UNIVERSITÀ DEGLI STUDI DI BOLOGNA
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di laurea triennale in Informatica

BILANCIAMENTO DI TRAFFICO VoIP SU RETI WIRELESS

Relatore:
Dott. Vittorio Ghini

Candidato:
Diego Rodriguez

ANNO ACCADEMICO 2008-2009 SESSIONE III

Dedico questa tesi alla mia famiglia
che mi ha sostenuto nei momenti
difficili e ha sempre creduto in tutto
ciò che ho fatto

Indice

Introduzione

Attualmente il panorama delle comunicazioni e' in continua evoluzione.
La presenza di reti wireless è in aumento sul territorio e la tecnologia VoIP vede sempre più applicazioni sia dal punto di vista lavorativo che personale. La possibilità di un utilizzo combinato di queste due tecnologie è resa complicata dalla presenza di due fattori concorrenti: L'elevata perdita di pacchetti insita nelle reti Wi-Fi e l'utilizzo di un protocollo inaffidabile come UDP da parte delle applicazioni VoIP.
Il nostro scenario ipotetico vede un dispositivo mobile (ad esempio un laptop dotato di una o più schede di rete) con la possibilità di spostamento nello spazio, che vuole instaurare una trasmissione VoIP con un altro dispositivo fisso sfruttando gli AccessPoint presenti nell'ambiente circostante.
Questa comunicazione avviene attraverso lo scambio di informazioni secondo questo modello: Il dispositivo mobile invia i dati di interesse all'AP che a sua volta li comunica al dispositivo fisso e viceversa. Il problema alla base di questo semplice meccanismo risiede nella mobilità e nell'utilizzo del protocollo UDP che non garantisce l'effettiva consegna dei pacchetti inviati. Riflettendo però sulle capacità del dispositivo mobile ci si accorge che è possibile sfruttare più schede di rete per l'invio dei dati e quindi utilizzare gli AP che hanno quantità di traffico differenti.
Il nostro studio ha come obiettivo lo sviluppo di un sistema per il bilanciamento di carico in grado di raccogliere le informazioni sullo stato delle connessioni presenti tra i due dispositivi e agire in base ad esse per distribuire i dati che una qualunque applicazione VoIP gli invierà localmente.
Di seguito una panoramica del documento:
Nel primo capitolo si descrive la realtà in cui si agisce, soffermandosi sulla tecnologia wireless, sui vari protocolli di comunicazione che costituiscono la gerarchia ISO OSI e sui tipi di crittografia più comuni al momento, WEP e WPA. Nel Secondo capitolo si espongono degli obiettivi, i possibili metodi per la risoluzione dei problemi riscontrati.
Nel Terzo capitolo si dettaglia la fase progettuale del sistema sviluppato e i problemi riscontrati. Nel Quarto capitolo vengono presentati i modelli astratti e l'implementazione del sistema realizzato secondo gli schemi descritti durante la Progettazione. Il Quinto capitolo riporta i risultati della valtuazione effettuata a seguito dei test. Nel Sesto capitolo si descrivono i possibili sviluppi futuri e si riportano le conclusioni.

Capitolo 1

Scenario

Oggi, in Italia e nel mondo, girando per le citta' è possibile avere accesso a reti wireless.
Una rete LAN wireless (WLAN - wireless local area network) consiste in una rete formata da due o più dispositivi connessi senza l'utilizzo di cavi. Essa utilizza tecnologie di trasmissione mediante onde radio come quella a espansione di spettro oppure la OFDM.
La tecnologia permette comunicazioni in un area limitata, consentendo mobilità all'interno di essa senza perdere l'accesso alla rete. In un contesto privato, le reti senza fili hanno avuto un intenso utilizzo grazie alla facilità di installazione e alla crescente diffusione di portatili e palmari. Alcuni tipi di esercizi come internet-cafè o centri commerciali hanno iniziato ad offrire questo tipo di servizio ai clienti. Nelle città sono nati progetti per piccole reti pubbliche, per esempio quelle di biblioteche e università, oppure grandi reti civiche che coprono le vie del centro.

1.1 Reti di nuova Generazione

Molte tecnologie wireless sono recentemente emerse, quail Wireless LAN (WLAN), WiMAX (worldwide interoperability for microwave access), e reti cellulari. Le WLAN, basate sulla famiglia IEEE 802.11 (IEEE 802.11a/b/g/n) si diffondono rapidamente per il loro basso costo, semplicità d'utilizzo, e alta capacità trasmissiva. Spesso queste WLANs sono gestite indipendentemente, come diverse sottoreti, da diverse organizzazioni e fornitori di servizi Internet (ISPs). Tali WLANs coprono ormai larghe aree, intere città, mediante numerosi access points (APs).Tradizionalmente, la mobilità può essere classificata in diversi tipi: host, network, session, e personal.

Host Mobility La capacità di un dispositivo di cambiare il proprio punto di accesso alla rete senza interruzione delle connessioni esistenti. E' il piu' importante dei tipi di mobilità, ed Iil suo supporto verrà descritto più avanti.

Network Mobility La capacità di un gruppo di dispositivi di cambiare tutti assieme il loro punto di accesso alla rete senza interruzione delle connessioni esistenti tra il gruppo. Protocolli come NEMO e OptiNets forniscono supporto alla mobilità di reti. NEMO è lo standard attuale mentre OptiNets è una sua estensione che fornisce instradamento ottimale per i nodi della rete mobile.

Session Mobility Permette ad un utente di di trasferire una comunicazione attiva da un dispositivo ad un altro. Session Initiation Protocol (SIP) possiede capacità di supportare tale mobilità poichè usa indirizzi tipo email per identificare utenti e sessioni.

Personal Mobility La capacità di un utente di accedere ad Internet usando un proprio identificatore personale per ricevere servizi in accordo con un proprio profile utente, mentre

si sposta usando un terminale o un altro servizio. Viene supportata da SIP.

1.1.1 Host Mobility and Seamless services

In reti tipo WLANs, un nodo mobile (MN) può accedere ad Internet attraverso un AP da una qualunque locazione. Però, quando il MN si sposta mentre usa diverse applicazioni, può subire numerosi passaggi (handovers) tra diverse sottoreti IP a causa della limitata copertura di ogni AP, sicchè la comunicazione può essere interrotta a causa di un cambio di indirizzo IP del MN. Per supportare host mobility nonostante il cambio di indirizzo IP, schemi di gestione della mobilità quail Mobile IP (MIP) e Mobile IPv6 (MIPv6) sono stati realizzati a livello network. Questi due protocolli lavorano permettendo ad un MN di essere raggiungibile mediante un indirizzo IP statico, chiamato home address, indipendentemente dalla locazione del MN. Un più recente protocollo proposto dall'IETF, Proxy Mobile IPv6 (PMIPv6), si affida solo all'infrastruttura di rete per supportare la host mobility. Al contrario di MIP/MIPv6, PMIPv6 permette ad un nodo mobile di cambiare punto di accesso senza interruzione delle connessioni senza richiedere al MN di essere coinvolto nella gestione della mobilità. Diversamente, il protocollo Host Identity Protocol (HIP), con le sue estensioni per la gestione di più NIC (multihoming) e con le estensioni per la mobilità, fornisce funzionalità di roaming ai MN, ma richiede che entrambi i due host alle due estremità delle comunicazioni utilizzino HIP.
Infine, Il supporto alla host mobility nonostante il cambio di indirizzo IP, viene fornito anche a livello trasporto mediante il protocollo mSCTP (mobile Stream Control Transmission Protocol).

Però, il degrado delle comunicazioni durante l'handover non può essere evitato qualunque schema di gestione della mobilità venga usato, tra quelli attualmente esistenti. Più nel dettaglio, il MN non può mandare o ricevere pacchetti durante un handover a causa delle operazioni che avvengono a livello 2 e 3. Inoltre, la qualità delle comunicazioni è molto sensibile ai cambiamenti delle condizioni del mezzo trasmissivo radio. Quindi, per fornire handover senza interruzioni (seamless handover), I seguenti tre requisiti dovrebbero essere soddisfatti:

• Inizio dell'handover in base a individuazione di cambiamenti di condizioni wireless.
• Eliminazione delle interruzioni di comunicazione prodotte dall'handover
• Selezione delle WLAN ottime.

Per fornire mobilità orientate ai servizi in ubiquitous networks, bisogna considerare attentamente la qualità delle comunicazioni durante l'handover, così come viene percepita dalle applicazioni. Molte applicazioni esistenti impiegano o Transmission Control Protocol (TCP, per applicazioni non real-time) o User Datagram Protocol (UDP, per applicazioni real-time) come protocollo di trasporto. In applicazioni non real-time, come file transfer (FTP), la misura più importante delle performance è il goodput, mentre la perdita di pacchetti, il RTT (round trip time), e il jitter sono importanti in applicazioni real-time quali Voice over IP (VoIP). A causa di tali differenze tra applicazioni non-real-time e real-time, bisognerebbe disporre di meccanismi di handover che considerino i requisiti delle applicazioni.

Per fornire seamless handover, è indispensabile un criterio per decidere se e quando

effettuare l'handover in base soprattutto ai cambiamenti delle condizioni del link wireless. Purtroppo, nelle reti con MIP, un MN individua i suoi movimenti in base al ricevimento di pacchetti di tipo router advertisement (RA), che sono spediti in broadcast poco frequentemente da un AP (solitamente uno al secondo). Questa bassa frequenza aumenta la latenza della decisone per l'handoff, causando degrado della qualità percepita dalle applicazioni. D'altro lato, nonostante alcuni schemi avanzati di MIP, come Fast Mobile IP (FMIP) e Hierarchical Mobile IP (HMIP), siano stati proposti per migliorare le prestazioni delle comunicazioni durante gli handover, questi schemi non considerano i criteri decisionali per gli handover.
In uno studio recente, nonostante mSCTP sia stato riconosciuto capace di supportare l'host mobility durante l'handover, gli aspetti della decisione riguardanti l'handover non sono stati discussi dettagliatamente. Quindi a tutt'oggi, mSCTP non è ancora in grado di fornire risposte alle necessità di decidere se e quando effettuare l'handover.

In ubiquitous WLANs, le comunicazioni spesso degradano a causa di:

1. Riduzione della potenza del segnale;
2. Interferenze radio con altre WLAN.

Un criterio decisionale per l'handover dovrebbe tenere conto di questi due aspetti.
La forza del segnale è spesso usata come misura delle prestazioni del link wireless. Però, un MN ha difficoltà a individuare degrado di prestazioni poichè la potenza del segnale fluttua fortemente. Inoltre, NIC di venditori diversi forniscono indicazioni diverse sulla potenza del segnale. Infine, un MN dovrebbe essere in grado di individuare la presenza di interferenze radio, ma ancora non è possibile.

Un primo approccio cross-layer per migliorare le prestazioni di handover orrizzontali (tra AP di una stessa tecnologia) si basa sulle informazioni del livello datalink. Più recentemente, Kashihara propone di usare il numero di ritrasmissioni dei frame come criterio per decidere se effettuare un handover, che garantisca continuità.

1.1.2 Heterogeneous Networks

Il Third Generation Partnership Project (3GPP) sviluppa standard per supportare seamless host mobility tra reti wireless di terza (3G) e quarta (4G) generazione, e reti WLAN basate sulla famiglia di protocolli IEEE 802.11, e altre reti ancora. Il consorzio 3GPP ha definito il IP Multimedia Subsystem (IMS) per supportare servizi basati su IP quali streaming, Voice over IP (VoIP), e videoconferencing.

IMS è l'elemento chiave dell'architettura 3G, che rende possible fornire accesso ubiquo da reti cellulari a tutti i servizi Internet. IMS è il componente che deve fornire supporto ai servizi multimediali (e.g., voice e/o video) basandosi su commutazione di pacchetto con qualità di servizio e fornendo autenticazione, autorizzazione e accounting (AAA). Per raggiungere questi obiettivi, IMS usa protocolli quali SIP, nel ruolo di protocollo di controllo di sessione, ed RTP/RTCP per il trasporto di media real-time, come video e audio. Il progetto di IMS design è basato su IPv6.

Aspetti di internetworking tra entità IMS e generici host, così come il roaming tra reti IMS e

non-IMS, sono ancora in fase di studio. Secondo 3GPP devono essere considerati 5 livelli di interconnessione:

1. pagamenti e gestione dell'utente,
2. controllo di accesso alle reti,
3. controllo di accesso ai servizi,
4. continua disponibilità dei servizi,
5. servizio senza interruzioni.

3GPP include I primi tre livelli nella Versione 6, mentre gli ultimi due verranno sviluppati nelle future versioni. In particolare, il livello 4 fornisce la capacità di mantenere una sessione attiva quando l'utente si sposta tra due diverse reti di accesso (da una WLAN ad una 3G UTRAN e vice versa), anche se è ammesso che l'utente percepisca l'avvenire dell'handoff (a causa di perdite e ritardi). Il quinto livello invece fornisce continuità seamless, cioè senza che l'utente si accorga di interruzioni.

IMS vuole fornire supporto a servizi realtime attraverso differenti tecnologie di reti di accesso. Però il supporto alla mobilità solleva aspetti relativi non solo ad handoff e interruzioni, ma anche alla qualità di servizio, poiché diverse tecnologie wireless forniscono diverse prestazioni in termini di bit rate, ritardi, perdite di pacchetti, etc.
Un meccanismo end-to-end adattivo nelle applicazioni può offrire un utile contributo alla gestione della QoS quando le reti sottostanti non dispongano di un sistema comune di gestione della QoS. Questo approccio può essere implementato introducendo conoscenza e gestione della mobilità ai diversi livelli dello stack di rete. Per permettere di supportare seamless handover tra reti di accesso eterogenee, lo standard IEEE802.21 definisce un framework (Media Independent Handover services, MIH) per scambiare informazioni tra strati diversi, indipendentemente dalla tecnologia utilizzata.

Esistono approcci alternativi per fornire servizi seamless operando a livello applicazione: tra questi, vanno per la maggiore quelli che si affidano al protocollo SIP per la fase di controllo (signaling). La soluzione MMUSE segue questo approccio proponendo un supporto al vertical handovers (handover tra reti eterogenee) basato su SIP che cerca di non interrompere la continuità dei servizi multimediali, usando dei proxy SIP collocati al bordo delle reti di accesso.

E' evidente dalla discussione presentata che ogni tipo di protocollo/architettura usato è adatto ad uno specifico tipo di mobilità da supportare. La soluzione proposta quindi cerca di individuare lo scenario di mobilità in cui opera il MN, e di dispiegare I protocolli più adatti in quella situazione.

1.2 Contesto della Ricerca

Lo studio riguarda la mobilità mediante connettività wireless in un ambiente vasto (per esempio una città). Per fare questo è necessario avere una copertura globale dell'area, anche attraverso varie reti differenti, e accesso a tutti gli hot spot presenti lungo il percorso come si può notare in questa immagine che rappresenta i punti di accesso del servizio fonera.

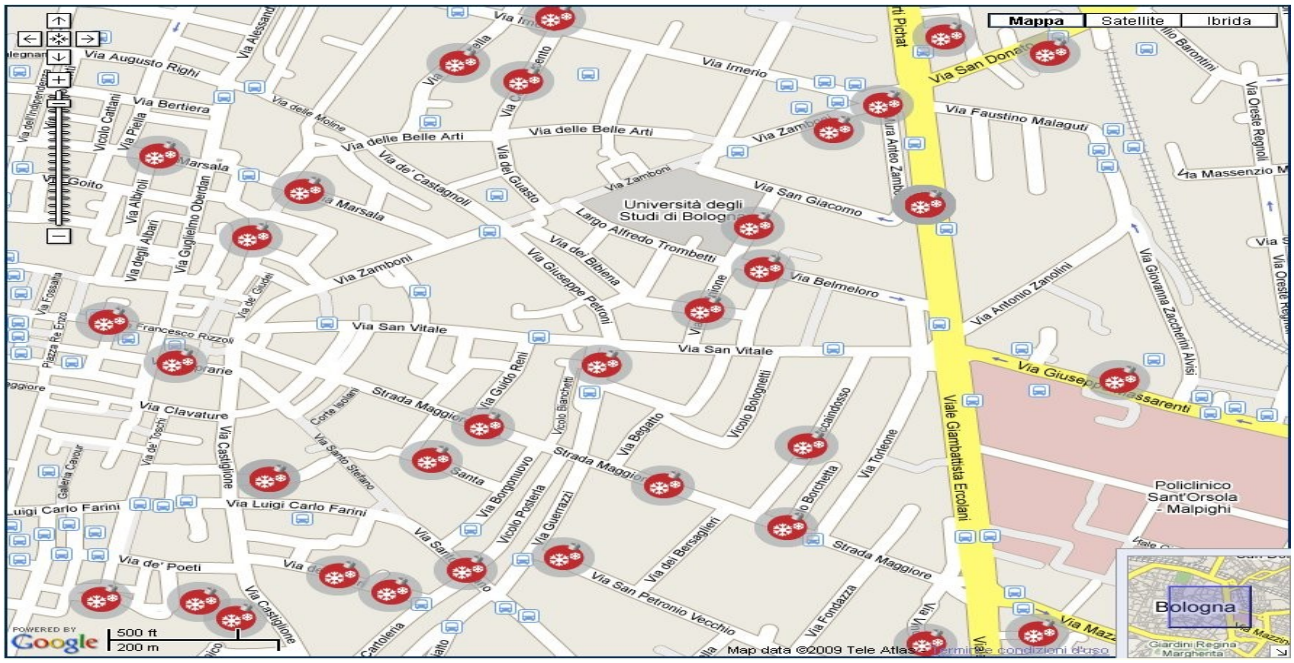

Figura 1.1: Hot Spot Fon a Bologna (maps.fon.com)

Precisamente la ricerca risponde all'esigenza di avere un'interazione effettiva con il flusso dei dati in un contesto UDP. Il protocollo UDP è privo di una connessione effettiva e non permette di avere informazioni riguardanti la sorte dei pacchetti inviati. L'utilizzo di questo protocollo viene imposto dal tipo di servizio che si vuole sfruttare; nel nostro caso la scelta del protocollo VoIP ha reso necessario affrontare il problema.
Data la facile reperibilità e la grande diffusione delle porte USB in quasi tutti i dispositivi mobili, la preferenza è andata alle schede WNIC USB. Il nostro scopo è ottenere il maggior numero di informazioni possibili sulla sorte dei pacchetti trasmessi per poi utilizzarle come base per stabilire quale è la connessione "migliore". Affinchè ciò avvenga svilupperemo una libreria per il bilanciamento di carico che stabilirà come inviare i dati e secondo quali parametri scegliere il canale di comunicazione (AP). Lavoreremo quindi un gradino sotto il livello applicazione:

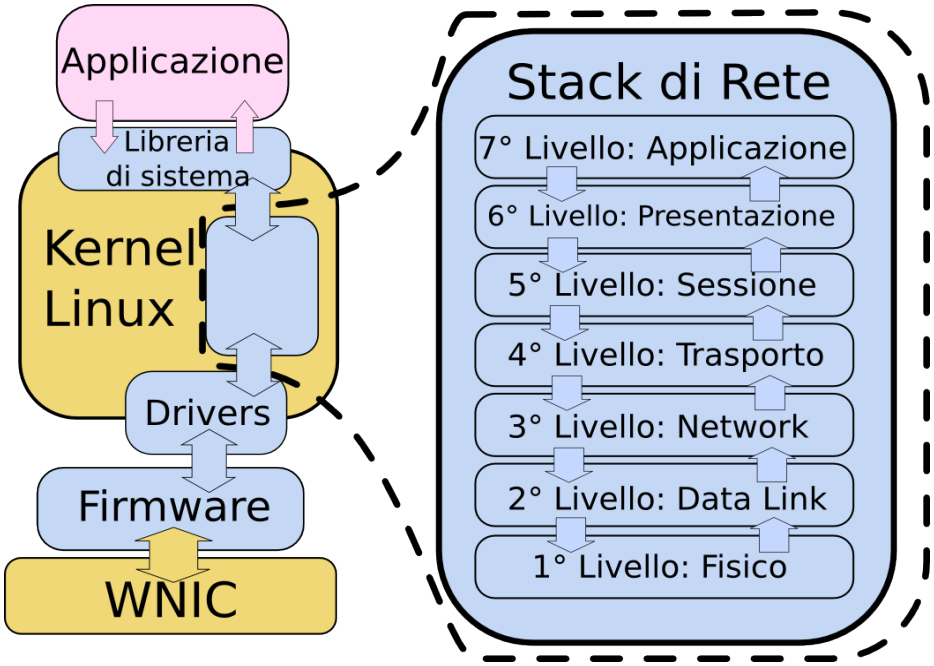

Figura 1.2: Meccanismo di interazione

Per capire quali sono i servizi presenti e le problematiche da affrontare e risolvere, di seguito si trovano le spiegazioni dei dispositivi e dei protocolli utilizzabili.

1.3 Tecnologia Wireless 802.11

Standard Lo IEEE802.11 o Wi-Fi definisce uno standard per le reti WLAN sviluppato dal gruppo 11 dell'IEEE 802. In esso viene definito il livello fisico e MAC del modello ISO-OSI e specificata sia l'interfaccia tra client e access point che quella tra più client. All'interno della famiglia, i protocolli dedicati alla trasmissione delle informazioni sono a, b e g. Gli altri standard, cioè c, d, e, f, h, ecc. riguardano estensioni dei servizi base e miglioramenti di servizi già disponibili. Il primo protocollo adottato su larga scala è stato il “b”; in seguito si sono diffusi “a” e soprattutto “g”. La sicurezza è stata inclusa in uno standard a parte, l'802.11i.
L'802.11b e 802.11g utilizzano lo spettro di frequenze (banda ISM) relativo ai 2,4 GHz. E' importante tenere in considerazione che trovandosi ad operare su bande con frequenze già utilizzate da altri apparecchi, i dispositivi b e g possono essere influenzati da telefoni cordless, ripetitori audio/video per distribuire programmi televisivi satellitari oppure altri apparecchi all'interno di un appartamento che utilizzano la banda di frequenze sopracitata.
L'802.11a utilizza la banda ISM dei 5,8 GHz.

Sicurezza Per quanto riguarda la sicurezza, le versioni originali dei protocolli 802.11 erano basati sulla crittografia WEP (Wired Equivalent Privacy). Nel 2001 un gruppo dell'università di Berkeley presentò uno studio nel quale descriveva le falle di sicurezza del protocollo 802.11. Il gruppo si concentrò sull'algoritmo di cifratura utilizzato dal WEP (**RC4**), che nell'implementazione scelta per lo standard 802.11 era molto debole e facilmente forzabile. Per ovviare al problema l'IEEE si mise al lavoro per progettare un'evoluzione dello standard. Contemporaneamente anche la Wi-Fi Alliance cercò una soluzione al problema. Per questo nel 2003 Wi-Fi Alliance annunciò la nascita del WPA e nel giugno del 2004 l'IEEE rilasciò le specifiche dell'IEEE 802.11i.
Il nuovo standard rendeva le reti wireless molto sicure e la Wi-Fi Alliance lo adottò subito sotto il nome di WPA2 che abbandona l'RC4 come algoritmo di codifica per passare al più sicuro Advanced Encryption Standard. Le varie opzioni di sicurezza disponibili comportano livelli di complessità crescente per la configurazione e l'adeguamento dei firmware e dei driver di AP e schede di rete wireless. In ambito privato sono di solito disponibili minori competenze, per cui capita di trovare reti wireless non crittate o che al massimo utilizzano la WEP. Queste reti sono insicure e possono essere forzate con semplicità, permettendo l'intercettazione del traffico wireless e l'accesso abusivo alla rete.

Vantaggi Molte reti riescono a fornire la cifratura dei dati e il roaming potendosi spostare dalla copertura di un access point ad un altro senza una caduta della connessione internet al di fuori del raggio di azione che delimita un hot-spot. Diversamente dal cellulare, l'esistenza di uno standard certificato garantisce l'interoperabilità fra apparecchio e rete anche all'estero. Senza i costi della cablatura (essendo tecnologia wireless) si ha una rapida e facile installazione (ed espansione successiva) della rete. La presenza di parecchi produttori ha creato una notevole concorrenza abbassando di molto i prezzi iniziali di questa tecnologia.

Svantaggi Il tempo di latenza delle schede wireless è superiore a quelle basate su cavo e rientra nell'ordine di 1-3 ms (da notare che le connessioni GPRS/UMTS hanno latenze nell'ordine di 200-400ms). Uno svantaggio delle connessioni Wi-Fi 802.11a/g è presente nella stabilità del servizio che per via dei disturbi sul segnale talvolta può essere discontinuo. Infatti questo può essere disturbato da forni a microonde, telefoni cordless nelle vicinanze che quando sono in funzione disturbano la frequenza operativa di 2,4-2,5 GHz. Da non trascurare il fatto che secondo alcuni recenti studi, scienziati affermati dichiarano che le radiazioni Wi-Fi sono nocive al nostro organismo.

1.4 Dispositivi

1.4.1 Access Point

Un Access Point è un dispositivo che permette all'utente mobile di collegarsi ad una rete wireless. Questo, collegato fisicamente ad una rete cablata (oppure via radio ad un altro access point), riceve ed invia un segnale radio al client, permettendo così la connessione. Un AP può avere varie modalità di funzionamento.

Root Mode E' la modalità classica di funzionamento dell'AP, ovvero quando esso è collegato alla rete cablata e funziona da punto di accesso per i nodi mobili wireless (Laptop, Cellulari, Palmari, ecc.)

Bridge Mode Consiste nalla creazione di un link wireless tra due (point-to-point) o più (point-to-multipoint) access point. Tipicamente quando un access point funziona in bridge mode non svolge la funzione di root, anche se alcuni costruttori permettono questa possibilità di funzionamento in contemporanea.

Repeater Mode Consente di aumentare il raggio di copertura di una rete wireless. Questo riduce drasticamente il troughput.

1.4.2 Schede di rete wireless

Un controller di rete wireless (WNIC - Wireless network interface controller) è una scheda di rete che permette ad un computer di utilizzare un' infrastruttura di rete non cablata. Questa lavora ai primi due livelli del modello ISO-OSI. Il dispositivo utilizza un'antenna e la comunicazione avviene attraverso le microonde.

Funzionamento Esistono due essenziali modalità di funzionamento, conosciute come *infrastruttura* e *ad-hoc*. Nel modello a *infrastruttura* è necessaria la presenza di un AP. Questo gestirà tutto il trasferimento dei dati e si comporterà da hub centrale. In questa modalità tutti i client devono essere connessi all'AP con lo stesso SSID (service set identifier). In caso di cifratura devono essere a conoscenza della chiave oppure di un mezzo per l'autenticazione.
In una rete *ad-hoc* non è necessario un AP poichè ogni dispositivo può interfacciarsi direttamente con gli altri nodi della rete. Anche in questo caso tutti i client devono essere impostati con lo stesso SSID sullo stesso canale. Le schede di rete utilizzano le specifiche IEEE 802.11. Vecchi dispositivi utilizzano standard antiquati, quelli recenti permettono sia

l'utilizzo di standard attuali che la retrocompatibilità. E' importante notare che il range del dispositivo varia in base all'ambiente circostante (ostacoli presenti nell'area) e alla qualità dell'antenna. Le fonti ambientali che influiscono sul funzionamento sono soprattutto quelle elettriche (frigoriferi, contatori, ecc.) oppure veri e propri ostacoli strutturali (muri, gabbie metalliche, ecc.). La velocità massima di trasferimento si può ottenere solo se realmente vicini alla fonte della trasmissione. I dispositivi negoziano dinamicamente la velocità di trasmissione per minimizzare la perdita di pacchetti (e quindi i tentativi di ritrasmissione). In un desktop computer una WNIC è solitamente connessa mediante il bus PCI, ma esistono alternative come USB, PC Card e soluzioni integrate come Mini PCI e PCI Express Mini.

Dispositivi USB

L'Universal Serial Bus è uno standard di comunicazione seriale che consente di collegare diverse periferiche ad un computer. Progettato per consentire a più periferiche di essere connesse usando una sola interfaccia standardizzata ed un solo tipo di connettore, permette l'hot plug. I device USB WNIC sono di facile reperibilità e semplici da utilizzare. Spesso le loro caratteristiche non li inseriscono al top della categoria, ma hanno vantaggi non trascurabili. Primo fra tutti, mediante una prolunga USB è possibile spostare il device per ottenere una ricezione migliore. In effetti in un ambiente chiuso la ricezione varia anche solo spostandosi di un metro.

Dispositivi PCMCIA

Si definisce PCMCIA la porta o il dispositivo nato per l'espansione delle funzionalità dei computer portatili. Lo standard delle PC Card nasce nel 1991 dalla fusione dei due standard JEIDA 4.1 e PCMCIA 2.0. Originariamente le PC Card furono pensate come slot di espansione per la memoria dei portatili, successivamente nello stesso formato vennero creati svariati dispositivi come schede di rete, modem e hard disk. E' da notare che la funzionalità originaria è andata in disuso. Esistono tre formati di PC Card e vari accessori per l'estensione di esse. Tutti i tipi hanno la stessa dimensione e numero di pin (68), la differenza sta nello spessore, rispettivamente 3.3mm, 5.0mm, e 10.5mm. Mediante la variazione di spessore è possibile ottenere device PCMCIA contenenti componenti di grosse dimensioni, e grazie alle estensioni le card possono sporgere dal connettore, permettendo per esempio ad un antenna wireless di essere esposta.

Dispositivi Mini PCI e PCI Express Mini

Mini PCI E' un bus standard che permette di connettere dispositivi periferici al computer. E' una evoluzione del PCI bus (Peripheral Component Interconnect) progettato per computer portatili e altri dispositivi di piccole dimensioni.

PCI Express Mini Card E' l'evoluzione dell' interfaccia PCI Express usando il bus Mini PCI. Sviluppata da PCI-SIG, l'interfaccia supporta connettività PCI Express, USB 2.0 e Firewire. Ogni device può scegliere l'interfaccia che ritiene più appropriata al suo scopo. I device WLAN con queste connessioni sono quelli integrati nei laptop.

1.5 Layer e protocolli

1.5.1 Stack di rete

Affinchè un' infrastruttura di rete possa funzionare in modo corretto ed efficace, è necessaria una strutturazione dettagliata del software che la deve gestire. Per ridurne la complessità, quasi tutte le reti sono gestite da una gerarchia di livelli (ognuno creato sulla base del sottostante) che astraggono e risolvono i problemi presenti. Questo meccanismo basato sui layer è un principio comunemente utilizzato nell' informatica e si basa sulla possibilità di trattare ogni strato come una black box, potendolo utilizzare semplicemente conoscendone l'interfaccia e il formato dell'informazione in uscita dopo l'elaborazione. Tra due postazioni la comunicazione avviene per livelli, cioè un dato livello nella macchina A comunicherà con il livello corrispondente nella macchina B. Le regole e le convenzioni utilizzate in questa comunicazione vengono definite come il protocollo del layer. La violazione di questi protocolli rende la comunicazione difficile se non impossibile. In mezzo ad ogni coppia di livelli adiacenti è presente un'interfaccia in cui vengono definite le operazioni e i servizi forniti dal livello sottostante a quello superiore. Oltre a minimizzare la quantità di informazioni necessarie per la comunicazione, rende possibile la modifica integrale dell'implementazione di un livello (mantenendo ovviamente intatta l'interfaccia) senza farlo sapere ai livelli adiacenti. La specifica di questa infrastruttura deve contenere tutte le informazioni che permettono la creazione di essa dal punto di vista sia hardware che software. Da notare che implementazione e specifiche di due macchine differenti possono essere diverse, l'unico fattore rilevante risede nell'utilizzo di un linguaggio comune che permetta la comunicazione. Per esempio tra due architetture con specifiche e implementazioni differenti che sanno interpretare e utilizzare il protocollo IP la comunicazione avviene senza problemi.

1.5.2 Livello Data Link (livello 2 del modello OSI)

Il livello Data Link si pone il problema della connessione diretta di due apparecchi. Si hanno quindi due sistemi connessi mediante un canale di comunicazione che si comporta come un cavo, per esempio un cavo coassiale, una linea telefonica, oppure una connessione punto-punto mediante microonde. Il canale dovrà comportarsi necessariamente come un cavo nel quale ogni sequenza di bit inviati viene ricevuta esattamente in quell'ordine. Il problema non è triviale data la possibilità di errori dovuti alla tecnologia. Un altro aspetto non trascurabile consiste nella presenza di ritardi di propagazione del segnale e nel fatto che la velocità di trasmissione è finita. Questi limiti influenzano notevolmente il funzionamento dei protocolli.

Mac 802.11

Il protocollo MAC (Media Access Control) è un sottolivello del Data Link Layer. Permette l'indirizzamento univoco e l'accesso ad un canale condiviso. Questo sottolivello funge da interfaccia tra il sottolivello LLC (Logical Link Control, rientra anchesso nel DL Layer) e il livello fisico (primo layer del modello OSI). Inoltre emula un canale di comunicazione full-duplex in una rete ad accesso multiplo rendendo possibile comunicazioni in unicast,

multicast e broadcast. In una comunicazione point-to-point full-duplex il livello MAC non è necessario, ma viene quasi sempre incluso in questi protocolli per questioni di compatibilità.

Indirizzamento Gli indirizzi gestiti da questo layer vengono chiamati indirizzi fisici o indirizzi MAC. Un identificativo MAC è un numero di serie univoco associato alla scheda di rete, rendendo possibile la consegna dei pacchetti ad un nodo all'interno della sottorete.

Meccanismo di accesso al canale Il meccanismo fornito da questo sottolivello è anche conosciuto come multiple access protocol e rende possibile a più nodi di collegarsi e condividere lo stesso mezzo fisico. Questo protocollo scova o previene le collisioni dei pacchetti in transito mediante un meccanismo di contesa del canale oppure attraverso l'allocazione di risorse riservate per la creazione di un canale logico. Il protocollo più diffuso nelle reti wired è chiamato CSMA/CD (Carrier sense multiple access con collision detection). Viene utilizzato all'interno dello stesso dominio di collisione, per esempio una sottorete gestita da un hub o un bus Ethernet. Parlando delle reti wireless troviamo il CSMA/CA (Carrier sense multiple access con collision avoidance). Riguardo il CSMA, quando un nodo mobile desidera trasmettere per prima cosa deve ascoltare il canale per un tempo prestabilito per verificare che non vi siano attività. Se esso è in stato idle allora si ha il permesso di trasmettere, in caso contrario si deve posticipare l'invio dei dati. Nello specifico con il CSMA/CA (LocalTalk) una volta che il canale è libero, la station che vuole trasmettere invia un segnale che specifica alle altre stazioni di non trasmettere e solo successivamente inizia l'invio dei dati. La prevenzione delle collisioni migliora le performance del CSMA. In caso di canale occuppato il tempo di attesa prima del test successivo assume un valore casuale, e questo riduce notevolmente la probabilità di collisioni. La necessità della prevenzione delle collisioni nasce dal fatto che in una rete wireless non si riesce ad ascoltare il canale durante la trasmissione dei propri dati, questo rende impossibile scovare le collisioni una volta avvenute.

1.5.3 Livello Network (livello 3 del modello OSI)

Nel trasferimento dei pacchetti dalla sorgente alla destinazione, questo livello si preoccupa di tutti i dettagli relativi al percorso. La consegna deve avvenire indipendentemente dalla struttura della rete e dal numero di sottoreti presenti. E' da notare che questo è l'ultimo livello in cui si ha a che fare con una trasmissione end-to-end. Per questo è necessario che il layer conosca la topologia di reti e sottoreti per potere scegliere il giusto percorso fra esse, soprattutto nel caso in cui la sorgente e la destinazione si trovino in reti differenti.

Protocollo IPv4

Il protocollo IPv4 è descritto nell'IETF RFC 791 pubblicato per la prima volta nel settembre 1981. E' un protocollo data-oriented e viene usato nelle reti di tipo packet switched. E' di tipo best-effort, infatti non garantisce la consegna dei pacchetti a destinazione. Non controlla la correttezza dei dati ed è possibile ricevere pacchetti duplicati o non nello stesso ordine di spedizione.

Spazio degli indirizzi Gli indirizzi IPv4 sono a 32 bit, per questo vi sono solo 4.294.967.296 indirizzi possibili. Da tenere presente che alcuni di essi sono di tipo riservato, per esempio circa 18 milioni sono destinati alle reti private e circa 16 milioni sono indirizzi

broadcast. Nonstante il numero apparentemente elevato, questo spazio di indirizzamento col passare del tempo si è dimostrato insufficiente. Il NAT (Network address translation) ha permesso di allontanare l'inevitabile. La soluzione effettiva al problema la si trova nello sviluppo del protocollo IPv6. Il formato umanamente leggibile di un indirizzo IPv4 consiste in una quadrupla di numeri separati da un punto (92.168.10.1).

NAT Un metodo semplice per permettere a tutti i computer di una sottorete di dialogare con tutta la rete pur non possedendo un IP fisso visibile dall'esterno è proposto dal NAT (Network address translation). Il metodo consiste nell'avere un solo gateway con IP fisso 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 perchè 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.

ARP Il protocollo IP necessita di conoscere la struttura logica della rete. Infatti se un nodo A invia un pacchetto (contenente l'indirizzo IP di sorgente e destinazione) al nodo B situato in una rete differente, il device che collega le due reti deve avere modo per conoscere entrambe le loro strutture. Questo deve avere una tabella dinamica dei MAC di entrambe le sottoreti e la giusta associazione MAC-IP. Questa associazione viene fatta mediante il protocollo ARP (Address Resolution Protocol) e lo scambio di messaggi propagati in broadcast.

Frammentazione e riassemblaggio Per rendere il protocollo più tollerante alle differenze delle varie sottoreti si è elaborato il concetto di frammentazione. Grazie a questo ogni device ha la possibilità di spezzare i dati in più pacchetti. Questo è 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 è di 65,535b, ma la dimensione tipica dell MTU di una rete Ethernet è di 1500b. Questo comporta che per inviare tutto il payload del pacchetto IP sono necessari 45 frame ethernet.

Frammentazione Quando un device riceve un pacchetto IP esamina la destinazione e sceglie su quale interfaccia instradarlo. Ogni interfaccia ha un MTU specifico, quindi è possibile effettuare la frammentazione che tiene conto della dimensione dell'header (minimo 20b e massimo 60b). Ogni segmento ha la sua intestazione IP con queste variazioni:

- la dimensione totale del pacchetto viene modificata con la dimensione del segmento
- il flag MF (more fragments) viene posto a 1 per tutti i frammenti eccetto l'ultimo
- il fragment offset viene sistemato in accordo alla posizione del segmento nel pacchetto IP completo utilizzando blocchi di 8byte

Riassemblaggio Alla ricezione di un pacchetto IP è necessario controllare il bit della frammentazione e l'offset. Se il bit è a 1 e l'offset è diverso da 0 allora è necessario immagazzinare i dati relativi al pacchetto. Alla ricezione del pacchetto con bit a 0 si potrà ricostruire l'originario utilizzando i campi offset. Una volta riassemblati tutti i segmenti nell'ordine corretto, il pacchetto IP viene passato al livello superiore dello stack di rete per

ulteriori elaborazioni.

1.5.4 Livello Trasporto (livello 4 del modello OSI)

Il livello di trasporto permette il dialogo strutturato tra mittente e destinatario. I protocolli definiti permettono la comunicazione end-to-end. Uno di questi protocolli è il TCP (Transmission Control Protocol). E' connection-oriented e permette al flusso di dati di essere recapitato senza errori. I frammenti dello stream vengono impacchettati e passati ai layer inferiori, sarà poi il ricevente che provvederà a riassemblarli e in caso di errore a chiederne il rinvio. Questo protocollo implementa anche un meccanismo per il flow control, cioè non permette ad una sorgente veloce di congestionare un ricevente lento. Il protocollo a cui siamo interessati però è l'UDP (User Datagram Protocol). Esso è di tipo inaffidabile (non è detto che i pacchetti arrivino a destinazione) e senza connessione. E' comunque abbondantemente utilizzato nelle applicazioni che necessitano di interattività e velocità. Per esempio audio e video vengono quasi sempre inviati attraverso UDP, questo perchè piuttosto che la ricezione di tutti i pacchetti, è molto più importante la regolarità e la continuità del flusso di dati.

Protocollo UDP

L'User Datagram Protocol è uno dei protocolli fondamentali di Internet. Le applicazioni possono creare e inviarsi dei messaggi brevi (conosciuti anche come datagram) attraverso degli appositi socket. UDP non garantisce l'affidabilità della connessione o che i pacchetti vengano ricevuti nell'ordine di invio. Infatti possono essere duplicati o non arrivare affatto senza nessuna segnalazione. 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 perchè preferiscono avere pacchetti mancanti invece che ritardi elevati. Inoltre viene utilizzato nel caso in cui sia necessaria una sincronia tra server e clients (es. giochi online) oppure si debbano gestire messaggi in broadcast o multicast. Un esempio di applicazioni che utilizzano UDP sono 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 seguente struttura:

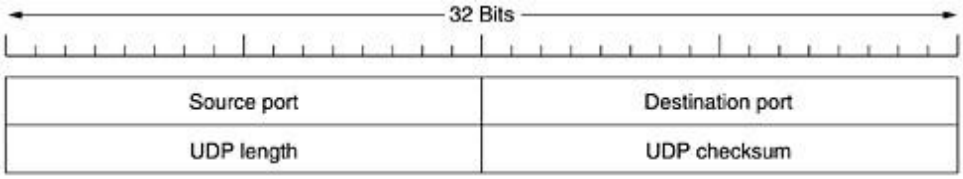

Figura 1.3: Header del pacchetto UDP

Caratteristiche del protocollo UDP è 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 è completamente indipendente dall'altro.

Inaffidabilità Quando un messaggio viene inviato, non è possibile sapere nulla sulla sua sorte.

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

Leggero Data la mancanza di controlli il suo header risulterà leggero in termine di bytes.

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.

1.5.5 Livello Applicazione (livello 7 del modello OSI)

Il livello applicazione contiene svariati protocolli normalmente utilizzati dagli utenti. Un esempio chiarificatore lo si trova nel protocollo HTTP (HyperText Transfer Protocol), che sta alla base del World Wide Web. La quantità di protocolli esistenti non permette di discuterne o farne una panoramica, perchè qualsiasi applicazione utilizzi la rete si basa su un protocollo di questo livello.

Protocollo VoIP

Con VoIP (Voice Over Internet Protocol) si intende il protocollo capace di gestire il passaggio da segnale audio (analogico) al flusso di pacchetti (digitale) sulla rete e viceversa. In questo modo è possibile effettuare delle telefonate utilizzando la rete Internet anzichè quella telefonica pubblica. Per questo è possibile telefonare a costi molto bassi in qualsiasi parte del mondo senza differenze geografiche. La qualità della telefonata è paragonabile a quella di una normale linea telefonica; basta solo disporre di una connessione internet a banda larga (ADSL o HDSL) con almeno 32 kbps di banda garantita.

Funzionamento Per poter funzionare sono richiesti due tipi di protocollo di comunicazione in parallelo:

- uno per trasportare la voce sottoforma di dati, normalmente viene adottato il protocollo RTP (Real-Time Transport Protocol)
- l'altro per riconvertire i dati in voce, per sincronizzarla e riordinarla quindi cronologicamente

Esistono diversi protocolli standard:

- SIP (Session Initiation Protocol) della IETF;
- H.323 della ITU;
- Skinny Client Control Protocol proprietario della Cisco;
- Megaco (conosciuto anche come H.248);
- MGCP;
- MiNET proprietario della Mitel;
- IAX, usato dai server Asterisk open source PBX e dai relativi software client.

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

Abbattimento costi Consente un abbattimento dei costi notevole, sia per chiamate

nazionali che internazionali o intercontinentali dato che il costo della chiamata praticamente non varia indipendentemente dal fatto che si parli nel proprio paese o dalla parte opposta del mondo.

Agevolazioni in campo lavorativo Può essere utilizzato sia all'interno di una stessa azienda, sia per telefonate più complesse, come teleconferenze in multi-point. Inoltre si può 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.

Servizi telematici Se si decide di utilizzare l' apparecchio telefonico apposito, a differenza dei normali apparecchi questo potrà essere sfruttato sia per normali conversazioni sia per ottenere l'accesso ad alcuni servizi internet, quali l'elenco telefonico, ottenendo una risposta in forma vocale o testuale.

Possibilità di Number Portability Passando alla tecnologia VoIP non è detto che si debba perdere il proprio numero telefonico, perchè, grazie alla Number Portability il numero resta tale e quale. La procedura che permette di trasferire numeri telefonici da un operatore all'altro è similare a quella dei vari gestori di telefonia mobile. E' possibile, inoltre, telefonare da un pc portatile via internet da qualsiasi luogo dotato di collegamento ADSL.

Maggiore privacy Le chiamate tramite VoIP sono più sicure delle normali chiamate, anche se non vengono criptate o cifrate. Per riuscire ad intercettare una comunicazione VoIP si deve aver accesso agli apparati che costituiscono la rete VoIP e/o la Rete Internet.

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.

Svantaggi Come tutte le tecnologie in via di sviluppo, e quindi da perfezionare, anche questa del VoIP presenta diversi svantaggi.

Qualità vocale Utilizzando la stessa linea per trasmettere sia voce che dati, la qualità della voce durante una telefonata potrebbe calare e, in caso di black-out elettrico non sarebbe possibile effettuare telefonate. Non sempre la qualità delle telefonate è eccellente a causa di due diversi problemi: il ritardo e la perdita di pacchetti dati. Il ritardo genera eco e sovrapposizione delle conversazioni. Per sopperire a questi problemi fastidiosi sarà necessario progettare degli apparecchi appositi in grado di cancellare l'eco. Per sopperire alla presenza di eco e disturbi quali fruscii, sono stati già creati software appositi. Per quanto riguarda la gestione dei silenzi, delle pause o di respiri prolungati, quindi mancanza di parole, sono stati creati altri software per eliminarli tenendo conto che, generalmente, una conversazione e' costituita da circa il 50% di tempi morti ed in questo modo si mantiene libera la linea da sovraccarichi inutili. Per quanto riguarda la perdita dei pacchetti, per ottenere una conversazione lineare questi devono rispettare l'ordine cronologico in cui vengono inviati. Una perdita di pacchetti voce pari o superiore al 10% non consente di svolgere una chiara comunicazione.

Banda insufficiente Il gestore della rete dovrebbe garantire una larghezza di banda sufficientemente ampia, per ridurre il tempo di latenza e le perdite di dati. Tuttavia ciò,

mentre è relativamente facile nelle reti private, è molto più difficile quando si usa internet come mezzo trasmissivo.

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

Il nostro contesto nel VoIP Lo scenario applicativo da noi preso in considerazione, vede utilizzato il VoIP in un contesto wireless. Per questo motivo diviene indispensabile risolvere il problema dell'interattività del controllo del flusso di dati con questo mezzo trasmissivo.

Protocollo SIP

Il protocollo SIP (Session Initiation Protocol) è un protocollo basato su IP, definito dalla RFC 3261, e impiegato principalmente per applicazioni di telefonia su IP o VoIP.
SIP gestisce in modo generale una sessione di comunicazione tra due o più entità, ovvero fornisce meccanismi per instaurare, modificare e terminare (rilasciare) una sessione. Attraverso il protocollo SIP possono essere trasferiti dati di diverso tipo (audio, video, messaggistica testuale, ecc). Inoltre, il SIP favorisce un'architettura modulare e scalabile, ovvero capace di crescere con il numero degli utilizzatori del servizio. Queste potenzialità hanno fatto sì che il SIP sia, oggi, il protocollo VoIP più diffuso nel mercato residenziale e business, sorpassando di molto altri protocolli quali H.323 ed MGCP.
Intorno al SIP sono sorti diversi tipi di utilizzatori, pensati per facilitare la fruizione della telefonia VoIP da parte di tutte le categorie di persone. Alcuni esempi sono gli ATA (Analog Telephone Adapters), capaci di convertire la segnalazione elettrica di un normale telefono analogico in un flusso di dati IP, e, dall'unione tra telefono tradizionale e ATA, gli IP Phones, telefoni dalle funzionalità elevate ai quali non arriva il doppino telefonico ma i cavi di rete. Evoluzione ulteriore sono i soft-phones, applicazioni software per personal computer che emulano le funzioni di un telefono VoIP.

Applicazione Esso trova applicazione nella telefonia su IP e nei servizi telefonici supplementari, nella video-comunicazione, nei giochi interattivi, nella messaggistica istantanea. Il protocollo è stato sviluppato, ufficialmente, a partire dal 1999 (RFC 2543 e 3261) per iniziativa di IETF e fa parte della Internet Multimedia Conferencing Suite.
Ad oggi l'utilizzo prevalente è nella telefonia su IP, per cui SIP è spesso sinonimo di un sistema dedicato a questa applicazione.

Descrizione Storicamente SIP utilizza il protocollo di trasporto UDP con porta di default 5060. Recenti revisioni di questo standard permettono anche l'utilizzo attraverso TCP e TLS.
Il protocollo SIP ha fondamentalmente le seguenti funzioni:

- Localizzazione degli utenti
    - acquisire le preferenze degli utenti
- Invitare gli utenti a partecipare ad una sessione:
    - negoziare le capability
    - trasportare una descrizione della sessione
- Instaurare le connessioni di sessione
- Gestire eventuali modifiche dei parametri di sessione
- Rilasciare le parti

- Cancellare la sessione in qualunque momento si desideri.

Il modello usato per la sintassi del protocollo SIP è text-based, derivato dall' HTTP. Per instaurare una sessione, avviene un three-way handshaking (concettualmente simile a quello che avviene con il protocollo TCP). Alcune delle caratteristiche importanti del protocollo SIP:

- è impiegabile sia in contesti client-server sia in contesti peer-to-peer.
- è facilmente estendibile e programmabile
- eventuali server possono essere sia stateless sia stateful.
- è indipendente dal protocollo di trasporto.

Un messaggio SIP è una *richiesta* o una *risposta*; una sequenza di una richiesta e una o più risposte è detta *transazione*: una transazione è identificabile da un *transaction-ID*, un identificativo che ne specifica la sorgente, la destinazione e il numero di sequenza. Il protocollo SIP supporta la mobilità ed è dialog-oriented: un *dialogo* è una relazione persistente tra entità paritetiche che si scambiano richieste e risposte in un contesto comune.

Overview sul protocollo SIP supporta cinque modalità per l'avvio e la terminazione di una connessione:

- **User location**: determinazione degli end system usati nella comunicazione;
- **User availability**: identificazione della disponibilità delle parti ad impegnarsi in una comunicazione;
- **User capabilities**: identificazione di media e parametri utilizzati;
- **Session setup**: avviso, instaurazione dei parametri di una sessione su chiamate;
- **Session management**: trasferimento e terminazione di una sessione, modifica dei parametri della sessione, e invocazione dei servizi.

SIP non è un sistema per le comunicazioni integrato verticalmente, ma piuttosto può essere usato come componente in altri protocolli IETF per costruire un'architettura multimediale completa (di solito includono il protocollo RTP per la comunicazione trasparente in real-time e fornitura di feedback in QoS, oppure RTSP per il controllo della consegna di contenuti in streaming). SIP non fornisce servizi, ma primitive per implementarli. Per esempio esso può individuare un utente e consegnargli un oggetto opaco alla sua locazione corrente.

Architettura di una rete SIP

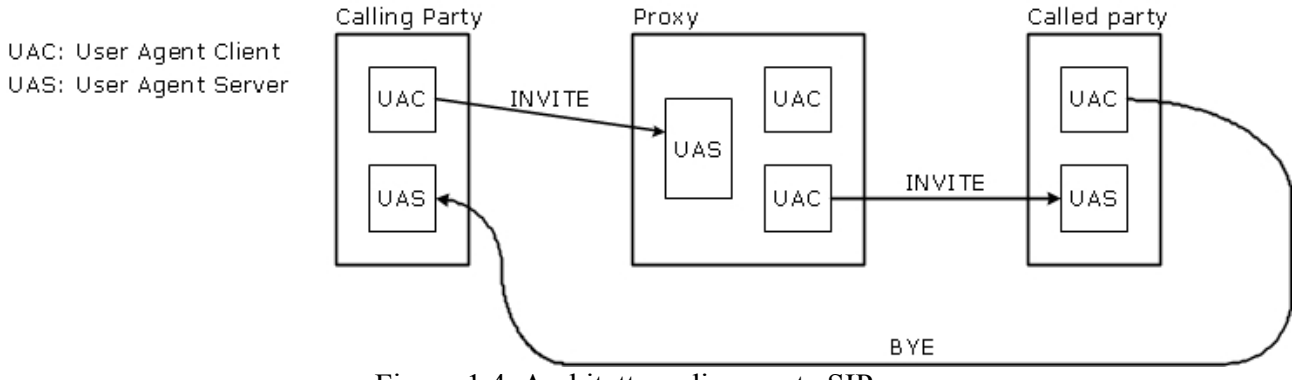

Figura 1.4: Architettura di una rete SIP

Le entità essenziali di una rete SIP sono:

- *SIP UserAgent* è un endpoint e può fungere da client o da server; i due ruoli sono dinamici, nel senso che nel corso di una sessione un client può fungere da server e viceversa. Quando funge da client, dà inizio alla transazione originando richieste. Quando funge da server, *ascolta* le richieste e le soddisfa, se possibile.
  Uno User Agent è in sostanza una macchina a stati, che evolve in dipendenza di messaggi SIP, e registra informazioni rilevanti del dialogo. Il dialogo ha inizio quando si risponde positivamente al messaggio di *Invite* e termina con un messaggio di *Bye*.
- *Registrar Server* è un server dedicato o collocato in un proxy. Quando un utente è iscritto ad un dominio, invia un messaggio di registrazione del suo attuale punto di ancoraggio alla rete ad un Registrar Server.
- *Proxy Server* è un server intermedio; può rispondere direttamente alle richieste oppure inoltrarle ad un client, ad un server o ad un ulteriore proxy. Un proxy server analizza i parametri di instradamento dei messaggi e "nasconde" la reale posizione del destinatario del messaggio - essendo quest'ultimo indirizzabile con un nome convenzionale del dominio di appartenenza.
  I proxy possono essere stateless o stateful. Quando uno User Agent invia sistematicamente le proprie richieste ad un proxy "vicino" (di default) allora il proxy viene detto *Outbound-Proxy*. Viceversa, un *Inbound-Proxy* è un proxy che instrada le chiamate entranti in un dominio. Infine un *Forking-Proxy* può instradare una stessa richiesta in parallelo o in sequenza a più destinazioni.
- *Redirect Server*: reinstrada le richieste SIP consentendo al chiamante di contattare un set alternativo di URIs.
- *Location Server*: è un database contentente informazioni sull'utente, come il profilo, l'indirizzo IP, l'URL.

Software Esistono diversi software in grado di sfruttare il protocollo SIP. Alcuni di questi sono software libero:

- Gizmo Project
- Whoupe
- Asterisk
- Ekiga
- Wengo
- Fring
- Google Talk

Protocollo NTP

Il **Network Time Protocol**, in sigla **NTP**, è un protocollo per sincronizzare gli orologi dei computer all'interno di una rete a commutazione di pacchetto, quindi con tempi di latenza variabili ed inaffidabili. L'NTP è un protocollo client-server appartenente al livello applicativo.

Funzionamento

L'NTP è uno dei più vecchi protocolli tuttora in uso, ed è giunto alla sua quarta versione. Fu

sviluppato presso l'università del Delaware da Dave Mills, che ne segue tuttora lo sviluppo. Il funzionamento si basa sul rilevamento dei tempi di latenza nel transito dei pacchetti sulla rete. Utilizza il tempo coordinato universale ed è quindi indipendente dai fusi orari. Attualmente è in grado di sincronizzare gli orologi dei computer su internet entro un margine di 10 millisecondi e con una accuratezza di almeno 200 microsecondi all'interno di una LAN in condizioni ottimali.
I diversi server NTP sono organizzati in una struttura gerarchica di "strati", dove lo strato 1 è sincronizzato con una fonte temporale esterna quale un orologio atomico, GPS o un orologio radicontrollato, lo strato 2 riceve i dati temporali da server di strato 1, e così via. Un server si sincronizza confrontando il suo orologio con quello di diversi altri server di strato superiore o dello stesso strato. Questo permette di aumentare la precisione, e di eliminare eventuali server scorretti.

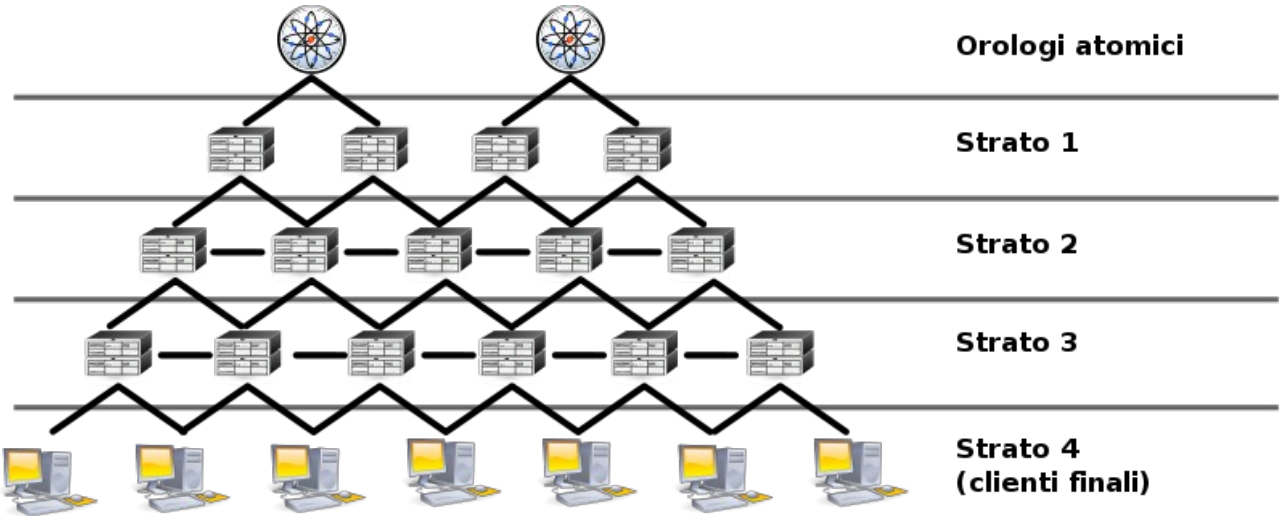

Figura 1.5: Struttura del protocollo NTP

Un server NTP è in grado di stimare e compensare gli errori sistematici dell'orologio hardware di sistema, che normalmente è di scarsa qualità. In una macchina che supporti l'NTP, il server è costituito da un processo operante a livello utente, così come da altre funzioni accessorie. Per ottenere le migliori prestazioni è utile che la parte relativa all'allineamento dell'orologio sia implementata nel kernel del sistema operativo, piuttosto che affidata ad un processo utente (tutte le recenti versioni del Kernel GNU Linux hanno questa caratteristica). Il dato a 64 bit scambiato dal protocollo è suddiviso in 32 bit che definiscono i secondi e altri 32bit per la parte decimale, potendo così rappresentare un intervallo di 2^{32} ed una risoluzione temporale teorica di 2^{-32} secondi. Ogni 2^{32} secondi l'intervallo temporale descritto dal protocollo si ripete, per cui è necessario specificare a quale periodo ci si riferisce. Questo non è un problema poiché 2^{32} secondi corrispondono a circa 136 anni.
I dettagli sul funzionamento dell'NTP sono definiti dalle RFC: RFC 778, RFC 891, RFC 956 e RFC 1305. L'NTP non deve essere confuso con i protocolli *daytime* (RFC 867) e *time* (RFC 868). Una versione semplificata di questo protocollo che non richiede la memorizzazione dei dati tra due successive comunicazioni è il **Simple Network Time Protocol**, **SNTP** (RFC 1361, RFC 1769 e RFC 2030), impiegato in sistemi embedded dove non è richiesta una grande precisione. Inoltre SNTP può essere usato su sistemi che agiscano solo da client NTP o solo da server (apparecchiature dedicate che ricevono l'ora da un orologio esterno e la ridistribuiscono via NTP).

Capitolo 2

Obiettivi

La connessione wireless, per sua stessa natura, non permette di effettuare alcun controllo sull'esito della trasmissione. Alcuni protocolli come il TCP implementano questa caratteristica a livello più alto, ma nel nostro caso l'utilizzo del protocollo UDP su cui si appoggerà quello VoIP non permette di avere delle garanzie riguardo la sorte dei pacchetti inviati. Il nostro studio si pone come obiettivo lo sviluppo di un sistema di bilanciamento di carico in grado di raccogliere le informazioni più rilevanti sul traffico di rete per poi sfruttarle a suo vantaggio durante l'invio dei dati ricevuti da un qualsiasi client VoIP.
Il sistema che verrà sviluppato dovrà interessare i due lati della trasmissione e verrà realizzato sotto forma di proxy, ossia tutti i bytes che verrano scambiati tra i due endpoints dovranno passare necessariamente per il bilanciatore di carico.
La situazione è quindi la seguente rappresentata in figura:

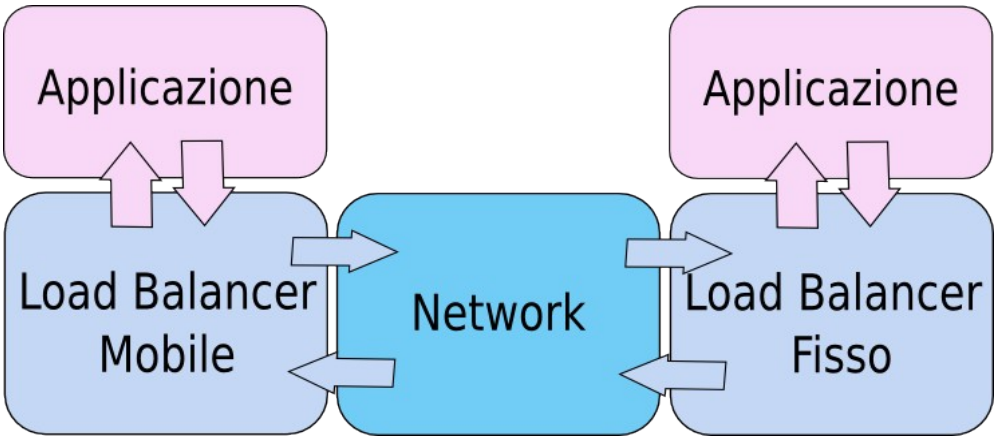

Figura 2.1: Organizzazione e relazioni tra applicazione ed i proxy

Come è possibile notare le applicazioni VoIP si appoggeranno al sistema di bilanciamento per l'invio e la ricezione dei dati. Avendo in mente questo schema è possibile produrre il software in grado di svolgere il ruolo di intermediario durante la trasmissione. Inoltre sono stati ritenuti molto importanti per lo sviluppo i seguenti fattori:

- Portabilità
- Indipendenza dall'Applicazione VoIP

Il software sviluppato dovrà essere in grado di funzionare su ogni sistema operativo o comunque fornire un algoritmo generale implementabile su una buona quantità di architetture differenti. Inoltre il protocollo che verrà progettato per permettere la raccolta delle informazioni e l'analisi della rete dovrà funzionare indipendemente dal tipo di client VoIP che si vorrà utilizzare, questo perchè sviluppare il sistema per un singolo applicativo limiterebbe di molto il supporto.

2.1 Le informazioni

L'invio dei dati tra i due bilanciatori comporterà la creazione di un protocollo, ossia un insieme di regole che serviranno ai due proxy per comprendersi chiaramente.
Inoltre questo protocollo servirà per analizzare costantemente la rete e raccogliere le seguenti informazioni:

- Consegna effettiva della frame inviata
- Quantità di pacchetti persa
- Latenza di ogni access point

Ogni volta che il bilanciatore mobile invia un pacchetto ad un determinato AP, è molto utile sapere se esso è stato trasmesso con successo. La motivazione che ci spinge all'utilizzo di questa informazione è la possibilità di scegliere un AP alternativo per garantire la consegna dei dati entro un limite massimo di 150 ms. Per ottenere questo tipo di controllo di flusso si è basato lo sviluppo sulle informazioni che un Kernel GNU Linux modificato potesse mettere a disposizione sulla consegna effettiva al prime hop dell'instradamento scelto.
Un altro problema che sorge dopo l'effettivo invio è dovuto al protocollo UDP che non assicura che il datagram inviato non vada poi perso nella rete. Per far fronte a tale problematica si è deciso di testare costantemente la rete end-to-end, cioè verificare quanti pacchetti sono stati ricevuti dal lato fisso su quelli effettivamente mandati dal lato mobile e viceversa (quindi solo quelli che non hanno avuto necessità di ritrasmissione).
Questo controllo eseguito ad intervalli regolari permetterà inoltre di verificare la latenza di un dato AP in quell'istante temporale, permettendo al sistema di stabilire quale è il canale di comunicazione più interattivo. Ottenute queste informazioni per ogni AP presente nella zona in cui il device mobile sta operando, si passa all'utilizzo vero e proprio di tali parametri per stabilire quale sia il canale migliore per la comunicazione secondo l'algoritmo che verrà progettato. L'architettura mobile è quindi strutturata in questo modo:

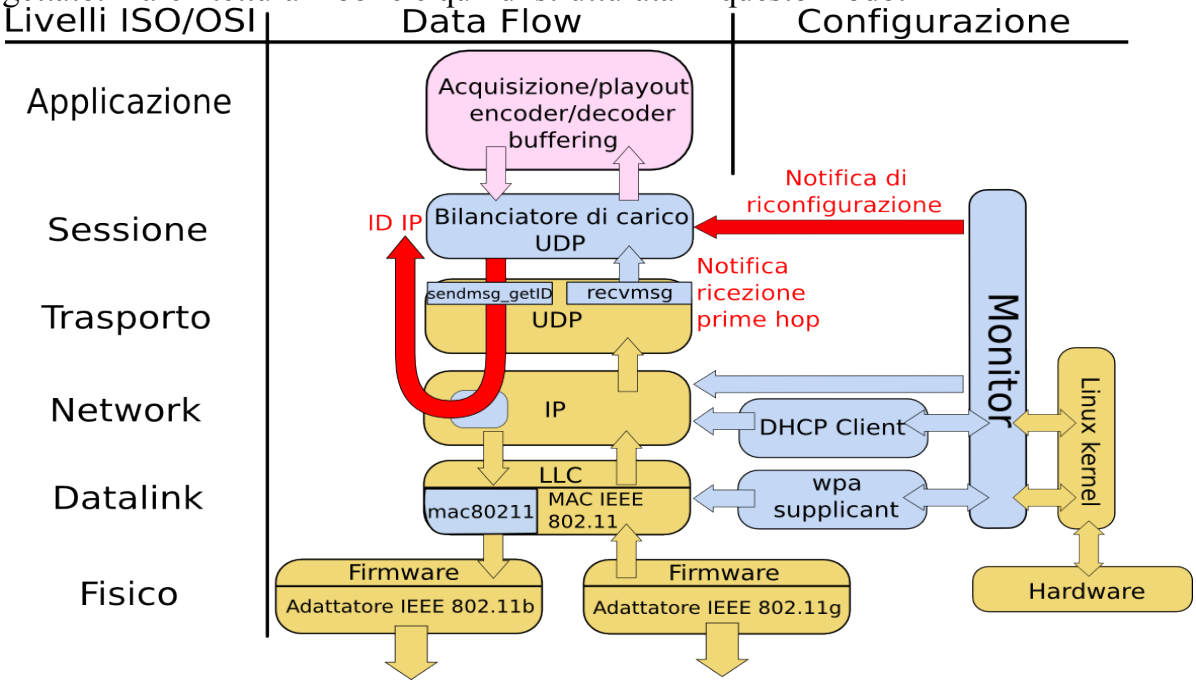

Figura 2.2: Architettura del sistema mobile

Dalla rappresentazione si evince che tramite la syscall modificata sendmsg_getID sarà possibile estrapolare e portare in userspace l'id di un pacchetto IP per poi effettuare il controllo di ricezione eseguita con successo da parte del prime hop tramite l'altra procedura recvmsg. Inoltre appare evidente che sarà necessario un modulo in grado di monitorare (Monitor) costantemente la presenza di nuovi AP con cui stabilire la connessione, in questo modo infatti il bilanciatore di carico UDP sarà informato dei nuovi canali di comunicazione e potrà quindi utilizzarli quando necessario.
La figura seguente invece rappresenta come sarà fatto il bilanciatore di carico UDP al suo interno:

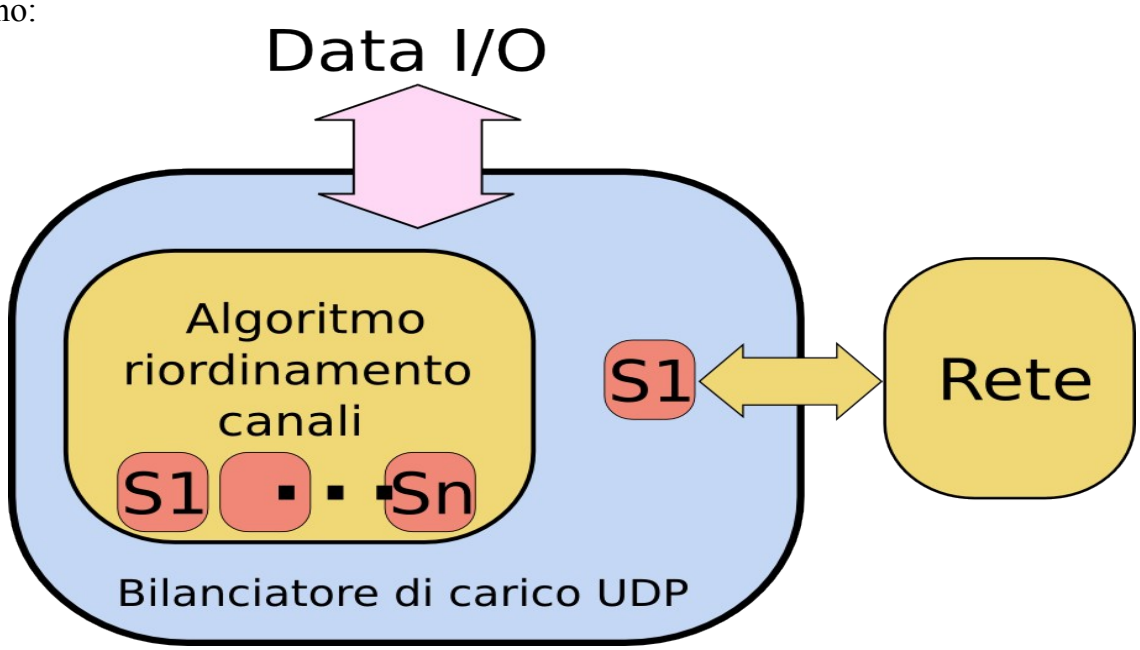

Figura 2.3: Composizione interna LoadBalancer UDP

Letti i dati che l'applicazione VoIP vuole spedire si procederà con l'invio del datagram all endpoint opposto utilizzando il primo dei canali di spedizione presenti (in figura chiamati S1...Sn). Durante il corso della trasmissione con la raccolta delle informazioni sarà possibile riordinare la lista dei canali secondo l'algoritmo progettato e quindi grazie alle statistiche ottenute potrebbe capitare che il canale ritenuto migliore sia messo in fondo alla coda mentre uno nuovo e più performante venga posto in testa. E' importante sottolineare che i nuovi socket aggiunti verranno testati almeno una volta per stabilire la loro affidabilità in termini di perdita pacchetti e latenza di trasmissione.

Capitolo 3

Progettazione

La tecnica del Load Balancing è un problema affrontanto molto spesso nell'ambito delle reti, in particolare quando c'è il bisogno di ottenere il miglior utilizzo delle risorse tramite la divisione del lavoro su più unità(CPU, hard disk, ecc..).
Utilizzando quindi più componenti invece che il singolo si aumenta l'affidabilità attraverso la ridondanza. Il servizio di bilanciamento di solito è ottenuto tramite un programma dedicato o un componente hardware (multilayer switch).
Una delle applicazioni più comuni del Load Balancing è provvedere un singolo servizio internet a più servers (server farm) cioè viene utilizzato un software che si mette in ascolto su una data porta e ricevuta una richiesta da parte di un Client la inoltra a uno dei servers.
In questo modo è possibile che il load balancer risponda direttamente al Client senza che esso sia a conoscenza di questa suddivisione del carico interno, inoltre i servers utilizzati non vengono contattati direttamente dal Client (per motivi di sicurezza).
L' insieme dei servers a cui il Load Balancer fa riferimento è detto cluster e i servers sono chiamati nodi. Esistono diversi algoritmi di scheduling che vengono utilizzati nella scelta del nodo a cui inoltrare le richieste del Client. Alcuni molto semplici effettuano una scelta casuale o round robin, altri più complessi prendono in considerazione fattori come il carico di ciascun nodo, tempi di risposta, locazione geografica o il traffico recentemente assegnato.
Il punto comune nei differenti load balancer è la gestione del failover, ossia cosa bisogna fare nel caso di guasto di un dato nodo del cluster.
Solitamente viene adottata una strategia chiamata HeartBeat che si occupa proprio di verificare il corretto funzionamento dei nodi. Se tale meccanismo dovesse fallire allora viene avviata una procedura chiamata Convergenza che provvederà ad eliminare il guasto.
Avendo presente il generale funzionamento della tecnologia che viene impiegata nei bilanciatori di carico è stato possibile creare un algoritmo che riprende le idee fondamentali e le riorganizza per supportare il nostro caso.
La figura seguente illustra l'ambiente in cui il sistema verrà sviluppato:

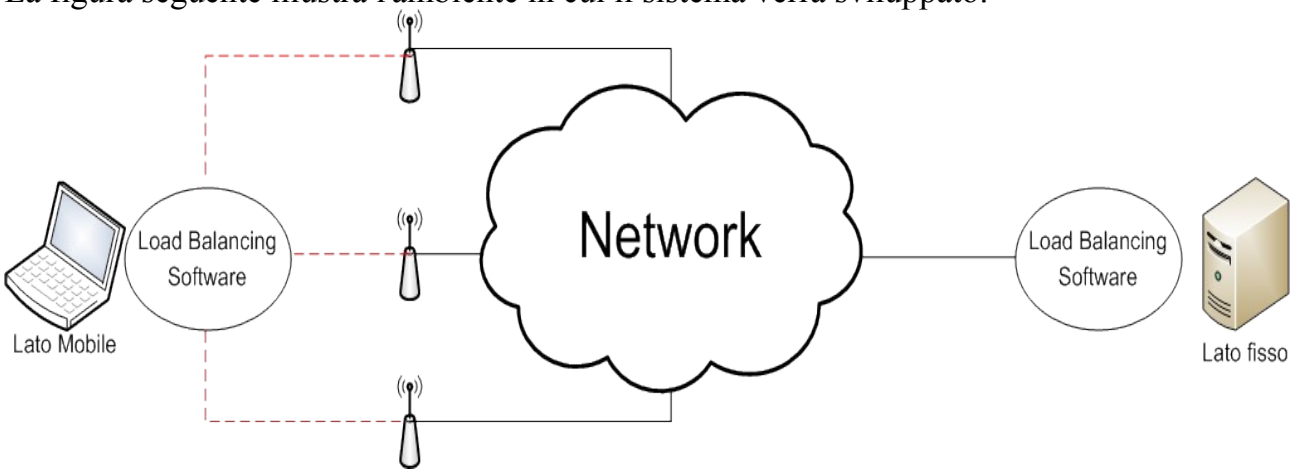

Figura 3.1: Situazione generale dell'ambiente in cui verrà sviluppato il sistema

La comunicazione avviene tra un computer mobile (es. un laptop) che ha la possibilità di muoversi liberamente nello spazio circostante e un elaboratore fisso.

Il software che verrà sviluppato su entrambi i lati dovrà stabilire quale è il canale di comunicazione migliore secondo i seguenti parametri:

- Percentuale di Perdita dei pacchetti
- Latenza della trasmissione

Infatti secondo alcune ricerche è emerso che questi due sono i fattori alla base di una chiara trasmissione vocale. E' ammessa una percentuale di pacchetti persi pari al 5% e una limite massimo di ritardo sui pacchetti voce spediti pari a 150ms oltre il quale la trasmissione risulta poco comprensibile. Per rispettare questi invarianti è stato studiato un meccanismo di raccolta delle informazioni che tiene traccia degli eventi che si sono verificati durante l'invio e la ricezione dei pacchetti. I seguenti argomenti sono fondamentali per una corretta comprensione del sistema sviluppato:

Notifica kernel gli elaboratori ideali che si presuppone vengano utilizzati per la comunicazione devo avere installato un Kernel GNU Linux modificato che fornisce le informazioni tramite syscall inerenti ai pacchetti IP inviati e sia in grado di stabilire se l'AP a cui è stato inoltrato il datagram UDP ha ricevuto con successo i dati.
Questo kernel infatti possiede una modifica allo stack di rete che sfruttando un meccanismo di notifica riguardo il prime hop dell' instradamento riesce a ottenere delle informazioni sul corretto invio dei pacchetti.

HeartBeat come accennato precedentemente un load balancer generico per comprendere lo stato attuale di uno dei suoi nodi sfrutta un meccanismo di heartbeat. Nel nostro caso si è deciso di procedere con l' heartbeat in maniera "temporizzata" cioè avviando la procedura relativa ogni volta che passa un certo periodo di tempo affinchè sia possibile ottenere informazioni aggiornate sullo stato della trasmissione. All'interno dei datagram di tipo heartbeat verrà inserita la quantità attuale di pacchetti ricevuta dall'endpoint opposto affinchè sia possibile determinare quanto è affidabile l'instradamento scelto. Alla ricezione della risposta dell'heartbeat inviato, sarà calcolata la latenza (ciò avviene per ogni pacchetto ricevuto) e si avranno a disposizione le informazioni inerenti a ciascun socket. Il lato fisso invece non è predisposto per iniziare la procedura di heartbeat, esso infatti si lascerà "guidare" dal lato mobile per capire a quale indirizzo deve inoltrare i messaggi contenenti voce e in assenza di pacchetti dati sarà comunque in grado di comprendere quale è il miglior canale di comunicazione secondo le informazioni acquisite durante la trasmissione.

Errori di trasmissione durante l'invio di un qualunque pacchetto può capitare che la notifica del kernel risulti negativa o assente perchè l'AP che riceve i nostri dati potrebbe non essere raggiungibile/guasto o al momento sovraccarico. Per gestire al meglio questo genere di problematica sono stati introdotti dei contatori di errori di trasmissione per ogni socket che vengono aggiornati in base all'esito di un invio. Essi sono stati utilizzati per controllare l'invio eseguito dal mobile all' AP (notifica kernel) e dal mobile al fisso (hearbeat). Se tali contatori oltrepassano un certo limite allora il socket relativo verrà eliminato perchè "giudicato" malfunzionante.

Chiave di riconoscimento ogni volta che viene inviato un pacchetto esso conterrà al suo interno una "chiave" che verrà confrontata con quella in possesso dall'altro endpoint che riceverà i dati. Tale chiave viene generata dal fisso e scambiata quando si stabilisce la connessione e rimane invariata fino al termine della trasmissione. L' utilizzo di questo meccanismo è fondamentale nel caso reale in cui il lato mobile continua a cambiare indirizzo perchè aggancia nuovi access point e quindi il lato fisso ricevendo pacchetti da indirizzi differenti avrà bisogno di stabilire quali sono stati realmente inviati dall' endpoint

opposto. In realtà la questione è molto più complessa nell'ambito della sicurezza in quanto sarebbe facile per un possibile aggressore catturare un pacchetto, studiarlo e comprendere che quel campo del messaggio serve per il riconoscimento. La soluzione utilizzata mette da parte la sicurezza ma non esclude che in un futuro sviluppo si possa sostituire la chiave con una firma digitale da utilizzare insieme alle chiavi scambiate sempre nella fase iniziale della comunicazione.

Miglior Socket per stabilire quale sia il miglior socket su cui inviare i dati, il bilanciatore di carico ordinerà la sua lista sockets secondo le informazioni ottenute tramite il meccanismo di heartbeat. Le funzioni di ordinamento dettagliatamente descritte nel capitolo sull' implementazione stabiliranno secondo quali parametri un socket è ritenuto migliore o eguale rispetto ad un altro socket. Al termine della fase di sort la lista sarà ordinata dal migliore al peggiore quindi basterà accedere all'elemento in testa per inviare i dati.

Ora che sono chiari i parametri su cui il sistema di bilanciamento si basa è possibile riassumere le ipotesi dell'algoritmo:

- La comunicazione tra i due Client è divisa in due parti, una wireless (Mobile - Access point) e una wired (Access point - Fisso).
- Ogni pacchetto può non essere correttamente ricevuto dall'Access point.
- Ogni pacchetto se inoltrato dall'Access point può perdersi nella rete a causa dell' inaffidabilità del protocollo UDP.
- Non è detto che i pacchetti arrivino in ordine sul lato Fisso a causa del protocollo UDP.
- Per Ogni corretta ricezione da parte dell'access point, viene inviato un pacchetto di notifica al Client mobile (questa informazione e i relativi dati sono reperibili tramite syscall).
- La mancata ricezione della notifica per n-volte consecutive su un dato socket del lato mobile viene considerata come morte del socket.
- Sul Client Mobile è possibile aggiungere/rimuovere in qualsiasi momento un'interfaccia di rete.

Le nostre esigenze in termine di prestazioni sono:

- Trasmissione veloce (ritardo massimo per pacchetto 150 ms).
- Percentuale di perdita pacchetti (massima 5%).
- Mancanza di una connessione effettiva perchè le attuali applicazioni VoIP utilizzano questo protocollo e l'IP del lato mobile è soggetto a continui cambiamenti (utilizzo di protocollo UDP).

3.1 Schema generale

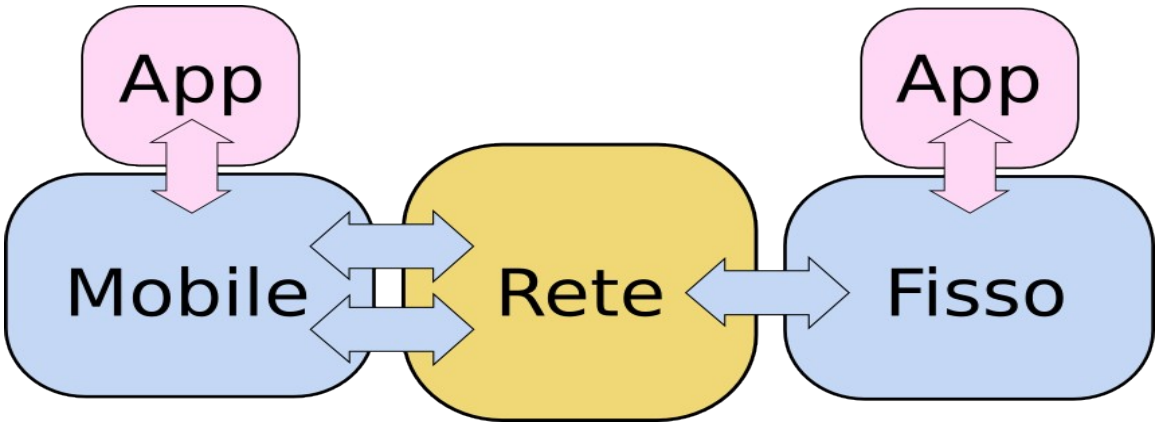

Figura 3.2: Schema base di progettazione

Il diagramma illustra la situazione in cui il sistema di load balancing verrà sviluppato.
I cerchi del grafico indicano le diverse entità presenti nel modello:

- Applicazioni VoIP (rosa)
- Bilanciatori di carico (celeste)
- Rete (giallo)

Le applicazioni VoIP si connetteranno ai bilanciatori di carico per inviare e ricevere i dati dal relativo endpoint. Il tipo di protocollo utilizzato è il TCP perchè in questo modo le applicazioni non devono effettuare ulteriori operazioni di riordino messaggi e i bilanciatori devono funzionare indipendentemente dal tipo di dati che riceveranno/invieranno. La connessione inoltre è locale anche se nulla vieta di posizionare i bilanciatori su altri elaboratori ma non è il nostro caso.
I cerchi celesti rappresentano le unità effettivamente sviluppate e tramite essi sarà possibile gestire correttamente e in maniera efficiente il bilanciamento di carico.
Come detto nelle ipotesi il protocollo utilizzato dai due Client è UDP che non garantisce l'effettiva consegna dei dati in maniera ordinata ad un dato indirizzo.
Per questo motivo è necessario sviluppare un protocollo aggiuntivo che riesca a gestire le problematiche presenti nella rete.
E' evidente inoltre che il sistema è asimmetrico in quanto il lato mobile utilizzerà uno o più access point per comunicare con il lato fisso che invece riceverà solo su un unico indirizzo statico. Infine il cerchio giallo rappresenta la rete che al suo interno contiene tutti gli instradamenti utilizzati per gestire la comunicazione tra i due endpoints.
Lo schema inoltre rappresenta il modello utilizzato durante i tests effettuati localmente per assicurare il regolare funzionamento dell'algoritmo.

3.2 Analisi di sviluppo dell'algoritmo

L'analisi delle tecniche impiegate ha permesso di stabilire delle associazioni tra gli elementi appartenenti al caso generale del load balancing e il nostro.
Le seguenti immagini permettono di visualizzare le analogie presenti:

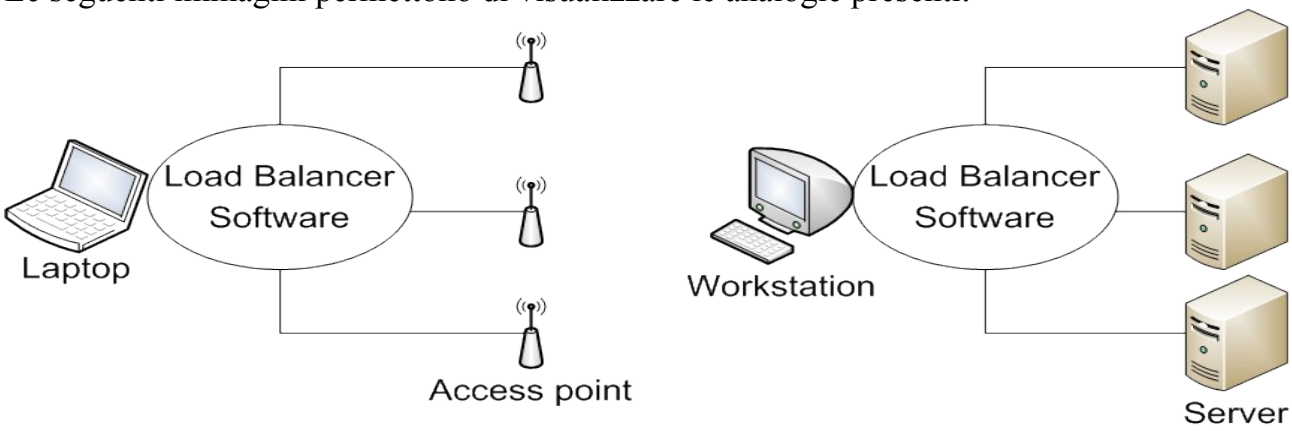

Figure 3.3: Analogie tra il nostro e il caso generale

- Il Client Mobile è come se fosse una workstation mobile
- Gli Access points sono il corrispondente dei Servers di una situazione generale

Nel nostro caso il load balancer non è fisso ma è mobile in quanto il Client si muoverà nell'ambiente che lo circonda. Tale differenza è molto importante perchè nel nostro caso il lato Fisso dovrà essere in grado di inoltrare le risposte correttamente anche nella situazione in cui il Client mobile dovesse cambiare l'indirizzo di rete(aggancio ad un nuovo access point, aggiunta o rimozione di un'interfaccia di rete). Nella situazione generale invece l'indirizzo IP del computer che utilizza il Load Balancing software è statico e quindi questo problema non è presente. Per far fronte a questa problematica si è deciso di utilizzare una chiave di riconoscimento che permetterà al Client Fisso di rispondere in maniera corretta al Mobile.
Nel nostro caso inoltre il carico non va distribuito su dei servers ma principalmente sugli access point che si occuperanno di inoltrare i dati al Fisso.
Per quanto riguarda la procedura di HeartBeat si è deciso che sarebbe stata necessaria una modifica che avrebbe mantenuto intatta la funzionalità.

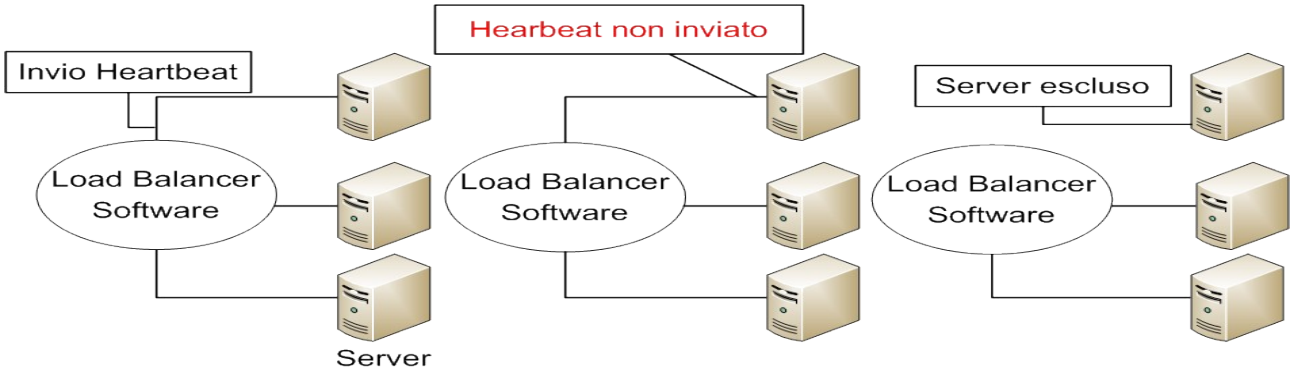

Figura 3.4: Procedura di HeartBeat nel caso generale

Come è possibile vedere a seguito di una failover la Convergenza eliminerà il nodo guasto dal cluster.

Nel nostro caso invece si è stabilito un limite che rappresenta il numero consecutivo di volte di fallimento della procedura oltre il quale viene attivata la Convergenza.

3.3 Sviluppo dell'algoritmo

Completata l'analisi delle due situazioni si è deciso di strutturare l'algoritmo in due fasi:

1)Handshake
2)Trasmissione dati & HeartBeat

La fase uno di Handshake si occupa di scambiare tra Client Mobile e Fisso i dati fondamentali che serviranno in futuro a garantire una corretta trasmissione dei pacchetti.
Come accennato precedentemente viene scambiata una chiave (generata dal Fisso) che il Mobile dovrà utilizzare durante la trasmissione per farsi riconoscere. Utilizzando appunto il protocollo UDP privo di una connessione effettiva sarebbe impossibile per il lato fisso determinare con chi sta dialogando realmente.
La figura seguente permette di comprendere meglio la fase uno sul lato mobile:

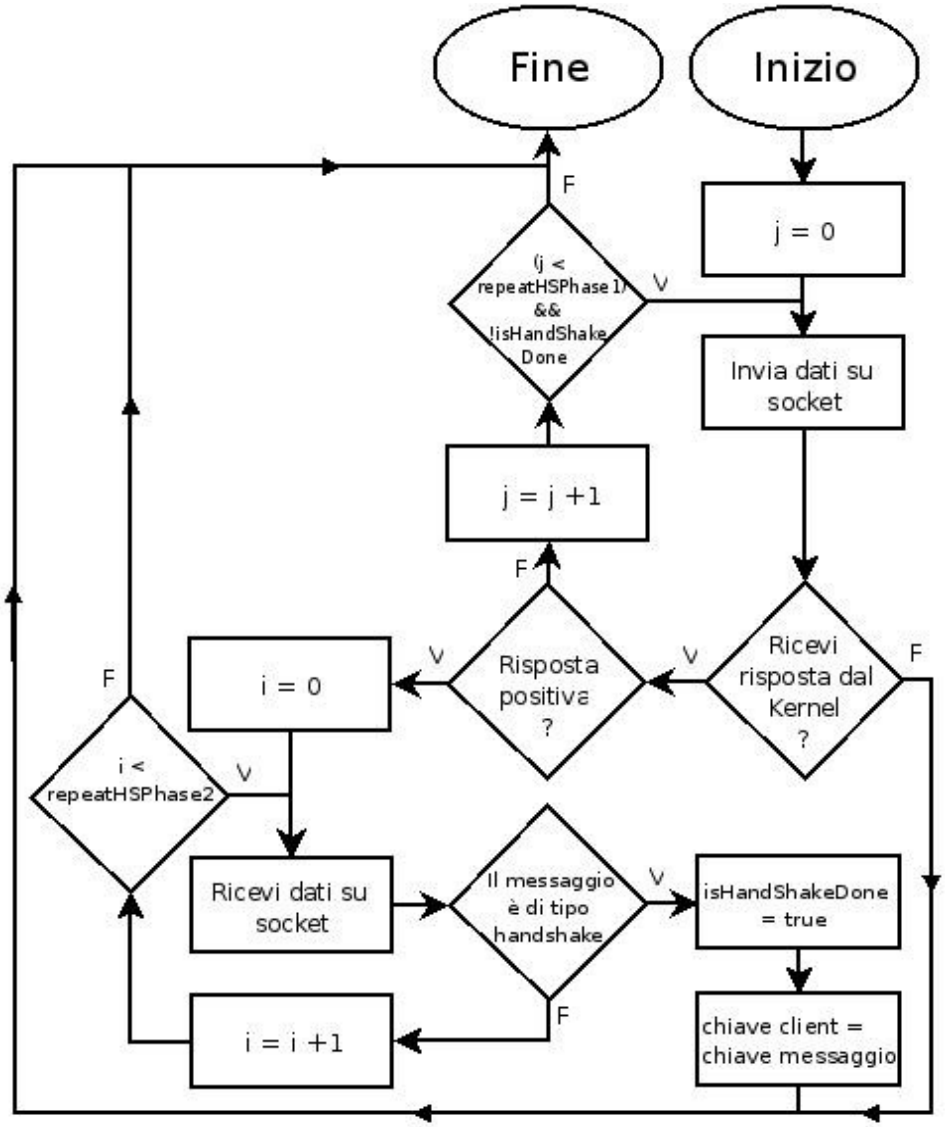

Figura 3.5: Diagramma di flusso per la procedura HandShake sul lato mobile

Il diagramma rappresenta i passi fondamentali che il load balancer software sul lato Mobile effettua per stabilire una connessione autenticata con il Fisso.
L'HandShake viene eseguito una ed una sola volta e può capitare che a causa dei problemi di rete sia necessario ripetere alcuni passi dell'algoritmo. Per questo motivo tale fase è stata divisa in due piccole sottofasi:

- Invio richiesta di HandShake
- Ricezione termine di HandShake

Per ogni sottofase è ammesso un limite massimo(in figura è chiamato con il nome di repeatHSPhase1 e repeatHSPhase2 per ciascuna sottofase rispettivamente) che rappresenta quante volte essa può essere ripetuta. Se la sottofase non si completa entro il relativo limite l'HandShake fallisce e il Client Mobile non potrà iniziare la seconda fase dell'algoritmo.
Il seguente diagramma invece rappresenta la fase di HandShake sul Client Fisso:

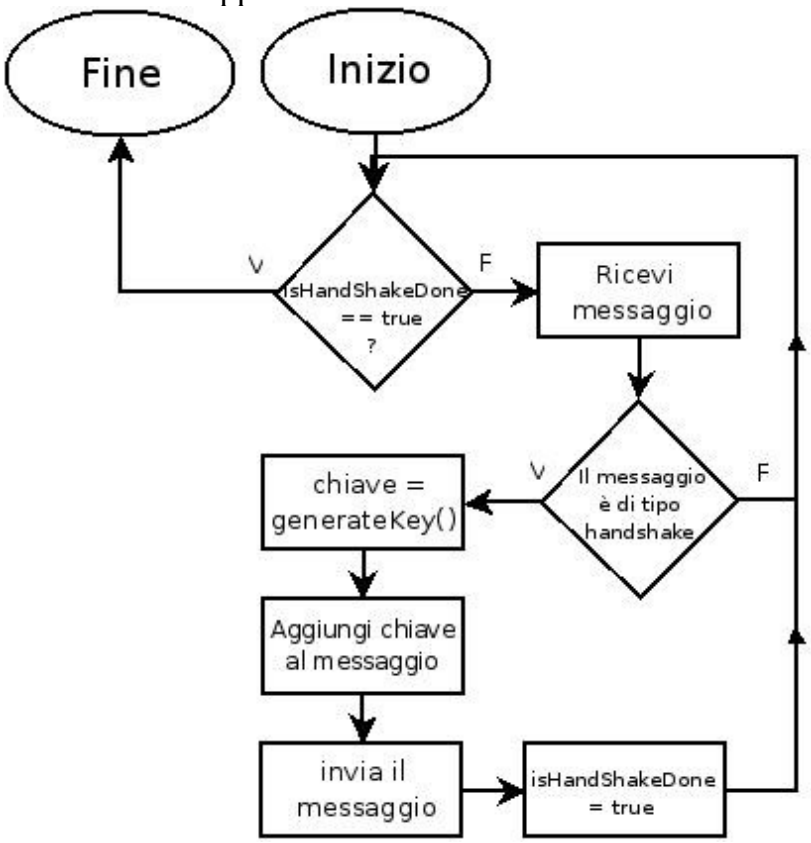

Figura 3.6: Diagramma di flusso per la procedura HandShake sul lato fisso

A differenza del Client Mobile, il Fisso attenderà a tempo indeterminato finchè non riceverà un messaggio di inizio handShake.
A causa dei problemi di rete e del protocollo UDP potrebbe capitare che la risposta inviata dal Fisso vada persa. Per questo motivo quindi sono stati introdotti i limiti delle sottofasi sul lato mobile e si è garantita la gestione di pacchetti di tipo handShake sul lato fisso anche durante la seconda fase dell'algoritmo.
La fase due si occupa invece della trasmissione effettiva dei dati e dei tests che saranno effettuati tramite la procedura di HeartBeat.
Il seguente diagramma illustra l'algoritmo generale implementato dal software del lato fisso e mobile per mandare i dati:

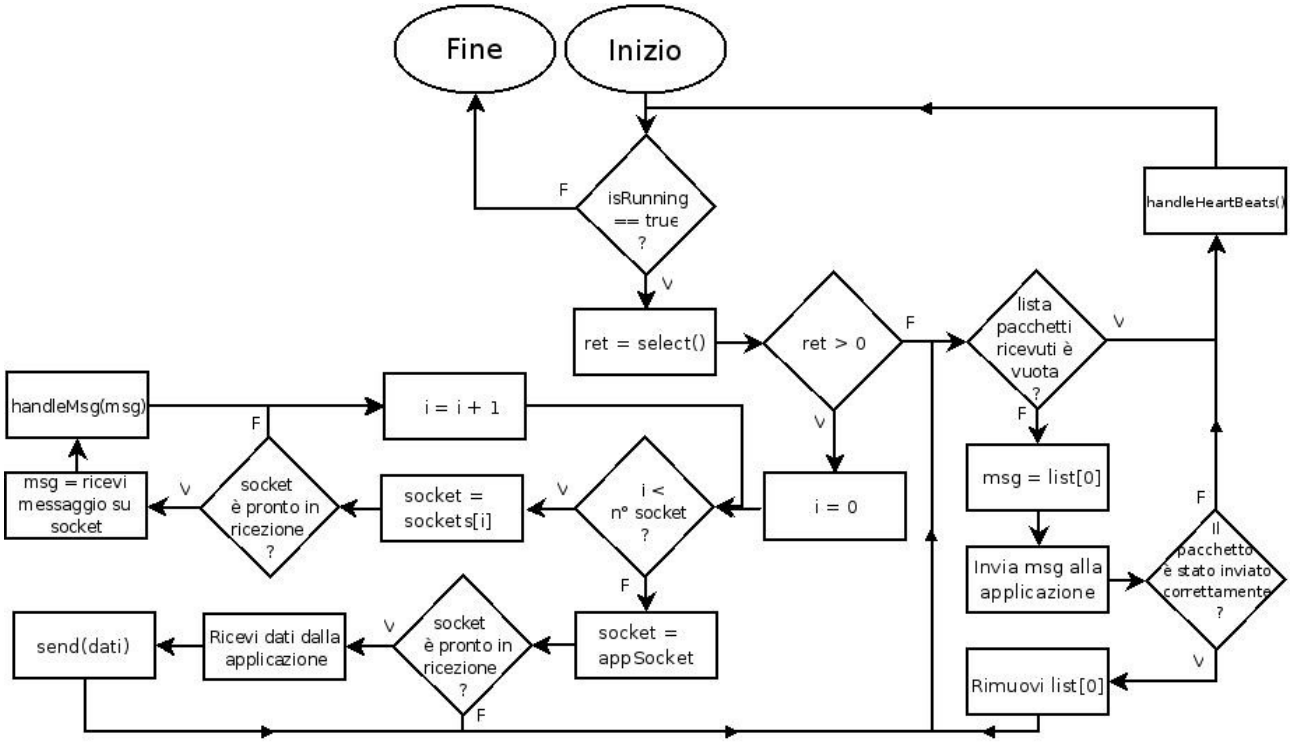

Figura 3.7: Diagramma di flusso per l'algoritmo generale di entrambi i bilanciatori

Tramite la funzione select() si chiede al sistema operativo quali socket sono pronti in lettura in modo tale da poter ricevere i dati da essi.
La fase di ricezione prevede prima il controllo dei sockets utilizzati per la trasmissione tra i due endpoints (mobile e fisso) e poi del socket utilizzato dall'applicazione VoIP.
Dopo aver ricevuto un messaggio esso verrà analizzato tramite una funzione chiamata handleMsg che agirà in maniera “differente” in base al tipo di Client.
Lo stesso ragionamento viene effettuato alla ricezione dei dati dall'applicazione VoIP utilizzando invece la funzione send.
Terminata questa fase si passa al controllo della lista dei pacchetti ricevuti e nel caso in cui essa contega i dati trasmessi dall'altro endpoint si procederà con l'invio di essi all'applicazione VoIP. L'ultimo passo che l'algoritmo esegue a fine ciclo è rappresentato dalla funzione handleHeartBeats() che periodicamente si occuperà di gestire gli heartbeats (invio e ricezione a seconda del tipo di Client).
Avendo analizzato i passi fondamentali dell'algoritmo generale si passa ora all'analisi delle funzioni e relativi diagrammi di flusso che ciascun Client implementerà a suo modo.

3.4 Lato Mobile

Per quanto detto durante la presentazione dello schema generale, il Client mobile è composto da un'applicazione VoIP che utilizza per la comunicazione e da un processo che si occupa di bilanciare i dati secondo l'algoritmo generale.
Analizziamo ora le procedure sostituite sul lato mobile:

1)handleMsg
2)send
3)handleHeartBeats

Iniziamo con la procedura di handleMsg illustrata in questo diagramma di flusso:

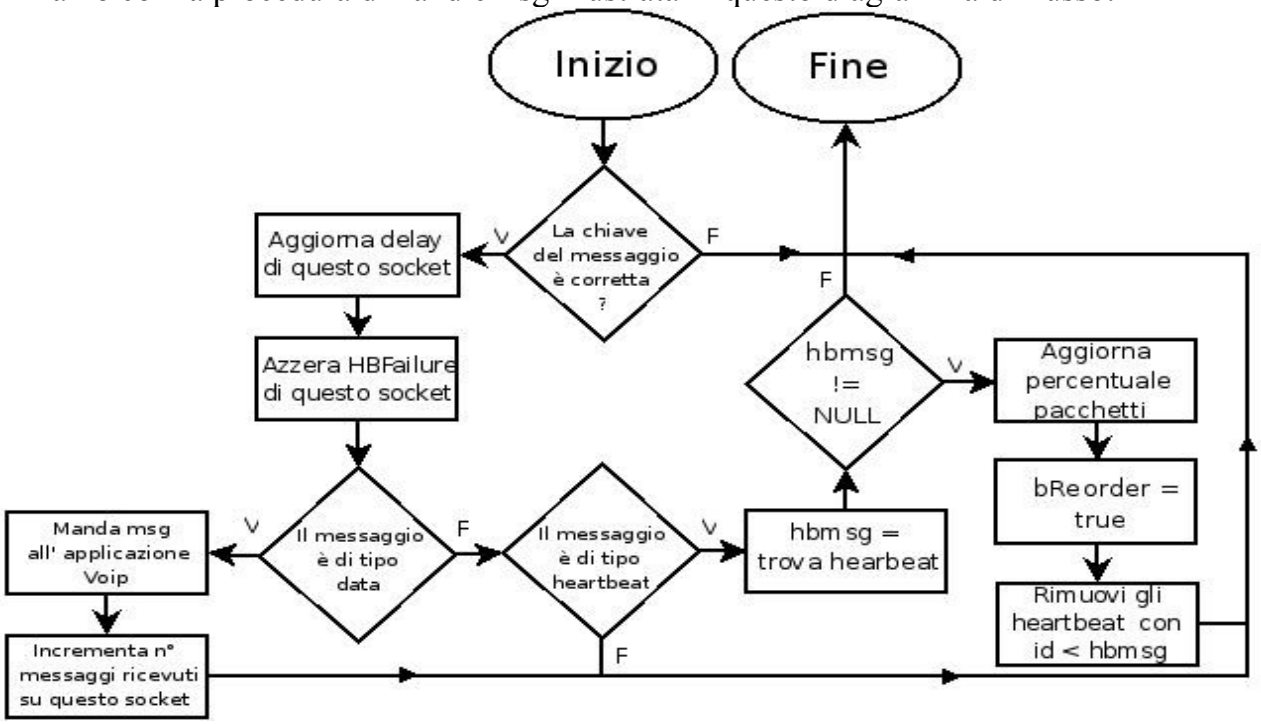

Figura 3.8: Diagramma di flusso per la procedura handleMsg sul lato mobile

Inizialmente si controlla la validità del pacchetto confrontando la chiave contenuta in esso. Nel caso in cui fosse valido si aggiorna il delay calcolando la media con il precedente valore e si azzera il contatore di HearBeat falliti in quanto la ricezione del pacchetto indica che il socket è vivo. A questo punto se il messaggio è di tipo dati sarà spedito direttamente all'applicazione VoIP e poi si incrementerà il numero di messaggi ricevuti su questo socket;
se il messaggio è di tipo heartbeat allora occorrerà ricercare nella lista di hb del socket l'heartbeat inviato precedentemente e nel caso in cui venisse trovato si procederà aggiornando la percentuale dei pacchetti, settando il flag bReorder del bilanciatore a true e rimuovendo tutti gli heartbeat con id minore di quello ricevuto perchè non ci servono le informazioni relative ad un istante precedente a questo.
Passiamo ora al diagramma di flusso della procedura send:

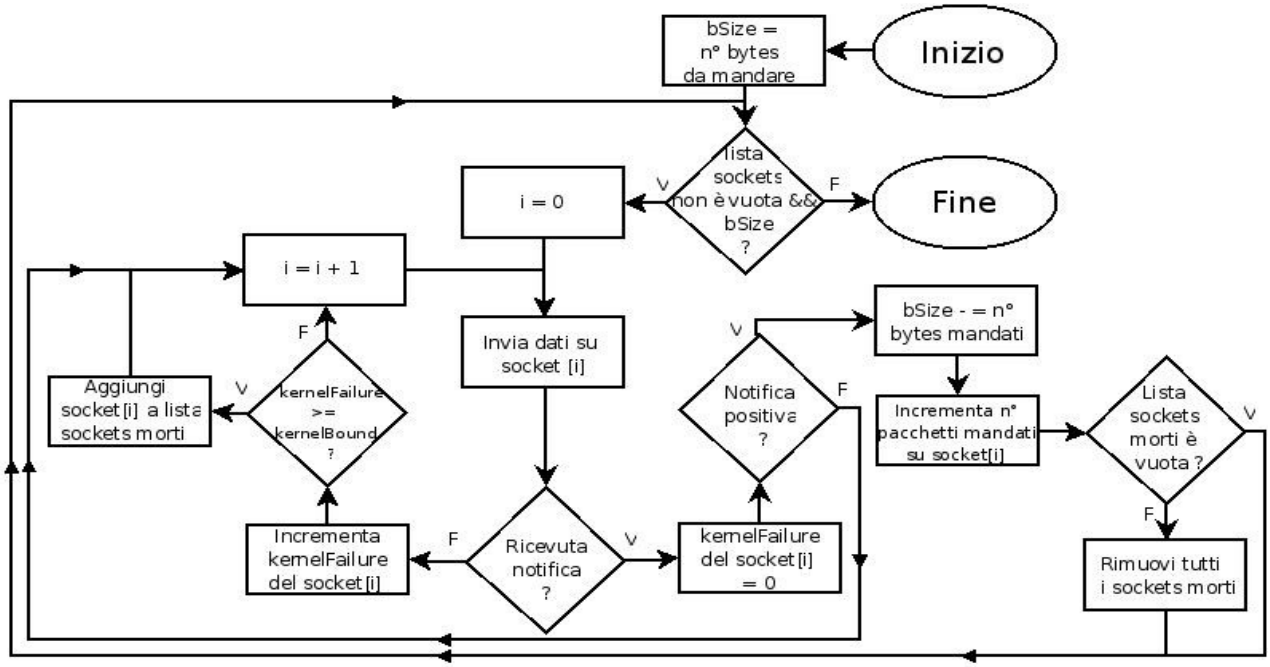

Figura 3.9: Diagramma di flusso per la procedura send sul lato mobile

I bytes letti dal socket dell'applicazione vengo incapsulati nei messaggi che il client mobile spedisce al client fisso. La procedura si occupa quindi di creare e inviare un certo numero di pacchetti in base alla quantità di dati che gli verrà passata come parametro.
Dopo aver effettuato l'invio si richiede al kernel se è stata ricevuta la notifica (positiva o negativa) dell'access point. Nel caso in cui non fosse stata ricevuta la notifica verrà incrementato il contatore di notifiche fallite (kernelFailure) e se maggiore del limite ammesso l'attuale socket sarà aggiunto ad una lista temporanea di sockets ritenuti morti. In caso di ricezione viene azzerato il kernelFailure per quel dato socket e se la notifica è positiva si incrementa il numero di pacchetti inviati e si eliminano gli eventuali sockets presenti nella lista temporanea. E' importante notare che la lista dei sockets viene ordinata dal "migliore" al peggiore durante la procedura di handleHeartBeats rappresentata in figura 3.10 secondo i parametri che vengono aggiornati a seguito della ricezione di un heartbeat.

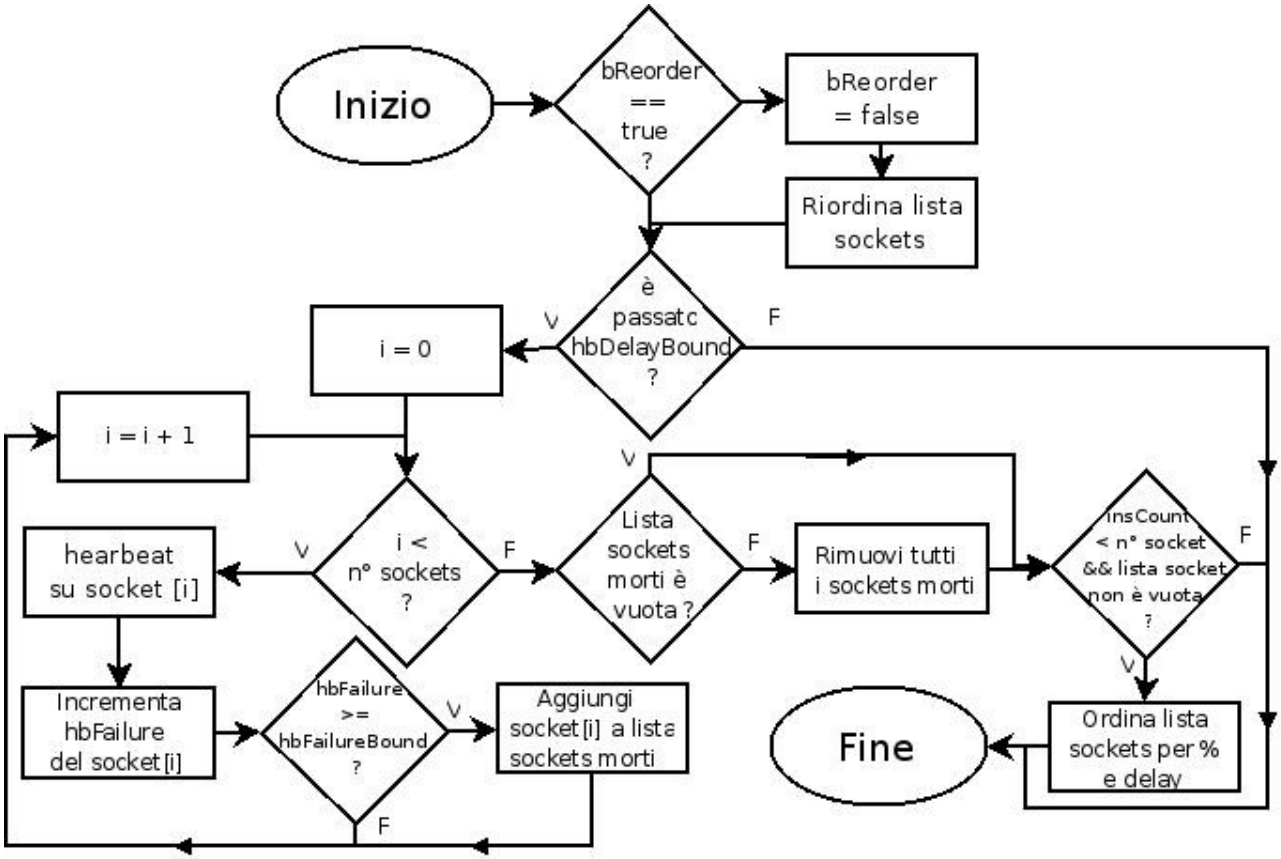

Figura 3.10: Diagramma di flusso per la procedura handleHeartBeats sul lato mobile

Inizialmente si controlla il flag bReorder che indica se precedentemente è stato ricevuto un pacchetto di heartbeat e quindi in caso positivo si provvede a riordinare la lista sockets. Per ottenere le informazioni sullo stato attuale della rete viene eseguito un controllo ogni hbDelayBound millisecondi. Su ogni socket verrà avviata la procedura di heartbeat che è stata interpretata in maniera leggermente differente rispetto allo standard.
Si è pensato di aggiungere all HeartBeat la quantità di pacchetti ricevuti fino a quel momento su un dato socket affinchè sia possibile testare l'affidabilità di un dato canale di trasmissione. All'interno della funzione si procederà con l'invio dell'heartbeat di cui ne verrà salvata una copia contenente la quantità di pacchetti mandata e il suo id perchè quando arriverà la risposta dal lato fisso si dovrà fare riferimento all'istante temporale in cui sono state richieste le informazioni. Terminata questa procedura si incrementa l'hbFailure e se maggiore del limite ammesso (hbFailureBound) si aggiungerà il socket ad una lista temporanea di sockets ritenuti morti che verrà svuotata a fine controllo.

3.5 Lato Fisso

Il Client fisso esegue come il Mobile lo stesso algoritmo generale fornendo una propria versione delle seguenti procedure:

1)handleMsg
2)send
3)handleHeartBeats

Analizziamo ora tramite diagramma di flusso tali procedure inziando con la handleMsg:

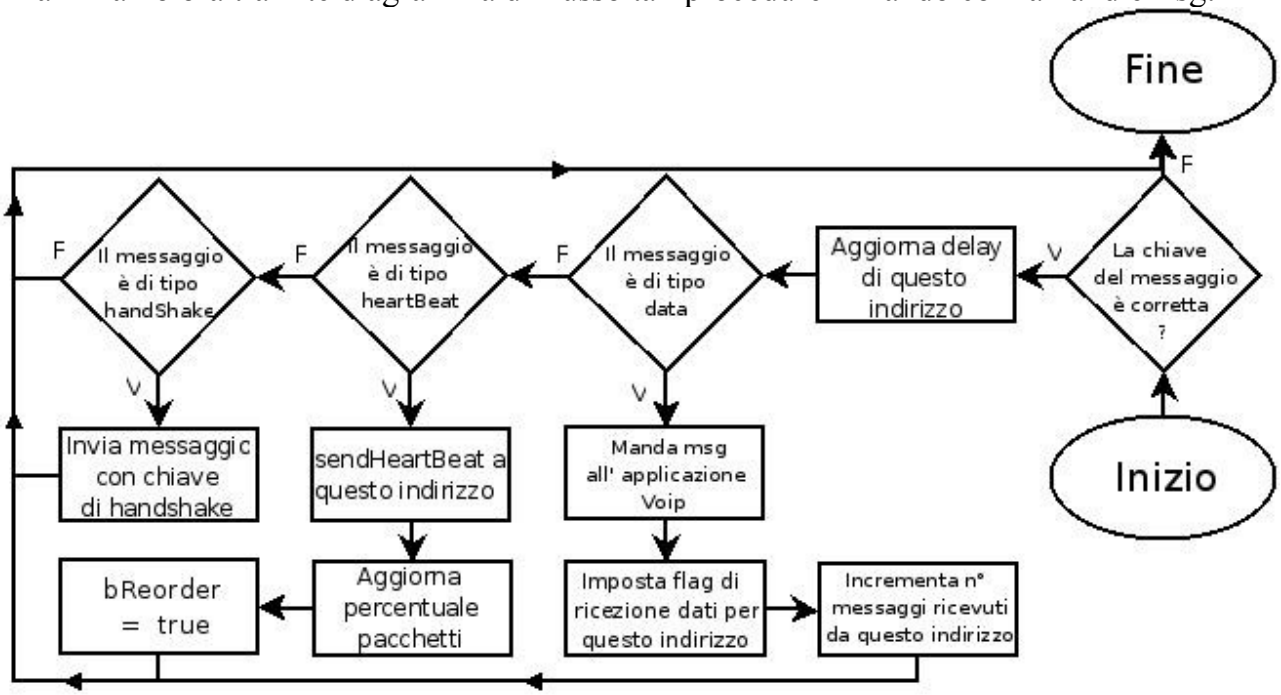

Figura 3.11: Diagramma di flusso per la procedura handleMsg sul lato fisso

Come già visto per il lato Mobile, una volta che è stato ricevuto un messaggio da un dato indirizzo bisogna stabilire se esso è valido e in caso positivo aggiornare il delay presente sull'instradamento utilizzato dall'indirizzo di provenienza. Dopodichè se ne valuterà la tipologia: se esso è di tipo data si cerca di mandarlo immediatamente all'applicazione, si imposterà il flag che indica che tale socket è attivo e si incrementa il numero di messaggi ricevuti. Se è di tipo heartBeat si manderà una risposta secondo quando scritto in precedenza e si procederà con l'aggiornamento delle percentuali e l'assegnamento a bReorder per avviare l'ordinamento durante la handleHeartBeats. Infine se è di tipo handshake si invierà la chiave di riconoscimento al client che la ha richiesta. E' essenziale notare che a differenza del lato mobile, il lato fisso riceverà/invierà messaggi su un solo socket e dovrà operare su strutture dati rappresentanti gli indirizzi da cui riceve i pacchetti e sopratutto non avvierà mai la procedura di heartbeat per ricevere informazioni sui pacchetti.
Per eseguire l'invio dei dati all'endpoint mobile, il lato fisso stabilisce una propria versione della funzione send illustrata in figura 3.12.

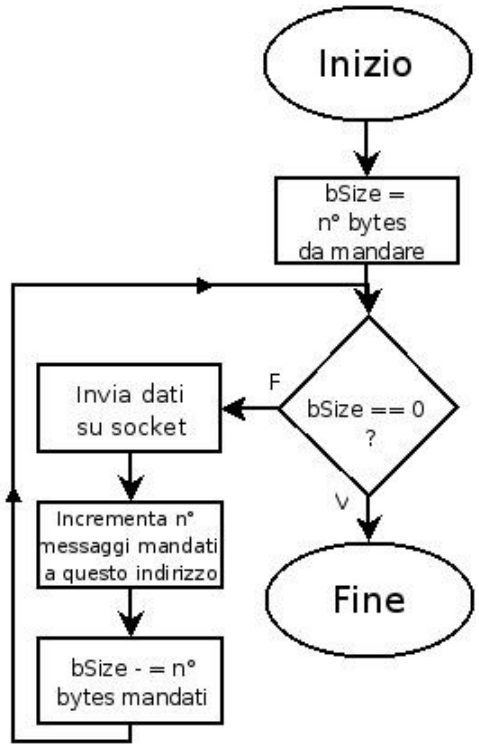

Figura 3.12: Diagramma di flusso per la procedura send sul lato fisso

E' evidente che tale procedura è relativamente più semplice rispetto a quella implementata dal lato mobile perchè il lato fisso non ha alcun tipo di feedback (notifica del kernel) inerente alla spedizione ad un dato indirizzo e quindi inoltrerà il messaggio una sola volta.
Per comprendere meglio come viene effettuata la scelta dell'indirizzo al quale vengono inviati i messaggi bisogna analizzare l'ultima procedura handleHeartBeats:

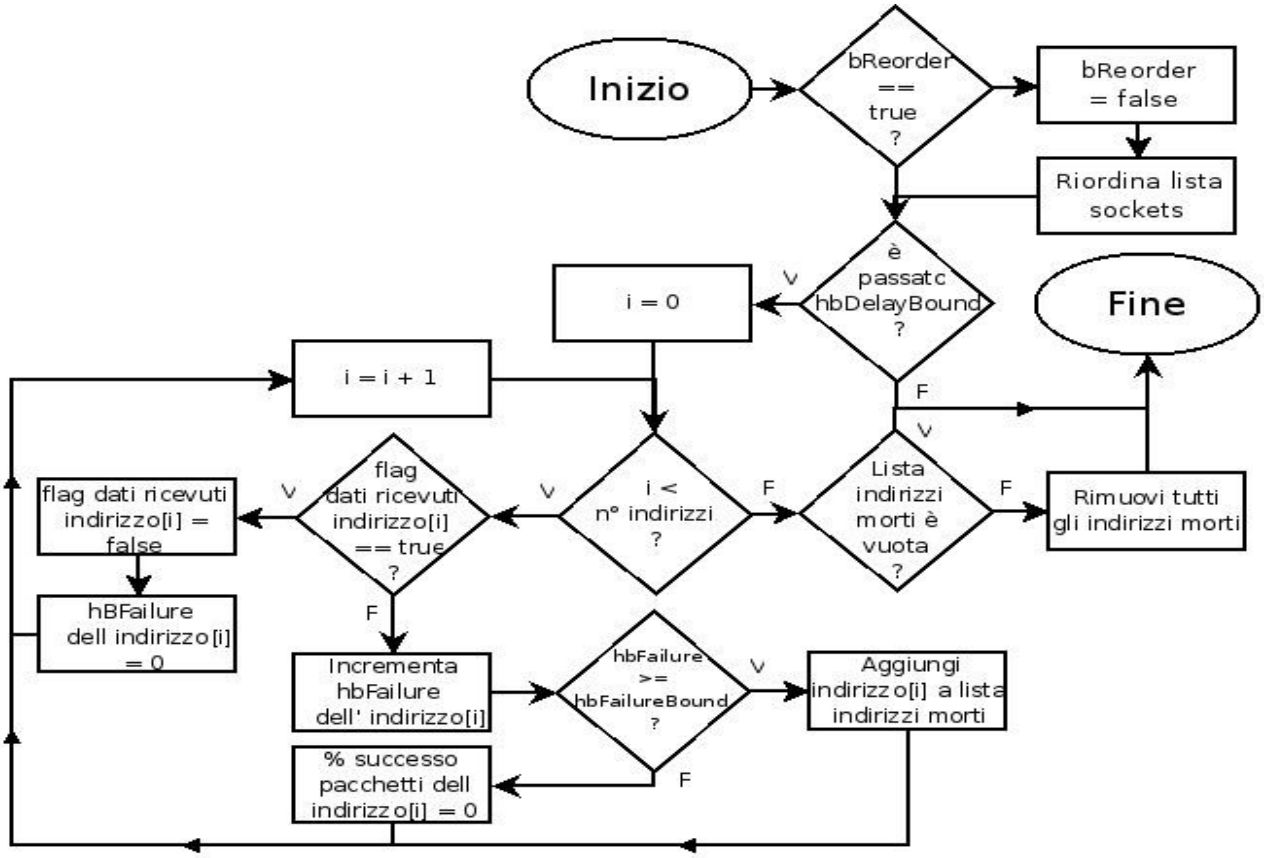

Figura 3.13: Diagramma di flusso per la procedura handleHeartBeats sul lato fisso

Come visto sul lato mobile anche qui viene controllato il flag bReorder per determinare se bisogna riordinare la lista degli indirizzi conosciuti, In seguito si verifica che siano passati hbDelayBound millisecondi e in caso positivo viene accertato che per ogni indirizzo conosciuto è stato ricevuto almeno un pacchetto dati. La grande differenza che è possibile notare rispetto al lato mobile consiste nella procedura di heartbeat, infatti il sistema è asimmetrico e quindi periodicamente il fisso riceverà le informazioni per comprendere meglio la situazione della rete.

Capitolo 4

Implementazione

L'idea base della realizzazione è stata quella di preparare una libreria statica in modo tale che sarebbe stato possibile effettuare il bilanciamento di carico indipendentemente dall'applicazione utilizzata per la trasmissione VoIP.
Per implementare i diagrammi di flusso creati durante la Progettazione si è scelto di utilizzare come linguaggio di programmazione il C/C++ perchè è object-oriented e garantisce una buona efficienza. L'utilizzo degli oggetti è stato fondamentale per poter passare dalla visione astratta a quella concreta del codice, inoltre l'incapsulamento ha reso più semplice ed elegante la rappresentazione dei modelli ideati.
Piuttosto che semplici funzioni spartane richiamabili dal modulo che bilancia il carico vengono istanziati degli oggetti per la gestione dei canali di trasmissione.
Durante la Progettazione sono emersi gli elementi cardine dell'Implementazione che sono:

- Load Balancer Software sul lato mobile
- Load Balancer Software sul lato fisso
- Sockets

Su questi soggetti sono state create delle Classi messe a disposizione delle applicazioni che faranno utilizzo della libreria; esse sono:

- `LBClient`
- `LBMobileClient` e `LBFixedClient`
- `Socket`

Per i tests di simulazione sono state create altre tre classi di supporto:

- `Delayer`
- `Channel`
- `Parser`

Queste sono state utilizzate durante la realizzazione del modulo `delayer.cpp` che simula ciò che avviene ai pacchetti durante la trasmissione.

4.1 Socket

La classe `Socket` rappresenta il vero e proprio canale di comunicazione che viene utilizzato per scambiare i dati tra i due endpoints.
In realtà è una rappresentazione astratta del concetto vero e proprio perchè infatti definisce al suo interno due metodi `virtual` per garantire operazioni polimorfiche a run-time da

parte della classe `LBClient`:

- `virtual size_t send(void * aBuffer, size_t aBufferSize);`
- `virtual size_t recv(void * aBuffer, size_t aBufferSize);`

Visto che il protocollo utilizzato per la trasmissione è UDP si è deciso di creare un'ulteriore classe chiamata `SocketUDP` affinchè i metodi `send` e `recv` fossero implementati tramite le funzioni di sistema `sendto` e `recvfrom`. Per quanto riguarda la connessione tra le applicazioni VoIP e i moduli di bilanciamento è stata creata un'ulteriore classe chiamata `SocketTCP` che implementa i metodi di invio e ricezione tramite le syscall standard `send` e `recv`.
Altri due metodi chiamati `sendInTime` e `recvInTime` sono presenti nella classe `Socket` e sono implementati tramite template method:

- `size_t sendInTime(void * aBuffer, size_t aBufferSize, long *const aTime);`
- `size_t recvInTime(void * aBuffer, size_t aBufferSize, long *const aTime);`

Essi sono stati utilizzati per permettere ricezione e invio entro un certo margine di tempo `aTime` espresso in microsecondi perchè come detto durante la Progettazione la nostra necessità principale è che i pacchetti arrivino entro 150 ms.
Oltre a questi metodi sono presenti alcuni membri definiti `protected`:

```
1)float successPercentage;
2)long int delay;
3)short int hbFailure, kernelFailure;
4)long int totalSentPacketQuantity;
5)struct sockaddr_in serverAddr;
6)struct sockaddr_in clientAddr;
```

(1) e (2) sono molto importanti perchè contengono i valori su cui l'algoritmo di sort si baserà per ordinare la lista dei sockets contenuta nella classe `LBClient`, essi vengono inizializzati dal costruttore rispettivamente a 100.0 e -1 affinchè il Socket venga testato almeno una volta dal bilanciatore (in questo modo infatti l'algoritmo di sort metterà in prima posizione tale Socket perchè ritenuto il migliore);(3) sono i contatori di fallimenti attuali relativi alla procedura HeartBeat e alla notifica di invio del Kernel; (4) è il n° totale di pacchetti che è stato mandato su questo Socket; (5) e (6) sono utilizzati dalle funzioni `send` e `recv` in quanto rappresentano rispettivamente il destinatario dei dati e l'indirizzo dell'ultimo Client da cui si è ricevuto un messaggio.

4.2 LBClient

La classe che gestisce il bilanciamento di carico è la `LBClient`.
Pur essendo una classe che contiene membri puri virtuali (infatti rappresenta il modello astratto che i vari tipi di client erediteranno) essa mette a disposizione dei metodi pubblici per l'aggiunta e la rimozione dei Socket. Essi sono `addSocket` e `removeSocket` i cui prototipi sono i seguenti:

- `long int addSocket(int aSock, struct sockaddr_in* const aAddress, const int aType);`
- `bool removeSocket(const long int aSid);`

Il metodo `addSocket` permette l'aggiunta di un socket alla lista contenuta all'interno del bilanciatore specificando il file descriptor, l'indirizzo del destinatario dei dati e il tipo di Socket; come valore di ritorno verrà restituito il Sid (Socket Id, identificatore univoco per ogni Socket) in modo tale che l'applicazione potrà in un secondo momento rimuovere il Socket dalla lista tramite il metodo `removeSocket`.
Come detto durante la Progettazione, la fase uno di HandShake viene immediatamente avviata alla prima invocazione di `addSocket`.
Se la fase uno terminerà con successo la variabile booleana `isHandShakeDone` verrà impostata a `true` per evitare che successive aggiunte di sockets causino la ripetizione di tale fase. Si è deciso inoltre che tutti i file descriptor rappresentanti i sockets avranno settato il flag `O_NONBLOCK` per permetterci una maggiore libertà di implementazione e negare il blocco a seguito di una `recv`. Inoltre il nuovo socket verrà aggiunto ad un fd_set che sarà utilizzato dalla funzione `select` all'interno dell'algoritmo generale.
Dopo che l'handshake si è completato bisogna aggiungere il socket TCP su cui ricevere/mandare i dati dell'applicazione VoIP, per questo motivo esiste il metodo `addAppSocket` che potrà essere invocato una sola volta:

- `long int addAppSocket(int aAppSockFd, struct sockaddr_in* const aAddress)`

Questa funzione è molto simile alla `addSocket` in quanto contiene le stesse operazioni con la grande differenza che a fine procedura viene avviato il thread che esegue l'algoritmo generale. Durante l'implementazione però è emerso un problema che probabilmente risiede nel paradigma OOP perchè infatti non è possibile se non tramite opportuni artifici avviare un thread che esegua un metodo di un oggetto all'interno dello stesso.
In relazione allo standard POSIX (si è utilizzata la libreria pthread) per creare un thread bisogna passargli l'indirizzo di una funzione che abbia il seguente prototipo:

```
void * threadFunction(void *);
```

Mentre in C++ un 'equivalente implementazione è la seguente:

```
void *ClassName::threadFunction(void *);
```

Come è possibile notare le due funzioni però hanno tipo differente.
Per bypassare gli errori segnalati dal compilatore (g++ nel nostro caso) si è ricorso all'utilizzo di un metodo statico che restituisse l'indirizzo della funzione secondo il prototipo da noi richiesto per mezzo di un cast. Purtroppo però risolvendo questa problematica ne è venuta fuori un'altra inerente alle regole di scope; ogni volta che durante i test si tentava di accedere ai membri privati contenuti all'interno della classe `LBClient` il programma crashava e andava in `segmentation fault.`
Probabilmente per i principi di incapsulamento dati del paradigma OOP il tentativo di accesso allo stack del processo principale sollevava tale errore relativo alla memoria.
La soluzione a questo problema di scope è nata per sbaglio da un'altra idea e forse è stato scoperto un bug (probabilmente non dipendente dal g++ o dalla libreria pthread perchè anche anche windows soffre di tale problema pur utilizzando la libreria di sistema per creare i threads).
Visto che alla creazione di un thread si può passare tramite cast a void* qualsiasi tipo di argomento, si era pensato inizialmente di passargli `this` che in C++ viene utilizzato per indicare l'indirizzo dell'oggetto corrente.
All'interno della funzione eseguita dal thread sarebbe poi stato possibile dereferenziare tale puntatore e accedere ai dati che ci servivano appartenenti all oggetto `LBClient.`
La cosa strana è che passando `this` come argomento non c'è bisogno di alcun tipo di dereference né tantomeno l'utilizzo dei metodi dell'oggetto per accedere ai dati.
Questo hack si è rilevato fondamentale per realizzare il meccanismo di ricezione pacchetti indipendentemente dalle operazioni che il modulo che utilizza la libreria sta eseguendo.
Tornando alla funzione `addAppSocket` per creare il thread di ricezione si è utilizzata la seguente procedura:

```
pthread_create(&lbThread,NULL,
               LBClient::getClientThreadEntry, this)
```

Che utilizza `LBClient::getClientThreadEntry` per ottenere il prototipo corretto per lo standard POSIX e `this` per bypassare le regole di scope.
Il thread così instanziato esegue sostanzialmente l'algoritmo generale descritto nella fase di Progettazione. Il suo compito è quello di gestire lo scambio dei messaggi ricevuti dall'altro endpoint e inoltrarli all'applicazione VoIP finchè la variabile booleana `isServerRunning` ha valore `true.`
Durante il processo di trasmissione come è stato affrontato nel capitolo riguardante la Progettazione, saranno invocate le funzioni di cui ciascuna sottoclasse avrà fornito una opportuna implementazione; esse sono:

```
- virtual bool handShake(Socket* const aSocket) = 0;
- virtual size_t send(void * aBuffer, size_t
                                      aBufferSize) = 0;
- virtual void handleHeartBeats() = 0;
- virtual void handleMsg(LBMsg& aMsg, Socket * aSocket)
                                                  = 0;
```

L'algoritmo generale è stato sviluppato tramite template method, per questo motivo tutti

questi metodi sono virtuali puri affinchè se ne imponga l' implementazione.
Inoltre per motivi di sincronizzazione è stato aggiunto alla classe un mutex per evitare che le operazioni di aggiunta/rimozione lascino la lista sockets in uno stato inconsistente.

4.3 LBMobileClient

Questa classe eredita la `LBClient` ed è stata utilizzata dal modulo che si occupa di bilanciare il carico sul lato mobile. Oltre ai metodi virtuali che deve implementare, sono stati aggiunti altri due metodi molto importanti:

- `virtual short int sendMsgOnSocket(LBMsg& aMsg, Socket*const aSocket);`
- `short int heartBeat(Socket* const aSocket);`

Il primo è utilizzato durante l'invio dei dati agli access points e serve a stabilire se è stata ricevuta la notifica dal Kernel. Esso è stato reso virtual per permettere alle sottoclassi di modificarne il comportamento in modo tale che venga supportato anche il caso reale (durante i tests infatti la notifica è rappresentata come un datagram UDP ricevuto dal `Delayer`). E' importante notare che il valore di ritorno deve essere sempre uno dei seguenti:

- 0 (ricevuta notifica negativa)
- 1 (ricevuta notifica positiva)
- 2 (notifica non ricevuta)

Questo perchè le funzioni di invio invocano questo metodo al loro interno e basano le loro scelte su tale valore.
Il secondo metodo invece è utilizzato nella procedura `handleHeartBeats` e serve per utilizzare il meccanismo di heartbeat. Al suo interno questa funzione si occuperà di mandare un messaggio contenente la quantità di pacchetti ricevuta fin a quel momento dal lato fisso e in seguito salverà una copia di tale messaggio per farvi riferimento nel momento in cui verrà ricevuta la risposta dall'endpoint opposto. Per quanto riguarda i metodi virtuali puri essi sono stati implementati secondo quanto detto durante la fase di Progettazione senza particolari differenze. E' importante però vedere il funzionamento dell'algoritmo di sort utilizzato per riordinare la lista dei sockets al termine della procedura `handleHeartBeats;` tramite il metodo sort delle liste appartenenti alla libreria STL è possibile passare come argomento una funzione che prenda come parametri due elementi aventi lo stesso tipo degli oggetti di cui la lista è composta. Per questo motivo è stata creata una funzione chiamata `compareSocket` che verrà passata come argomento al metodo `sort`:

```
static bool compareSocket(Socket* aSocket1, Socket*
                                                aSocket2){
    float p1 = aSocket1->getSuccessPercentage(),
          p2 = aSocket2->getSuccessPercentage();
    long int d1 = aSocket1->getDelay(),
```

```
                d2 = aSocket2->getDelay();
        return compareWrapper(p1, p2, d1, d2);
    }
```

Questa funzione presi due Socket come parametri invocherà al suo interno una seconda funzione `compareWrapper` che effettua il controllo vero e proprio e stabilisce quale dei due canali viene prima nell'ordinamento:

```
    static bool compareWrapper(const float p1,const float p2,
                    const long int d1, const long int d2){
    if((p1 == p2) && (d1 < d2))
        return true;
    else if(p1){
        if((p1 > p2) && (d1 < SORT_DELAY_MAX))
            return true;
        else if((d2 > SORT_DELAY_MAX) && (d1 < d2))
            return true;
    }
    return false;
}
```

Il controllo eseguito stabilisce quando i dati relativi al primo socket sono migliori rispetto a quelli del secondo; i casi possibili sono i seguenti:

- Stessa percentuale e delay minore.
- Percentuale maggiore e delay sotto la soglia massima (150ms nel nostro caso).
- Percentuale minore ma delay delay del secondo socket oltre la soglia massima.

Come è possibile notare la macro `SORT_DELAY_MAX` rappresenta il limite massimo oltre il quale un socket è considerato valido. E' logico pensare che presi due canali, il primo con perdita di pacchetti nulla ma con forte latenza e il secondo mediamente stabile ma molto veloce, comporterà una scelta basata sulla velocità in quanto permetterà comunque che la comunicazione continui anche se sembra che secondo l'ordine dei confronti è stata ritenuta più importante la stabilità rispetto all'interattività della trasmissione. Dopo che il riordinamento della lista è terminato si avrà in testa il Socket con le statistiche migliori ed esso verrà utilizzato al successivo invio dei dati.

4.4 LBFixedClient

L'interlocutore della classe `LBMobileClient` prende il nome di `LBFixedClient`.
In modo analogo a quanto fatto sul lato mobile l'applicazione che necessiterà di ricevere il flusso dati VoIP dovrà connettersi al modulo bilanciatore che sfrutterà questa classe per gestire i dati. In maniera simmetrica per quanto riguarda la gestione dell heartbeat, è stato aggiunto un nuovo membro funzione:

```
– bool sendHeartBeat(Socket* const aSocket, LBAck*
                                aAck ,const long int aSid);
```

Esso viene invocato dalla procedura `handleMsg` nel caso in cui sia stato ricevuto un messaggio di inizio heartbeat. Il suo compito è quello di ricercare i dati richiesti dal lato mobile in base al sid contenuto nel messaggio (per distinguere univocamente una data connessione si è deciso di aggiungere un campo all'interno dei messaggi scambiati che rappresenta il sid di provenienza del pacchetto). Per ricercare le informazioni quindi è stato aggiunto un altro membro funzione:

- `CLIData * findClientBySid(const long int aSid);`

Esso procederà con la ricerca della struttura dati `CLIData` avente per sid `aSid` e ne ritornerà l'indirizzo. `LBFixedClient` contiene quindi una lista di strutture dati che sono l'equivalente delle informazioni che la classe `LBMobileClient` distribuisce all'interno degli oggetti `Socket`. Tale lista viene aggiornata in due occasioni:

- Ricezione di un messaggio con sid sconosciuto (aggiunta)
- Fallimento della procedura di hearbeat (rimozione)

Nel primo caso se un messaggio ha un sid di cui non si hanno a disposizione le informazioni, verrà creato un nuovo `CLIData` e sarà poi aggiunto alla lista.
Nel secondo caso invece se la struttura dati ha fallito il controllo per `hbFailureBound` volte consecutive durante la procedura di `handleHeartBeats` essa sarà rimossa.
Un'altra differenza presente rispetto al lato mobile consiste nell'utilizzare un unico oggetto Socket per inviare i dati perchè è possibile modificare l'indirizzo al quale si spedirà il messaggio tramite il metodo `setServerAddr`.
In maniera analoga a `LBMobileClient`, questa classe dovrà essere in grado di comprendere quale è l'indirizzo migliore a cui spedire i dati. In base ad una precedente ricezione di una richiesta di heartbeat, all'inizio della procedura di `handleHeartBeats` verrà riordinata la lista dei `CLIData` tramite il metodo sort che prenderà in input la seguente funzione:

```
static inline bool compareClient(CLIData * aCliData1,
                                 CLIData * aCliData2){
    float p1 = aCliData1->successPercentage,
          p2 = aCliData2->successPercentage;
    long int d1 = aCliData1->delay,
             d2 = aCliData2->delay;
    return compareWrapper(p1, p2, d1, d2);
}
```

Questa funzione al suo interno richiama la `compareWrapper` già vista in precedenza nella `LBMobileClient`. Al termine del riordino la lista dei `CLIData` avrà in testa la struttura il cui indirizzo è ritenuto il migliore che non necessariamente dovrà coincidere con quelo scelto sul lato mobile. Per questo motivo il lato fisso sceglierà l'ultimo indirizzo valido da cui ha ricevuto un messaggio; potrebbe capitare però che il lato mobile non invii dati per un determinato periodo perchè l'interlocutore non sta parlando, allora grazie ai pacchetti di heartBeat spediti periodicamente il lato fisso potrà comunque determinare quale è il migliore canale di trasmissione.

4.5 Delayer

La classe `Delayer` è stata creata per permettere di testare in locale l'algoritmo creato senza aver bisogno di un vero access point fisico e due computer che si scambiano le informazioni. Il suo compito è quello di fornire dei metodi per l'aggiunta e la rimozione di canali di trasmissione e garantirne l'accesso tramite un'interfaccia che servirà per modificare i parametri relativi al delay, drop e lost.
Per quanto riguarda le operazioni sulla lista di `Channel` sono disponibili i seguenti metodi pubblici:

```
- int addChannel(int aSock1, int aSock2,
                 struct sockaddr_in* const servAddr1,
                 struct sockaddr_in* const servAddr2,
                 const float aDropPercentage = DROP_PERCENT,
                 const float aLostPercentage = LOST_PERCENT,
                 const int aDelayBase = base,
                 const int aDelayBound = bound);
- bool removeChannel(const int aCid);
```

Il primo metodo prende in input i seguenti parametri: i sockets che rappresentano i due estremi del canale; gli indirizzi a cui mandare i bytes ricevuti dall'estremo opposto; la percentuale di drop dei pacchetti ricevuti utilizzata per simulare l'impossibilità di inviare un messaggio ricevuto da parte dell'access point; la percentuale di lost dei pacchetti per simulare la perdita di un pacchetto su un determinato instradamento; il ritardo base che verrà aggiunto per simulare il tempo trascorso durante il trasporto nella rete; il limite massimo di ritardo che può essere aggiunto ad un messaggio.
Quando il metodo sarà chiamato si creerà un oggetto `Channel` con gli argomenti passati e tramite la funzione `startThread` si avvierà il thread con lo stesso artificio visto nella classe `LBClient`. Il valore di ritorno della procedura è un identificatore univoco chiamato cid che potrà essere utilizzato successivamente come parametro per la `removeChannel` e anche nei seguenti metodi:

```
- void setDropPercentage(const int aCid, const float
                                        aDropPercentage);
- void setLostPercentage(const int aCid, const float
                                        aLostPercentage);
- void setDelay(const int aCid, const int aDelayBase,
                                    const int aDelayBound);
```

Essi possono essere invocati per cambiare i valori utilizzati durante i calcoli sul drop, lost e delay del canale. In particolare occorre ricordare che il calcolo del ritardo fornisce valori appartenenti ad un intorno del valore base (base ± bound).

4.6 Channel

La classe `Channel` rappresenta un canale di comunicazione generico.
Nel nostro caso non è stata data l'opportunità di creare canali basati su protocollo TCP perchè ci interessava soltanto testare l'algoritmo secondo le ipotesi stabilite. E' stata quindi utilizzata la classe `SocketUDP` per rappresentare i due estremi del canale.
Il `Delayer` quando istanzierà un oggetto `Channel` provvederà a passargli, oltre ai normali parametri visti in precedenza, una coppia di socket pair perchè il thread che sarà avviato tramite il metodo `startThread` eseguirà una `select()` bloccante e per evitare problemi di memoria e sincronizzazione sarà necessario risvegliarlo quando l'oggetto `Delayer` verrà distrutto.
Alla Ricezione di un messaggio si eseguono azioni differenti in base al tipo di estremo:
se l'estremo è sul lato mobile allora si invierà la notifica Kernel di cui si è discusso nel precedente capitolo, si calcolerà se il messaggio deve essere inviato correttamente e in caso positivo si eseguirà un test per stabilire se il pacchetto deve essere perso nella rete e infine si aggiungerà ad una lista di pacchetti ordinata secondo il ritardo.Se l'estremo è sul lato fisso allora si simula la perdita nella rete e in caso negativo si ritarda il pacchetto.
Inoltre per controllare l'andamento della rete in base ai parametri di creazione sono stati aggiunti dei metodi per ottenere i valori inerenti alle quantità di pacchetti droppati, persi e ricevuti. In sostanza il compito di un `Channel` è ricevere dati sui sockets, eseguire i calcoli inerenti alla rete, inoltrare il messaggio sull estremo opposto.

4.7 Parser

Durante i tests si è presentata la necessità di effettuare delle prove con parametri variabili a runtime. Per questo motivo il modulo `delayer.cpp` istanzia un oggetto di tipo `Delayer` e i relativi `Channel` di comunicazione e poi un oggetto `Parser` che servirà a scindere le stringhe lette sullo standard input rappresentanti i comandi per modificare i parametri che i canali adoperano per eseguire i calcoli della rete.
All'interno della classe `Parser` è presente una lista di `string*` contenente i tokens attuali della stringa analizzata tramite il metodo `tokenize` ed è possibile accedere a tali elementi utilizzando la funzione `getNextToken`.

4.8 Strutture dati

La struttura base utilizzata per lo scambio dei messaggi tra lato mobile e lato fisso è chiamata `LBMsg`. Essa contiene i seguenti membri:

```
1) long int seqNumber;
2) long int keyword;
3) long int sid;
4) short int type;
```

```
5) timeval time;
6) unsigned int  wDataSize;
7) char data[dataSize];
```

(1) il numero di sequenza del pacchetto che viene incrementato ad ogni invio; (2) la chiave di riconoscimento scambiata durante la fase di handshake; (3) il sid del socket che ha spedito il messaggio; (4) il tipo di messaggio (`hbPacketStart, hbPacketEnd, hsPacketStart, hsPacketEnd, dataPacket, kernelAck`); (5) il tempo di spedizione utilizzato per ricalcolare per ogni pacchetto ricevuto la latenza del canale; (6) la dimensione dei dati dati spediti; (7) i dati veri e propri.
Tutto ciò che viene scambiato tra i due endpoints sarà incapsulato nella struttura dati appena descritta. Durante la fase di heartBeat si inserirà nel campo `data` dell' `LBMsg` la struttura `LBAck` fatta in questo modo:

```
1) long int ackSeqNumber;
2) long int recvPacketCount;
```

(1) è il numero di sequenza dell pacchetto heartbeat e (2) è il numero di pacchetti ricevuti dall' estremo opposto. Per gestire le liste di messaggi ricevuti ma che hanno avuto problemi durante l'invio all'applicazione è stata utilizzata la struttura AppPacket contenente i seguenti dati:

```
1) long int seqNumber;
2) unsigned int  wDataSize;
3) char data[dataSize];
4) char * pData;
```

(1), (2), (3) sono gli stessi identici membri della struttura dati `LBMsg` mentre (4) rappresenta la posizione corrente dei dati che dovranno essere scritti successivamente sul socket dell'applicazione perchè potrebbe capitare che il buffer di invio non sia abbastanza capiente per contenenere tutti i bytes da spedire.
L'ultima struttura dati che viene utilizzata sul lato fisso è la `CLIData` composta dai sequenti membri:

```
1) struct sockaddr_in address;
2) long int recvPacketCount;
3) long int sentPacketCount;
4) long int delay;
5) float successPercentage;
6) long int sid;
7) short int hbFailed;
8) bool bPacketRecv;
```

(1) l'indirizzo da cui si è ricevuto un messaggio; (2) e (3) quantità di pacchetti ricevuti e mandati da questo indirizzo; latenza presente sull'instradamento utilizzato per raggiungere questo indirizzo; (5) percentuale di successo di invio; (6) sid di questo indirizzo; (7) contatore di heartbeat falliti; (8) valore booleano indicante se è avvenuta la ricezione di almeno un messaggio.

Capitolo 5

Prove e Risultati

Lo studio del sistema implementato è stato integrato con delle prove simulative e pratiche volte a verificare il corretto funzionamento del bilanciatore di carico in varie situazioni.
Per i tests simulativi è stato realizzato un modulo chiamato delayer (come scritto nei precedenti capitoli) che simula tutto quello che può accadere durante il percorso che un datagram UDP compie per arrivare a destinazione. Con questo si intendono tutti i problemi relativi alla trasmissione che sono:

- **La mancata ricezione del datagram UDP da parte dell' AP**
  infatti il meccanismo di notifica kernel è stato implementato nel caso simulativo tramite un pacchetto spedito allo stesso indirizzo da cui sono stati ricevuti i dati (ovviamente solo per il lato mobile).

- **La possibilità che il datagram UDP vada perso durante il tragitto**
  data la scarsa affidabilità del protocollo utilizzato si è voluto simulare anche questo tipo di problematica che può verificarsi durante il trasporto dall'AP al lato fisso, tramite un calcolo casuale viene stabilito se il pacchetto verrà spedito all'endpoint opposto.

- **Il ritardo a cui ogni datagram UDP è soggetto durante la trasmissione**
  per simulare la latenza il modulo delayer mantiene una lista di pacchetti che devono essere spediti. Essa è ordinata in modo crescente e le strutture dati che contiene hanno un campo indicante la quantità di tempo che devono aspettare i dati prima di essere effetivamente spediti.

- **La possibilità che uno degli AP diventi guasto**
  è possibile infatti che durante la trasmissione l'AP si sovraccarichi/guasti per questo motivo è stata data la capacità di chiudere un canale di comunicazione a runtime affinchè venisse verificato il comportamento dei proxy in questo tipo di situazione.

Per quanto rigurarda le prove sperimentali esse sono state effettuate nell'area urbana del Dipartimento di Informatica all'università di Bologna. E' stato scelto come lato mobile un laptop dotato di due interfacce wireless IEEE 802.11g che possono essere associate a due differenti AP disponibili nell'area.

5.1 Metriche

Nelle varie prove effettuate sono stati analizzati i fattori ritenuti più importanti:

- **Affidabilità**
  si è verificato che le informazioni raccolte durante la trasmissione dei datagrams venissero utilizzate correttamente dall'algoritmo progettato, infatti tramite i file di log è stato possibile controllare la validità delle prove eseguite.

- **Latenza**
  è stato analizzato il tempo che intercorre tra l'invio di un pacchetto e l'effettiva ricezione dall'endpoint opposto. Inoltre si è accertato che il sistema scegliesse il canale di comunicazione in base alle informazioni ottenute tramite l'heartbeat.

- **Perdita pacchetti**
  si è stabilito quanto fosse il rapporto tra pacchetti inviati e ricevuti in totale su un dato tragitto sfruttando un certo access point in modo tale che venisse testata l'efficienza dell'algoritmo sviluppato.

5.1.1 Parametri

Per avere informazioni complete circa l'attendibilità del nostro sistema sono state effettuate più prove variando i seguenti parametri:

- **Traffico di background**
  alcuni test sono stati eseguiti con traffico nullo cioè soltanto con due utenti connessi agli access points mentre altri invece hanno previsto l'aggiunta di dispositivi che generassero traffico pari a 5Mbps sugli AP interessati.

- **Dimensione dati inviati**
  sono state effettuate prove con pacchetti di dimensione variabile dai 150-2048 bytes per vedere il comportamento durante la frammentazione.

- **Latenza & Scarto**
  è stato possibile modificare sul modulo delayer i valori che influenzano le quantità casuali che vengono generate per aggiungere ritardo e scartare i pacchetti.

- **Bounds di ordinamento**
  Per quanto detto durante la fase di progettazione è presente un define che rappresenta il limite entro il quale un socket è ritenuto valido. Modificando contemporaneamente tale valore e la latenza, si è verificato il comportamento dell'algoritmo di ordinamento dei canali.

5.2 Prove

5.2.1 Prove simulative

Come detto precedentemente le prove simulative sono state svolte su un laptop dove sono stati eseguiti gli applicativi che nel caso reale vengono utilizzati sul lato mobile e fisso durante una trasmissione VoIP. Tali applicazioni utilizzando i bilanciatori di carico inviano e ricevono i pacchetti dal modulo delayer che rappresenta la rete.
Le prove sono state caratterizzate da continui cambiamenti inerenti alle percentuali di perdita dei pacchetti e ritardo aggiuntivo per capire se l'agoritmo stesse operando in maniera ottimale.

Esempio di output applicativo load balancer

Recv, id:191, delay:97, sid:1
Recv, id:192, delay:91, sid:1
Recv, id:193, delay:108, sid:0
Recv, id:194, delay:107, sid:0
Recv, id:195, delay:111, sid:0
Recv, id:196, delay:111, sid:0
Recv, id:197, delay:100, sid:0
Recv, id:198, delay:111, sid:0
Recv, id:199, delay:120, sid:0
Recv, id:200, delay:114, sid:0
Recv, id:201, delay:111, sid:0
Recv, id:202, delay:112, sid:0
Recv, id:203, delay:100, sid:0
Recv, id:205, delay:80, sid:1
Recv, id:204, delay:130, sid:0
Recv, id:206, delay:94, sid:1
Recv, id:207, delay:94, sid:1
Recv, id:208, delay:109, sid:1
Recv, id:209, delay:80, sid:1
Recv, id:210, delay:104, sid:1

Come è possibile notare l'output permette di visualizzare l'id del pacchetto mandato (non quello del datagram UDP), il tempo passato per arrivare all'endpoint opposto e l'id del socket da cui si è ricevuto il messaggio. Se i dati superano i 150 millisecondi allora viene stampata la scritta “TEMPO SUPERATO”. Questo estratto di codice fa riferimento ad una situazione con i seguenti parametri:

- Nessuna perdita di pacchetti da parte del protocollo UDP
- Nessun errore nella trasmissione di una frame ad un AP
- AP 0 con un delay aggiuntivo compreso tra 100 ± 10 ms
- AP 1 con un delay aggiuntivo compreso tra 80 ± 10 ms

5.2.1 Risultati prove simulative

La tabella seguente rappresenta il valore dei parametri utilizzati per configurare il delayer durante i test:

Access Point	Drop	Delay	Lost
AP 0	2,00%	80 ± 10	2,00%
AP 1	2,00%	50 ± 10	10,00%

Secondo la configurazione ci aspettiamo che l'algoritmo scelga l'access point con ritardo maggiore (entro la soglia di 150ms di latenza massima) ma con minore percentuale di perdita di pacchetti. I risultati ottenuti dalle prove simulative sono rappresentati dai seguenti grafici:

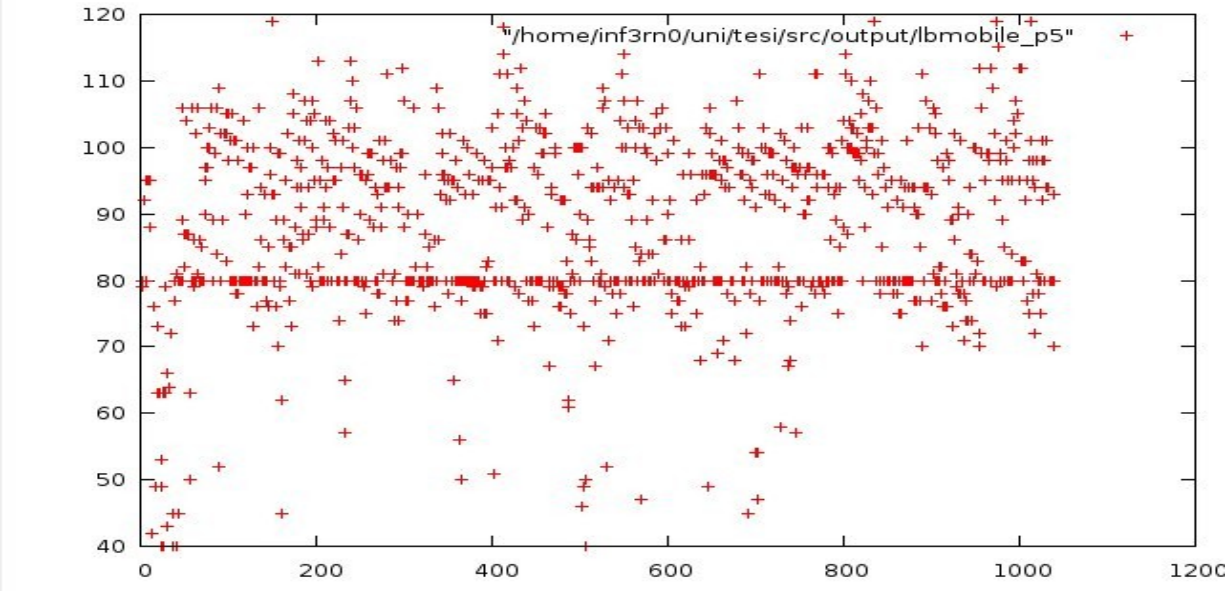

Figura 5.3: Grafico sui risultati ottenuti dai test inerenti il lato mobile

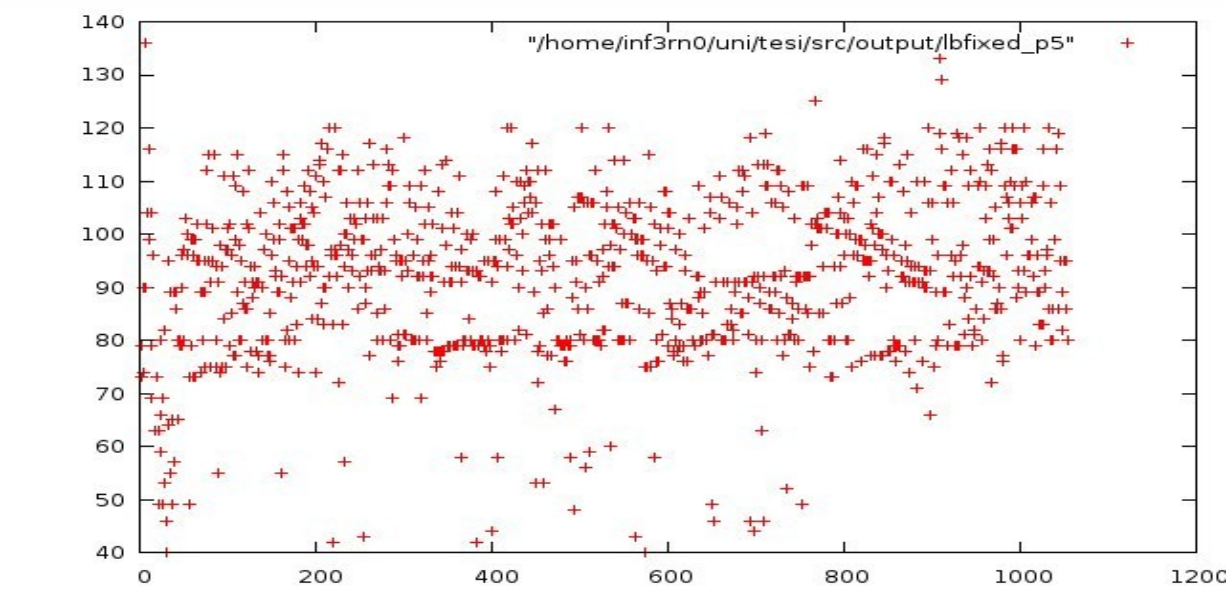

Figura 5.4: Grafico sui risultati ottenuti dai test inerenti il lato fisso

I risultati sull'andamento complessivo della trasmissione sono abbastanza soddisfacenti considerando che il modulo che simula la rete ha restituito i seguenti dati a fine comunicazione:

```
Channel{
     cid:0
     drop%:2
     lost%:2
     delaybase/bound:80000-10000
     socket 1 drop/lost/recv packet:49/41/1159
     socket 2 drop/lost/recv packet:0/44/1070
}
Channel{
     cid:1
     drop%:2
     lost%:10
     delaybase/bound:50000-10000
     socket 1 drop/lost/recv packet:6/11/122
     socket 2 drop/lost/recv packet:0/14/110
}
```

La scelta del canale è stata eseguita con successo in quanto il bilanciatore ha deciso che era meglio inoltrare i pacchetti su un canale più lento ma con una probabilità minore di perdita durante il trasposto all'endpoint opposto.

Capitolo 6

Sviluppi Futuri e Conclusioni

Il sistema di bilanciamento di carico sviluppato può essere migliorato e modificato sotto diversi aspetti in previsione anche dei nuovi protocolli che porteranno diversi cambiamenti alla normale trasmissione dei dati. La nuova versione IPv6 del noto protocollo IP porterà a cambiamenti sostanziali nella scelta dell'access point e nella formattazione dei dati sopratutto per quanto riguarda le nuove modifiche che andranno apportate al kernel per estrapolare le informazioni che verrano poi utilizzate dal bilanciatore di carico.
Inoltre si potrebbe pensare di espandere ulteriormente l'architettura considerata per permettere la comunicazione tra due client mobili o due client fissi visto che il nostro sistema prevedeva due elaboratori appartenenti alle due differenti categorie. Un altro aspetto importante potrebbe essere la modifica del protocollo ideato, si potrebbe infatti tenere conto di altre informazioni come la posizione geografica, il traffico assegnato ad un determinato access point e altri fattori simili.

6.1 IPv6

L'ICANN rese disponibile il protocollo IPv6 sui root server DNS dal 20 luglio 2004, ma solo dal 4 febbraio 2008 iniziò l'inserimento dei primi indirizzi IPv6 nel sistema di risoluzione dei nomi. Si prevede che il protocollo IPv4 verrà utilizzato fino al 2025 circa, per dare il tempo necessario ad adeguarsi e correggere gli eventuali errori. Oltre a rispondere all'esigenza dell'aumento dello spazio di indirizzamento, l'IPv6 incorpora alcuni protocolli che prima erano separati, come l'ARP, ed è in grado di impostare automaticamente alcuni parametri di configurazione della rete, come per esempio il default gateway. Inoltre supporta nativamente il QOS e introduce l'indirizzamento anycast, che permette ad un computer in rete di raggiungere automaticamente il server disponibile di un dato tipo (un DNS, per esempio) più vicino anche senza conoscerne a priori l'indirizzo. Per quanto riguarda il miglioramento del servizio, le principali migliorie sono:

- header di lunghezza fissa (40 byte)
- pacchetti non frammentabili dai router
- eliminazione del campo checksum

Queste tre novità alleggeriscono il lavoro dei router, migliorando l'instradamento e il throughput. IPv6 è la seconda versione dell'Internet Protocol ad essere ampiamente sviluppata, e costituirà la base per la futura espansione di Internet. Proprio per questo motivo un argomento molto interessante riguarda il porting da IPv4 a IPV6 volto ad aggiornare le modifiche effettuate al Kernel GNU Linux in modo tale da poter usufruire delle informazioni inerenti ad un invio avvenuto con successo durante la trasmissione.

6.2 Architettura

Come scritto precedentemente l'architettura del sistema preso in considerazione dovrà considerare come sviluppo futuro il supporto per comunicazioni tra due dispositivi mobili e due dispositivi fissi. La seguente figura rappresenta un ipotetico scenario con più elaboratori in comunicazione:

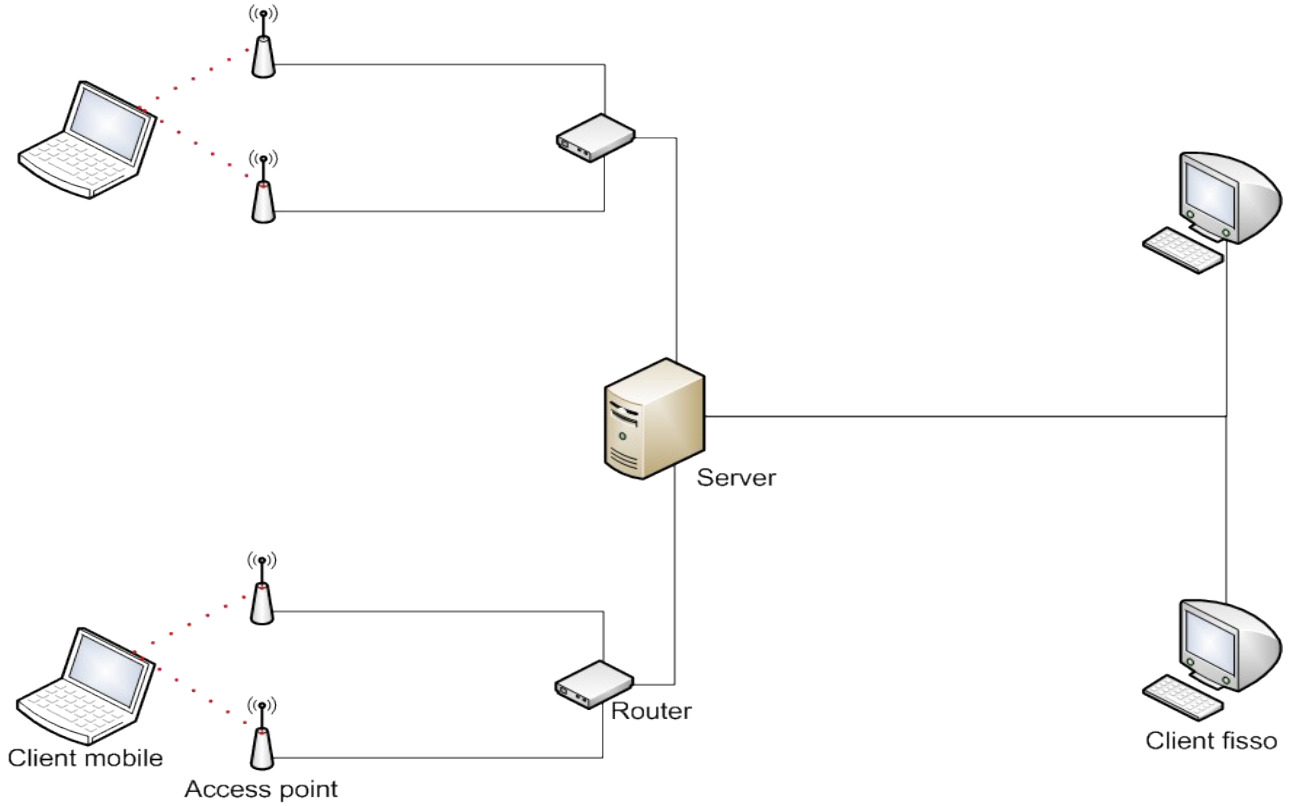

Figura 6.1: Ipotetico scenario inerente ad un possibile cambio di architettura.

Determinare il tipo di dispositivo utilizzato è molto importante in quanto potrebbe facilitare e rendere più affidabili le comunicazioni. Infatti se pensiamo a due clients mobili che vogliono stabilire una trasmissione VoIP sfruttando il protocollo UDP, entrambi in continuo movimento all'interno di un ambiente, sorge spontaneo il problema del continuo cambio di indirizzo IP dovuto alla presenza di nuovi access point durante il percorso effettuato. Per risolvere questa problematica si potrebbe ad esempio utilizzare un server centrale (in grado di interpretare le strutture dati utilizzate dai bilanciatori di carico) che tenga traccia degli attuali indirizzi disponibili associati ad un dato host. Per quanto riguarda la connessione tra due clients fissi si potrebbe pensare di cambiare totalmente protocollo e quindi utilizzare TCP che anche se meno leggero di UDP garantirebbe la consegna effettiva e l'ordinamento dei pacchetti spediti. Un fattore che inoltre dovrebbe essere valutato nel cambio dell'architettura riguarda la latenza che fin ora è stata sempre considerata secondo l'ipotesi che l'elaboratore che riceve i dati è lo stesso che effettua la chiamata. Nello scenario futuro probabilmente andrebbero fatte delle valutazioni per stabilire le tempistiche per ogni segmento di cui l'instradamento è composto e di conseguenza ideare un meccanismo in grado di analizzare la latenza di ognuno di essi.

6.3 Conclusioni

Il panorama attuale vede le comunicazioni in continua evoluzione. Le reti wireless sono in aumento sul territorio e la tecnologia VoIP viene impiegata in molte applicazioni pubbliche e private. L' utilizzo combinato di questi due mezzi è complicato dal problema dell' elevata perdita di pacchetti insita nelle reti Wi-Fi e l' utilizzo di un protocollo inaffidabile come UDP da parte del VoIP. Il nostro scenario ipotetico rappresenta la trasmissione che avviene tra due dispositivi, uno mobile connesso a più AP con la possibilità di spostamento nello spazio senza perdita di connettività a patto di copertura del percorso, uno fisso con indirizzo IP statico. Questa comunicazione basata sullo scambio di pacchetti è mediata dall' AP che ogni volta che riceve dei dati dal lato mobile provvede a comunicargli l' avvenuta ricezione tramite notifica (Acknowledgement).
Il lavoro svolto è stato mirato a creare un sistema di bilanciamento di carico in grado di stabilire quale è l'AP avente le condizioni migliori al quale inviare i pacchetti. Per determinare il mezzo di comunicazione più sicuro si è sviluppato un algoritmo che classifica i canali secondo velocità e quantità di pacchetti persi sia sull'intero tragitto che dal dispositivo mobile all'AP. L' utilizzo di un Kernel GNU Linux modificato a livello di stack di rete permette di recuperare le notifiche che l'AP invia come risposta alla ricezione di dati da inoltrare al lato fisso, in questo modo la comunicazione gode di un'interattività migliore.
Il meccanismo di acknowledgement implementato a basso livello è affiancato da quello di Heartbeat che è attualmente utilizzato nei moderni bilanciatori di carico per stabilire il corretto funzionamento di un nodo all'interno del cluster usato per il trasporto dati.
Nel nostro caso si è deciso di aggiungere all'Heartbeat la quantità di pacchetti ricevuta fino a quel momento su un dato canale affinchè si avesse un ulteriore informazione da considerare nel classificare gli AP. L'altro fattore tenuto in considerazione dal bilanciatore di carico è la latenza calcolata grazie al supporto del protocollo NTP che permette di sincronizzare gli orologi di sistema dei due dispositivi della trasmissione. Alla ricezione di ogni pacchetto su entrambi gli endpoints si procede calcolando la quantità di tempo passata e in seguito viene aggiornata la latenza del canale considerato. Il lavoro svolto ha generato un modulo da integrare all'interno di un sistema completo utilizzato sui dispositivi della comunicazione (in particolare il lato mobile) che sfruttii il bilanciatore di carico come strumento per la trasmissione dei dati. I risultati simulativi forniscono inoltre alcuni semplici e indicativi risultati circa il comportamento e il possibile utilizzo del sistema così creato.
Il bilanciamento di carico è un problema presente fin dall'inizio nell’ambito delle reti e solitamente è risolto tramite la suddivisione del carico su più elaboratori. Nel nostro caso però le unità su cui dividere il carico sono gli AP e quindi si è in una situazione particolare rispetto al caso generale. In futuro sarà possibile migliorare sotto molti aspetti la tecnologia sviluppata secondo il profilo della sicurezza, stabilità e velocità della trasmissione grazie all’evoluzione degli standard attuali, con la possibilità di integrare il bilanciatore in scenari differenti con la presenza di numerosi dispositivi mobili in comunicazione.

Bibliografia

[1] H. Koga, S. Kashihara, Y. Fukuda, K. Iida, Y. Oie, “A quality-aware VoWLAN architecture and its quantitative evaluations”, in *Special Issue on Voice over Wireless Local Area Network, IEEE Wireless Communications*, 13(1), February 2006.

[2] E. Ziouva, T. Antonakopoulos, “A dynamically adaptable polling scheme for voice support in IEEE802.11 networks”, in *Computer Communications* 26(2), pp. 129-142, February 2003.

[3] L. Qiu, P. Bahl, A. Adya, “The Effect of First Hop Wireless Bandwidth Allocation on End-to-End Network Performance”, *in Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, Miami, Florida, USA, pp. 85-93, 2002.

[4] IEEE Std. 802.11b-1999, “Higher-Speed Physical Layer (PHY) extension in the 2.4 GHz band,” IEEE Standard for Information Technology, 1999.

[5] IEEE Std. 802.11g-2003, “Further Higher-Speed Physical Layer Extension in the 2.4 GHz Band”, IEEE Standard for Information Technology, 2003.

[6] IEEE Std. 802.11n-2007, “Higher Throughput Improvements using MIMO”, IEEE Standard for Information Technology, 2007.

[7] IEEE Standard for Local and metropolitan area networks, “Corrigendum 1 Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE Std 802.16e-2005”, 28 Feb 2006.

[8] IEEE Standards Association, IEEE 802.11e, available at http://standards.ieee.org/getieee802/download/802.11e-2005.pdf, 2008.

[9] IEEE, “Media Independent Handover”, IEEE Draft Standard 802.21, 2008, work in progress.

[10] FONERA, http://www.fon.com/, 2008.

[11] ITU-T Recommendation G.114, “One-way Transmission Time”, May 2003.

[12] Network Working Group, “RTP: A Transport Protocol for Real-Time Applications”, July 2003.

[13] A. da Conceicao, L. Jin, D. A. Florencio, F. Kon, “Is IEEE 802.11 ready for VoIP?|”, *in Proceedings of 8th Workshop on Multimedia Signal Processing*, October 2006.

[14] K. Hwangnam, J.C. Hou, “Improving protocol capacity for UDP/TCP traffic with model-based frame scheduling in IEEE 802.11-operated WLANs”, *in IEEE Journal in*

Selected Areas in Communications, 22(10), ISSN: 0733-8716, pp. 1987-2003, December 2004.

[15] P. Wang, W. Zhuang, "A Token-Based Scheduling Scheme for WLANs Supporting Voice/Data Traffic and its Performance Analysis", in *IEEE Transactions Wireless Communications*, 7(4), April 2008.

[16] V. Devarapalli et al., "Network Mobility (NEMO) Basic Support Protocol," IETF RFC 3963, Jan. 2005.

[17] M. Luoto, T. Sutinen, Cross-Layer Enhanced Mobility Management in Heterogeneous Networks, Proc. of ICC '08. IEEE International Conference on Communications, pp. 2277-2281, May 2008.

[18] F. Galan Marquez, M. Gomez Rodriguez, T. Robles Valladares, T. de Miguel, L.A. Galindo, "Internetworking of IP Multimedia Core Networks between 3GPP and WLAN," IEEE Wireless Comm. Magazine, vol. 12, no. 3, pp. 58-65, June 2005.

[19] H. Fathi, R. Prasad, and S. Chakraborty, "Mobility Management for VoIP in 3G Systems: Evaluation of Low-Latency Handoff Schemes", IEEE Wireless Comm. Magazine, vol. 12, no. 2, pp. 96-104, Apr. 2005.

[20] M. Bernaschi, F. Cacace, G. Iannello, A. Pescape`, and S. Za, "Seamless Internetwoking of WLANs and Cellular Networks: Architecture and Performance Issues in a MobileIPv6 Scenario," IEEE Wireless Comm. Magazine, vol. 12, no. 3, pp. 73-80, June 2005.

[21] S. Gundavelli et al., "Proxy Mobile IPv6," draft-ietfnetlmm-proxymip6-01.txt, IETF, June 2007, work in progress.

[22] R. Moskowitz and P. Nikander, "Host Identity Protocol (HIP) Architecture," IETF RFC 4423, May 2006.

[23] Wedlund E., Schulzrinne H., "Mobility Support using SIP," Proc. 2nd ACM Int'l. Wksp. Wireless Mobile Multimedia, Seattle, WA, Aug. 1999.

[24] 3GPP, "Feasibility Study on 3GPP System to Wireless Local Area Network (WLAN) Internetworking," Technical Report TR 22.934, v. 6.2.0, Sept. 2003.

[25] 3GPP, "IP Multimedia Subsytem (IMS); Stage 2," Technical Report, TS 23.228, Mar. 2006.