatto al processo che si vuole controllare. I moduli possono essere caricati o all'avvio di umView o durante la sua esecuzione mediante semplici comandi che verranno trattati nella sezione 3.1.2
I moduli possono modificare il comportamento di diverse famiglie di oggetti tra cui pathname, filesystem e protocol family. Questi ultimi sono di particolare interesse per il progetto che stiamo trattando, dal momento che rappresentano il supporto alla virtualizzazione della system call socket. Il riferimento agli oggetti disponibili in un certo istante viene memorizzato in una Hash Table che verrà consultata nel momento in cui il processo controllato effettua una chiamata a sistema. A questo punto un componente specifico di umView, il dispatching layer, si occuperà di reindirizzare la chiamata verso il modulo opportuno. Se non esiste un modulo in grado di gestire la chiamata, allora verrà eseguito il codice della system call originale.

3.1.2 Moduli

In umView l'aggiunta di un modulo permette ad un processo di modificare il proprio punto di vista sulle system call chiamate. Infatti, quando un processo che vive all'interno di umView effettua una chiamata di sistema, il codice che viene eseguito non sarà necessariamente quello delle system call così come le conosciamo, ma piuttosto quello definito all'interno dei moduli caricati in quel momento.
Prima di presentare i moduli utilizzati, vediamo un esempio che ci aiuta a

capire meglio la logica di funzionamento di umView. I comandi che seguono vengono eseguiti in sequenza all'interno un terminale:

`umview -p umnet bash`
Manda in esecuzione umView. L'opzione -p viene utilizzata per caricare i moduli a riga di comando. In questo caso viene caricato il modulo umnet necessario per la gestione dello stack virtuale di rete. Il parametro finale (bash) sta proprio ad indicare che l'applicazione da controllare è la Bourne-Again Shell [7].

`mount -t umnetlink none /dev/net/mydev`
Si utilizza il comando di sistema mount per collegare il modulo umnetlink al dispositivo /dev/net/mydev

`exec mstack /dev/net/mydev bash`
Ora lanciamo il comando mstack sul dispositivo precedentemente creato: questo ha l'effetto di cambiare lo stack di default con quello gestito dal modulo umnetlink.

A questo punto tutti i comandi che verranno lanciati dentro il nuovo ambiente avranno una visione su ciò che è stato definito nel modulo umnetlink.

Vediamo ora una breve panoramica sui moduli trattati nel progetto:

- umnet: viene incluso alla partenza di umView e definisce uno stack di rete per l'applicazione che si vuole controllare. Nel nostro caso, questo modulo viene sempre incluso perchè rappresenta la base su cui appoggiano i moduli successivi.

- umNETDIF: definisce uno stack per la famiglia di protocolli ipv4 e ipv6. All'interno di questo modulo sono state ridefinite le system call che riguardano la comunicazione e la gestione delle interfacce di rete.

- umnetlink2: derivato dal modulo precedente, definisce uno stack per la famiglia di protocolli netlink. All'interno di questo modulo sono state ridefinite le system call che riguardano la configurazione delle interfacce di rete.

3.1.3 ptrace

La parte core di umView è costituita dalla system call ptrace (ovvero process trace) che, come suggerisce il nome, permette ad un processo di controllare l'esecuzione di un altro processo.
La system call ptrace viene tipicamente utilizzata all'interno di applicazioni che necessitano di implementare funzioni di breakpoint debugging e system call tracing. L'aspetto interessante di ptrace, e fondamentale per lo sviluppo del progetto così com'è stato pensato, consiste nella possibilità di controllare altre applicazioni avendo i semplici diritti di utente.

3.2 iproute2

iproute2 [8] è una suite di applicazioni pensate per la gestione avanzata della configurazione di rete e per il controllo del traffico TCP/IP a livello datalink. Elemento fondamentale della suite sono i socket netlink considerati una potente interfaccia di configurazione dinamica dello stack di rete. Gli strumenti messi a disposizione da iproute2 si propongono come moderna alternativa agli ormai datati tool di gestione della rete (noti come "net-tools" [9]) che però continuano ad essere ampiamente utilizzati negli script di configurazione. La tabella 3.1 mostra il riferimento incrociato tra i net-tools più comuni e le alternative proposte da iproute2.

All'interno del *monitor* verranno sostanzialmente utilizzati i comandi ip route e ip rule per configurare dinamicamente le tabelle di instradamento delle

Descrizione	net-tools	iproute2
Configurazione delle interfacce	ifconfig	ip addr, ip link
Tabelle di routing	route	ip route
Tabelle arp	arp	ip neigh
Tunneling	iptunnel	ip tunnel
Multicast	ipmaddr	ip maddr

Tabella 3.1: net-tools iproute2 cross-reference

interfacce presenti nel sistema. In particolare, nel modulo dhcpc.c, possiamo trovare alcuni esempi di utilizzo dei comandi

`ip rule add from 217.202.64.71 table 10001`
Aggiunge alla tabella identificata dall'indice 10001 la rotta 217.202.64.71
`ip route add default via 192.168.1.1 dev wlan0 table 10001`
Aggiunge il default gateway alla tabella 10001

E' importante sottolineare che l'utilizzo di iproute2 richiede, in alcuni casi, i diritti di root per poter svolgere operazioni di modifica della configurazione di sistema. Questo è uno dei motivi per cui l'applicazione *monitor* termina in partenza se viene messa in esecuzione con i soli diritti di utente.

3.3 wpa_supplicant

Un supplicant [10] è un'entità hardware o software che si trova ad un capo di una LAN[1] punto a punto ed ha il compito di gestire le procedure di autenticazione con l'entità che si trova all'altro capo. wpa_supplicant [11] è un supplicant WPA ovvero un'applicazione software che permette all'host, nel quale è in esecuzione, di autenticarsi presso una rete IEEE 802.11 mediante

[1]Local Area Network

i protocolli WEP[2], WPA[3] e WPA2.
L'applicazione *monitor* utilizza wpa_supplicant per configurare a livello data-link le interfacce di rete di tipo wi-fi; il livello network verrà invece configurato dal DHCP[4] client integrato nel *monitor*. Naturalmente wpa_supplicant per potersi autenticare presso un WPA authenticator ha bisogno delle credenziali di accesso. Queste vengono scritte nel file di configurazione di wpa_supplicant il cui percorso è indicato in monitor.conf (sezione 6.8).
Per poter comunicare con il *monitor* wpa_supplicant mette a disposizione un'interfaccia di controllo che consente di aprire un socket tra le due applicazioni e ricevere notifiche sullo stato dei dispositivi che si stanno controllando. In particolare il *monitor* userà le seguenti API[5]:

- struct wpa_ctrl* wpa_ctrl_open (const char *ctrl_path);
  apre il socket che servirà per comunicare con wpa_supplicant. In input viene passato il riferimento a wpa_supplicant nella directory /var/run e verrà restituita la struttura wpa_ctrl così definita:

```
struct wpa_ctrl {
    int s;
    struct sockaddr_in local;
    struct sockaddr_in dest;
};
```

  dove s è il file descriptor del socket appena aperto, local la porta di ascolto del *monitor* e dest la porta di ascolto di wpa_supplicant, entrambe su indirizzo locale 127.0.0.1 (se non specificato diversamente).
- int wpa_ctrl_request (struct wpa_ctrl *ctrl, ...); [12]
  esegue l'associazione con un dispositivo wi-fi; da ora in poi il *monitor* riceverà da wvdial le notifiche sulle variazioni di stato del dispositivo.

[2]Wired Equivalent Privacy
[3]Wi-Fi Protected Access
[4]Dynamic Host Configuration Protocol
[5]Application Program Interface

- int wpa_ctrl_recv (struct wpa_ctrl *ctrl, char *reply, size_t *reply_len);
  permette di leggere le notifiche pendenti. La contestualizzazione di questa chiamata, all'interno della logica di controllo del *monitor*, verrà evidenziata nella sezione 6.9.
- void wpa_ctrl_close (struct wpa_ctrl *ctrl);
  chiude il socket precedentemente aperto da wpa_ctrl_open.

3.4 wvdial

Wvdial [13] è un'applicazione nata per gestire connessioni punto a punto tramite modem analogico o digitale. Il modem dovrà essere in grado di interpretare i comandi Hayes [14] presenti nel file di configurazione di wvdial. In particolare, nello scenario che stiamo considerando è previsto un modem di tipo digitale in grado di gestire connessioni UMTS [15]. Il file di configurazione di wvdial sarà quindi specifico per il dispositivo che si sta pilotando e per il gestore di telefonia che fornisce il servizio di connessione dati.
Wvdial gestisce fondamentalmente le operazioni di autenticazione del dispositivo UMTS a livello datalink, mentre demanda al demone pppd le operazioni di connessione punto a punto. Il demone pppd dovrà quindi essere in esecuzone nel sistema nel momento in cui wvdial viene invocato.
Anche in questo caso il livello network viene configurato dal DHCP client integrato nel *monitor*.
Attualmente wvdial non prevede API o altri sistemi che consentano l'integrazione delle funzionalità offerte all'interno di applicazioni di terze parti. Poiché il *monitor* ha necessità di comunicare con wvdial, è stato necessario modificare i sorgenti di quest'ultimo (versione 1.60) in modo da aggiungere un canale di comunicazione bidirezionale sulla porta 8889. Nel file di configurazione del *monitor* (sezione 6.8) viene indicato il percorso in cui si trova l'eseguibile ottenuto dalla compilazione dei sorgenti modificati.

Capitolo 4

Obiettivi

L'obiettivo del progetto è quello di controllare il traffico di rete di una o più applicazioni al fine di utilizzare l'interfaccia di rete *più conveniente* in quel momento, senza generare interruzioni o frammentazioni della comunicazione. Bisogna sottolineare che, nel contesto del sistema, il comportamento appena descritto deve riguardare la singola applicazione, devono cioè essere definite delle politiche a livello di applicazione e non di dispositivo. Va inoltre puntualizzato che il terminine "*più conveniente*" non si riferisce semplicemente alla disponibilità o alle prestazioni[1] del dispositivo, ma coinvolge soprattutto il tipo di applicazione in essere. Ad esempio, se dobbiamo eseguire il download di un file di grandi dimensioni è meglio scegliere l'interfaccia più veloce, mentre se stiamo utilizzando un'applicazione che realizza VoIP è meglio scegliere l'interfaccia più stabile, che garantisce cioè un flusso di dati costante. Possono inoltre essere definite delle politiche di gruppo che vengono applicate a prescindere dal tipo di applicazione. Per esempio ad ogni interfaccia può essere attribuita una priorità che dipende dal costo del collegamento ad essa associato.
Lo scenario descritto nel capitolo 2 rappresenta il punto dal quale siamo partiti per sviluppare il progetto. Attualmente il punto debole dei dispositivi

[1] per esempio ampiezza di banda o potenza del segnale

portatili[2] continua ad essere l'autonomia della batteria, spesso troppo limitata. Sappiamo inoltre che le stime sull'autonomia del dispositivo fornite dai costruttori non prevedono l'utilizzo costante di connessioni wi-fi ed i valori sono centrati su applicazioni di uso comune quali browser oppure visualizzatori di testi ed immagini.
A fronte di questo, l'obiettivo principale è quello di utilizzare degli strumenti privi di funzionalità inutili; ad esempio sarebbe semplice realizzare quello che stiamo proponendo mediante una macchina virtuale completa, ma l'impegno di risorse sarebbe tale da rendere i consumi non compatibili con l'autonomia dei dispositivi mobili.
Nel capitolo 5 vengono presentate diverse soluzioni che tengono conto di questo aspetto cercando di realizzare al meglio gli obiettivi che ci siamo posti.

4.1 Monitor

Vediamo nello specifico quali sono gli obiettivi che vorremmo raggiungere per realizzare il monitor di sistema.
Il *monitor* dovrà svolgere due funzioni principali:

1. fornire informazioni sullo stato delle interfacce di rete alle applicazioni che ne fanno richiesta.
2. controllare e configurare le interfacce di rete in modo da mantenere una coerenza logica tra il loro stato e le relative tabelle di routing.

Per poter essere indipendenti dal contesto della distribuzione utilizzata e dalle convezioni adottate dai sistemi operativi[3], sarà necessario riconoscere le interfacce di rete in base alla famiglia di appartenenza quindi ethernet, wi-fi, umts, bluetooth per citare le più comuni. Questo si rende necessario

[2] laptop, tablet, smartphone
[3] in previsione di un possibile porting

dal momento che ogni famiglia ha bisogno di procedure specifiche per essere configurata a livello datalink e network.

Inoltre nella fase iniziale il monitor dovrà installare e configurare un'interfaccia virtuale[4], mentre al termine del processo l'interfaccia dovrà essere disinstallata e la configurazione di sistema ripristinata.
Infine come evidenziato in figura 5.3 il *monitor* dovrà essere indipendente dal resto dell'architettura in modo da rendere la stessa più modulare possibile.

[4]tunnel a livello IP, quindi tipicamente TUN0

Capitolo 5

Architettura

L'idea di base per realizzare quanto descritto negli obiettivi è quella di catturare alcune system call[1] chiamate dall'applicazione che si vuole controllare, al fine di modificarne il comportamento.

Se dal punto di vista concettuale questa descrizione risulta semplice e lineare, dal punto di vista pratico la definizione dell'architettura ha richiesto la valutazione di tre soluzioni. Nelle sezioni che seguono verranno spiegate le ragioni che ci hanno spinto a scartare le prime due soluzioni e ad accettare l'ultima come migliore compromesso funzionale.
Ricordiamo gli attori principali del sistema:

applicazione da controllare è l'oggetto del lavoro; potrebbe essere un' applicazione qualsiasi, ma l'architettura viene impiegata al meglio solo se l'applicazione scelta utilizza intensamente le comunicazioni di rete. Sarà un processo figlio di umView.

umView gestisce la virtualizzazione delle system call (socket call nel nostro caso).

proxy [16] o sistema di controllo del traffico locale; nasconde le interfacce di rete reali all'applicazione che si sta controllando (e solo ad

[1] In particolare le system call del gruppo sys_socketcall

essa) in modo da veicolare il traffico sull'unica interfaccia virtuale messa a disposizione dal proxy stesso. La scelta dell'interfaccia fisica da utilizzare può essere fatta dal proxy oppure da un modulo specifico di umView.

TUN0 è l'interfaccia virtuale utilizzata per modificare il flusso dati secondo logiche proprie di ciascuna architettura; l'installazione di TUN0 ha il duplice scopo di realizzare un collegamento virtuale con le applicazioni e di istanziare un descrittore reale.

5.1 Singolo proxy

Come si evince dalla figura 5.1 l'applicazione lanciata da umView vede come unica interfaccia di rete TUN0. Le interfacce reali vengono nascoste e non saranno in nessun modo raggiungibili dall'applicazione. In questo caso sarà il proxy a veicolare in traffico da TUN0 verso l'interfaccia reale più conveniente in quel momento. Il proxy dovrà inoltre controllare il traffico in ingresso in modo da inoltrare a TUN0 solo i pacchetti destinati all'applicazione.

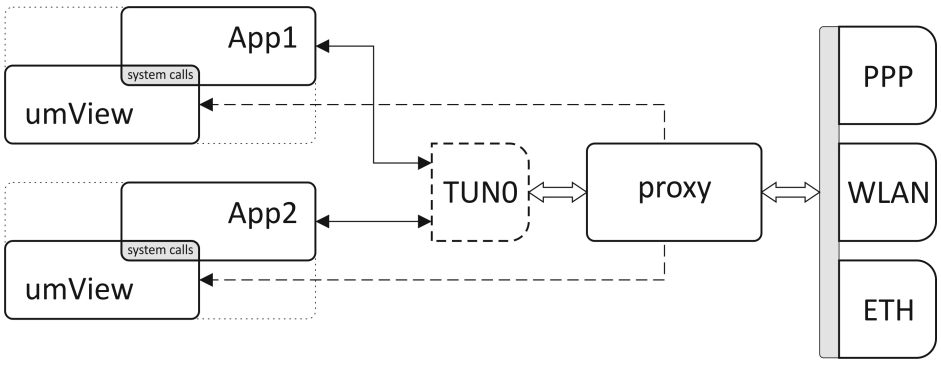

Figura 5.1: Architettura preliminare con singolo proxy

Anche se le figure 5.1 e 5.2 presentano alcune similitudini, la logica di funzionamento è totalmente diversa. Nella soluzione con singolo proxy si vuole implementare un modulo da collegare ad umView in grado di intercettare le

socket call di tipo netlink e limitare la visione dell'applicazione alla sola interfaccia di rete TUN0. Invece, come vedremo nella sezione 5.2, la soluzione con proxy dedicato userà prevalentemente socket INET [17].
L'analisi finale ha evidenziato alcuni punti di forza:

- architettura semplice: la disposizione lineare dei moduli riflette un flusso lineare di dati.
- intervento poco invasivo: le socket call da modificare sono un piccolo sottinsieme delle system call messe a disposizione dal kernel.

Sono stati però individuati anche alcuni punti deboli:

- possibile overload dovuto al passaggio di strutture di controllo tra umView e proxy; nell'ipotesi molto probabile che il sistema venga utilizzato su un dispositivo mobile, l'overload si tradurrebbe in un aumento dei consumi con conseguente calo di autonomia del dispositivo stesso.
- architettura non scalabile; per ogni applicazione bisogna attivare un'istanza di umView, mentre tutto il traffico continuerebbe ad essere veicolato attraverso l'unica interfaccia TUN0. A questo punto diventa difficile associare il flusso dati in ingresso all'applicazione corrispondente.
- la presenza di un singolo proxy rende difficile differenziare le politiche riferite alla qualità del servizio (QoS). In questo caso verrebbe addottata un'unica politica per tutte le applicazioni in essere.

Le ultime due considerazioni ci hanno portato a scartare questa soluzione a favore dell'architettura descritta nella sezione 5.2 che prevede un proxy dedicato per ogni applicazione.

5.2 Proxy dedicato

La soluzione con proxy dedicato non è un semplice aggiornamento della versione con singolo proxy, tant'è che il principio di funzionamento è stato pro-

fondamente rivisto. In questo caso si vuole implementare un modulo da collegare ad umView in grado di intercettare le chiamate di tipo sys_socketcall [18]. L'applicazione controllata da umView sarà in grado di vedere tutte le interfacce di rete presenti sulla macchina, ma l'interazione con esse verrà mascherata dal modulo sopra citato.

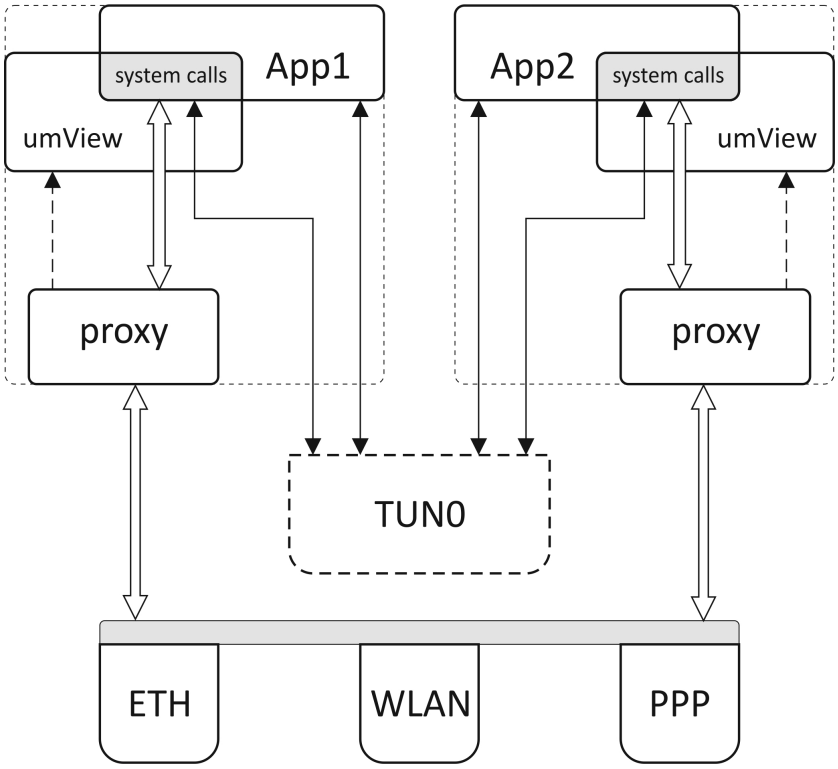

Figura 5.2: Architettura preliminare con proxy dedicato

Per semplicità in figura 5.2 sono state rappresentate due istanze di umView, ma lo stesso modello può essere esteso ad un maggior numero di istanze. Come evidenziato dalla linea tratteggiata, sia l'applicazione che il proxy sono processi figli di umView. Questo consente di realizzare con semplicità un canale di comunicazione interno[2] per lo scambio dei messaggi di controllo. Questa soluzione risolve il problema di possibile sovraccarico descritto

[2]il canale può essere realizzato tramite pipe

nell'architettura con singolo proxy; inoltre umView potrebbe essere opportunamente modificato in modo da gestire il parametro relativo alla QoS da associare all'applicazione. Possiamo ipotizzare la seguente sintassi:

```
$umview -p umnet -qos x app1
$umview -p umnet -qos x app2
```

dove `qos` è la chiave che identifica il parametro e `x` è il valore associato ad una delle otto classi di QoS definite dall'ITU [19].
La presenza di un proxy per ogni istanza di umView risolve l'ultimo problema evidenziato nella precedente architettura relativo alla difficoltà di associare il flusso dati in ingresso all'applicazione corrispondente. In questo caso il problema non sussiste dal momento che umView, applicazione e proxy fanno parte dello stesso nucleo e quindi i riferimenti sono ben delineati.
Durante lo studio di fattibilità si è visto che le tecniche usate per mascherare le interfacce reali non sono risultate particolarmente efficaci sia dal punto di vista dei risultati che dal punto di vista della stabilità di sistema. Venendo a mancare l'elemento centrale dell'architettura si è pensato di procedere con la soluzione descritta nella sezione 5.3 che ha come caratteristiche salienti quella di essere più lasca e modulare rispetto la presente.

5.3 Soluzione definitiva

L'ultima versione dell'architettura ha visto una profonda ristrutturazione del proxy a cui è stato dato il nome *monitor* (capitolo 6) per caratterizzare meglio il ruolo che ricopre. Inoltre è stato aggiunto ad umView il modulo *umNETDIF*[3] che ha il compito di gestire il traffico di rete dell'applicazione da lui controllata.
Si nota subito la diversa collocazione dei moduli: come da specifiche iniziali, umView e applicazione continuano a lavorare in user space, mentre il *monitor*,

[3]Oggetto del lavoro di Raffaele Lovino

avendo necessità di modificare le tabelle di routing e configurare le interfacce di rete, viene eseguito in kernel space.
Il *monitor* mette a disposizione un socket a cui *umNETDIF* si può collegare per ottenere informazioni sullo stato dei dispositivi di rete tutte le volte che sugli stessi si verificherà una variazione a livello network o datalink [20].
Per semplicità in figura 5.3 è stata rappresentata una sola istanza di

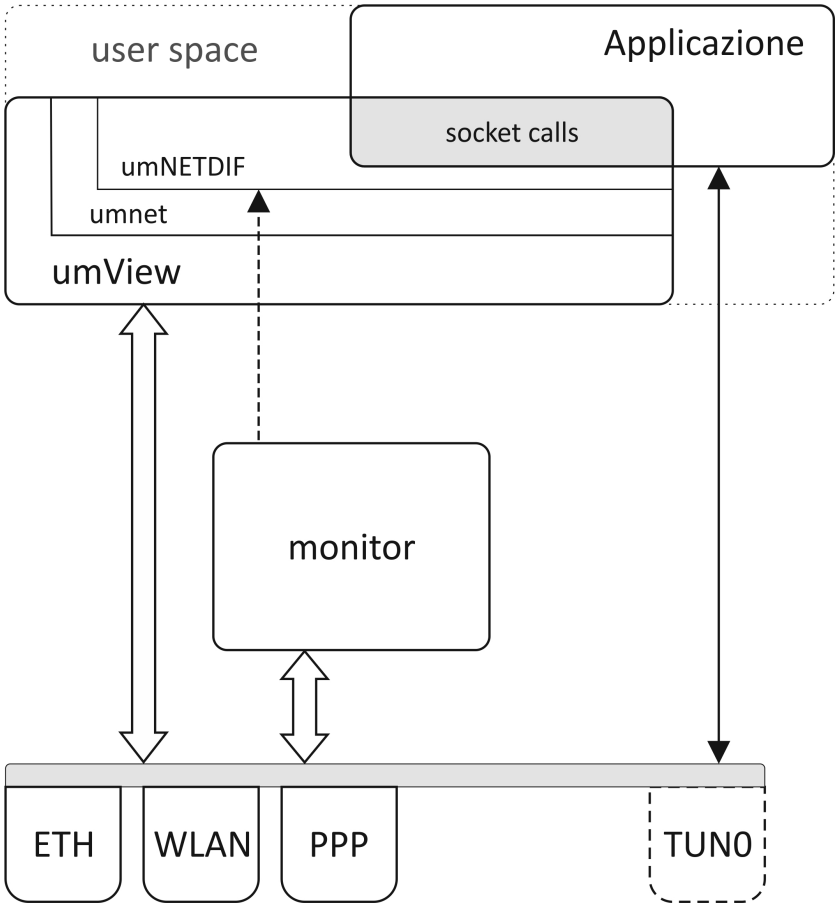

Figura 5.3: Architettura definitiva

umView, ma l'architettura consente la presenza contemporanea di più istanze di umView con relativa applicazione, mentre il *monitor* rimane comunque attivo su singola istanza.
La logica di instradamento dei pacchetti è la seguente:

- *monitor* e *umNETDIF* sono collegati e quindi *umNETDIF* conosce lo stato dei dispositivi di rete presenti sulla macchina. TUN0 è stata attivata e configurata come interfaccia di default.
- L'applicazione spedirà i pacchetti verso TUN0.
- umView intercetta l'operazione di scrittura e cede il controllo della system call a *umNETDIF* il quale provvederà ad incapsulare i pacchetti e a redirigerli verso l'interfaccia di rete più conveniente.
- I pacchetti in ingresso seguiranno il percorso inverso e l'applicazione avrà l'impressione di comunicare esclusivamente tramite TUN0.

Come già anticipato, l'architettura non è priva di difetti: citiamo quelli più evidenti:

- L'applicazione ha visibilità su tutte le interfacce di rete e quindi è potenzialmente in grado di accedervi, ma in assenza di indicazioni specifiche utilizzarà TUN0.
- Il *monitor* imposta delle politiche di instradamento globali che inevitabilmente riguarderanno anche le applicazioni non coinvolte in questo tipo di gestione.

Nonostante tutto, questa soluzione è sembrata essere la più conveniente poiché presenta il giusto equilibrio tra prestazioni e difficoltà di sviluppo.

Capitolo 6

Monitor

Andiamo ora ad approfondire ed analizzare il ruolo che il *monitor* occupa all'interno dell'architettura generale, la logica di funzionamento, i compiti principali che svolge e quelli che demanda ad altre applicazioni.
Anche se il nome può trarre in inganno, il *monitor* non è un semplice supervisore, ma è parte attiva nel processo di scelta e configurazione delle interfacce di rete presenti nel sistema. Le funzionalità principali del *monitor* possono essere riassunte come segue:

- Supervisione di tutte le interfacce di rete presenti nel sistema
- Filtro per le interfacce indesiderate
- Configurazione delle interfacce a livello datalink e network
- Definizione delle politiche di instradamento dei pacchetti
- Comunicazione attiva con le applicazioni locali interessate

Vedremo ora una breve descrizione dell'architettura interna del *monitor* ponendo in evidenza i moduli che meglio ne caratterizzano il funzionamento.

6.1 Architettura

Come si evince dalla figura 6.1 l'architettura è composta dai seguenti blocchi funzionali:

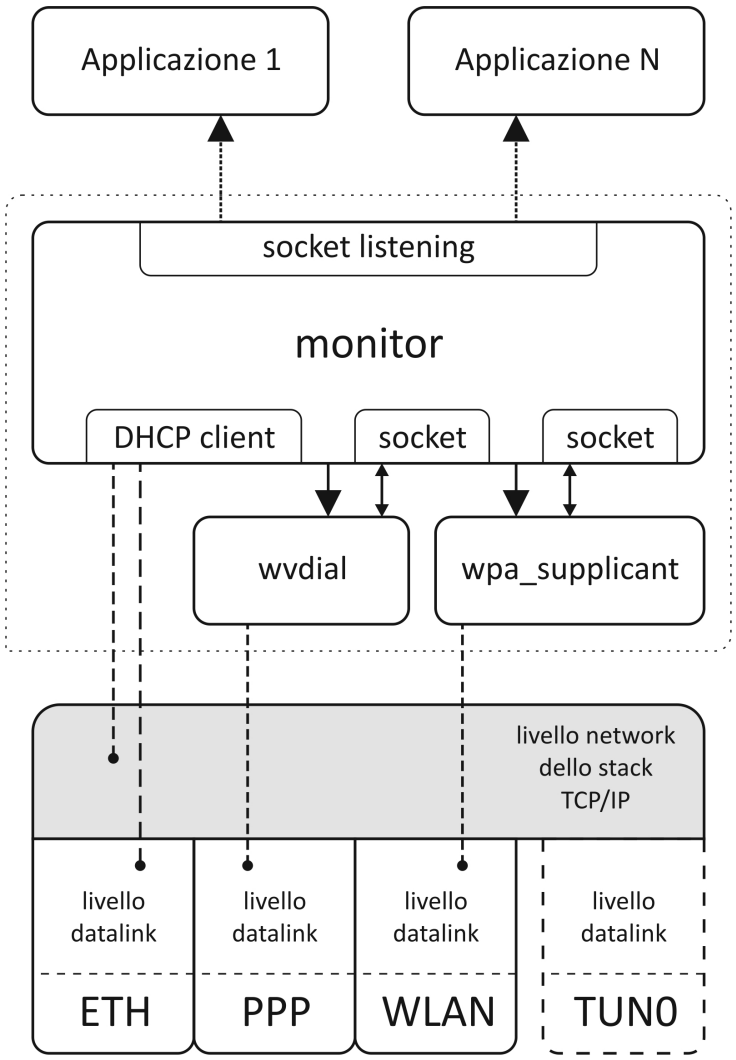

Figura 6.1: Architettura del monitor

socket listening viene utilizzato per comunicare con le applicazioni locali. Quando la configurazione delle interfacce cambia, il *monitor* comunica alle applicazioni collegate il nuovo assetto.

DHCP client è integrato nel *monitor* e serve per configurare il livello network delle interfacce di rete.

wpa_supplicant viene attivato solo in presenza di interfacce wireless (802.11). Serve per configurare il livello data link come descritto nella sezione 3.3

wvdial viene attivato solo in presenza di interfacce punto-punto in modalità UMTS. Serve per configurare il livello datalink delle stesse come descritto nella sezione 3.4

tun0 interfaccia virtuale creata ed attivata dal *monitor* nella fase di inizializzazione. Sarà l'interfaccia di default per tutte le applicazioni del sistema.

6.2 Socket listening

Lo scopo principale del *monitor* è quello di fornire alle applicazioni che ne fanno richiesta informazioni aggiornate sullo stato delle interfacce. Questo tipo di servizio assume un significato specifico nell'architettura che stiamo presentando, ma può essere utilizzato per scopi totalmente diversi da quelli per cui è stato progettato e questo grazie alla natura modulare del progetto nel suo insieme e all'indipendenza funzionale del *monitor*.
Le applicazioni che si collegano al *monitor* riceveranno come prima informazione il numero delle interfacce attualmente attive e successivamente, per ogni interfaccia, una struttura dati msgElem_t così definita:

```
struct msgElem_t {
    unsigned short link_type;
    unsigned short priority;
    unsigned long ip;
    unsigned long netmask;
    unsigned long gw;
    int if_index;
```

```
    int advRouting;
    int cfg_seq;
    char ifname [IFNAMSIZ];
};
```

- **link_type** tipo di interfaccia (1 = [W]LAN ; 2 = PPP)
- **priority** definisce il tipo di priorità (0 = minima ; 1 = normale)
- **ip** indirizzo locale dell'interfaccia
- **netmask** indirizzo di maschera della sottorete. Questo parametro non viene utilizzato se il tipo di interfaccia è PPP (link_type==2)
- **gw** Se link_type==2 definisce l'indirizzo del nodo remoto. Altrimenti definisce l'indirizzo del default gateway della sottorete
- **if_index** identificativo univoco dell'interfaccia
- **advRouting** viene settato a 0 se ci sono stati problemi nella definizione delle regole di instradamento
- **cfg_seq** definisce l'associazione tra configurazione ed interfaccia
- **ifname** nome che il sistema assegna al dispositivo

Le applicazioni per collegarsi al *monitor* dovranno conoscere il numero di porta, che viene passato al *monitor* a linea di comando, ed anche la struttura dati appena descritta, mentre l'indirizzo ip è quello di localhost (tipicamente 127.0.0.1). Nel momento in cui la configurazione di qualche interfaccia cambia, il *monitor* controlla tutte le applicazioni che sono in ascolto e ad ognuna spedisce le informazioni aggiornate sullo stato delle interfacce senza aspettare alcun tipo di conferma. Il *monitor* è in grado di rilevare cambi di configurazione delle interfacce sia a livello datalink, per esempio quando il dispositivo non è più disponibile, sia a livello network, per esempio quando cambia l'indirizzo ip o il gateway [21] di riferimento.
Le applicazioni per monitorare lo stato del socket dovranno presumibilmente implementare un ciclo di controllo mediante la system call select o simile, in modo da catturare le informazioni nel momento in cui vengono rese disponibili.

6.3 DHCP Client

Il DHCP è un protocollo di rete di livello applicativo usato per assegnare agli host che ne fanno richiesta una configurazione IP così definita:

- Indirizzo IP assegnato
- Maschera di sottorete
- Default gateway
- Indirizzo IP del server DNS[1]
- Nome del dominio DNS
- Indirizzi IP dei server NTP[2]
- Altri parametri opzionali e di minore importanza

La configurazione viene calcolata dal server DHCP della sottorete in base a politiche specifiche, associate per esempio al MAC[3] address dell'interfaccia, oppure in base a politiche di gruppo. In quest'ultimo caso il server dispone di un numero circoscritto di configurazioni da assegnare dinamicamente.
L'altro attore è il DHCP client, che ha il compito di interrogare il server, gestire la risposta e configurare il livello network del dispositivo per il quale sta eseguendo la richiesta.
Server e client utilizzano rispettivamente le porte 67 e 68 ed il protocollo UDP.
Il *monitor* integra un DHCP client modificato con il quale configura il livello network dei dispositivi di rete della macchina locale e definisce le politiche di instradamento dei pacchetti. Si è scelto di integrare il DHCP client per poter accedere direttamente alle sue funzionalità e personalizzarne il comportamento.
Per ogni dispositivo viene lanciata un'istanza del DHCP client sotto forma di thread parallelo a quello principale. La figura 6.2 mostra come il DHCP client si comporta in presenza di un dispositivo ethernet.

[1]Domain Name System
[2]Network Time Protocol
[3]Media Access Control

E' importante sottolineare che, essendo il DHCP client stato integrato all'interno del monitor, la sua attivazione e successiva disattivazione coincidono rispettivamente con l'esecuzione e la terminazione di un thread parallelo.

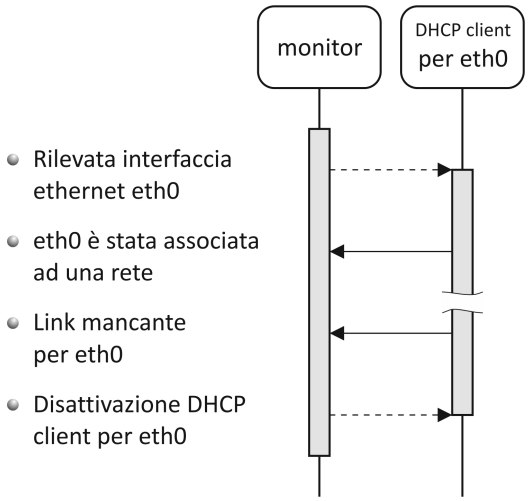

Figura 6.2: Interazione tra *monitor* e DHCP client durante la gestione di un dispositivo ethernet

6.4 wpa_supplicant: configurazione dei dispositivi wi-fi

I driver dei dispositivi wi-fi fanno sì che i dispositivi stessi, a livello data link e più precisamente al livello LLC[4], vengano visti come interfacce ethernet. Se da un lato questa semplificazione comporta ovvi vantaggi di gestione, dall'altro può essere di ostacolo nel momento in cui ci sia la necessità di riconoscere la natura del dispositivo che si sta considerando. Il *monitor* deve poter operare in entrambe le modalità, infatti:

1. il file di configurazione prevede la possibilità di includere (o escludere) il singolo dispositivo identificandolo mediante il nome oppure median-

[4]Logical Link Control - Sottolivello superiore del data link layer

te la famiglia di appartenenza (ethernet, wi-fi, umts). In quest'ultimo caso, quando un dispositivo deve notificare un cambio di stato, il *monitor* deve essere in grado di riconoscere a quale famiglia appartiene per sapere se considerarlo o meno.

2. il *monitor* applica politiche di instradamento diverse a seconda della famiglia a cui il dispositivo appartiene.

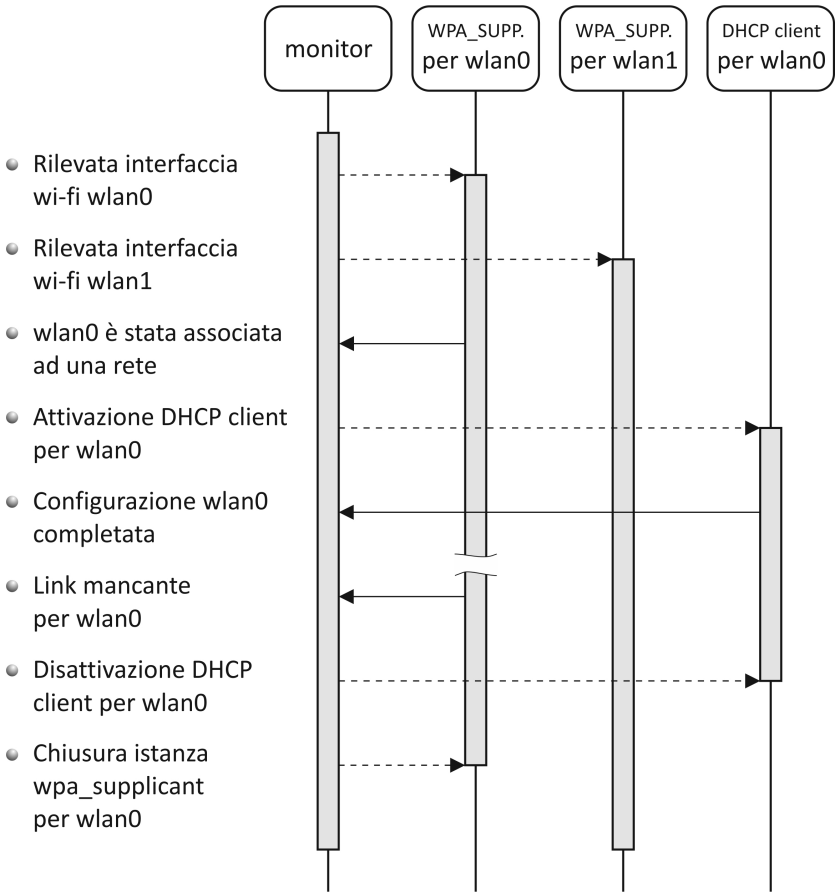

Figura 6.3: Interazione tra *monitor* e wpa_supplicant durante la gestione dei dispositivi wi-fi

Per questo sono state implementate tre funzioni, descritte nella sezione 6.7 che, preso in input il nome del dispositivo, ricavano l'indice ad esso associato e riconoscono la famiglia di appartenenza restituendo 1 nel caso ci sia corrispondenza e 0 altrimenti.

La figura 6.3 evidenzia i soggetti coinvolti nelle operazioni di aggiornamento di sistema nel momento in cui una scheda wi-fi cambia di stato.
E' importante ricordare che, rispetto al *monitor*, wpa_supplicant è un processo figlio, mentre il DHCP client è un thread parallelo.

6.5 wvdial: configurazione dei dispositivi UMTS

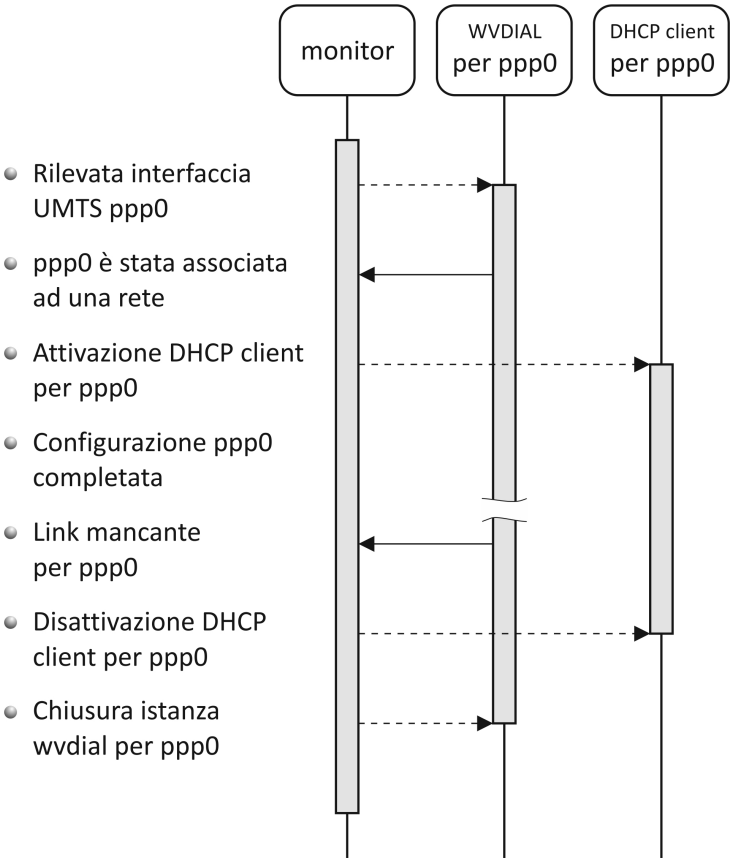

Figura 6.4: Interazione tra *monitor* e wvdial durante la gestione dei dispositivi umts

Come si evince dalla figura 6.4 l' interazione tra *monitor* e wvdial è del tutto simile a quella descritta nella sezione 6.4. L'architettura modulare permette

infatti di mantenere coerenza a livello logico e quindi la gestione delle diverse famiglie di dispositivi viene fatta semplicemente differenziando le chiamate ai moduli funzionali.

6.6 TUN

Nella fase di inizializzazione del *monitor* (sezione 6.9) è prevista l'installazione e la configurazione di un'interfaccia virtuale di tipo TUN. La scelta di utilizzare un'interfaccia di tipo TUN anzichè TAP [22] deriva dal fatto che, mentre la prima è in grado di trasmettere pacchetti TCP/IP, la seconda viene usata per trasmettere frames ethernet. Quindi l'interfaccia TAP fa passare qualsiasi protocollo, mentre noi siamo interessati a limitare il traffico al solo protocollo IP.
Siccome sulla macchina locale non è possibile escludere la presenza di una rete virtuale punto-punto, l'interfaccia virtuale installata dal *monitor* avrà TUNH0 come nome di default in modo da evitare conflitti. Sarà possibile cambiare questo valore andando a specificarne uno diverso nel file di configurazione monitor.conf.
Come evidenziato nel capitolo 7 la configurazione delle tabelle di routing sarà tale da fare risultare TUNH0 l'interfaccia di riferimento per tutte le applicazioni che adottano politiche di default nella trasmissione e ricezione dei pacchetti TCP/IP. D'altra parte TUNH0 non è fisicamente né tantomeno logicamente collegata ad un altro end point. Ci dovrà quindi essere un'applicazione (nel nostro caso umView) che controlla costantemente TUNH0 e veicola il suo traffico verso l'interfaccia fisica che in quel momento risulta essere più vantaggiosa. Questa architettura può essere scalata mantenendo TUNH0 come unica interfaccia di riferimento.
Le funzioni riguardanti la gestione dell'interfaccia TUN sono contenute nei moduli local_tun.c e tun.c. L'indirizzo di defaut 10.254.254.253 può essere cambiato modificando il file di configurazione monitor.conf.

6.7 Riconoscimento automatico delle interfacce

Allo stato attuale il monitor è in grado di riconoscere automaticamente le interfacce delle famiglie:

- ethernet 802.3
- wireless 802.11
- ppp in modalità UMTS

Il riconoscimento automatico si è reso necessario causa la mancata standardizzazione del nome dei dispositivi. Generalmente ai prefissi ETH, WLAN, PPP vengono rispettivamente associate le interfacce ethernet, wi-fi e modem (UMTS nel nostro caso), ma chi possiede i diritti di amministratore ha la libertà di cambiare questi riferimenti. Per farlo basta accedere al file 70-persistent-net.rules contenuto nella directory /etc/udev/rules.d, trovare il MAC dell'interfaccia e modificare il nome assegnato dal sistema. Di seguito è riportato un esempio che descrive la stringa di configurazione dell'interfaccia a cui è stato assegnato il nome eth0:

```
# PCI device 0x8086:/sys/devices/pci0000:00/0000:00:03.0 (e1000)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
    ATTR{address}=="08:00:27:fa:dd:a6", ATTR{dev_id}=="0x0",
    ATTR{type}=="1", KERNEL=="eth*", NAME="eth0"
```

E' sufficiente cambiare il valore identificato dalla chiave NAME per variare il nome dell'interfaccia.
Avendo capito che non è possibile fare affidamento sul nome del dispositivo per identificarne la famiglia di appartenenza, è stato necessario scrivere tre funzioni in grado di svolgere questo compito utilizzando la system call ioctl [23] che ricordiamo essere utilizzata per il dialogo con le periferiche. Di seguito sono riportate le intestazioni delle tre funzioni:

- int is_wired (char* ifname);
  controlla se il dispositivo appartiene alla famiglia ethernet 802.3 [24] mediante una chiamata ioctl di tipo SIOCETHTOOL
- int is_wireless (char* ifname);
  controlla se il dispositivo appartiene alla famiglia wireless 802.11 mediante una chiamata ioctl di tipo SIOCGIWNAME
- int is_umts (char* ifname);
  controlla se il dispositivo è di tipo UMTS mediante uno scambio di messaggi basati sullo standard Hayes.

Come si può notare le funzioni hanno come unico parametro formale il nome dell'interfaccia. Partendo da questo viene ricavato, prima l'identificativo unico del dispositivo prendendo le informazioni dalla directory indicata nella stringa di configurazione (/sys/devices/...), e poi viene fatto un test specifico (per esempio con ioctl) per dedurre la famiglia di appartenenza. Le tre funzioni vengono testate in cascata e la sequenza si blocca nel caso il risultato sia positivo[5]. Sono stati fatti diversi test per verificare che i risultati non producessero né falsi negativi né tantomeno falsi positivi, che porterebbero ad associazioni multiple tra famiglie e dispositivo.

6.8 File di configurazione

Come è facile immaginare monitor.conf è il file di configurazione del *monitor*, è editabile mediante un normale editor di testo e definisce una serie di parametri nella forma : CHIAVE = VALORE
Il file di configurazione viene letto e valutato alla partenza del processo. Qualsiasi anomalia viene segnalata mediante un messaggio di errore puntuale e il processo termina.
All'interno sono presenti 3 sezioni:

[5]Le funzioni restituiscono 1 in caso di corrispondenza e 0 altrimenti

Generale contiene i parametri che non hanno una particolare caratterizzazione.

Interfacce contiene i parametri relativi alla definizione delle interfacce ammesse. In particolare i parametri con suffisso _AUTO definiscono le politiche per il riconoscimento automatico di famiglie di interfacce.

Path contiene i percorsi relativi ai file di configurazione e alle applicazioni utilizzate dal *monitor*. L'esistenza dei percorsi viene verificata alla partenza del *monitor* in modo da evitare comportamenti anomali durante il funzionamento.

6.9 Flow chart

Vediamo ora il diagramma di flusso relativo al main del *monitor* e riferito al file init.c. Per motivi di sintesi non sarà possibile presentare tutti i dettagli che accompagnano le funzioni principali, ma sarà comunque possibile capire la logica entro la quale il *monitor* opera.
Di seguito vegono descritte le funzioni principali rappresentate nel diagramma di flusso di figura 6.5:

1. Viene letto il file di configurazione e verificata la validità formale dei dati. Se qualche test fallisce il processo termina ed il controllo viene restituito alla shell. Si rimanda alla sezione 6.8 per maggiori dettagli.
2. Nella fase di inizializzazione vengono compiuti i seguenti passi:
   - apertura del socket che verrà utilizzato per comunicare con le applicazioni locali alla macchina
   - apertura del socket per wvdial (nel caso venga utilizzato)
   - controllo del possesso dei privilegi di root: se questo test fallisce il processo termina
   - apertura di un socket netlink necessario per ricevere informazioni riguardo lo stato dei link

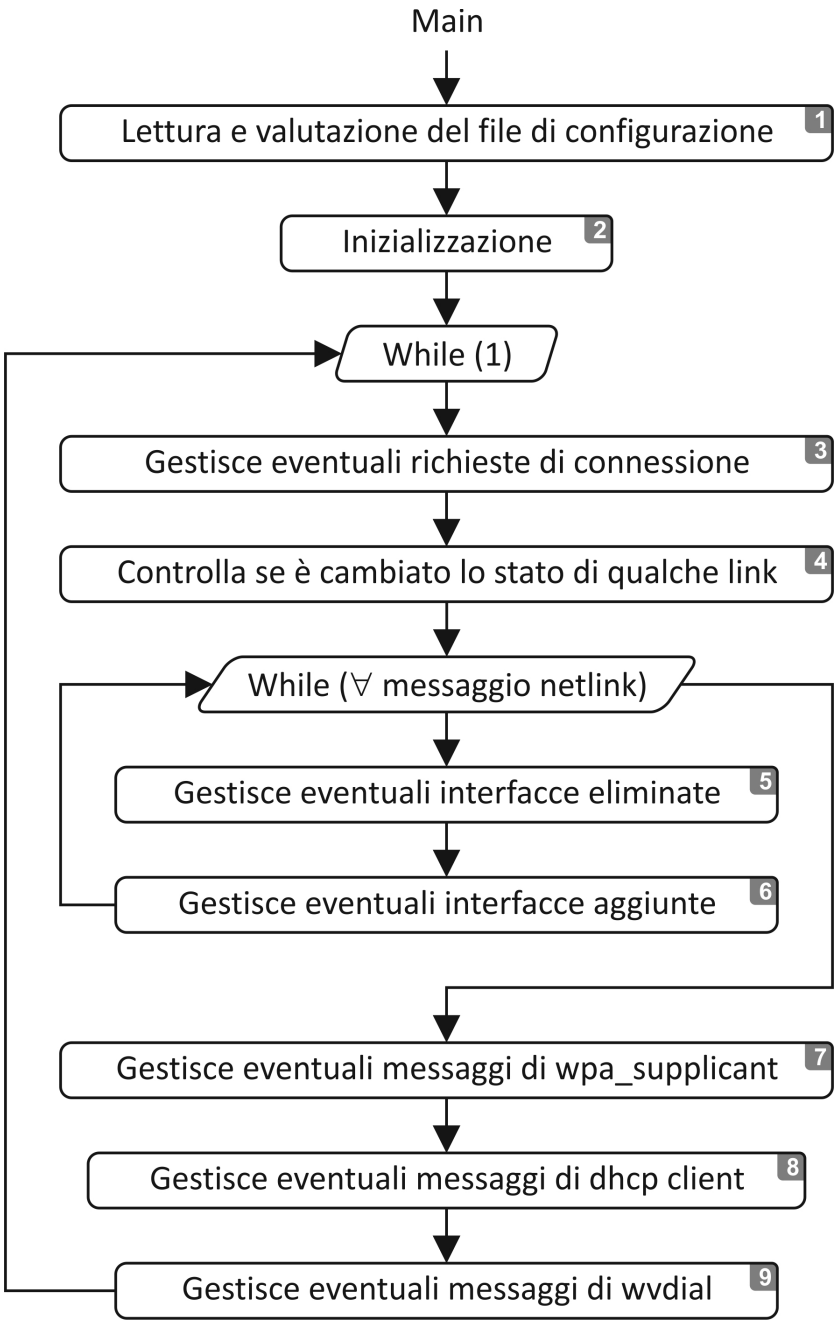

Figura 6.5: Logica di funzionamento del monitor

- creazione e configurazione dell'interfaccia TUNH0

3. La richiesta di connessione può arrivare da un'applicazione locale oppure da wvdial. In entrambi i casi si procederà, prima ad aprire un socket e poi:
   - nel caso la richiesta arrivi da un'applicazione locale verranno spedite all'applicazione stessa le informazioni riguardanti lo stato delle interfacce (vedi sezione 6.2)
   - nel caso la richiesta arrivi da wvdial, dopo la connessione, il *monitor* riceverà un aggiornamento sullo stato delle interfacce UTMS. A sua volta il *monitor* informerà tutte le applicazioni collegate in quel momento riguardo la situazione aggiornata.
4. Vengono controllate eventuali variazioni a livello datalink limitatamente alle interfacce ammesse dal file di configurazione. Tipicamente in questo step viene rilevata l'aggiunta o la perdita di un'interfaccia come descritto nei punti 5 e 6. In tutti i casi il DHCP client aggiorna le tabelle di routing in modo da mantenere coerente lo stato attuale.
5. Viene terminata l'istanza del DHCP client, wpa_supplicant o wvdial a seconda che l'interfaccia persa sia di tipo ethernet, wi-fi o umts.
6. Viene attivata un'istanza del DHCP client, wpa_supplicant o wvdial a seconda che l'interfaccia aggiunta sia di tipo ethernet, wi-fi o umts.
7. I messaggi provenienti da wpa_supplicant riguardano fondamentalmente la notifica dei link tolti e aggiunti oppure il risultato delle operazioni di configurazione delle interfacce wi-fi.
8. Il *monitor* riceve notifiche dal DHCP riguardo l'avvenuta configurazione delle interfacce a livello network. Vengono gestiti esclusivamente i messaggi status_up, status_down, terminate, open_raw_socket, close_raw_socket, update_dns.
9. Come il punto 7, ma con riferimento alle interfacce UMTS.

Capitolo 7

Valutazioni

Al momento sono state tracciate le linee guida del progetto e sono stati impostati i moduli principali, ma mancano ancora le funzioni che realizzano in concreto l'interazione tra i moduli. Questa è la ragione per cui, nel presente capitolo, verranno trattati esclusivamente i test eseguiti sul modulo chiamato *monitor*, argomento specifico di questa tesi.

7.1 Strumenti

Tutti i test sono stati fatti utilizzando il sistema operativo GNU/Linux [25] con kernel versione 3.2.0 eseguito, per semplicità, nella macchina virtuale di Oracle, VirtualBox [26] versione 4.3.12, mentre la distribuzione di riferimento è Ubuntu [27] versione 12.04.5 LTS.
Sul fronte del software, oltre al *monitor* sono stati utilizzati gli strumenti presentati nelle sezioni 3.2, 3.3 e 3.4.
Vediamo invece i dettagli dell'hardware:

piattaforma hardware : utilizzando una macchina virtuale l'hardware sottostante perde parzialmente di significato; l'output del comando *uname*

-i ci informa riguardo le caratteristiche della piattaforma che risulta essere a 32bit e i386 compatibile.

interfaccia ethernet : viene emulata da VirtualBox mediante una scheda Intel PRO/1000 MT (82540EM); il bit rate è ovviamente limitato dalla piattaforma sottostante che, nel nostro caso, accetta connessioni fino a 1GiBps.

interfacce wi-fi : sono stati utilizzati due adattatori wi-fi LAN modello Digicom 8E4213 con interfaccia USB[1] e compatibili con gli standard 802.11g e 802.11b.

interfaccia UMTS : è stato utilizzato un adattattore UMTS LAN modello HAUWEI K3765 con interfaccia USB e compatibile con gli standard HSDPA[2] e HSUPA[3]

E' importante sottolineare che VirtualBox, pur astraendosi dal sistema sottostante, previa autorizzazione ha la possibilità di prendere il controllo delle periferiche USB. E' interessante monitorare l'hardware collegato al sistema reale per vedere in diretta la rimozione delle periferiche USB da parte di VirtualBox durante la fase di boot e la conseguente installazione di queste nel sistema virtuale. L'evidente vantaggio consiste nel poter utilizzare esattamente l'hardware riferito alle periferiche USB.

7.2 Test

I test sono serviti per verificare il comportamento del monitor in relazione agli obiettivi descritti nella sezione 4.1. In particolare abbiamo potuto suddividere i test in due categorie, quelli rivolti alle applicazioni e quelli eseguiti sui dispositivi di rete.

[1] Universal Serial Bus
[2] High-Speed Downlink Packet Access
[3] High-Speed Uplink Packet Access

7.2.1 Comunicazione con le applicazioni

Questa serie di test ha avuto come obiettivo quello di verificare il rispetto delle specifiche descritte nella sezione 6.2. Per raggiungere lo scopo è stata scritta una semplice applicazione chiamata test_crtl_socket che si collega alla porta 3001 di localhost[4] e attende prima il numero delle interfacce e poi un record[5] per ogni interfaccia di rete attiva nel sistema. Nell'esempio che segue vediamo le informazioni che stampa test_crtl_socket dopo essersi collegata al *monitor* ed aver ricevuto le informazioni sullo stato dei dispositivi di rete che, nel caso specifico, erano costituiti da un'interfaccia ethernet ed un adattatore wi-fi LAN:

```
2 interfacce presenti
1 - IP 10.0.2.15, NETMASK 255.255.255.0, GW 10.0.2.2, \
    IF_INDEX 2 - eth3, advRouting 0
2 - IP 10.0.2.16, NETMASK 255.255.255.0, GW 10.0.2.2, \
    IF_INDEX 3 - wlan0, advRouting 0
```

Si ricorda che il *monitor* fornirà informazioni limitatamente ai dispositivi di rete selezionati nel file di configurazione (sezione 6.8).

Per semplicità l'applicazione di test si scollega subito dopo aver ricevuto i dati e termina l'esecuzione a seguito della scrittura delle informazioni. Uno sviluppo futuro dell'applicazione potrebbe prevedere la possibilità di mantenere in essere il collegamento in modo da ricevere notifiche tutte le volte che cambia la configurazione di qualche interfaccia di rete.

7.2.2 Dispositivi di rete

Questa serie di test è servita per verificare il comportamento del *monitor* al variare dei dispositivi di rete collegati alla macchina. Come descritto nella

[4] per i test il *monitor* viene configurato anch'esso sulla porta locale 3001

[5] il tracciato record è riportato nella sezione 6.2

sezione 6.2 il *monitor* è in grado di rilevare variazioni sia a livello network che a livello datalink. Dobbiamo quindi precisare con quali modalità abbiamo potuto agire sui singoli dispositivi:

ethernet : essendo la scheda di rete interna non siamo riusciti a scollegarla; abbiamo invece potuto variare il livello network agendo sul cavo RJ45.

wi-fi : in questo caso abbiamo potuto variare sia il livello datalink, collegando o scollegando il dispositivo USB, sia il livello network agendo sulla presenza o meno dell'access point.

umts : in questo caso abbiamo potuto variare solo il livello datalink, collegando o scollegando l'adattatore USB, mentre non è stato possibile schermare il dispositivo rispetto al segnale proveniente dal ponte radio.

	0	1	3	7	f	e	c	8	-	2	6	4	5	d	9	b	a	-	-
eth0	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0
wlan0	0	0	1	1	1	1	0	0	0	1	1	0	0	0	0	1	1	0	0
wlan10	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0
ppp0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0

Tabella 7.1: sequenza di test riferiti alle interfacce di rete; le colonne identificate dal simbolo - si riferiscono a configurazioni ripetute.

La tabella 7.1 mostra la sequenza di test eseguiti sul *monitor*, dove 0 significa dispositivo assente, mentre 1 significa dispositivo presente. La tabella deve essere letta da sinistra a destra e come si può notare ogni configurazione differisce dalla precedente esattamente per un dispositivo, questo per minimizzare le variabili in essere in caso di errore.
Sono state coperte tutte le possibili combinazioni di 2 elementi presi a gruppi di 4 per un totale di 16 test significativi, mentre 3 sequenze sono state ripetute per rispettare l'invariante del punto precedente. Bisogna però sottolineare che, essendo significativo anche l'ordine di apparizione dei dispositivi, una serie completa di test richiederebbe 64 sequenze corrispondenti alle possibili

disposizioni semplici di 4 elementi distinti, per gruppi di cardinalità 1, 2, 3 e 4.

Durante il periodo di attività il *monitor* stampa una sequenza di messaggi di log corrispondenti al susseguirsi degli eventi che si verificano. In particolare è possibile vedere quando un dispositivo di rete si collega e quando ha terminato la fase di configurazione. Di seguito riportiamo la sequenza dei messaggi rilevati nel momento in cui un'interfaccia di tipo ethernet completa la fase di configurazione:

```
...
netmask 255.255.255.0
LOG_DHCP - LOG_DHCP - eth3: Aggiungo default gw
/sbin//route add default gw 10.254.254.253 dev tunh0
LOG_DHCP - LOG_DHCP - eth3: Creo tabella x interfaccia
LOG_DHCP - LOG_DHCP - eth3: UPDATE_DNS spedito
LOG_DHCP - LOG_DHCP - eth3: STATUS_UP spedito
Wed Oct 29 12:31:14 2014
 Dispositivo configurato, interfaccia [eth3], if_index [2]
////////////////////////////////////////////////
eth3 - IP 10.0.2.15  LAN
IF_INDEX 2, advRouting 0
NETMASK 255.255.255.0 GW 10.0.2.2  cfg_seq 1
////////////////////////////////////////////////
...
```

7.3 Conclusioni

Tutte le prove effettuate hanno dato esito positivo, hanno cioè riportato dei risultati in linea con le aspettative. Si vuole però sottolineare che il numero di test è stato limitato rispetto ai possibili scenari ed inoltre il numero di dispositivi testati rappresenta una piccola parte di quelli disponibili in commercio.

Nonostante ciò riteniamo significativi i test descritti nelle sezioni 7.2.1 e 7.2.2 ed in particolare:

- per quanto riguarda le applicazioni possiamo dire di non aver avuto difficoltà nel gestire il dialogo con il *monitor*: il sistema utilizzato è molto semplice e di conseguenza sufficientemente robusto da poterlo considerare affidabile.
- per quanto concerne invece i dispositivi possiamo concludere che i risultati sono tali da ritenere il sistema sufficientemente stabile e reattivo sia rispetto alla presenza/assenza dei dispositivi di rete, sia rispetto alle variazioni di qualità del canale di comunicazione.

Capitolo 8

Sviluppi futuri

Rimane ancora molto da fare per raggiungere gli obiettivi descritti nel capitolo 4 sia per ciò che riguarda la parte generale che per il modulo chiamato *monitor*.

A livello generale manca ancora la parte riguardante la scelta dell'interfaccia di rete da utilizzare: come abbiamo discusso nel capitolo 4, questa deve essere la "*più conveniente*" e quindi deve essere parametrizzata sul tipo di applicazione che si sta considerando in un certo istante, senza dimenticare eventuali politiche di gruppo che inevitabilmente faranno aumentare la complessità dell'algoritmo di decisione.
Un altro punto importante ancora da sviluppare riguarda la gestione dei pacchetti, da quando vengono spediti dall'applicazione a quando realmente raggiungono l'interfaccia di rete che dovrà farsi carico della spedizione e viceversa. In questo caso sarà necessario incapsulare i pacchetti che l'applicazione spedisce al fine di aggiungere le informazioni necessarie al riconoscimento dell'applicazione stessa; queste informazioni verranno utilizzate per risalire alla sorgente nel momento in cui il pacchetto torna indietro. Ricordiamo infatti che i pacchetti spediti dall'applicazione conterranno l'indirizzo IP di TUN0, mentre l'interfaccia reale sulla quale i pacchetti transiteranno avrà inevitabilmente un indirizzo diverso.

Per concludere la parte generale bisogna ricordare che attualmente umView non ha un modulo in grado di interrogare il *monitor* al fine di ottenere le informazioni sullo stato delle interfacce di rete. Anche questo aspetto dovrà essere implementato nella maniera più opportuna: una soluzione potrebbe essere quella di gestire l'attesa dei messaggi in un thread parallelo mediante l'uso della system call select o poll.

8.1 Monitor

Il lavoro principale di questa tesi ha riguardato il modulo chiamato *monitor*; entreremo quindi nel dettaglio di ciò che riteniamo debba essere ancora fatto per ottenere un'unità funzionale allineata con gli obiettivi descritti nella sezione 4.1.

Attualmente il *monitor* è in grado di riconoscere e gestire le interfacce di rete delle famiglie ethernet (802.3), wi-fi (802.11) e UMTS. Uno sviluppo futuro non potrà prescindere dall'ampliare la gamma dei dispositivi aggiungendo per esempio gli standard Bluetooth [28] e GSM [29], per citare i più noti.

Un altro aspetto ancora da sviluppare riguarda le politiche di instradamento dei pacchetti, un termine generico che può essere specificato dai seguenti 3 punti:

1. l'attuale approccio basato sulle tabelle di routing limita la possibilità di valorizzare le caratteristiche della singola applicazione. Il limite riguarda fondamentalmente l'aumento di prestazioni, un aspetto che può essere secondario per alcune applicazioni (ad esempio quelle che realizzano messaggistica istantanea), ma diventa determinante per le applicazioni che gestiscono un flusso costante, e spesso elevato, di dati (come potrebbe essere lo streaming video o la telefonia VoIP). L'evoluzione di questo aspetto dovrà tenere conto dei limiti appena descritti cercan-

do di trovare una soluzione che probabilmente prescinde dall'uso delle tabelle di routing.

2. un'aspetto collegato al precedente riguarda la scelta dell'interfaccia di rete; la definizione globale delle politiche di instradamento non consente di creare un legame tra la singola applicazione e l'ordine dei dispositivi di rete da utilizzare. La soluzione più facile da realizzare consiste nel demandare la decisione ad un modulo di umView; in questo caso sarebbe molto semplice per umView ottenere informazioni sulle caratteristiche dell'applicazione che sta controllando e comportarsi di conseguenza. Una soluzione più evoluta consiste invece nel centralizzare l'algoritmo di decisione in modo da gestire ogni applicazione in relazione alla situazione globale. L'algoritmo di decisione trova la sua naturale collocazione nel *monitor*, dal momento che questo viene eseguito in singola istanza.
3. l'ultimo aspetto riguarda l'attuale impossibilità di applicare le regole solo alle applicazioni controllate da umView. Uno sviluppo futuro dovrà prevedere un'architettura in grado di lasciare inalterato il comportamento delle applicazioni non espressamente coinvolte nel processo.

Bisogna specificare che i punti 1 e 2 riportano considerazioni sulla QoS, ma mentre il primo sviluppa l'aspetto logico, strettamente legato alla definizione delle tabelle di routing e del protocollo di livello trasporto, il secondo pone l'accento sull'aspetto fisico, legato cioè alle prestazioni del mezzo trasmissivo.

Ricordiamo che il *monitor* incorpora un dhcp client opportunamente modificato che utilizza per configurare il livello network delle interfacce di rete. Nell'ottica di realizzare un collegamento con un altro endpoint, il dhcp server viene tipicamente collocato in una macchina terza oppure su un dispositivo dedicato come potrebbe essere il router della sottorete locale. Quindi attualmente la macchina che ospita il *monitor* deve appartenere ad un'infrastruttura in grado di gestire perlomeno l'assegnamento centralizzato degli indirizzi IP.

Uno sviluppo futuro potrebbe prevedere connessioni ad-hoc realizzate tramite interfacce ethernet, utilizzando cavo patch incrociato di categoria 5 o 6, oppure tramite le interfacce wi-fi dei dispositivi. In questo caso sarà necessario prevedere un meccanismo che sostituisca le funzionalità del dhcp client e server.

Per concludere ricordiamo che il *monitor* prevede un thread di controllo in grado di intercettare il traffico di TUN0. Attualmente non viene fatta alcuna operazione sui pacchetti in transito, ma uno sviluppo futuro potrebbe prevedere operazioni di prerouting utili al modulo di umView per velocizzare o migliorare la scelta dell'interfaccia di rete fisica da utilizzare.

Conclusioni

Il presente lavoro è servito ad impostare le basi di un sistema in grado gestire il traffico in mobilità all'interno di una rete di access point abilitati a tale scopo. L'idea è una diretta conseguenza della crescente diffusione dei dispositivi mobili e i dati presentati nell'introduzione, in particolare nella tabella 1.2, confermano nel lungo periodo l'utilità del lavoro svolto.

Inizialmente è stato presentato uno scenario reale che ha messo in evidenza le caratteristiche peculiari del sistema. Si è potuto notare che le connessioni umts non sono state escluse, poichè verranno utilizzate al bisogno nel caso in cui la rete wi-fi venga a mancare. E' stato poi presentato umView quale componente strategico dell'architettura.

Le caratteristiche di umView sono state riprese nella sezione relativa agli strumenti, ne è stata messa in evidenza la struttura modulare ed è stato discusso il ruolo che la system call ptrace occupa al suo interno. Siamo poi scesi nel dettaglio degli strumenti utilizzati dal *monitor* ed in particolare è stata presentata la suite di applicazioni iproute2, non solo in relazione al presente lavoro, ma anche in riferimento agli ormai datati net-tools . Si è infine vista l'importanza di wpa_supplicant e wvdial quali strumenti utilizzati per configurare il livello datalink rispettivamente dei dispositivi wi-fi e umts.

Nel presentare gli obiettivi è stata fatta una panoramica di carattere generale allo scopo di contestualizzare il lavoro riguardante il *monitor*. In particolare sono stati messi in evidenza i due aspetti caratteristici di questo modulo, riferiti uno alle applicazioni ed l'altro alle interfacce di rete.

L'architettura del sistema è stata definita dopo aver valutato altri due modelli che non sono poi risultati adatti allo scopo. Inizialmente è stato valutato il modello con singolo proxy che, a fronte di una struttura semplice e lineare, ha mostrato alcune incertezze nel momento in cui abbiamo cercato di scalarlo aggiungendo due o più istanze di umView. Il modello con proxy dedicato ha risolto alcuni problemi della versione precedente, ma è risultato instabile nel momento in cui si è tentato di mascherare le interfacce di rete mediante l'uso di socket netlink. Infine abbiamo pensato di progettare l'architettura avendo come linea guida l'indipenza funzionale dei moduli. Siamo riusciti ad ottenere un risultato accettabile aggiungendo il modulo chiamato *monitor* e mettendo in comunicazione le varie parti mediante socket.

Siamo poi entrati nel merito del *monitor*, descrivendone prima l'architettura e poi alcuni dettagli sui moduli che la compongono, ponendo particolare attenzione all'interazione che questi hanno con l'ambiente esterno. Seguono i motivi che ci hanno spinto ad introdurre l'interfaccia virtuale TUN ed il ruolo che questa occupa all'interno dell'architettura. Sono state affrontate le problematiche relative al riconoscimento automatico delle interfacce di rete ed infine è stato presentato e commentato il flow chart che descrive la logica di funzionamento del *monitor*.

I test sono serviti a dimostrare la correttezza e la validità del lavoro svolto. In particolare sono stati fatti due tipi di test, uno riferito alle applicazioni ed uno rivolto alle interfacce di rete. Sebbene tutte le prove siano state significative, se non altro per l'oggettività del risultato, quelle riferite alle interfacce non hanno coperto tutti i possibili casi. Riteniamo comunque che i risultati ottenuti siano sufficienti per affermare che il prodotto sia affidabile allo stato attuale dell'arte.

In conclusione sono stati elencati i punti deboli dell'architettura generale e del *monitor*, allo scopo di fornire degli spunti per possibili futuri aggiornamenti.

Il lavoro svolto, oltre ad avere avuto un interesse intrinseco legato all'attività di ricerca, è servito per tracciare le linee guida di un progetto molto

ambizioso, che ci auguriamo possa essere realizzato in un futuro non troppo lontano.

Bibliografia

[1] Internet Live Stats: http://www.internetlivestats.com/internet-users

[2] The Statistics Portal: http://www.statista.com/statistics/218532/global-smartphone-penetration-since-2008

[3] Hotspot: http://www.gsmarena.com/glossary.php3

[4] IEEE 802.11: http://standards.ieee.org/getieee802/download/802.11-2012.pdf

[5] Linux syscall reference: http://syscalls.kernelgrok.com

[6] Virtual Square: http://wiki.v2.cs.unibo.it

[7] The GNU Bourne-Again SHell: http://tiswww.case.edu/php/chet/bash/bashtop.html

[8] IProute2: http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2

[9] Net-tools: http://www.linuxfoundation.org/collaborate/workgroups/networking/net-tools

[10] Supplicant: https://en.wikipedia.org/wiki/Supplicant_(computer)

[11] wpa_supplicant: http://w1.fi/wpa_supplicant

[12] wpa_supplicant code reference: http://mirror.umd.edu/roswiki/doc/diamondback/api/wpa_supplicant/html/wpa__ctrl_8c_source.html

[13] wvdial: https://wiki.archlinux.org/index.php/Wvdial

[14] Hayes: http://www.zoomtel.com/documentation/dial_up/100498D.pdf

[15] UMTS: https://en.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System

[16] Proxy: https://en.wikipedia.org/wiki/Proxy_server

[17] Berkeley sockets:
https://www.princeton.edu/ãchaney/tmve/wiki100k/docs/Berkeley_sockets.html

[18] sys_socketcall: http://lxr.free-electrons.com/source/net/socket.c

[19] ITU-T Y.1541: http://www.itu.int/rec/T-REC-Y.1541/en

[20] ISO/IEC standard 7498-1:1994: http://standards.iso.org/ittf/
PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip

[21] Gateway: http://tldp.org/LDP/nag/node27.html

[22] TUN TAP: https://www.kernel.org/doc/Documentation/networking/tuntap.txt

[23] ioctl: https://www.gnu.org/s/libc/manual/html_node/IOCTLs.html

[24] IEEE 802.3: http://www.ieee802.org/3

[25] GNU/Linux: https://www.gnu.org

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

[27] Ubuntu: http://www.ubuntu.com

[28] Bluetooth: http://www.bluetooth.com/Pages/Bluetooth-Home.aspx

[29] Global System for Mobile communications: http://www.gsma.com

Ringraziamenti

Vorrei anzitutto ringraziare mia moglie Elena a cui è dedicata questa tesi e tutto il lavoro che abbiamo fatto in questi anni di studio. Grazie per avermi dato il coraggio necessario ad iniziare, fiducia in ogni momento e speranza nel raggiungere questo traguardo.

Anche a voi Margherita e Lidia, mie bimbe, future studenti dedico questo lavoro, così un giorno, leggendo queste pagine capirete cosa intendevo tutte le volte che ho detto: "... devo andare a studiare".

Ringrazio i miei genitori e mio fratello, per avere sempre gioito con me dei risultati ottenuti.

Ringrazio i miei compagni di studio Federico, Luca e Vincenzo per avermi fatto sentire un po' più giovane di quello che sono.

Ringrazio infine il Professor Vittorio Ghini, insegnante mirabile, per avermi accompagnato al termine di questo cammino universitario.