UNIVERSITÀ DEGLI STUDI DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

Corso di Laurea in Scienze dell'Informazione

Sede di Bologna

BoAT,
UN TOOL SOFTWARE PER SUPPORTARE VOCE A PACCHETTI SU INTERNET: CONTROLLO ADATTIVO DEL BIT RATE

Tesi di Laurea in:

Sistemi per l'Elaborazione dell'Informazione I

Presentata da: Relatore Chiar.mo:

MASSIMILIANO QUIETI *Prof. MARCO ROCCETTI*

Correlatore:

Dott. VITTORIO GHINI

Sessione Prima

ANNO ACCADEMICO 1998 - 1999

Indice

Capitolo 1

Introduzione e obiettivi della tesi

Negli ultimi due decenni, l'informatica ha subìto una notevole accelerazione nel suo processo di evoluzione. Fino ai primi anni '80, il computer era visto come un oggetto, il cui utilizzo era quasi esclusivamente di pertinenza degli ambienti scientifici e matematici. I calcolatori erano di grosse dimensioni e i costi, li rendevano accessibili solamente ad aziende e a centri universitari e governativi. Negli anni successivi, l'evoluzione tecnologica, ha portato ad un abbattimento dei costi di produzione e ad una notevole diminuzione delle dimensioni dei calcolatori, si è passati dai grandi mainframe, ai piccoli personal computer. Il PC è diventato in breve tempo, uno strumento di uso comune (come può esserlo la TV e il telefono) e attorno ad esso, sono nati e si sono sviluppati una serie infinita di servizi e strumenti.

L'esplosione di Internet ha dato lo spunto decisivo, al processo di diffusione dell'informatica. La possibilità, per chiunque, di poter inviare e ricevere messaggi e documenti, ha incrementato notevolmente la popolarità dei computer, sia per un utilizzo per così dire "domestico", sia per lavoro.

Oltre ai servizi più comuni come la posta elettronica, le reti di computer sono oggi utilizzate anche per trasmettere audio e video. Le potenze di calcolo delle nuove CPU e la disponibilità di una banda di trasmissione sempre più ampia, rendono possibile l'utilizzo di applicazioni distribuite per la trasmissione di dati multimediali.

Accanto a questo tipo di applicazioni, sta nascendo tutta una serie di nuove esigenze, che debbono essere prese in considerazione per sfruttare al meglio questo tipo di servizio. Al contrario di applicazioni tradizionali come la posta elettronica, quelle di tipo multimediale, sono soggette a vincoli di funzionamento dipendenti dal tempo, vengono perciò definite di

tipo *real-time*. Per applicazioni di questo tipo, si è costatato che il ritardo sperimentato dai dati nell'attraversamento della rete, non deve essere superiore ad alcune centinaia di millisecondi, pena la perdita di interattività nella comunicazione [MKT 98].

Le suddette applicazioni che utilizzano Internet per la trasmissione, si affidano al protocollo IP (Internet Protocol), il quale è un servizio di tipo *best effort*; ossia non è fornita alcuna garanzia, per quanto riguarda i ritardi di trasmissione e per eventuali perdite di pacchetti. Con un servizio di tipo best effort, se una connessione è libera da congestione, i pacchetti non vengono scartati e i ritardi accumulati dipendono esclusivamente, dal tempo di propagazione del segnale e dai ritardi di serializzazione accumulati al passaggio in ogni router.

Al contrario, su una connessione nella quale è presente una congestione rilevante, i pacchetti risentono dei ritardi di accodamento all'interno dei router, di conseguenza, aumentano i ritardi end-to-end di trasmissione e, nel momento in cui le code dei router sono piene, i pacchetti vengono scartati.

In uno scenario di questo tipo, la trasmissione di audio ne risente notevolmente in termini di qualità sonora.

Si possono focalizzare quindi due aspetti fondamentali, dai quali dipende la qualità di una comunicazione audio:

- la quantità dei pacchetti persi sulla rete
- i ritardi sperimentati dai pacchetti stessi durante la trasmissione

Il primo aspetto compromette l'intelligibilità, ossia la comprensione delle frasi; il secondo impedisce l'interattività della comunicazione. E' stato dimostrato attraverso studi approfonditi, che ritardi superiori ai 400-500 ms., impediscono l'interattività della conversazione e che le percentuali di perdite, non debbono superare il 9-10%, pena una diminuzione nell'intelligibilità della conversazione stessa [Bol 93].

Un'altra complicazione che rende lo scenario ancora più instabile, deriva dal fatto che le condizioni della connessione, non sono prevedibili a priori e

possono variare molto velocemente, a causa della presenza di altre connessioni.

Problemi di questo tipo come abbiamo detto, deteriorano la qualità dell'audio e debbono quindi essere risolti, se si vuole veramente offrire un servizio di telefonia su Internet.

Le soluzioni sono diverse: un primo approccio potrebbe essere quello di modificare i servizi offerti dalla rete, ad esempio cambiando la politica dei router, passando da una gestione delle code di tipo FIFO, ad una strategia che distingue tra i vari flussi di traffico dei pacchetti. In questo modo, si potrebbero privilegiare i flussi di traffico nei quali il tempo è un fattore importante.

Un esempio in questa direzione, è la definizione del protocollo RSVP [Zha 93], che è stato standardizzato, con l'obiettivo di fornire una classificazione e identificazione dei flussi di dati. RSVP chiede ai router di catalogare i pacchetti in classi, per ottenere le risorse necessarie [HCBO 99]. In questo modo, alle applicazioni viene concesso il tipo di servizio richiesto. Lo svantaggio di servizi di questo tipo, è che implicano il cambiamento dell'architettura della rete stessa.

Un secondo approccio che permette di trovare una soluzione, senza dover modificare i servizi offerti dalla rete, consiste nel fare in modo che sia l'applicazione stessa, a compensare gli effetti indesiderati di un servizio di tipo best effort. In pratica, ciò consiste nel progettare l'applicazione in maniera tale, che si adatti continuamente allo stato della connessione, tramite la quale avviene l'invio dei pacchetti.

L'applicazione audio da noi sviluppata, si propone di limitare gli svantaggi derivanti da un servizio di tipo best effort qual è IP, attraverso l'implementazione di meccanismi, che permettono di adattare il comportamento dell'applicazione stessa alle variazioni della rete, con l'obiettivo di minimizzare l'incidenza negativa delle perdite e dei ritardi sulla qualità dell'audio.

Un meccanismo adattivo per il controllo del tempo di playout, permette che l'applicazione si adatti in modo dinamico ai ritardi di rete, utilizzando i

periodi di silenzio per sondare lo stato di congestione del traffico, durante i quali settare dinamicamente i ritardi di playout.

Un meccanismo per il controllo adattivo del bit rate permette, tramite un codec adattivo, di variare il data rate di trasmissione in base allo stato della rete, in modo da garantire sempre la migliore qualità uditiva possibile e minimizzare i pacchetti persi. Nei capitoli successivi i due meccanismi sopra menzionati, verranno esaminati in maniera più approfondita.

Un altro aspetto che deve essere preso brevemente in considerazione, per concludere, riguarda il supporto real-time fornito dal sistema operativo alle applicazioni. I sistemi operativi di tipo *general purpouse* come Linux, sono progettati per gestire l'esecuzione di applicazione di ogni genere. Questo comporta che le applicazioni di tipo real-time, che hanno la necessità di andare in esecuzione periodicamente (nell'ordine di qualche decina di millisecondi), vengono invece schedulate in modo concorrente, come le normali applicazioni non real-time (ad esempio editor di testo, compilatori ecc.). Questa gestione dei processi, contribuisce ulteriormente a deteriorare le prestazioni di un'applicazione audio distribuita. E' stato dimostrato che i ritardi introdotti dal sistema operativo, sono equivalenti a quelli di rete.

Vedremo in seguito, che per limitare questo aspetto negativo, si è fatto uso di un server (sviluppato durante un precedente lavoro di tesi), il cui compito è quello di permettere, all'applicazione che ne fa richiesta, l'utilizzo di una percentuale maggiore di CPU, allo scopo di limitare il più possibile i ritardi introdotti dal sistema operativo, su cui viene lanciata l'applicazione audio distribuita.

Questa tesi ha come obiettivo quello di illustrare la progettazione, la realizzazione e la valutazione sperimentale del meccanismo di controllo adattivo del bit rate, utilizzato all'interno dell'applicazione audio. Il meccanismo, si propone di regolare la quantità di bit utilizzati per la codifica delle informazioni audio, in base allo stato della rete, in modo da garantire, in ogni istante, la miglio qualità uditiva possibile.

Possiamo riassumere, attraverso la figura 1-1, la struttura dell'applicazione audio da noi sviluppata, illustrando le componenti

principali del tool. In grigio sono evidenziate le parti che verranno trattate approfonditamente, all'interno di questa tesi, in capitoli successivi; mentre le parti rimanenti sono trattate in [Bal 99].

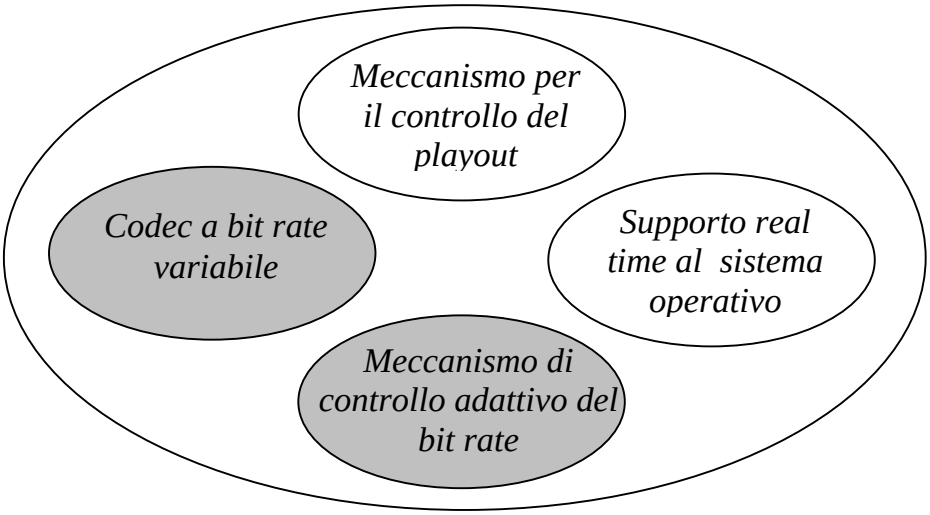

Figura 1-1: Le componenti principali che costituiscono l'applicazione BoAT.

Concludiamo il primo capitolo, riassumendo, attraverso il grafico di figura 1-2, i risultati ottenuti grazie all'utilizzo del nostro sistema di controllo del bit rate. Nel grafico si confronta l'andamento del bit rate regolato dal nostro meccanismo adottato in BoAT, con quello sviluppato dall'INRIA in Free Phone. Senza scendere nel dettaglio (una più ampia analisi del meccanismo verrà fatta nei capitoli quattro e cinque), possiamo ugualmente notare come il nostro sistema sia incline ad avere oscillazioni meno frequenti, rispetto al sistema di Free Phone. L'andamento più stabile e le variazioni meno repentine nel bit rate, incidono in misura minore sulla qualità dell'audio. Il nostro sistema di controllo tende ad utilizzare la banda disponibile con una maggiore cautela, sfruttando sempre una quantità di banda minore rispetto a Free Phone. Il bit rate con il nostro meccanismo si attesta sui 43 Kbit/sec, mentre Free Phone raggiunge i 49 Kbit/sec. La minore quantità di byte spediti si traduce tuttavia, in una sensibile diminuzione delle perdite, dovute ai periodi di elevato carico della rete. Il nostro sistema, infatti, tende a reagire prima agli stati di congestione della rete, abbassando il bit rate, e anche quando si tratta di ripristinare il rate di

trasmissione, nei momenti in cui la rete migliora, si aumenta il bit rate con maggiore cautela. Nell'esperimento mostrato in figura 1-2, le perdite nel caso di Free Phone, si attestano al 10,5% circa, mentre con il nostro meccanismo, le perdite sono limitate al 6,3%, con evidenti benefici per la qualità della conversazione.

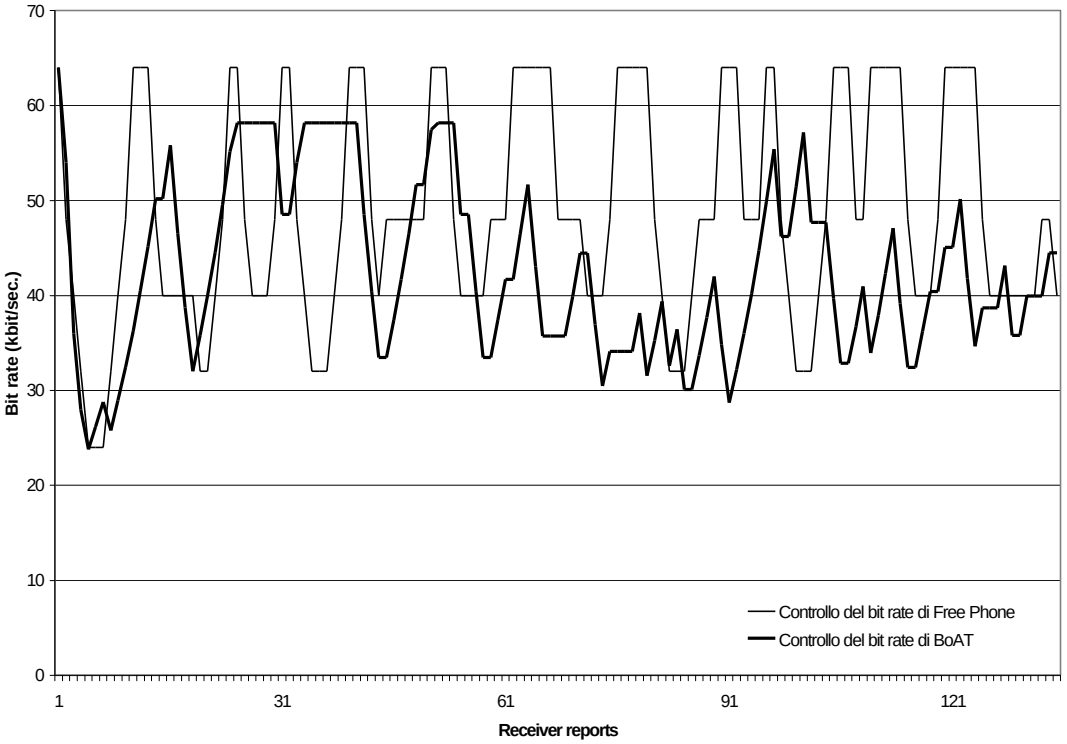

Figura 1-2: Meccanismo di controllo del bit rate. Confronto tra BoAT e Free Phone.

In questo capitolo abbiamo semplicemente voluto, accennare ai risultati ottenuti, grazie all'utilizzo di un nuovo meccanismo di controllo adattivo del bit rate, sviluppato nell'applicazione BoAT. Nei capitoli successivi, analizzeremo in modo più rigoroso ed approfondito il sistema di controllo, giustificando tutte le scelte di progettazione effettuate e mostrando i risultati ottenuti.

Struttura della tesi:

Le tesi è organizzata nel seguente modo:

Nel secondo capitolo verrà prima fatta una breve panoramica sullo stato dell'arte degli audio tool attualmente in circolazione, poi verranno discusse alcune problematiche relative alla trasmissione di audio su Internet. Essendo il capitolo prettamente introduttivo, il lettore può esimersi dalla lettura, qualora sia già a conoscenza delle problematiche e dei vincoli, relativi ad un'applicazione audio.

Nel terzo capitolo, si illustrerà l'applicazione audio sviluppata, analizzandone la struttura e focalizzando l'attenzione sui punti più importanti, che verranno poi approfonditi nei capitoli successivi.

Nel quarto e nel quinto capitolo, si analizzerà il meccanismo per il controllo adattivo del bit rate in maniera approfondita, sia dal punto di vista teorico, sia per quanto concerne la sperimentazione.

Il capitolo sesto concluderà questo lavoro di tesi. Verranno riassunti i risultati ottenuti e si forniranno dei suggerimenti per sviluppi futuri.

Capitolo 2

Stato dell’arte della trasmissione audio

In questo capitolo si analizzano prima gli aspetti relativi alla trasmissione di audio su reti a commutazione di pacchetto, dopo di che si fornisce una panoramica sullo stato dell’arte delle applicazioni audio attualmente in circolazione. Analizziamo inoltre, le tecniche più importanti relative alla codifica dell’audio e due protocolli recenti che sono utilizzati nella trasmissione audio, al fine di migliorarne le prestazioni: *IP Multicast* e *RTP*. In ultimo analizziamo le problematiche e le relative soluzioni della trasmissione audio su Internet.

2.1 La trasmissione audio

Abbiamo già avuto modo di dire nel capitolo precedente, che i due fattori che influenzano maggiormente una comunicazione audio su Internet, sono i ritardi e le perdite. Tuttavia, esistono altri aspetti importanti, su cui vale la pena prestare attenzione. Iniziamo illustrando la struttura di un’applicazione audio generica.

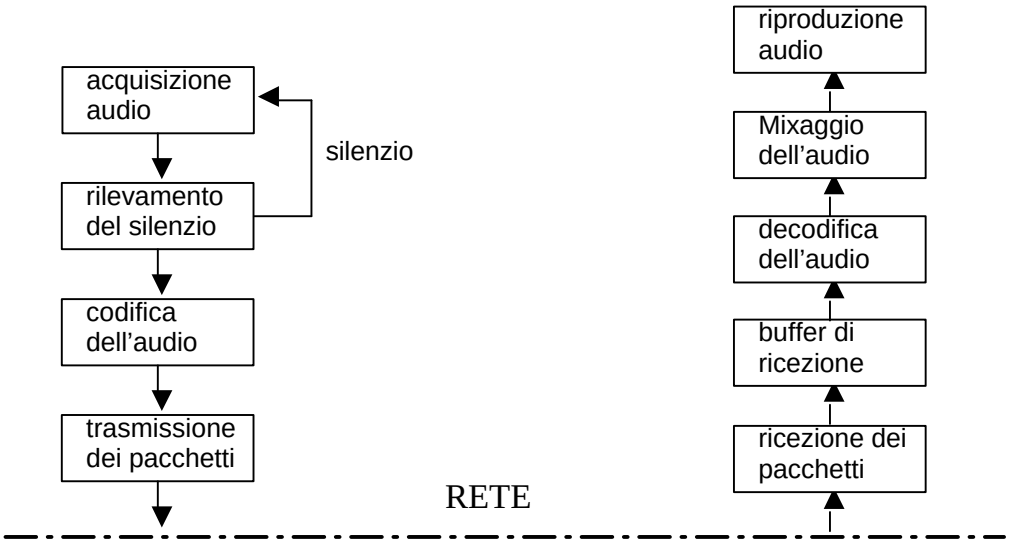

Figura 2-1: Struttura generale di un'applicazione audio.

Un'applicazione audio (come si evince dalla Figura 2-1), può essere suddivisa in due parti ben distinte: una prima parte (a sinistra in figura), svolge la funzione di acquisizione della voce e trasmissione; una seconda (a destra in figura), svolge la funzione di ricezione della voce ed emissione tramite la scheda audio. Analizziamo ora più approfonditamente le due parti.

2.1.1 Le fasi di spedizione dell'audio

Vediamo le varie fasi che portano dall'acquisizione dell'audio, alla trasmissione sulla rete.

Acquisizione dell'audio

Questa fase, avviene tramite la lettura dei campioni messi a disposizione dalla scheda audio, che utilizza una tecnica di campionamento e discretizzazione del segnale elettrico, proveniente in genere dal microfono. La lettura dei campioni è eseguita ad intervalli periodici (in genere ogni 20-40 ms.).

Rilevamento del silenzio

Il compito svolto da questa fase, è quello di rilevare i periodi di silenzio all'interno di una conversazione. Il rilevamento può avvenire utilizzando tecniche semplici, che consistono nello stabilire una soglia sotto la quale, ciò che si legge dalla scheda audio è considerato silenzio, oppure utilizzando tecniche più complesse e precise, che eseguono un'accurata analisi, distinguendo tra rumore di fondo e il segnale audio vero e proprio. Un meccanismo di *silence detection* è importante all'interno di un'applicazione audio poiché, evitando di trasmettere i periodi di silenzio, diminuisce l'occupazione di banda. Infatti, esiste una stretta correlazione tra la banda disponibile e ritardi end-to-end, limitando la quantità di informazioni trasmesse, si può avere una diminuzione dei ritardi stessi. E' stato dimostrato attraverso studi approfonditi, che in media durante una conversazione, ogni interlocutore parla solamente per il 40-50% del tempo a disposizione [JEA 97], di conseguenza evitando di trasmettere nei periodi

di silenzio, si può avere un risparmio nell'utilizzo della banda fino al 50-60 per cento.

Un ulteriore fattore, che porta a considerare fondamentale un meccanismo di rilevamento del silenzio, all'interno di un'applicazione audio, consiste nel fatto che molti dei meccanismi per il controllo del playout, utilizzano proprio i periodi di silenzio per regolare dinamicamente il tempo di emissione dell'audio (come vedremo nel capitolo 3, anche il meccanismo da noi implementato sfrutta i periodi di silenzio), per evitare di produrre elongazioni o contrazioni all'interno di parole. Un efficiente silence detector, migliora di conseguenza anche le prestazioni del sistema di controllo del playout.

Codifica dell'audio

La codifica ha lo scopo di comprimere i dati numerici prodotti con la fase precedente, al fine di diminuire il data rate emesso. Nel paragrafo 2.3 verrà fatto un breve studio sulle tecniche di codifica.

Trasmissione dei pacchetti

A questo punto, il flusso audio viene "pacchettizzato" e ogni pacchetto contiene una quantità di audio che può variare da 20, 40 e anche 80 ms. E' importante definire una giusta dimensione della quantità di audio, inserita all'interno di ogni pacchetto: una quantità troppo bassa, comporta la spedizione di un numero maggiore di pacchetti, con il conseguente aumento dell'*overhead* associato all'header del protocollo; d'altronde la spedizione di una quantità di audio troppo elevata, può portare ad un aumento nei ritardi end-to-end e ad un degrado maggiore nell'intelligibilità del parlato, qualora vi siano perdite.

In genere, le applicazioni audio si affidano al protocollo di trasporto UDP (*User Datagram Protocol*) [UDP] per la trasmissione delle informazioni, in quanto il protocollo TCP (*Transfer Control Protocol*) [TCP] è si più affidabile, ma questa affidabilità la si paga in termini di aumento dei ritardi, dovuti ai numerosi controlli che vengono effettuati sui

pacchetti trasmetti. Esistono tuttavia, anche applicazioni audio che si basano sul TCP; un esempio è Real Audio [RealAudio].

Il protocollo UDP, offre inoltre la possibilità (al contrario di TCP) di effettuare trasmissioni di tipo multipoint che, come vedremo più avanti, permettono di effettuare audioconferenze. Per quanto riguarda tutte le funzionalità che UDP non implementa, come la pacchettizzazione dei dati, il riordino dei pacchetti, la gestione delle perdite ecc., esse vengono lasciate all'applicazione, la quale decide come gestirle.

2.1.2 Le fasi di ricezione dell'audio

Vediamo le varie fasi che portano dalla ricezione dei pacchetti, all'emissione dell'audio.

Ricezione dei pacchetti

La parte di ricezione, svolge il lavoro di raccolta dei pacchetti disponibili sulla rete e di memorizzazione dell'informazione audio, contenuta in essi, all'interno del buffer di ricezione.

Decodifica dell'audio

La decodifica consiste nel decomprimere le informazioni utilizzando l'algoritmo inverso a quello adottato per la codifica. Generalmente le informazioni audio ottenute dopo la decodifica, hanno una qualità inferiore rispetto alle originali prima della codifica. Maggiore è la compressione, maggiore è la perdita di qualità.

Mixaggio

Nel caso in cui l'applicazione supporti il multicast, è possibile ricevere informazioni provenienti da sorgenti sonore diverse. Compito di questo modulo è di gestire le diverse sorgenti audio che arrivano al ricevente, come un unico flusso di informazioni.

Riproduzione dell'audio

Questa è la fase conclusiva di tutto il processo, le informazioni sono inviate alla scheda audio, che provvederà a trasformarle in segnali elettrici e

ad inviarle al canale di uscita. E' importante che in fase di riproduzione venga mantenuta la stessa periodicità nell'emissione dell'audio, adottata in fase di acquisizione dell'audio dal sender, onde evitare sia di introdurre silenzi artificiali all'interno del discorso, sia di non rispettare i periodi di pausa all'interno della conversazione.

2.2 Le applicazioni multimediali distribuite

L'obiettivo dichiarato, da parte degli sviluppatori di applicazioni di questo tipo, è quello di fornire un servizio in tutto e per tutto simile a quello della telefonia normale, con in più alcuni vantaggi, primo fra tutti i costi nettamente inferiori. Il modo più semplice di fare telefonia su rete a commutazione di pacchetto, è quello denominato servizio *phone-to-phone* che corrisponde ad una normale telefonata fatta tramite Internet. Un altro tipo di utilizzo, se vogliamo più interessante del precedente, è quello della audioconferenza, ossia la possibilità, da parte di più persone, di colloquiare tra loro contemporaneamente.

Possiamo suddividere quindi le applicazioni audio in due grandi categorie: *unicast* e *multicast*.

2.2.1 Applicazioni di tipo Unicast

Sono applicazioni di tipo *point-to-point*, che offrono la possibilità di effettuare una comunicazione telefonica su Internet. Il vantaggio principale di questo tipo di servizio è ovviamente il costo nettamente inferiore rispetto ad una telefonata normale. La possibilità di poter effettuare chiamate dall'altra parte del pianeta, al costo di una telefonata urbana, ha portato molte società di informatica e telecomunicazioni, a sviluppare applicazioni e servizi di questo tipo. A parte i costi, vi sono ad ogni modo altri fattori da tenere in considerazione, se si vuole rendere concorrenziale la telefonia su Internet, nei confronti di quella normale. In primo luogo, i ritardi devono essere minimi per assicurare l'interattività, la comunicazione deve essere di tipo *full-duplex* (entrambi gli interlocutori debbono avere la possibilità di parlare contemporaneamente come in una normale telefonata), ed inoltre, la

qualità della voce deve essere comparabile con quella che si ottiene con le normali linee telefoniche, avvicinandosi il più possibile alla qualità della codifica PCM (*Pulse Code Modulation*).

2.2.2 Applicazioni di tipo Multicast

Applicazioni di questo tipo, permettono di effettuare conferenze multimediali tramite Internet, evitando di utilizzare linee dedicate come ISDN, le quali hanno costi notevolmente più elevati. L'introduzione di *IP Multicast* e *RTP* (vedi par. 2.4 e par. 2.5), hanno reso più agevole, condurre conferenze tra un largo numero di partecipanti. Come nel caso delle applicazioni di tipo Unicast, gli utenti devono percepire con buona qualità l'audio; oltre alle problematiche già riscontrate nelle applicazioni point-to-point, ve ne sono di nuove, come ad esempio la gestione di diverse sorgenti sonore, provenienti da partecipanti diversi.

Negli ultimi anni, sono stati sviluppati diversi tool, che permettono di effettuare trasmissioni audio multicast su Internet, i quali possono essere suddivisi in applicazioni di prima e di seconda generazione. Fra i primi abbiamo:

- VAT (Video Audio Tool) [VAT]: è stato uno dei primi tool a supportare MBone. Sviluppato da Steve McCanne [JacMcc 92].
- NeVoT (Network Voice Terminal) [NeVoT]: è stato sviluppato all'università del Massachusset da Henning Schulzrinne [Sch 92].
- IVS (Interrnet Videoconference System) [IVS]: è un passo avanti rispetto ai due precedenti, in quanto ha introdotto per primo l'utilizzo di RTP.

Tra quelli di seconda generazione i più importanti sono sicuramente:

- RAT (Robust Audio Tool) [RAT]: sviluppato all'*University College of London* da Isidor Kouvelas. Adotta RTP e nuovi meccanismi di controllo di qualità superiore [Per 98].
- Free Phone [FreePhone]: sviluppato da Andres Vega Garcia presso l'INRIA, si propone di migliorare RAT, grazie all'introduzione di nuovi meccanismi di controllo [Gar 96].

L'applicazione da noi sviluppata, è stata messa a confronto durante la fase di sperimentazione con i tool RAT e Free Phone, come vedremo successivamente.

2.3 Metodi di codifica dell'audio

In questo paragrafo forniamo una panoramica sulle varie metodologie di codifica, la quale è una componente fondamentale per la trasmissione di audio su IP. Vi sono due discipline che giocano un ruolo importante [MinMin 98]:

- *Analisi vocale*: è la parte che converte la voce in una forma più adatta ad essere manipolata dai computer e ad essere trasmessa su reti digitali. E' la fase di codifica (*coding*).
- *Sintesi vocale*: è la parte che effettua la riconversione dei dati, da una forma digitale ad una adatta all'uso umano. Queste funzioni, sono sostanzialmente inverse a quelle dell'analisi vocale.

La fase di analisi vocale che permette di trasformare un segnale analogico, in una rappresentazione numerica, si svolge in tre fasi distinte. La prima fase è quella di *campionamento del segnale*, che consiste nel prelevare dei valori dal segnale continuo, ad intervalli di tempo finiti e regolari. La seconda fase effettua la conversione analogico/digitale del segnale. Ad ogni valore d'ampiezza (associato ad ogni campione ricavato dalla fase precedente), viene associato un numero binario, tra quelli che il convertitore ha a disposizione. Questa fase viene detta di *quantizzazione*. Una volta che il segnale è stato trasformato in forma più appropriata, si passa alla fase di *codifica* vera e propria, la quale effettua normalmente anche una compressione.

Analizziamo ora i codificatori di segnale audio, raggruppandoli in tre grandi classi:

- *Waveform codec* o codificatori a forma d'onda.
- *Source codec*, più comunemente detti *vocoder*.
- *Hybrid codec*.

2.3.1 Waveform codec

I codificatori di questo tipo, vengono utilizzati ad elevati bit rate e possono trattare il segnale vocale ad alta qualità. Essi tentano di ricostruire un segnale che abbia la stessa forma d'onda dell'originale. In genere, il segnale vocale analogico viene campionato a *8000 Hz*, secondo la tecnica PCM che, per quanto riguarda la telefonia, è la tecnica più usata. Possiamo effettuare una distinzione tra i vari tipi di codec presenti in questa classe, a seconda della tecnica di quantizzazione utilizzata (fase che mappa valori continui in un numero finito di valori discreti).

La forma più semplice di codifica basata sul sistema PCM, è quella che fa uso di una *quantizzazione uniforme*. Comporta l'utilizzo di 20 bit per campione e si ottiene un data rate di 160 Kbit/sec.

Un data rate di 160 Kbit/sec può essere considerato troppo elevato, di conseguenza si può utilizzare una *quantizzazione logaritmica*, che consente una codifica meno onerosa dal punto di vista dei byte ottenuti. Questa variante, permette di limitare a soli 8 bit la dimensione dei campioni, generando un data rate di 64 Kbit/sec e restituendo ugualmente un segnale non distinguibile dall'originale. La quantizzazione logaritmica è un processo di compressione non lineare, tramite l'utilizzo di una funzione logaritmica. Sono stati definiti negli anni '60 degli standard, che sono ancora sfruttati, per questo tipo di quantizzazione: in America è in uso lo standard *μ-law*, mentre in Europa si adotta lo standard *A-law*.

Per ottenere una ulteriore diminuzione nel data rate, si può fare uso di algoritmi di analisi, per avere così quantizzatori a adattamento dinamico, in base alle variazioni dell'ampiezza del segnale vocale. Le tecniche basate su Pulse Code Modulation che fanno uso di una *quantizzazione adattiva*, vengono definite *APCM*.

Un'ultima classe di codificatori PCM è quella che fa uso di quantizzatori di tipo predittivo. Riuscendo a predire il valore di un campione, da quello precedente, allora l'errore che si commette tra la predizione e l'originale avrà una varianza inferiore, rispetto ai due campioni reali. Si può dunque tentare di quantizzare questo errore, con un numero di bit inferiore rispetto

a quelli che sarebbero necessari per il segnale originale di input. Questo è lo schema che sta alla base dei *Differential PCM* (DPCM).

Diciamo infine, che possono essere utilizzate tecniche adattive, insieme a tecniche di tipo predittivo, in questo caso si parla di codec di tipo *ADPCM*.

2.3.2 Source codec

Le tecniche a forma d'onda viste in precedenza, operano sul dominio del tempo. Un altro approccio consiste nel considerare il dominio delle frequenze. Questo è ciò che fanno i Source Codec (o Vocoder).

I Vocoder sono utili nel caso in cui il segnale da riprodurre sia sempre la voce umana. La codifica che si ottiene ha un data rate molto basso, ne segue che anche la qualità della voce risulta povera, simile a quella sintetica.

Questo genere di codificatori non cercano la somiglianza tra l'onda del segnale originale e quella ricostruita, ma tentano di riprodurre i suoni in ingresso. Alla sorgente il segnale viene analizzato per ricavare i parametri del sistema lineare e per ricavare la funzione cosi detta di *eccitazione* che simula il tratto vocale; questi dati vengono poi inviati al decoder che provvede poi alla vera e propria fase di sintesi.

Si ottiene un parlato intelligibile con bassissimi bit rate, sebbene, come abbiamo detto, la qualità di voce che si ricava non presenta sonorità naturali. Uno dei vocoder più usati è il *Linear Predictive Coder* (LPC), il quale ha un data rate di 5,6 Kbit/sec.

2.3.3 Hybrid codec

I codificatori ibridi tentano di combinare le tecniche waveform, con quelle di sintesi per ottenere prestazioni intermedie; qualità della voce discreta, data rate compresi tra 8 e 30 Kbit/sec. Un codec di questo tipo, standardizzato negli Stati Uniti è il *Codebook Excited Linear Prediction* (CELP). Il parlato viene prima sottoposto al predittore del tratto vocale, poi al predittore di tono. Il CELP è in grado di generare audio di qualità telefonica con soli 4,8 Kbit/sec, ha tuttavia lo svantaggio di avere un'alta complessità, quindi richiede macchine molto potenti per essere adottato in

applicazioni real-time. Concludiamo il paragrafo con la tabella 2-1 che riassume i principali codec ed il relativo data rate.

CODEC	***Data Rate (kbit/sec)***
Lineare a 16 bit	128
PCM	64
ADPCM6	48
ADPCM5	40
Microsoft DVI	32
ADPCM3	24
ADPCM2	16
CELP G728	16
GSM	13,2
CELP G729	8
LPC	5,6

Tabella 2-1: Data rate dei principali codec audio.

2.4 IP Multicast

IP Multicast è un'estensione del protocollo IP che fornisce un efficiente distribuzione dei dati molti a molti su Internet. In figura 2-2 è riassunto uno stack di protocolli per Internet Multimedia.

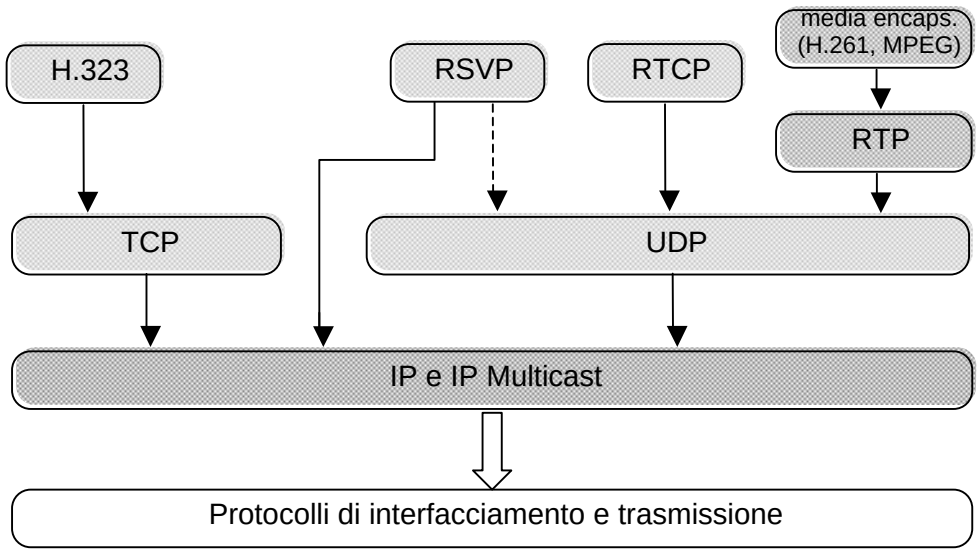

Figura 2-2: Stack di protocolli per Internet Multimedia.

La motivazione principale, che ha portato allo sviluppo di questo nuovo protocollo, è stata quella di fornire la possibilità ad un insieme di utenti di essere considerato come un'entità unica. Possiamo quindi riassumere IP Multicast nel seguente modo:

- tutti i messaggi che debbono essere inviati ai componenti del gruppo verranno inviati al gruppo stesso;
- i membri di un gruppo riceveranno tutti i dati inviati al gruppo;
- sono i router ad avere il compito di gestire la trasmissione delle informazioni a tutti i membri di un gruppo.

Con IP Multicast un gruppo viene indirettamente identificato da un singolo indirizzo denominato *IP class-D multicast address*. La classe degli indirizzi multicast va da *224.0.0.0* a *239.255.255.255*.

Un gruppo IP Multicast è scalabile, poiché le informazioni circa i membri del gruppo e gli eventuali cambiamenti ad esso, sono tenuti a livello locale nei router vicini ai componenti stessi. IP Multicast è la soluzione naturale per la gestione di conferenze su Internet, grazie all'efficienza nella trasmissione dei dati, i quali vengono replicati nei punti appropriati della rete, invece che nei sistemi finali.

Poiché i router attuali non sono abilitati alla gestione diretta di IP Multicast, è stata sviluppata una dorsale denominata *Multicast Backbone* (MBone), che gestisce il traffico di pacchetti di questo tipo. Possiamo considerare MBone come una rete virtuale software, costituita da tante sottoreti, all'interno della quale transita il traffico multicast. Si tratta per lo più di tipologie multicast LAN, tipo Ethernet. Queste sono collegate tramite canali point-to-point denominati *tunnel*. Gli estremi di ogni tunnel, sono in genere macchine utente, dotate di sistema operativo in grado di supportare IP Multicast e capaci di effettuare *multicast routing* tramite un processo demone chiamato *mrouted* [MacBru 94].

2.5 RTP (Real-Time Transport Protocol)

Un protocollo di trasporto di tipo point-to-point, per flussi di dati real-time è RTP [SCFJ 96], sviluppato dall'IETF (*Internet Engineering Task*

Force). Fornisce un formato standard per l'header dei pacchetti, con la possibilità di avere informazioni su tipo dei dati, numerazione dei pacchetti, sincronizzazione *intra-media* e *inter-media* e altro.

Di seguito (Figura 2-3), viene illustrato il formato dell'header di RTP, il quale è normalmente usato insieme al protocollo UDP, senza tuttavia aumentarne l'affidabilità. Ogni sorgente RTP è identificata da un *source id*, il quale è inserito in ogni pacchetto spedito. I *timestamp* inseriti nei pacchetti dipendono dal tipo di dato che viene trasmesso, ad esempio i timestamp dei pacchetti di una trasmissione di audio campionato a 8KHz e codificato PCM, hanno un *clock-rate* di 8KHz.

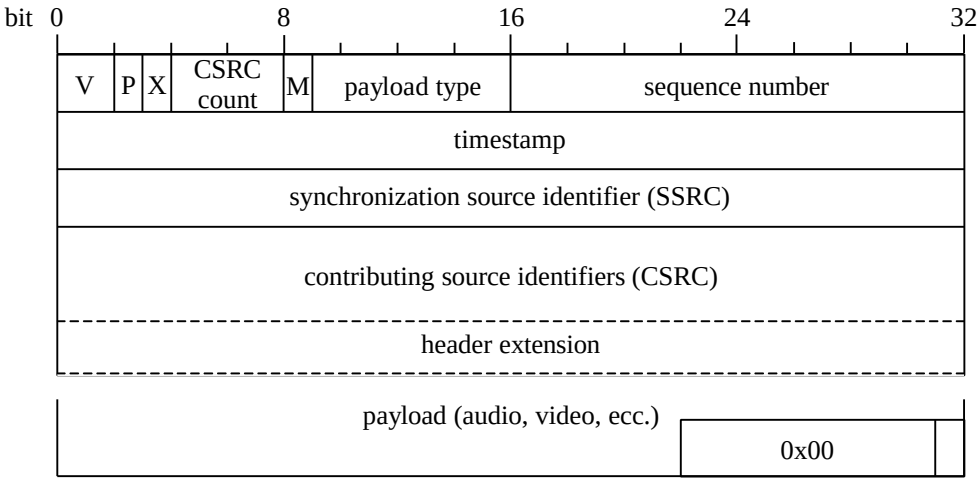

Figura 2-3: Formato header RTP.

Ogni flusso RTP è coadiuvato da pacchetti di tipo RTCP (*Real Time Control Protocol*). I pacchetti RTCP forniscono informazioni di controllo, come ad esempio la relazione tra l'orologio real-time del sender e i timestamp dei pacchetti RTP, per avere così una sincronizzazione tra flussi diversi (ad es. in una trasmissione di audio e video insieme).

RTCP inoltre fornisce periodicamente informazioni sui membri di un gruppo, tramite trasmissioni multicast e informazioni sulla qualità della connessione (percentuale di pacchetti persi, variazione dei ritardi ecc.).

Scendendo più nel particolare vediamo i campi principali dell'header di un pacchetto RTP:

- *Version*: numero di versione del protocollo, attualmente 2.
- *Payload type*: identificatore del tipo di dato trasportato.
- *Sequence number*: permette di riordinare in modo sequenziale i pacchetti RTP e di rilevare quelli persi.
- *Timestamp*: riporta l'istante di campionamento del primo byte dati del pacchetto.
- *Syncronization SouRCe (SSRC)*: identifica in modo univoco il mittente tramite un codice numerico.

I pacchetti RTCP, invece, non contengono dati ma solo informazioni di controllo. Per la gestione di questi pacchetti, il protocollo prevede l'utilizzo di una porta diversa da quella dei dati. Poiché i pacchetti RTCP vengono immessi sullo stesso canale dei dati, è opportuno non spedirne una quantità esagerata, IETF consiglia di utilizzare per i pacchetti di controllo il 5% della banda richiesta dall'applicazione. Vi sono differenti tipi di pacchetti RTCP, vediamo i più importanti:

- *Source Description (SDES)*: contengono informazioni come nome, indirizzo, e-mail e telefono del mittente.
- *Sender Report (SR)*: sono generati dagli utenti che spediscono audio e video tramite RTP. Descrivono l'ammontare dei dati spediti, forniscono una relazione tra i timestamp dei pacchetti RTP e il tempo assoluto per avere una sincronizzazione tra media diversi.
- *Receiver Report (RR)*: sono spediti da partecipanti ad una sessione RTP che stanno ricevendo dati. Trasportano informazioni statistiche come percentuali di perdite, variazione dei ritardi ecc.
- *Bye*: specifica l'abbandono di una sessione di trasmissione, da parte di un partecipante.

I Receiver Report in particolare, sono importanti poiché danno un riscontro sullo stato della qualità del servizio, permettendo quindi

adattamenti dinamici al data rate, in base allo stato corrente della connessione.

2.6 Problemi e soluzioni per la trasmissione audio su Internet

In questo paragrafo verranno analizzate brevemente, le due problematiche più importanti per ciò che riguarda la trasmissione in tempo reale su una rete a commutazione di pacchetto: i ritardi e le perdite. Come abbiamo già accennato nel primo capitolo, i ritardi sono un fattore cruciale per un'applicazione audio; abbiamo visto che ritardi troppo elevati minano l'interattività. Vi è inoltre un altro fattore collegato ai ritardi che rende difficile lo sviluppo di un'applicazione audio real-time: il *jitter*. Definiamo il jitter come la variazione dei ritardi end-to-end. Definiamo inoltre con il termine *playout delay* l'ammontare di tempo sperimentato dai pacchetti audio, dall'istante di tempo in cui sono generati alla sorgente, all'istante in cui vengono riprodotti alla destinazione [RGPSB 98].

Per un'applicazione real-time, il jitter ha un impatto negativo sulla stretta dipendenza temporale che deve essere mantenuta. Se durante una connessione i ritardi si mantenessero costanti, la destinazione disporrebbe di una sequenza di pacchetti con la stessa periodicità di quella generata alla sorgente e di conseguenza, non vi sarebbe nessun problema nel calcolare il tempo di emissione dell'audio (tempo di playout). Come abbiamo visto, purtroppo, il tipo di servizio fornito da Internet, non garantisce una situazione costante nel tempo: i ritardi variano durante una connessione, di conseguenza, la distribuzione dei tempi di arrivo dei pacchetti non è uniforme.

Le due situazioni sono ben illustrate nella figura 2-4, la quale illustra le operazioni di spedizione effettuate da colui che trasmette l'audio (*sender*) e da colui che lo riceve (*receiver*), in una situazione con ritardi costanti e in una con ritardi variabili.

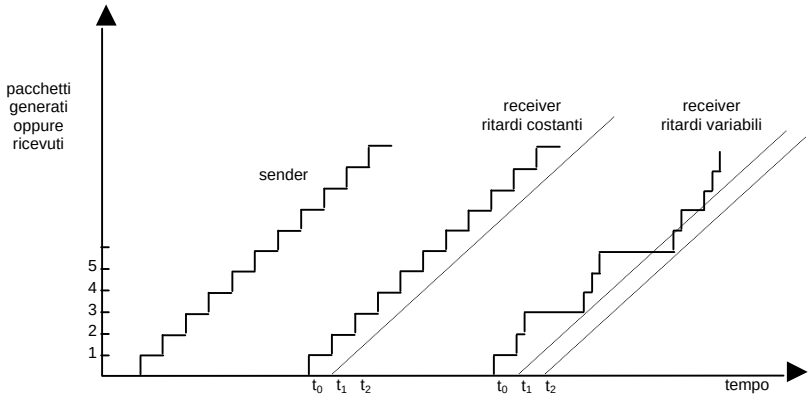

Figura 2-4: Generazione e ricostruzione di voce pacchettizata.

Al sender i pacchetti sono generati come risultato del campionamento periodico della sorgente audio. La regolarità dell'andamento a gradini del sender, indica che i pacchetti sono generati periodicamente alla sorgente. Qualora i pacchetti vengano ricevuti al receiver con ritardi costanti, l'andamento regolare è conservato e di conseguenza, è rispettata anche la periodicità nella riproduzione dell'audio. In questo caso il receiver può iniziare l'emissione dei pacchetti nel momento stesso in cui vengono ricevuti.

Nel caso di ritardi variabili, come si può vedere, l'andamento a gradini non è regolare, ciò significa che i pacchetti ci mettono tempi diversi per arrivare a destinazione, con la conseguenza che al receiver, la periodicità dell'audio non è rispettata. In questo caso si può cercare di attenuare il fenomeno della variazione dei ritardi, iniziando la riproduzione dei pacchetti ad un istante successivo. Ritardare l'emissione al tempo t_1 non è sufficiente in quanto, i pacchetti 6, 7 e 8 saranno ricevuti troppo tardi, a causa del ritardo elevato subìto dal pacchetto 6. L'inizio del playout, quindi, dovrebbe essere ritardato al tempo t_2, ma non vi è garanzia che pacchetti successivi non subiscano ritardi maggiori.

Questa instabilità è dovuta in gran parte alla politica di servizio FIFO utilizzata all'interno dei router. E' stato dimostrato che esiste una correlazione più stretta tra i ritardi e il numero di router che i pacchetti debbono attraversare, rispetto ai ritardi stessi e alla distanza che i pacchetti debbono percorrere [KBSSGM 98]. Ogni router quindi contribuisce con

una quantità *d* al ritardo totale end-to-end sperimentato da un pacchetto. Ossia, se un pacchetto arriva ad un router al tempo *t*, esso verrà ritrasmesso al tempo *t + d*, dove *d* varia in qualsiasi istante a seconda del traffico presente.

Se i ritardi di trasmissione non possono essere tenuti sotto controllo da parte di un'applicazione audio, al contrario la loro variazione può essere controllata. Uno dei compiti fondamentali che devono essere svolti, consiste, infatti, nell'attenuare gli effetti negativi dovuti alla variazione dei ritardi per i pacchetti che vengono ricevuti. In generale al momento della ricezione di un pacchetto, esso non sarà immediatamente riprodotto, ma ritardato di una quantità di tempo variabile. Questo evita il rischio di rimanere privi di audio da riprodurre con la conseguenza di introdurre silenzi artificiali all'interno della conversazione. Infatti, ad ogni istante all'interno del buffer di playout (buffer nel quale vengono inseriti i pacchetti in attesa di essere emessi), si ha un numero di pacchetti consecutivi, che debbono essere riprodotti in modo preciso e temporizzato.

Il meccanismo per il controllo del playout da noi implementato verrà discusso nel prossimo capitolo.

Come si diceva all'inizio del paragrafo, un altro fattore che rende precaria la qualità dell'audio è il numero di pacchetti persi. Le perdite di pacchetti sono causate da due fattori diversi; il primo è dovuto alla rete, il secondo al meccanismo di controllo del playout.

La prima situazione si verifica nel momento in cui la rete deve sopportare un traffico estremamente elevato. In queste situazioni è possibile che i router ricevano più dati di quanti riescano a gestirne, di conseguenza le code dove vengono temporaneamente inseriti i pacchetti in attesa di essere instradati, si saturano ed ogni ulteriore pacchetto ricevuto viene scartato. Va osservato che le macchine che vengono adibite a router, sono fornite di quantità di memoria sempre più elevate e di conseguenza la possibilità che vi siano perdite di questo tipo è al quanto remota.

Il secondo fattore che determina perdite, dipende invece dal meccanismo di controllo del playout. Se un pacchetto arriva a destinazione con un

ritardo troppo elevato, è probabile che venga scartato poiché non è più possibile riprodurlo in tempo, in quanto il ritardo che ha subìto ha superato il playout delay che era stato fissato per quel pacchetto. Anche in questo caso, come nel precedente i ritardi sono dovuti alla rete, ma a decretare la perdita di un pacchetto non è un qualsiasi router attraversato, bensì il meccanismo che regola la riproduzione dell'audio.

Per entrambi questi fenomeni, possono essere sviluppate tecniche all'interno di un'applicazione real-time, che tentano di attenuare l'impatto negativo che le perdite hanno sulla qualità dell'audio. Una prima soluzione consiste nel cercare di utilizzare in ogni istante una quantità di banda che non superi la capacità della connessione, con l'intento di non aumentare e possibilmente ridurre lo stato di congestione della rete. Si tratta quindi di implementare un meccanismo per il controllo del data rate (il meccanismo da noi utilizzato sarà illustrato nel capitolo 4), che in base alle informazioni sullo stato della rete diminuisce la quantità di dati spediti per limitare il numero di pacchetti persi e diminuire i ritardi. E' bene precisare che lo stato di congestione della rete non è determinato unicamente da una singola connessione, di conseguenza anche diminuendo l'occupazione di banda, molto probabilmente nella maggioranza dei casi non si ha un riscontro positivo sullo stato della rete.

Il meccanismo sopra descritto, utilizzato da solo non è sufficiente a ridurre l'impatto acustico che le perdite generano sulla qualità dell'audio percepito dagli utenti. E' necessario introdurre anche un meccanismo di ripristino dei pacchetti persi. I meccanismi più utilizzati a tal fine, possono essere ricondotti in due grandi classi: *Automatic Repeat Request* (ARQ) e *Forward Error Correction* (FEC).

La classe di meccanismi ARQ si basa sulla ritrasmissione dei pacchetti che non sono stati ricevuti dal destinatario. Non sono adatti per le applicazioni audio real-time, a causa del notevole incremento che si ha nei ritardi di latenza (dovuti all'attesa che i dati non ricevuti, vengano trasmessi nuovamente), e per l'impossibilità di utilizzo in ambienti multicast come MBone.

La classe di meccanismi FEC è un'alternativa attraente, essendo affidabile e soprattutto non portando ad aumenti di latenza. Consiste nella trasmissione contemporanea dei dati audio veri e propri e di informazioni ridondanti relative a dati inviati in precedenza, in modo da poter ricostruire a destinazione almeno una parte dei pacchetti audio perduti [BolGar]. L'efficacia di meccanismi di questo tipo, dipende dall'andamento delle perdite: vi è una maggiore efficienza nel caso in cui la media delle perdite consecutive è piccola. Esistono molte soluzioni che adottano meccanismi di FEC, alcuni di semplice implementazione, altri più complicati che coinvolgono operazioni di OR-esclusivo.

Utilizzare una tecnica di FEC, consiste come detto, nello spedire informazioni su uno o più pacchetti precedenti; come conseguenza vi può essere un incremento del data rate spedito. In genere per limitare questa quantità di dati spediti in più, si adotta la tecnica di utilizzare una codifica secondaria più povera, per le informazioni ridondanti, rispetto a quella primaria utilizzata per spedire nuovi dati. Ad esempio una scelta potrebbe essere di usare una codifica primaria PCM ed una secondaria GSM.

Un'ulteriore scelta da effettuare, riguarda il numero di livelli di ridondanza che si vogliono utilizzare: supponendo di dover spedire il pacchetto *n*, con un livello di ridondanza, insieme ad esso si spediranno anche le informazioni relative al pacchetto *n-1*, con due livelli si spediranno anche le informazioni relative ai pacchetti *n-1* e *n-2* eccetera. E' chiaro che maggiore è il numero di livelli di ridondanza, maggiore sarà la quantità di byte spediti.

Nella tabella 2-2 si riassumono alcune combinazioni di FEC: la prima colonna indica la codifica principale, le eventuali codifiche secondarie specificando con (*i*) che il pacchetto audio *n* porta informazioni ridondanti sul pacchetto *n-i*; la seconda colonna indica la quantità di dati spediti per le varie combinazioni e, nella terza colonna, il fattore di miglioramento per quanto riguarda la percentuale di perdite, che si ottiene con l'utilizzo del FEC con quella specifica combinazione. Ad esempio un fattore di miglioramento uguale a tre indica che la percentuale di perdite al

destinatario dopo l'utilizzo del FEC è un terzo rispetto al caso in cui il FEC non venga usato.

Come si può vedere dalla tabella l'utilizzo di una tecnica di FEC, comporta dei notevoli benefici per quanto riguarda la percentuale di perdite e l'aumento del data rate rimane contenuto.

Combinazioni	Data Rate (kbit/s)	Miglioramento
(PCM)	64	1
(PCM, LPC(1))	69	2,5
(PCM, LPC(2))	69	6
(PCM, LPC(1), LPC(2))	74	6
(PCM, LPC(1), LPC(3))	74	10
(PCM, LPC(1), LPC(2), LPC(3))	79	18

Tabella 2-2: Combinazioni di FEC.

Un meccanismo di FEC, introduce quindi, notevoli miglioramenti nella qualità dell'audio, grazie alla diminuzione rilevante della percentuale di perdite. Tuttavia all'interno di un'applicazione audio, è possibile utilizzare altre tecniche di sostituzione dei pacchetti persi, che non comportano né la ritrasmissione dei pacchetti non ricevuti (come con i metodi di classe ARQ), né l'invio di informazioni aggiuntive di ridondanza (come con i metodi di FEC). Esiste, infatti, la possibilità, di sostituire i vuoti lasciati dai pacchetti persi, direttamente al ricevente, tramite tecniche cosiddette di *packet substitution*. Consistono nel sostituire in fase di riproduzione, i vuoti lasciati dai pacchetti non ricevuti, con altre informazioni, come ad esempio silenzio e rumori bianchi, oppure attraverso sostituzioni più complesse come quelle di tipo *waveform* e di interpolazione. Sebbene questi sistemi siano più semplici da sviluppare, i benefici apportati alla qualità dell'audio, sono meno efficienti rispetto alla tecnica di Forward Error Correction [HSW 95].

2.7 Riassunto

In questo capitolo abbiamo affrontato, brevemente, le problematiche inerenti allo sviluppo di applicazioni audio in tempo reale, soffermandoci su

tutti gli aspetti più interessanti.

Concludiamo il capitolo, mostrando un grafico che riassume in base alle percentuale di perdite e alla dimensione media dei ritardi di playout, come questi influiscano sulla qualità dell'audio (vedi Figura 2-5).

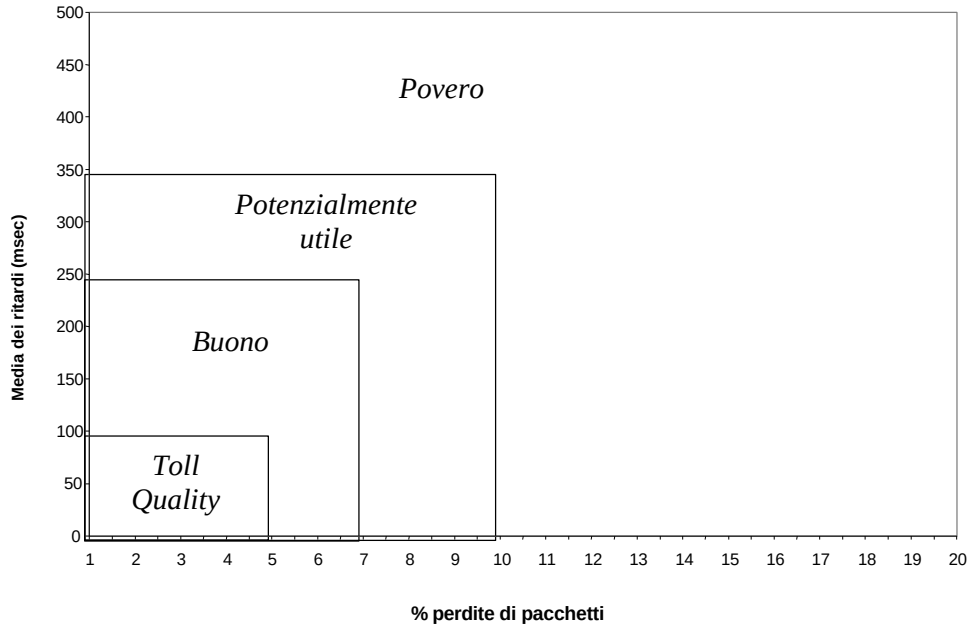

Figura 2-5: QoS in funzione delle perdite e dei ritardi.

La qualità migliore, definita *toll quality*, la si ottiene con perdite inferiori al cinque per cento e con ritardi minori di 100 ms. e corrisponde alla qualità telefonica. Al crescere dei ritardi e della percentuale di perdite, la qualità peggiora fino a diventare povera e quindi, non adatta ad una conversazione telefonica su rete a commutazione di pacchetto.

Capitolo 3

Sviluppo dell'applicazione audio

In questo capitolo analizziamo l'applicazione audio sviluppata. Ne descriviamo inizialmente le caratteristiche salienti e l'architettura. In seguito verranno analizzate le tre componenti fondamentali dell'applicazione: il meccanismo di controllo del playout, il server real-time e il meccanismo per il controllo del bit rate (il quale sarà poi ripreso più ampiamente nel capitolo successivo). In conclusione analizzeremo brevemente l'interfaccia utente.

3.1 Caratteristiche dell'applicazione

Il progetto *BoAT* (Bologna optimal Audio Tool) è stato sviluppato con l'obiettivo di testare in ambiente reale, quindi su rete a commutazione di pacchetto, i meccanismi per il controllo del tempo di playout e del bit rate. Di conseguenza, non tutte le problematiche relative all'implementazione di un programma di questo tipo sono state prese in considerazione. Vediamo brevemente le caratteristiche principali:

- L'applicazione gestisce connessioni di tipo unicast, in modalità *half-duplex*, su rete a commutazione di pacchetto.
- Sono gestiti tutti quei meccanismi, di cui si è parlato nel capitolo precedente, che permettono di migliorare la qualità dell'audio: ovviamente il controllo del playout e del bit rate, ma anche rilevamento del silenzio e FEC.
- E' stato affrontato e risolto il problema dei ritardi introdotti dal sistema operativo, mediante l'utilizzo di un *Server Soft Real Time*.
- La codifica delle informazioni audio avviene tramite un *codec*, che utilizza una tecnica di compressione basata sulla *trasformata wavelet*.

Attraverso l'uso del codec, possono venire simulati dal punto di vista del data rate alcuni dei principali tipi di codifica (PCM, DVI, GSM e LPC).

- L'applicazione è stata sviluppata per il sistema operativo Linux, sotto l'ambiente X-Window.

3.2 Architettura dell'applicazione

Possiamo vedere la struttura come composta da due parti distinte: la prima, che chiamiamo sender, è attiva quando l'applicazione è in modalità di trasmissione; la seconda, che chiamiamo receiver, è viceversa attiva quando ci si trova in modalità di ricezione dell'audio. Illustriamo in figura 3.1 la struttura dell'applicazione.

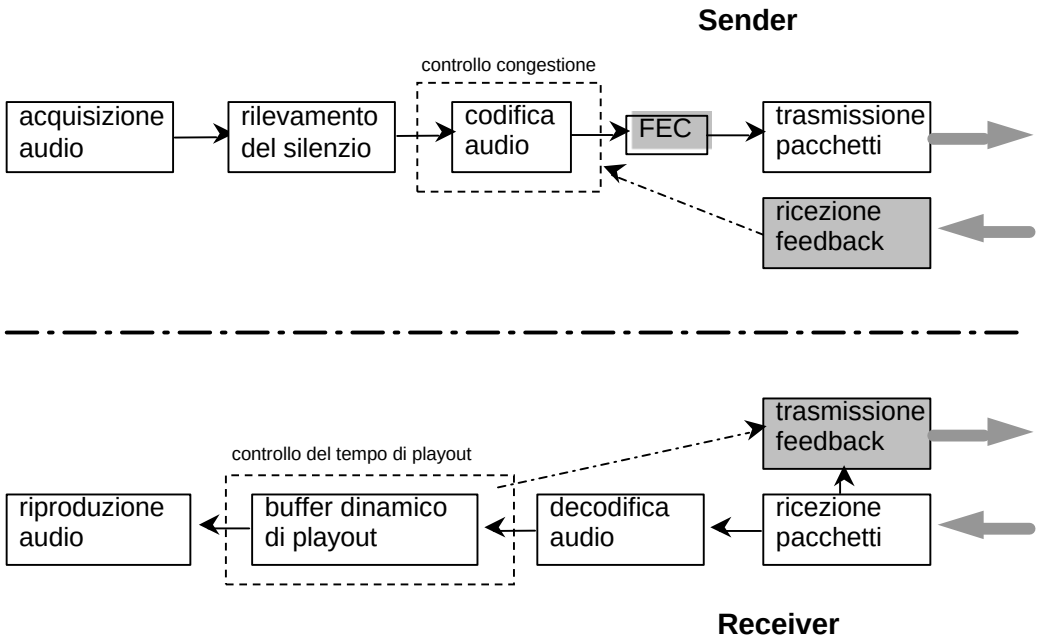

Figura 3-1: Struttura dell'applicazione audio.

Il flusso di una trasmissione audio ha inizio al sender con l'acquisizione dei dati attraverso la scheda audio, successivamente, dopo aver controllato che vi sia dell'audio da trasmettere, tramite il meccanismo di rilevamento del silenzio, le informazioni vengono codificate dalla componente di codifica, la quale è regolata dal meccanismo di controllo adattivo del bit

rate, che decide opportunamente, in base alle informazioni ricevute dal receiver (feedback), la quantità di byte che debbono essere usati per la codifica. Una volta che i dati sono stati compressi vengono aggiunte eventuali informazioni di ridondanza (FEC) ed infine, vi è l'inserimento dei dati in un pacchetto e la spedizione al receiver.

Il receiver nel momento in cui riceve un pacchetto dalla rete, provvede ad estrarne le informazioni per effettuare la decodifica (sia dei dati principali, sia delle eventuali informazioni di ridondanza), dopo di che tramite il meccanismo di controllo del playout decide se inserire le informazioni nel buffer di playout, oppure se scartarle nel caso siano arrivate troppo tardi. Periodicamente, le informazioni che si trovano nel buffer vengono inoltrate alla scheda audio per la riproduzione. Il receiver spedisce al sender informazioni di feedback, per informarlo sullo stato della connessione.

Vediamo ora più approfonditamente le varie parti che compongono l'applicazione, separatamente per sender e receiver.

3.2.1 La trasmissione

Acquisizione dell'audio

La scheda sonora come detto, trasforma i segnali elettrici in valori numerici (campioni), che vengono mantenuti in un buffer a disposizione dell'applicazione. Per ogni secondo vengono memorizzati nel buffer ottomila campioni, utilizzando la tecnica di campionamento *16 bit linear*, ogni campione è un valore a *16 bit*. L'applicazione effettua delle letture periodiche dal buffer della scheda audio ogni *40 ms.* estraendo *320* campioni, che corrispondono a *640 byte*. E' molto importante non ritardare la lettura dal buffer della scheda, per evitare di introdurre dei ritardi e, soprattutto, per evitare che il buffer stesso si riempia completamente e si perdano quindi dei campioni (*recording overrun*). Per impedire che ciò si verifichi, viene utilizzata una tecnica definita di *riallineamento*, ossia nel momento in cui l'applicazione si trova in ritardo rispetto alle letture periodiche che devono essere effettuate dal buffer, vengono letti un numero

di pacchetti consecutivamente, quanti ne servono per ripristinare la situazione; i dati estratti in questa fase non vengono spediti. In questo modo si ha una perdita di informazioni, abbiamo tuttavia considerato meno grave perdere qualche decina di millisecondi di audio, piuttosto che correre il rischio di introdurre ritardi e soprattutto, che si verifichino overrun del buffer della scheda audio.

Rilevamento del silenzio

La fase di rilevamento del silenzio è, come detto in precedenza, importante in quanto permette di spedire una quantità di dati inferiore ed inoltre, è durante le fasi di silenzio che il meccanismo di playout provvede all'aggiustamento del tempo di playout. La tecnica di rilevamento del silenzio da noi utilizzata, è la stessa che viene adottata nell'applicazione audio RAT. Dopo aver letto 40 ms. di informazioni dalla scheda audio, su di essi viene effettuato il calcolo della media dell'energia (per energia si intende il valore numerico di un campione). Nel caso in cui la media calcolata superi una precisa soglia, ciò che si è letto viene considerato audio. La soglia si adatta dinamicamente secondo i cambiamenti istantanei nell'intensità dell'audio.

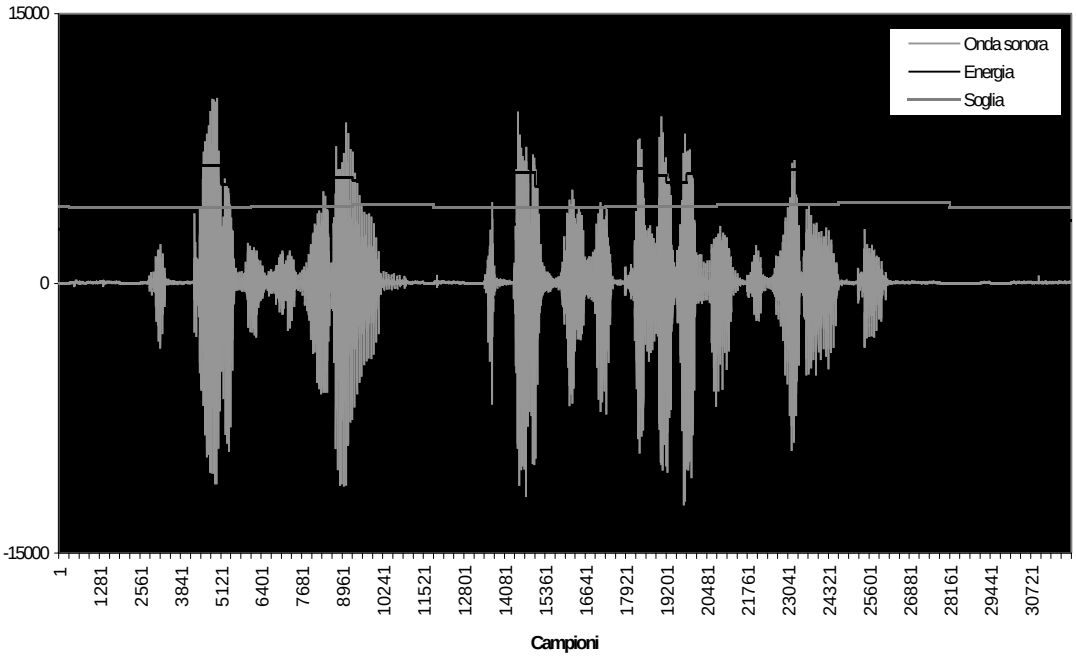

Figura 3-2: Comportamento del silence detector.

Il grafico di figura 3-2 illustra il funzionamento del silence detector. Sono presenti: l'intensità d'onda, dove valori vicini allo zero rappresentano silenzio, mentre valori più elevati (positivi e negativi) rappresentano periodi di parlato; la media dell'energia calcolata per ogni periodo di 40 ms. e la soglia. I periodi in cui la media dell'energia è posizionata sotto la soglia, coincidono con i periodi in cui l'intensità dell'onda è vicina allo zero, ossia i momenti di silenzio. E' importante notare come il silence detector rilevi quasi istantaneamente il passaggio da una fase di silenzio ad una di audio, evitando così di tagliare la parte iniziale o finale di una parola. Si può notare infine, come la soglia adattiva tenda a scendere durante i periodi di silenzio e, viceversa tenda a salire durante i periodi di audio, adattandosi quindi alle variazioni dell'intensità sonora.

Codifica dell'audio

La codifica delle informazioni avviene attraverso l'utilizzo di un codec a data rate variabile sviluppato da Francesco Naldi [Nal 98]. La quantità delle informazioni spedite cambia frequentemente in base ad alcuni fattori, i quali permettono come vedremo, di aumentare o diminuire il data rate. I fattori che regolano la quantità di informazioni che debbono essere spedite sono le perdite e i ritardi.

Il codec adotta la tecnologia wavelet, la quale è in genere utilizzata in algoritmi per la compressione di immagini. La struttura del codec è quindi simile a quella di un compressore di immagini:

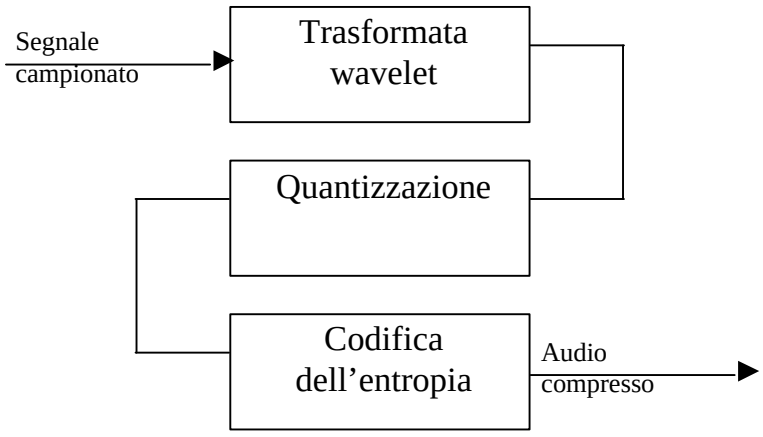

Figura 3-3: Struttura del codec a data rate variabile.

Vediamo brevemente le varie fasi che compongono la struttura del codec a data rate variabile.

- *Fase di trasformata wavelet*

Su ogni vettore di campioni (corrispondenti a 40 ms. di audio), viene eseguita una trasformazione che permette di spostare le informazioni nello spazio delle frequenze. Si opera in passi successivi, ad ogni passo otteniamo un vettore di dimensioni dimezzate (che si va ad aggiungere a quelli ricavati dai passi precedenti) ed uno con le frequenze più alte residue.

Graficamente possiamo rappresentare la fase di trasformata wavelet come segue:

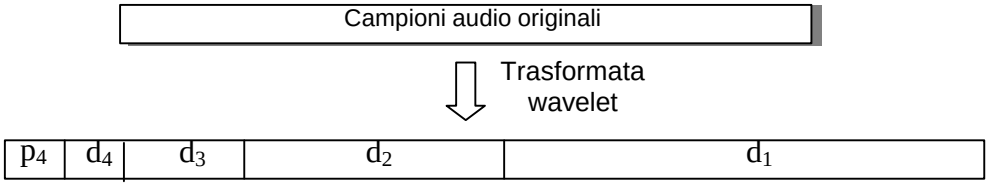

Figura 3-4: Trasformazione di un vettore di campioni audio attraverso la trasformata wavelet.

Ogni suddivisione ottenuta, rappresenta una banda di frequenze ben precisa e al suo interno, i coefficienti indicano la presenza o meno di tale banda nel segnale originale. Le frequenze sono disposte in ordine di altezza da destra verso sinistra, quindi lo spazio più piccolo contiene la banda delle basse frequenze fondamentali per il segnale originale.

Per risalire al numero di livelli di decomposizione possibili, occorre trovare il logaritmo della più alta potenza di due, la quale sia anche divisore del numero di campioni. Nel nostro caso 64 è la massima potenza binaria che divide 320 (numero di campioni), da cui avremo $log_2 64=6$. Ne consegue, che per ottenere il nuovo vettore sono necessari sei passi di decomposizione. Operata la trasformata wavelet, otteniamo un vettore di 320 coefficienti *floating point* che vanno quantizzati al fine di ridurre la dimensione delle informazioni da essi rappresentata.

- *Quantizzazione*

La quantizzazione ha la funzione di cambiare la rappresentazione dell'informazione per passare ad una più utile per la compressione. A causa del tipo di funzione usata, vi è una perdita di informazioni a causa della non invertibilità della funzione di quantizzazione, dovuto al fatto che valori diversi vengono mappati su un singolo valore e di conseguenza viene a mancare la condizione necessaria di iniettività.

Esistono diverse tecniche di quantizzazione in letteratura, dalla uniforme, alla statistica, alla vettoriale. Quella utilizzata nel codec è una quantizzazione ad approssimazioni successive, la quale possiede la caratteristica intrinseca di effettuare una codifica *embedded*. La codifica embedded è già utilizzata nelle immagini Jpeg di tipo progressivo, per intenderci quelle che su Internet cominciano ad essere visualizzate (a strati successivi), prima che sia stato completato il processo di ricezione dell'immagine.

La tecnica di quantizzazione ad approssimazioni successive, consiste nel raggruppare in ordine di importanza i bit che compongono la rappresentazione dei valori da quantizzare.

Dopo aver effettuato la fase di quantizzazione, si procede alla fase di codifica dell'entropia.

- *Codifica dell'entropia*

La fase di quantizzazione ha ridotto l'informazione audio ad un flusso di bit tramite un processo di codifica di tipo *lossy*. La codifica dell'entropia permette di eliminare l'informazione superflua dovuta ad una rappresentazione binaria non ottimale, in altre parole, tenta di renderla meno ridondante. La tecnica usata, denominata *run length*, è di tipo *lossless*, ossia non vi è alcuna perdita di informazione, ma solo una ottimizzazione dello spazio di memorizzazione utilizzato. Lo schema secondo il quale operano le tecniche run length, consta in una rappresentazione esclusivamente dimensionale dei gruppi di simboli omogenei.

Applicando questa metodologia ad un formato digitale binario, è possibile ricondurre una generica stringa di bit ad una semanticamente equivalente, composta dalle lunghezze delle sotto stringhe di bit uguali e adiacenti. Riportiamo di seguito un esempio di questa tecnica applicata ad una stringa binaria:

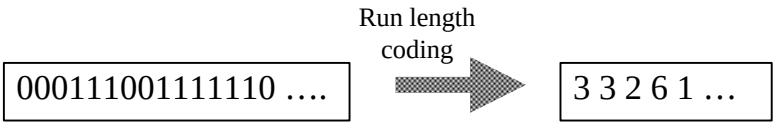

Figura 3-5: Passaggio da una codifica binaria a una run length.

La tecnica di compressione run length può essere notevolmente migliorata, utilizzando una ulteriore codifica per le lunghezze delle sotto stringhe. In altre parole si può utilizzare una tabella di simboli predeterminata, oppure costruita dinamicamente (la quale è più efficiente per quanto riguarda la compressione, ma troppo pesante per applicazioni real-time), per la codifica delle sotto stringhe. La tabella, mappa precise sotto stringhe binarie con una codifica numerica studiata appositamente. Nella tabella seguente è riportato il codice adottato nell'implementazione della procedura di run length:

Lunghezza sequenza	***Codice***
1..2	1x
3..6	01xx
7..14	001xxx
15..30	0001xxxx
31..62	00001xxxxx

Tabella 3-1: Codice run length delle lunghezze delle stringhe.

La caratteristica di poter essere integrati, rende il quantizzatore ad approssimazioni successive e la codifica run length, lo strumento ideale per la compressione in tempo reale dei coefficienti wavelet.

Dopo aver analizzato le tre fasi che compongono il codec a data rate variabile, possiamo darne una valutazione qualitativa e di performance. La caratteristica principale è quella di poter simulare la gran parte dei codec in circolazione, dal punto di vista del data rate trasmesso, ma anche quella di adottare codifiche intermedie. Il codec risulta quindi estremamente versatile e con un grado di raffinatezza ideale, per applicazioni di trasmissione audio su rete a commutazione di pacchetto. Dal punto di vista della qualità sonora, nonostante la tecnica sia di tipo lossy, utilizzando un data rate pari alla codifica PCM (64 Kbit/sec), non si ha uno scadimento nella qualità dell'audio rispetto all'originale prima della codifica. La fascia di utilizzo del codec è essenzialmente quella che va dai 64 Kbit/sec di PCM ai 13,2 Kbit/sec di GSM.

Per quanto riguarda le performance di esecuzione, diciamo innanzi tutto che la complessità del codec aumenta, con l'aumentare del data rate; infatti, maggiore precisione significa più passi di approssimazioni successive e quindi una fase di run length più lunga. Nel caso di un data rate pari alla codifica PCM, il tempo di esecuzione del codec, risulta intermedio tra i codec waveform e quelli ibridi e di sintesi. Si realizza così un compromesso tra velocità e qualità.

Per quanto concerne l'implementazione, il codec viene attivato tramite la funzione *wave_encoder*, alla quale debbono essere passati il vettore di 320 campioni da comprimere e il data rate, ossia la quantità di byte che dovranno essere restituiti dal codec dopo la codifica. Ad esempio supponendo di voler comprimere i dati ad una codifica equivalente a GSM (che corrisponde ad un data rate di 528 bit per 40 ms. di audio), i 640 byte di dati saranno compressi in 66 byte, per un fattore di compressione di circa 10 a 1.

Meccanismo di Forward Error Correction

Il meccanismo di FEC realizzato nella nostra applicazione, è ad un livello di ridondanza, ossia vengono inserite informazioni relative ai dati audio del pacchetto precedente. Prima di descrivere come è stato inserito il sistema di FEC all'interno di BoAT, ci sembra opportuno puntualizzare che

il FEC è stato inserito, solamente con lo scopo di migliorare la qualità globale dell'audio. Non vi è alcuna pretesa di considerare questa tecnica di ridondanza come la migliore tra tutte quelle disponibili, poiché non sono stati effettuati test di alcun tipo.

Il meccanismo può essere attivato o disattivato durante la conversazione in qualsiasi momento, da parte di colui che sta trasmettendo. La codifica secondaria utilizzata per le informazioni di ridondanza, ha un data rate equivalente a quello della codifica GSM (13,2 Kbit/sec). Ciò permette di avvalersi dei benefici del meccanismo di FEC, senza aumentare in maniera rilevante l'occupazione di banda. Un ulteriore accorgimento consiste nel disattivare automaticamente il meccanismo di FEC qualora il data rate scenda sotto la soglia dei 26 Kbit/sec, questo perché in una situazione di questo tipo, in cui la rete presenta una congestione rilevante, non è consigliabile aumentare la quantità di informazioni spedite, col rischio di congestionare ulteriormente la rete e avere come conseguenza un aumento nei ritardi e nelle perdite.

Dal punto di vista implementativo, per ogni periodo di audio trasmesso, viene mantenuta una codifica più povera che verrà inserita nel pacchetto successivo. Questo lo si ottiene agevolmente grazie al codec a data rate variabile: ogni codifica di *n* byte contiene al suo interno una successione di *n-1* codifiche incapsulate, di qualità via via decrescente, ottenute tralasciando semplicemente l'ultimo byte della precedente. Una volta effettuata la codifica primaria di un frame audio, si ottengono anche tutte le codifiche di livello inferiore, non serve quindi richiamare nuovamente il codec, per ottenere la codifica secondaria, ma basta estrarre il numero di byte necessari da quella primaria. Questo fa si che l'implementazione del meccanismo di FEC non pesi eccessivamente sull'applicazione, cosa fondamentale per un'applicazione come la nostra, dipendente dal tempo.

Trasmissione dei pacchetti

Una volta che le informazioni sono pronte per essere trasmesse, vengono inserite in un pacchetto e spedite. La struttura di un pacchetto dati è la seguente:

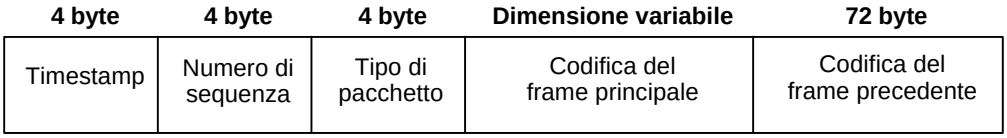

Figura 3-6: Struttura di un pacchetto dati.

Analizziamo brevemente i campi contenuti all'interno di un pacchetto:

- *Timestamp*: in questo campo di 4 byte viene memorizzato il tempo di spedizione. L'informazione viene poi utilizzata dal receiver, per regolare il tempo di emissione dell'audio contenuto nel pacchetto.
- *Numero di sequenza*: viene utilizzato per numerare progressivamente tutti i pacchetti spediti. Serve al receiver per calcolare i pacchetti persi, utilizzando la tecnica RTP (vedi par. 4.4).
- *Tipo di pacchetto*: utilizzato per distinguere i diversi tipi di pacchetto. Oltre a quello dati di figura 3-6 (utilizzato per la trasmissione dei dati audio), ci serviamo di altri pacchetti di controllo. L'elenco dei pacchetti è il seguente:
  - *Probe*, *Response* e *Installation*: utilizzati dal meccanismo di controllo del playout;
  - *Switch*: pacchetto che permette al sender di comunicare al receiver che gli ha ceduto la comunicazione.
  - *Receiver Report*: spedito dal receiver per informare il sender sullo stato della rete. E' un pacchetto di feedback, tramite il quale il sender decide il data rate per i pacchetti successivi.
  - *Hello*, *HelloResponse*: pacchetti che permettono di effettuare la sincronizzazione iniziale fra sender e receiver.
- *Codifica frame audio principale*: questo campo contiene i 40 ms. di audio che devono essere spediti. La lunghezza è variabile e dipende dalla quantità di byte utilizzati per la codifica.
- *Codifica frame audio precedente*: qualora il meccanismo di FEC sia attivato, contiene la codifica secondaria dei 40 ms. di audio precedenti. La dimensione di questo campo è fissata in 72 byte.

Nel momento in cui il pacchetto è pronto, viene trasmesso e il processo ricomincia.

3.2.2 La ricezione

Questa parte all'interno dell'applicazione audio si occupa della ricezione dei pacchetti e provvede all'emissione dell'audio. Vediamo di seguito le varie fasi dal momento in cui i pacchetti sono stati ricevuti.

Decodifica dell'audio

Questa fase si occupa di riportare le informazioni contenute nel pacchetto alla dimensione originaria. La fase di decodifica deve essere esattamente opposta a quella di codifica, è quindi importante sapere il data rate utilizzato in fase di compressione dei dati da parte del sender. Questa informazione è contenuta nei primi byte del vettore in cui si trova il frame audio codificato. Una volta appurato il data rate di codifica è possibile effettuare l'operazione inversa, che porta al ripristino della dimensione originale. Ovviamente le informazioni decodificate non coincidono con quelle precedenti la codifica, in quanto il codec come è stato detto in precedenza è di tipo lossy. Maggiore è la compressione effettuata, maggiore sarà la perdita di informazioni.

Una volta decodificate le informazioni, sia del frame principale, sia dell'eventuale frame inserito dal FEC, le informazioni vengono gestite dal meccanismo di controllo del playout.

Buffer di playout

Il buffer è regolato dal meccanismo di controllo del playout. Qui descriviamo solamente la struttura del buffer e il suo utilizzo, il meccanismo di controllo verrà discusso in un paragrafo successivo.

All'interno del buffer vengono inseriti i pacchetti audio in attesa di essere riprodotti. Ogni pacchetto ha una propria posizione, specificata dal timestamp contenuto nel pacchetto stesso. Questo fa si che, nel caso in cui i pacchetti non arrivino nello stesso ordine con il quale sono stati spediti, l'ordine è ripristinato all'interno del buffer di playout. Il buffer è composto

da un insieme di celle, ognuna delle quali può contenere 40 ms. di audio oppure essere vuota, nel caso non vi sia audio da riprodurre. La struttura del buffer come si può vedere in figura 3-7 è quella di una lista circolare.

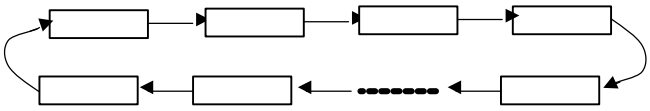

Figura 3-7: Buffer di playout.

La dimensione del buffer è variabile e viene regolata dinamicamente dal meccanismo di controllo del playout in base alle variazioni dei ritardi. Ad esempio, se i ritardi end-to-end aumentano, il meccanismo deve compensare tale aumento ritardando il tempo di emissione, per dare la possibilità ai pacchetti di arrivare in tempo per essere riprodotti. Come conseguenza, si ha che un pacchetto può rimanere per un tempo maggiore nel buffer, prima di essere schedulato per la riproduzione; ne consegue che il buffer deve essere in grado di contenere un numero di pacchetti superiore e quindi deve avere una capacità maggiore. In generale, la dimensione del buffer deve essere limitata inferiormente ad un numero di celle, equivalente ad una quantità di tempo, corrispondente al ritardo aggiuntivo, dopo il quale i pacchetti vengono riprodotti.

Riproduzione dell'audio

La gestione dell'emissione dell'audio viene regolata, come detto, dal meccanismo di controllo del playout. Ogni 40 ms. viene effettuata una lettura dal buffer per l'emissione di un pacchetto. E' importante mantenere la stessa periodicità che utilizza il sender in fase di lettura dei frame di dati, per non inserire dei ritardi artificiali all'interno delle parole. Questa temporizzazione è stata implementata attraverso l'utilizzo degli allarmi, facendo uso della funzione di sistema s*etitimer*.

Nel momento in cui scatta un allarme, viene eseguita una funzione di gestione, la quale controlla se nel buffer è presente il pacchetto che dovrebbe essere emesso. Nel caso sia già stato ricevuto, viene estratto dal buffer è avviene la scrittura sulla scheda audio. Dopo di che viene settato

nuovamente l'allarme ai 40 ms. successivi. Per sapere immediatamente qual è la cella del buffer che dovrebbe contenere il pacchetto da riprodurre, si fa uso di un cursore, chiamato *alarm_pointer*, il quale è sempre posizionato sulla cella che deve essere controllata nel momento in cui scatta l'allarme. Una volta che la cella è stata controllata, ed eventualmente il pacchetto è stato estratto, il cursore viene spostato alla cella successiva, che corrisponde ai prossimi 40 ms. Nel caso la cella sia vuota (questo può accadere se ci si trova in un periodo di silenzio, oppure se il pacchetto non è arrivato in tempo), non vi è alcuna emissione, ma viene solamente riattivato l'allarme e aggiornato il cursore *alarm_pointer*.

3.3 Meccanismo di controllo del playout e supporto real-time al S.O.

In questo paragrafo vengono descritti brevemente il meccanismo per il controllo del playout e il server per il supporto real-time al sistema operativo. Per uno studio più ampio ed esauriente, si faccia riferimento a [Bal 99].

3.3.1 Meccanismo per il controllo del playout

Per compensare i ritardi di rete variabili, un meccanismo di controllo del playout è necessario. L'idea consiste nel accodare in un buffer i pacchetti ricevuti, ritardandone l'emissione di una quantità di tempo oltre la ricezione del primo pacchetto. Tipici meccanismi per il controllo del playout, aggiustano in modo adattivo il playout delay, con l'obiettivo di tenere questo ritardo aggiuntivo il più piccolo possibile. Esiste una stretta relazione tra i ritardi addizionali imposti e l'ammontare dei pacchetti persi dovuti al loro arrivo in ritardo. Si deve quindi cercare un compromesso, poiché ritardi aggiuntivi troppo elevati diminuiscono il numero di pacchetti persi, concedendo loro più tempo per arrivare, ma minano l'interattività. Al contrario ritardi addizionali minimi, migliorano l'interattività, ma al tempo stesso fanno si che aumenti la possibilità che i pacchetti non arrivino in

tempo per essere riprodotti, creando problemi all'intelligibilità della conversazione.

Un segmento audio è costituito da periodi di energia (*talkspurt*), separati da periodi di silenzio, durante i quali non è generato audio. I meccanismi utilizzati, in genere mantengono costante il tempo di playout durante i talkspurt, aggiustandolo invece nei periodi di silenzio. Questo viene fatto con lo scopo di evitare di inserire interruzioni artificiali all'interno di parole, oppure di tagliare parti di audio. La figura 3-8 riassume questa tecnica di adattamento dinamico del playout.

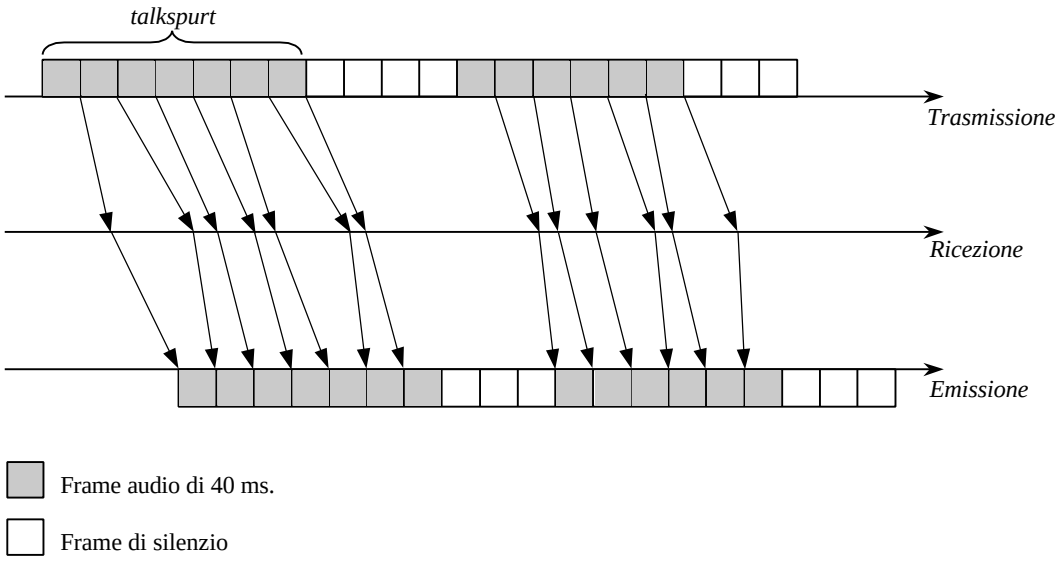

Figura 3-8: Riproduzione audio, tramite l'utilizzo di un meccanismo di controllo del playout.

Come si evince dalla figura, il periodo di silenzio tra i due talkspurt, è di lunghezza maggiore all'interno della fase di trasmissione. Mentre in fase di emissione è stato ridotto di un frame; ciò è dovuto all'intervento del meccanismo di controllo del playout, che ha sfruttato tale periodo di silenzio, per ricalcolare il tempo di emissione, anticipando di conseguenza, la riproduzione del secondo talkspurt.

I meccanismi per il controllo del playout adottati dalla maggior parte dei tool audio attualmente in circolazione, soffrono dei due seguenti problemi:

1. si fa uso di un meccanismo software esterno (ad esempio basato su NTP) per sincronizzare gli orologi sia del sender, che del receiver. Questo per evitare che orologi con frequenze diverse, portino il receiver ad avere problemi di *overflow* o *underflow* del buffer.
2. si assume che i ritardi sperimentati dai pacchetti su Internet, abbiano una distribuzione di tipo Gaussiano. Questa assunzione che viene usata in quasi tutti le applicazioni audio non è una congettura plausibile. Recenti studi sperimentali hanno indicato la presenza di frequenti picchi di ritardi end-to-end nella trasmissione audio [BCG 95].

Il meccanismo ideato recentemente presso il Dipartimento di Scienze dell'Informazione dell'Università di Bologna [RGPSB 98] e da noi implementato nella nostra applicazione audio, migliora tutti gli effetti negativi menzionati sopra, mantenendo valori soddisfacenti per il playout delay e per la percentuale di perdite dovute all'arrivo troppo tardi dei pacchetti.

Il meccanismo, è stato disegnato per adattare dinamicamente i ritardi di playout di un talkspurt, alle condizioni di traffico della rete, senza fare alcun tipo di assunzione sulla distribuzione dei ritardi end-to-end e senza fare uso, di un meccanismo esterno per la sincronizzazione degli orologi. Descriviamo ora la tecnica per l'aggiustamento dinamico del tempo di playout.

Si cerca di ottenere, ad intervalli periodici di circa un secondo, una stima del limite superiore dei ritardi sperimentati dai pacchetti durante una comunicazione audio. Questo limite è periodicamente calcolato, tramite lo scambio di pacchetti tra sender e receiver. Si fa uso di un protocollo di *handshake* a tre vie, tramite il quale, è possibile calcolare un *round trip time* (misura che indica la quantità di tempo, necessaria ad un pacchetto per effettuare il percorso sender-receiver-sender), il quale viene usato per stimare il limite superiore dei ritardi, tramite il quale il receiver aggiusta il playout delay per il talkspurt successivo e ridimensiona il buffer di playout. Il meccanismo proposto, garantisce che il ritardo di playout può essere aggiustato tra un talkspurt e l'altro, utilizzando un periodo di silenzio

sufficientemente lungo, senza introdurre *gap* o *time collision* (formalmente definiti in [RGPSB 98]).

Riassumendo, le innovazioni principali introdotte dal meccanismo per il controllo del playout sono le seguenti:

1. un tecnica interna e accurata, che permette di mantenere una stretta sincronizzazione tra gli orologi di sender e receiver;
2. un metodo che stima in maniera adattiva il tempo di playout per i pacchetti audio, con un *overhead* computazionale aggiuntivo minimo;
3. un metodo semplice ed esatto per ridimensionare il buffer di playout, basato sulle misure delle condizioni di traffico della rete sottostante.

Per quanto riguarda l'implementazione del meccanismo, periodicamente (ogni due secondi) viene effettuata la sincronizzazione tra sender e receiver, tramite l'utilizzo di un handshake a 3 fasi: il sender spedisce un pacchetto di *probe* al receiver (prima fase), il quale non appena lo riceve provvede a rispedirlo immediatamente al sender (pacchetto di *response*, seconda fase). Il sender all'arrivo del pacchetto di response, può effettuare il calcolo del *round trip time*. Dopo avere misurato il ritardo di andata e ritorno, il sender invia il valore calcolato al receiver (pacchetto di *installation*, terza fase), che lo utilizza per definire il nuovo tempo di playout, per i pacchetti successivi.

3.3.2 Supporto real-time al sistema operativo

Per preservare un comportamento strettamente dipendente dal tempo, le applicazioni multimediali richiedono al sistema operativo sottostante di fornire loro garanzie di Qualità del Servizio (QoS) di tipo real-time. Abbiamo già accennato nel capitolo 2, che sistemi operativi general purpouse, come Linux, non fornisco garanzie di questo tipo. Si deve quindi cercare di ridurre l'impatto negativo, che il meccanismo di schedulazione dei processi, di un sistema operativo non adatto ad applicazioni real-time, ha sulla qualità dell'audio. Un sistema operativo generico, ha ripercussioni negative sui due aspetti più importanti per quanto riguarda la qualità della riproduzione dell'audio, che sono:

1. il mantenimento della continuità dell'audio. Eventuali discontinuità nella riproduzione, sono dovute all'interruzione del trasferimento di audio alla scheda sonora. Si ha di conseguenza lo svuotamento del buffer della scheda audio, che comporta l'introduzione di silenzi artificiali, inopportuni, all'interno del flusso audio, peggiorando l'intelligibilità della conversazione.
2. mantenere minimi i tempi di attesa per quanto riguarda l'emissione dell'audio. Anomalie di schedulazione, possono portare all'aumento dei ritardi end-to-end e di conseguenza, ad una diminuzione nell'interattività della conversazione.

Esistono fondamentalmente due tecniche differenti, che si prefiggono di raggiungere questo obiettivo, direttamente al livello applicazione, senza quindi apportare modifiche al kernel del sistema operativo.

La prima tecnica che analizziamo brevemente, consiste nel implementare un meccanismo, direttamente all'interno dell'applicazione, con il compito di compensare le anomalie dovute alla schedulazione. Questa tecnica, definita *cushion*, si prefigge l'obiettivo di fornire in modo continuativo audio alla scheda sonora, mantenendo quindi la continuità di riproduzione, attraverso un comportamento adattivo, nei confronti delle anomalie di schedulazione [KouHar 97]. L'idea che sta alla base del meccanismo di cushion, è simile a quella del meccanismo di playout, consiste nel ritardare l'emissione dell'audio, tramite l'inserimento preventivo di periodi di silenzio nel buffer della scheda sonora, per prevenire eventuali interruzioni dovute a schedulazione. La quantità di silenzio che deve essere inviato alla scheda (*cushion size*), varia a seconda dello stato del sistema. Per quantificare questa quantità di silenzio, si effettua un monitoraggio del carico del sistema operativo. Attraverso questo monitoraggio è possibile sapere la lunghezza dei periodi di schedulazione e di conseguenza, variare in modo adattivo la dimensione del cushion. Anche la dimensione del cushion è normalmente modificata nei periodi di silenzio tra un talkspurt e

l'altro. Questa tecnica, permette quindi di mantenere la continuità dell'audio, introducendo tuttavia ritardi.

La seconda tecnica si prefigge invece di eliminare le anomalie di schedulazione, attraverso l'uso di meccanismi esterni, che si prefiggono di fornire ad applicazioni strettamente dipendenti dal tempo, una maggiore quantità di CPU, rispetto a quello che normalmente il sistema operativo assegna ad una qualsiasi applicazione.

Per avere garanzie di qualità del servizio di tipo real-time, la nostra applicazione fa uso di un software sviluppato da Marco Serra [Ser 98]. Si tratta di un server specifico per la schedulazione dei processi che elaborano continuous media in ambiente UNIX. Il server originariamente sviluppato per il sistema operativo Solaris, è stato modificato e riadattato per funzionare sotto Linux. Il server è costituito da tre componenti principali: il *broker*, il *dispatcher* e la *dispatch table*. Per accedere ai servizi forniti dal server, si fa uso di un client, tramite il quale debbono essere mandate in esecuzione le applicazioni che hanno necessità di una maggiore quantità di CPU. L'interazione tra le varie componenti del server e il client è mostrata in figura 3.9.

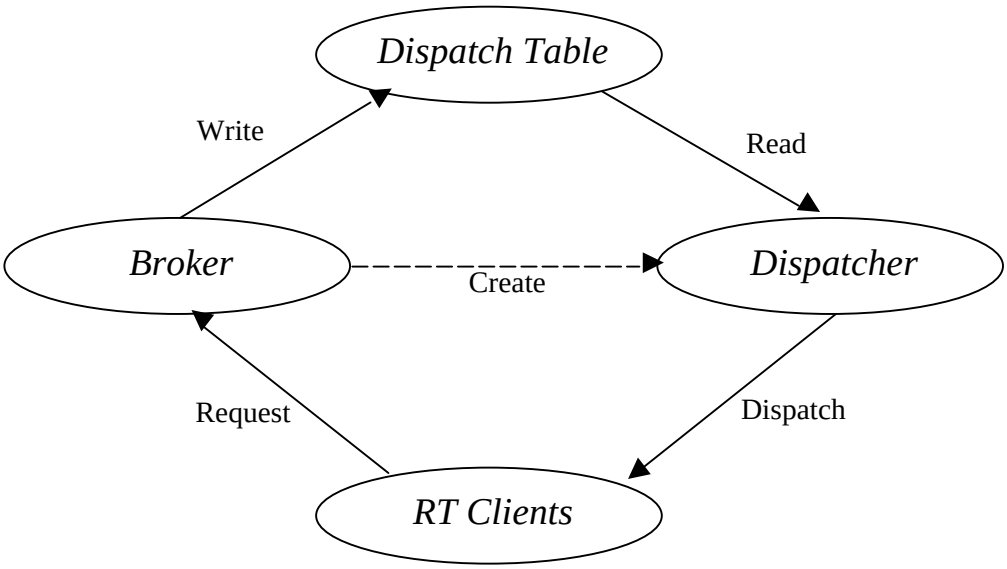

Figura 3-9: Architettura del server soft RT.

Il processo server

Il broker riceve le richieste dei processi real time (i client), ed esegue un controllo di ammissione per verificare se il nuovo processo può essere schedulato con successo. Se il controllo va a buon fine, il broker inserisce il nuovo processo nel gruppo di quelli RT, che aspettano di andare in esecuzione. Al nuovo processo viene assegnata una priorità di attesa, inoltre viene scritto nella dispatch table un nuovo ordine di schedulazione calcolato in base all'algoritmo prescelto, il *rate monotonic*.

Il broker, viene creato nel momento in cui si manda in esecuzione il server. E' un processo demone che deve avere i privilegi di root, poiché deve poter modificare le priorità dei processi in esecuzione. Rimane in attesa e nel momento in cui arriva una nuova richiesta di un processo, si sveglia e la gestisce nella maniera opportuna. Il broker non esegue l'effettivo dispatching dei processi RT, ma lascia questo compito al dispatcher.

Il dispatcher viene creato dal broker e ucciso quando non ci sono processi RT da schedulare. Contiene la memoria condivisa con la dispatch table. Si occupa della schedulazione dei processi RT presenti nella dispatch table, seguendone l'ordine di schedulazione, quindi non permette a nessun processo di monopolizzare l'uso del processore, in quanto il tempo di esecuzione dei processi RT e protetto l'uno dagli altri.

La dispatch table, infine, è uno spazio di memoria condivisa, nella quale il broker scrive la schedulazione calcolata e dal quale il dispatcher legge l'ordine con cui assegnare il processore ai processi.

Il processo client

Il client permette ai processi di accedere ai servizi forniti dal server. Una qualsiasi applicazione che vuole fare uso del server deve essere mandata in esecuzione attraverso il client (*cpu_cli*) in questo modo:

cpu_cli nome_programma <periodo in millisec.> <percentuale di CPU>.

Il periodo indica la dimensione dello slot di tempo, durante il quale il programma deve usare una percentuale di CPU specificata dal secondo

parametro. Ad esempio un periodo di 10 ms. e una percentuale uguale a 20, indicano che ogni 10 millisecondi, due debbono essere usati dal programma che ne fa richiesta.

La nostra applicazione utilizza un periodo di 40 millisecondi ed una percentuale uguale a 50.

3.4 Meccanismo per il controllo adattivo del bit rate

In questo paragrafo accenniamo brevemente al meccanismo per il controllo del bit rate, che sarà poi ripreso e affrontato in maniera più accurata nel capitolo 4.

La regolazione del bit rate viene effettuata al sender, in base alle informazioni che periodicamente (ogni 2 secondi) il receiver gli fornisce sullo stato della rete, tramite i *Receiver Report*. Il sender utilizza le informazioni di feedback che gli vengono fornite, per decidere il data rate più opportuno con il quale spedire i pacchetti audio successivi. L'obiettivo dell'utilizzo di un meccanismo di questo tipo, è quello di sfruttare ad ogni istante la banda disponibile, adattandosi allo stato di carico della rete.

Le informazioni che il receiver spedisce al sender, sono la percentuale di perdite di pacchetti e la variazione dei ritardi, nel periodo di monitoraggio di 2 secondi. Questi parametri descrivono esattamente lo stato in cui si trova la rete ad ogni istante, dal punto di vista del carico di trasmissione a cui è sottoposta, e il sender può quindi effettuare automaticamente le opportune regolazioni del bit rate.

Nel prossimo capitolo, analizzeremo gli aspetti teorici che stanno alla base dei meccanismi di controllo adattivo del bit rate e illustreremo la tecnica da noi utilizzata.

3.5 Interfaccia utente

Per finire, analizziamo brevemente l'interfaccia utente della nostra applicazione. L'interfaccia è stata sviluppata utilizzando le librerie *Tcl/Tk*

versione 8.0. Abbiamo deciso di utilizzare queste librerie, per motivi di semplicità e, soprattutto, per evitare di appesantire troppo l'applicazione.

L'interfaccia sviluppata è molto semplice, ma al tempo stesso permette di tenere sotto controllo tutti gli aspetti relativi all'applicazione. Nel momento in cui il tool viene eseguito, all'utente si presenta la finestra, illustrata in figura 3-10.

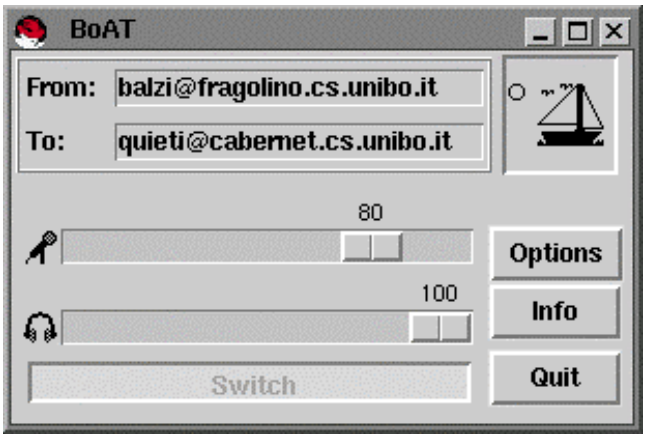

Figura 3-10: Finestra principale dell'applicazione audio.

Tramite questa finestra, l'utente può tenere sotto controllo i volumi di entrata (microfono) e di uscita (cuffie o casse). In alto sono presenti informazioni relative ai partecipanti alla conversazione. Sono presenti inoltre, quattro pulsanti:

- *Switch*: questo pulsante è attivato solamente nell'interfaccia di colui che sta parlando (sender), tramite il quale può decidere di passare la linea al suo interlocutore. Dopo che è stato premuto, il sender, diventa automaticamente receiver e, viceversa, il receiver diviene sender. Possiamo vedere la funzione del pulsante switch, simile a quella del tasto presente nei walkie-talkie, che permette di cedere la linea da un interlocutore ad un altro.
- *Options*: questo pulsante permette di accedere ad una finestra di opzioni, tramite la quale è possibile variare alcune caratteristiche dell'applicazione.

- *Info*: fornisce informazioni sugli autori.
- *Quit*: termina l'esecuzione dell'applicazione audio.

Analizziamo brevemente la finestra delle opzioni illustrata in figura 3-11.

Figura 3-11: Finestra delle opzioni.

In questa finestra sono presenti diverse opzioni di scelta, che possono essere utilizzate per modificare il comportamento dell'applicazione. Le opzioni di sincronizzazione sono utilizzate esclusivamente dal receiver, mentre la scelta del data rate e l'opzione di controllo della ridondanza, sono di esclusiva pertinenza del sender.

- **Synchronization options**

Queste opzioni permettono al receiver di modificare il comportamento del meccanismo per il controllo del playout, abilitando o disabilitando il rilevamento delle *spike* e per consentire che le installazioni vengano forzate (vedi [Bal 99]).

- **Data rate**

L'utente può scegliere qui che tipo di meccanismo per il controllo del bit rate utilizzare, potendo scegliere tra quello implementato da noi (*Automatic B&Q*) e quello utilizzato in Free Phone [Gar 96].

E' inoltre possibile scegliere l'utilizzo di codec fissi, escludendo quindi il meccanismo per il controllo del bit rate. La scelta di un codec fisso non significa che l'applicazione utilizza quel codec specifico (ad esempio PCM), ma ne simula solamente il data rate (ad esempio 64 Kbit/sec) tramite il codificatore a data rate variabile.

- **Redundancy control**

Utilizzato dal sender per attivare o disattivare il meccanismo di Forward Error Correction.

3.6 Riassunto

In questo capitolo abbiamo analizzato a grandi linee la struttura della nostra applicazione, descrivendo tutte le componenti e soffermandoci in particolare, sui meccanismi per il controllo del playout e del bit rate e sul server real-time.

Nel capitoli successivi, analizzeremo in modo approfondito, il sistema di controllo adattivo del bit rate, valutandone anche le prestazioni.

Capitolo 4

Meccanismo di controllo adattivo del bit rate

Il presente capitolo, descrive in dettaglio il sistema da noi proposto per la regolazione della quantità di byte, che debbono essere utilizzati in fase di trasmissione delle informazioni audio, in funzione dello stato della rete di comunicazione. In primo luogo consideriamo dettagliatamente gli aspetti teorici che stanno alla base del meccanismo sviluppato. In seguito analizziamo alcune soluzioni proposte in letteratura, che affrontano le problematiche relative alla congestione; infine, illustriamo il meccanismo da noi sviluppato. Da ora in avanti i termini bit rate e data rate, avranno il medesimo significato: la quantità di bit utilizzati per la codifica delle informazioni audio.

4.1 Aspetti teorici

Abbiamo già visto che il servizio *best effort* offerto da Internet, non è compatibile con le richieste di un'applicazione per la trasmissione di audio in tempo reale. Sebbene, a prima vista, le problematiche che inficiano la qualità dell'audio, sembrino dovute a elementi totalmente estranei all'applicazione, poiché le condizioni della rete sono strettamente legate alle applicazioni che ne fanno uso, è necessario analizzare come queste ultime, si adattano alle condizioni della rete sottostante.

L'accesso alla rete è regolato dall'uso di protocolli, che possiamo suddividere principalmente in *connection-oriented* (TCP) e *connection-less* (UDP).

Per quanto concerne TCP, esso fornisce direttamente alle applicazioni diversi meccanismi di controllo, che regolano l'immissione e la dimensione dei pacchetti su una rete. Per evitare la congestione, l'attuale standard TCP

consiglia l'impiego di due tecniche: *partenza lenta* e *riduzione moltiplicativa* [Ste 97], che sono correlate e facilmente applicabili. Sappiamo che per ogni connessione, TCP deve ricordare la dimensione del finestra ricevuta (cioè la dimensione del buffer annunciata nei riscontri iniziali). Per controllare la congestione, TCP gestisce un secondo limite chiamato *limite della finestra di congestione*, o più semplicemente *finestra di congestione*. In un istante qualsiasi, TCP si comporta come se la dimensione della finestra sia:

Finestra_ammessa = min (annuncio_ricevitore, finestra_congestione)

Nel caso di una connessione non congestionata, la dimensione della finestra di congestione è uguale a quella del ricevitore. L'eventuale riduzione della finestra di congestione, comporta una riduzione del traffico che TCP immetterà sulla connessione. Per stimare la dimensione della finestra, TCP presuppone che la maggior parte delle perdite di pacchetti, siano dovute alla congestione e, di conseguenza, adotta la strategia della riduzione moltiplicativa, che consiste nel dimezzare la finestra di congestione, via via che le perdite aumentano (fino ad un limite minimo). Per quei pacchetti che restano nella finestra ammessa, si esegue un *backoff esponenziale* del timer di ritrasmissione. Poiché TCP riduce la finestra di congestione della metà per ogni perdita, esso riduce esponenzialmente la finestra se la perdita continua. In altre parole, se la congestione è probabile, TCP riduce esponenzialmente sia il volume di traffico, sia la velocità di ritrasmissione. Se la perdita continua, alla fine limiterà la trasmissione ad un singolo datagramma e continuerà a raddoppiare i valori di *timeout* prima di ritrasmettere. L'idea quindi, è quella di fornire una rapida diminuzione del traffico, per consentire ai router di smaltire il numero di pacchetti che sono presenti nelle code.

Nel momento in cui la situazione ritorna alla normalità, il protocollo deve ripristinare una situazione normale di trasmissione. La scelta utilizzata in TCP è quella cosiddetta di partenza lenta. Il ripristino della quantità di dati che debbono essere spediti, avviene gradualmente, questo per evitare di congestionare nuovamente la rete. Il ripristino a partenza lenta, è un

meccanismo adattivo, che consiste nell'aumentare il traffico di un pacchetto, ogni volta che arriva un riscontro, per un pacchetto precedentemente spedito. La finestra di congestione, viene quindi ampliata di un pacchetto, ogni volta che si riceve un riscontro di un pacchetto spedito in precedenza; se ad esempio la finestra di congestione ha dimensione iniziale 1, nel momento in cui arriva il riscontro, TCP porta la finestra a 2, trasmette due pacchetti e attende riscontro. Nel momento in cui arrivano i due riscontri, TCP porta la finestra ad una dimensione di 4 pacchetti e provvede a spedirli. Come si può vedere quindi, il ripristino della situazione avviene ugualmente in tempi rapidi. Infatti, se supponiamo *N* la dimensione della finestra del ricevitore, sono necessari solamente *log2N* viaggi di andata e ritorno, prima che TCP possa trasmette consecutivamente *N* pacchetti. TCP adotta un ulteriore accorgimento di tipo conservativo, per evitare di congestionare nuovamente la rete. Nel momento in cui, la dimensione della finestra di congestione raggiunge la metà della sua dimensione originale, TCP entra in una fase di *anticongestione* e rallenta la rapidità di incremento. Durante questa fase, la dimensione della finestra aumenta di un solo pacchetto alla volta e solo se, tutti i segmenti nella finestra sono stati ricevuti.

Il protocollo UDP, al contrario di TCP, non effettua alcun controllo sulla quantità di informazioni che vengono spedite; il controllo viene delegato all'applicazione. In altre parole, è lasciata libertà a colui che sviluppa l'applicazione, di decidere la quantità di byte che vuole spedire, senza che il protocollo di trasporto possa porre delle limitazioni. Le applicazioni che utilizzano UDP, vanno quindi dotate di opportuni meccanismi per limitare la quantità di risorse utilizzate.

Si tratta ora di decidere, che tipo di tecnica utilizzare, per affrontare lo stato di congestione della rete. Esistono due correnti di pensiero, che individuano differenti modi per affrontare la situazione:

- Aumentare l'immissione di pacchetti e quindi la richiesta di ampiezza di banda.

- Diminuire l’immissione di pacchetti, attendendo che il traffico diminuisca.

Il primo approccio, è senza dubbio di facile utilizzo e, sicuramente, si raggiunge il risultato di aumentare la quantità di pacchetti inviati e ricevuti. L’effetto negativo di una scelta di questo tipo, sta nel fatto che aumentando la quantità di dati spediti, quasi sicuramente vi è, di riflesso, un aumento anche nel numero delle perdite e nei ritardi. Questo comporta un degrado nella qualità dell’audio, per quanto riguarda le applicazioni che trasmettono audio in tempo reale. Inoltre questa scelta, non tiene conto del fatto che la rete è una risorsa che deve essere condivisa: se tutte le applicazioni adottassero questo principio, si arriverebbe in breve tempo al blocco quasi totale del traffico.

Il secondo metodo per affrontare la congestione della rete, può quindi essere considerato più appropriato. Rimane da decidere, in quale modo diminuire la quantità di dati che debbono essere trasmessi. Una prima scelta, potrebbe essere quella di diminuire la frequenza di spedizione (come avviene in TCP), con l’intento di diminuire il numero di pacchetti che transitano all’interno della rete. Una tecnica di questo tipo, è da considerarsi, tuttavia, deleteria per una applicazione audio real-time, che ha come vincolo principale, quello di mantenere la periodicità sia durante la lettura dei campioni, sia durante la riproduzione. Se ad esempio, si sceglie una dimensione di 40 ms. per i frame audio, il numero di pacchetti che devono essere spediti al secondo, è univocamente determinato e non può essere assolutamente diminuito (né tanto meno aumentato).

Questo vincolo, ci porta a considerare come scelta più appropriata, per lo sviluppo di un meccanismo di controllo del bit rate, quella che, al contrario, limita la quantità di byte per ogni pacchetto audio e non il numero di pacchetti trasmessi.

Dopo avere analizzato gli scenari, nei quali un’applicazione audio deve muoversi, dobbiamo prendere in considerazione, una qualche tecnica che ci permetta di regolare il data rate. L’idea più efficiente è, probabilmente, quella di un meccanismo a finestra, simile a quello adottato in TCP.

Possiamo dare ora una definizione più rigorosa, del compito che deve svolgere un meccanismo di controllo adattivo del bit rate: è una funzione che, prese in input la disponibilità delle risorse e la qualità del servizio richiesta, genera uno stato di mediazione ottimale, al quale l'applicazione stessa deve uniformarsi.

Per quanto riguarda la disponibilità delle risorse, il tutto si riconduce alla valutazione della quantità di banda disponibile. Come abbiamo detto in precedenza, la banda è una risorsa condivisa, inoltre lo stato della rete, muta molto velocemente, di conseguenza, non c'è un modo diretto per appurare in ogni istante quant'è la banda disponibile. L'unica possibilità, per appurare la disponibilità di risorse su Internet, è lo scambio di messaggi di controllo da parte delle applicazioni che usufruiscono del servizio. In questa maniera, si può avere un riscontro oggettivo sullo stato della rete.

Per quanto riguarda la qualità del servizio, l'obiettivo è quello di trasmettere audio alla massima qualità offerta dall'applicazione. Il limite superiore, è dato dalla frequenza di campionamento e dalla quantità di bit, utilizzati per la rappresentazione dei campioni. Per una applicazione audio real-time, il data rate massimo è in genere limitato a 64 Kbit/sec, valore che permette di avere una buona qualità di voce. L'obiettivo del meccanismo di controllo del bit rate, è quello di variare il data rate di trasmissione, in base alle condizioni della rete, in modo da garantire sempre la miglior qualità uditiva possibile.

Sebbene il meccanismo per il controllo del bit rate, sviluppato in questa tesi, sia orientato all'utilizzo all'interno di un'applicazione di tipo unicast, ci sembra opportuno fare brevemente, alcune considerazioni sulle problematiche inerenti a questo argomento in ambiente multicast. La trasmissione di audio (ed anche video), da una particolare sorgente ad un insieme di destinatari, generalmente effettuata tramite IP Multicast, pone il problema della scarsa efficienza di un controllo del data rate di tipo *source-based*. In una trasmissione di tipo multicast, non è possibile definire un singolo data rate, il quale sia giusto per tutti i riceventi, a causa

dell'eterogeneità della rete, di conseguenza non tutte le richieste di banda dei destinatari, possono essere soddisfatte con un unico rate di trasmissione. E' quindi necessario che il data rate di trasmissione sia *receiver-driven*, ossia gestito direttamente dai riceventi, per eliminare il problema delle diverse condizioni della rete. Sono allo studio diverse tecniche per cercare di trovare una soluzione al problema. Tra le tante, citiamo il protocollo *Receiver-driven layered Multicast* illustrato in [MJV 96].

4.2 Caratteristiche dei meccanismi di controllo del bit rate

I meccanismi di controllo del data rate, permettono di adattare la quantità di dati trasmessi, allo stato della connessione, con lo scopo di non sovraccaricare la rete e di limitare i ritardi e la percentuale di perdite.

Allo stato attuale, possiamo suddividere tali meccanismi in manuali, semiautomatici e automatici, a seconda dell'influenza che l'utente può avere su di essi. Due tra i più interessanti tool, per la trasmissione di audio su rete a commutazione di pacchetto, RAT e Free Phone, utilizzano meccanismi diversi. Il primo adotta un sistema manuale, il secondo invece integra tutte e tre le tecniche.

Possiamo riassumere un meccanismo per il controllo del data rate, come composto da due fasi distinte: una prima fase di rilevamento della situazione e una seconda fase di reazione. A seconda del tipo di meccanismo implementato (manuale, semiautomatico o automatico), le due fasi possono venire sviluppate in modi diversi.

Se il controllo è manuale (come in RAT), spetta all'utente, rendersi conto delle variazioni, che si presentano nelle condizioni della rete e successivamente, tentare di porvi rimedio, aggiustando il data rate. Purtroppo, per l'utente non è affatto semplice dare una valutazione sullo stato della rete, in quanto i fattori da tenere in considerazione sono diversi. Nel caso di una conversazione di tipo telefonico (*full-duplex*), gli interlocutori, possono accorgersi di un eventuale peggioramento nello stato della rete, dalla caduta della qualità dell'audio e di conseguenza agire sul meccanismo per diminuire il data rate. Può capitare invece che la

trasmissione sia unilaterale, ossia da un parte si trasmette e dall'altra si riceve (ad esempio audioconferenza multicast). In questo caso, colui che sta trasmettendo, non ha la possibilità di avere un riscontro qualitativo su ciò che sta spedendo, di conseguenza qualora la qualità dell'audio peggiori, non può apportare dei correttivi per porre rimedio alla situazione.

Quella appena illustrata, non è l'unica difficoltà nell'utilizzo di un meccanismo di controllo manuale. Un altro fattore da tenere in considerazione, è la velocità con la quale le condizioni della rete mutano, può accadere infatti, che nel momento in cui l'utente rileva un peggioramento nella qualità dell'audio (dovuto a maggiori ritardi o ad un numero di perdite più elevato), e cerca di porvi rimedio diminuendo il data rate, la situazione della rete può essere cambiata nuovamente e di conseguenza la modifica effettuata non risultare coerente.

L'unico fattore che può costituire un vantaggio nell'utilizzo di un meccanismo manuale, è quello che consente a colui che trasmette, di avere sempre sotto controllo il data rate con il quale sta trasmettendo, cosa che non può avvenire con un sistema completamente automatico.

Un sistema automatico, esclude i limiti riscontrati in quello manuale, è tuttavia di difficile messa a punto, poiché occorre prendere in considerazione diversi parametri prima di poter definire un giusto meccanismo.

Analizziamo ora brevemente, le tecniche adottate dai due tool audio più importanti: RAT e da Free Phone.

RAT utilizza un meccanismo manuale; all'utente viene fornito un insieme di cinque codec diversi, tra i quali scegliere quello più adatto in ogni istante. I codec a disposizione sono i seguenti: 16 bit linear (128 Kbit/sec), PCM μ-law (64), DVI (33), GSM (13,2) ed infine LPC (5,6). E' giusto osservare, che per agire sui codec è necessario avere specifiche conoscenze in materia, cosa che non è detto abbiano tutti quelli che fanno uso di applicazioni di questo tipo.

Free Phone integra tutte e tre le metodologie, all'interno del suo meccanismo. Nella modalità manuale, l'utente può scegliere tra una gamma

di codec predefiniti: PCM μ-law (64 Kbit/sec), ADPCM6 (48), ADPCM5 (40), ADPCM4 (32), ADPCM3 (24), ADPCM2 (16), GSM (13,2) ed infine LPC (4,8).

Nella modalità semiautomatica, sono messe a disposizione dell'utente, 13 combinazioni differenti tra le quali poter scegliere coppie di codec, uno per la codifica primaria e uno per quelle secondarie (eventualmente nessuno qualora non si utilizzi una tecnica di FEC).

Il meccanismo automatico, fa uso di due soglie di intervento: HWM (*High Water Mark*) e LWM (*Low Water Mark*). Corrispondono rispettivamente, ad un limite superiore ed inferiore della percentuale di pacchetti persi. Una volta attivato il meccanismo automatico, l'applicazione agisce aumentando il data rate, se le perdite scendono sotto la soglia inferiore, diminuendolo se superano la soglia superiore. L'utente deve solamente variare le percentuali di soglia, che di *default* sono il 5% per LWM e il 15% per HWM.

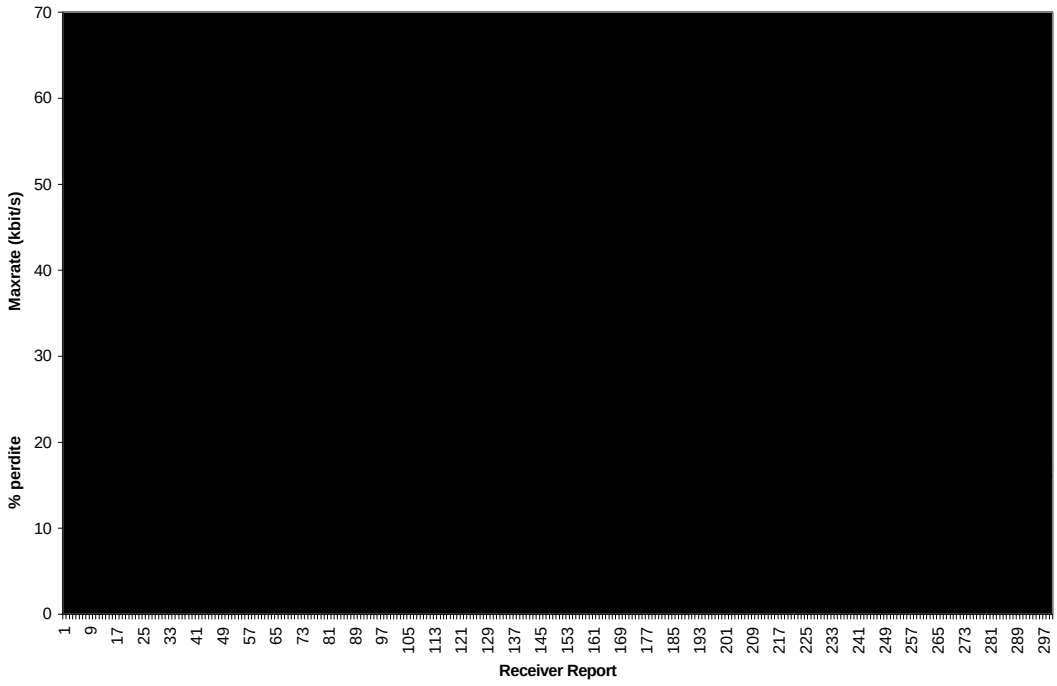

Figura 4-1: Sistema di controllo del bit rate adottato da Free Phone.

In figura 4-1 è illustrato il funzionamento del sistema di controllo adottato da Free Phone. Si può notare innanzi tutto, come il data rate abbia un andamento a gradini, ciò è dovuto al fatto che si fa uso di codec fissi, ognuno dei quali ha un data rate distinto. Una scelta di questo tipo, a nostro avviso, porta ad avere dei repentini cambi nella qualità dell'audio, dovuti alle differenze in termini di data rate tra i vari codec; tali cambi repentini possono risultare poco gradevoli a colui che ascolta.

Il meccanismo di controllo di Free Phone è ad andamento discreto, e spesso fatica ad interpretare correttamente lo stato della rete. Un motivo che determina questa difficoltà, nel valutare il data rate, è dovuto al fatto che il meccanismo sviluppato nel tool dell'INRIA, fa uso solamente delle perdite istantanee, trascurando invece il jitter. La varianza dei ritardi è, a nostro avviso, un parametro fondamentale, per riuscire a valutare correttamente lo stato della rete.

Il meccanismo automatico di Free Phone verrà ripreso in seguito, per metterlo a confrontarlo con quello da noi sviluppato.

4.3 Misurazione del jitter

In questo paragrafo, analizziamo gli aspetti teorici che stanno alla base del parametro jitter. Abbiamo visto in precedenza che con questo termine, si indica la varianza dei ritardi, fenomeno causato dalla permanenza variabile dei pacchetti, all'interno dei buffer dei router e dai percorsi diversi seguiti dagli stessi. Il calcolo del jitter sarebbe possibile, qualora possedessimo le misurazioni dei ritardi stessi.

Definiamo con $R_i = A_i - T_i$ il ritardo sperimentato dal pacchetto *i*, dove A_i rappresenta l'istante di arrivo e T_i l'istante di partenza. La difficoltà consiste nella valutazione di questa differenza. I tempi di partenza e di arrivo, generalmente vengono misurati con i timer delle macchine locali, i quali è probabile, non siano sincronizzati tra loro e, di conseguenza, non risulta possibile dare una stima dei ritardi sperimentati dai pacchetti.

Esiste la possibilità di effettuare una sincronizzazione esterna, attraverso l'utilizzo dell'algoritmo NTP (Network Time Protocol) [Mil 94], il quale

permette la sincronizzazione di due macchine remote. Non è molto diffuso e manca di precisione.

Escludendo la sincronizzazione delle macchine, l'unica misura, che ci permette di avere una idea della variazione dei ritardi (ma non dei ritardi stessi), è il *relative transit time* (r.t.t.). E' ottenuto, come nel caso precedente, sottraendo all'istante di arrivo del pacchetto, il valore dell'istante di partenza (che deve essere contenuto nel pacchetto stesso), senza porre vincoli di sincronizzazione tra le due macchine. Si ottiene quindi una misura relativa, che messa in relazione con le misure dei pacchetti precedenti, ci consente di ottenere una stima della variazione dei ritardi. In letteratura, sono state sviluppate diverse funzioni che permetto di stimare il jitter, utilizzando i relative transit time.

A causa dell'impossibilità di conoscere a priori tutti i relative transit time, poiché l'insieme di questi valori cresce man mano che pervengono nuovi pacchetti, non possiamo ottenere una misura istantanea del jitter, usando la formula classica della varianza statistica:

$$s = \frac{\sum_{i=1}^{n} t_i^2}{n} - \left(\frac{\sum_{i=1}^{n} t_i}{n} \right)^2$$

Tuttavia è possibile, ad ogni successivo pacchetto ricevuto, calcolare una stima del jitter. Analizziamo ora due metodi diversi per permettono di stimare la varianza dei ritardi.

4.3.1 Algoritmo di RTP

Il primo algoritmo, è proposto in RFC 1889 relativo al protocollo RTP [SCFJ 96], e fornisce una stima della varianza statistica dei pacchetti di dati RTP. Il jitter *J*, è definito come la deviazione media delle differenze *D*, tra i relative transit time di due pacchetti. Diamo una definizione formale, del algoritmo utilizzato in RTP.

Sia R_i il tempo di arrivo del pacchetto i e S_i il relativo timestamp di partenza, si definisce con $D(i,j)$, $i < j$, la differenza tra due relative transit time.

$$D(i,j) = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i)$$

Per ogni pacchetto i, che viene ricevuto da parte del ricevente, il jitter viene aggiornato utilizzando la formula seguente:

$$J = J + \frac{\left(\left|D(i-1,i)\right| - J\right)}{16}$$

Questa funzione, è un ottimo stimatore del primo ordine e, il fattore *1/16*, permette di avere una buona riduzione del rumore, mantenendo tuttavia un ragionevole rapporto di convergenza. Questa tecnica di calcolo del jitter è stata sviluppata per essere indipendente dall'utilizzo di RTP, può quindi essere utilizzata con tecniche di *report* differenti.

4.3.2 Algoritmo del gradiente stocastico

La seconda funzione che vogliamo analizzare, si basa sul noto algoritmo del gradiente stocastico [RKTS 94], che fornisce un metodo semplice per stimare sia i ritardi, sia la varianza. Una prima funzione calcola una media pesata dei ritardi [Pos 81], nel seguente modo:

$$\hat{d}_i = a * \hat{d}_{i-1} + (1-a) * d_i$$

dove, per ogni i:

$\hat{d}_i$ è la media pesata dei relative transit time, calcolata all'i-esimo pacchetto.

d_i è il relative transit time dell'i-esimo pacchetto.

a rappresenta la memoria dello stimatore, il valore scelto è 0,998002.

La media pesata dei ritardi, è poi usata per calcolare la stima della varianza s, utilizzando la formula suggerita da Van Jacobson [Jac 88]:

$$s_i = a * s_{i-1} + (1-a) * \left| \hat{d}_i - d_i \right|$$

Per completezza, mostriamo anche una variante al metodo appena descritto, suggerita da Mills [Mil 83], che fa uso di due stimatori diversi per il calcolo della media pesata dei ritardi.

$$\hat{d}_i = \begin{cases} b * \hat{d}_{i-1} + (1-b) * d_i & se\ d_i > \hat{d}_{i-1} \\ a * \hat{d}_{i-1} + (1-a) * d_i & altrimenti \end{cases}$$

con *a* sempre uguale a 0,998002 e *ß* pari a 0,75.

L'idea è quella di usare due differenti meccanismi: uno per le fasi di crescita dei ritardi, durante i quali si da più peso (tramite *b*) ai valori istantanei; ed uno per i periodi di riduzione dei ritardi, durante i quali si da più peso alla storia passata (tramite *a*). Il sistema di calcolo della varianza rimane invariato.

Per concludere il paragrafo, mostriamo in figura 4-2, il confronto tra il metodo di calcolo utilizzato in RTP e quello suggerito da Van Jacobson.

L'andamento della varianza, calcolata con lo stimatore del primo ordine, presenta troppe oscillazione, di conseguenza un campionamento di questa funzione ogni 2 secondi, non fornisce informazioni apprezzabili per un meccanismo di controllo del bit rate. Per quanto riguarda il metodo che fa uso del gradiente stocastico, da un lato si ha un andamento più stabile rispetto al precedente, dall'altro la reazione ai mutamenti della rete, risulta essere troppo lenta e quindi anch'esso, non soddisfa le nostre necessità. Si noti anche come i due metodi non siano direttamente confrontabili, per quanto riguarda il calcolo del valore del jitter; ciò è dovuto soprattutto al fattore di riduzione del rumore applicato in RTP.

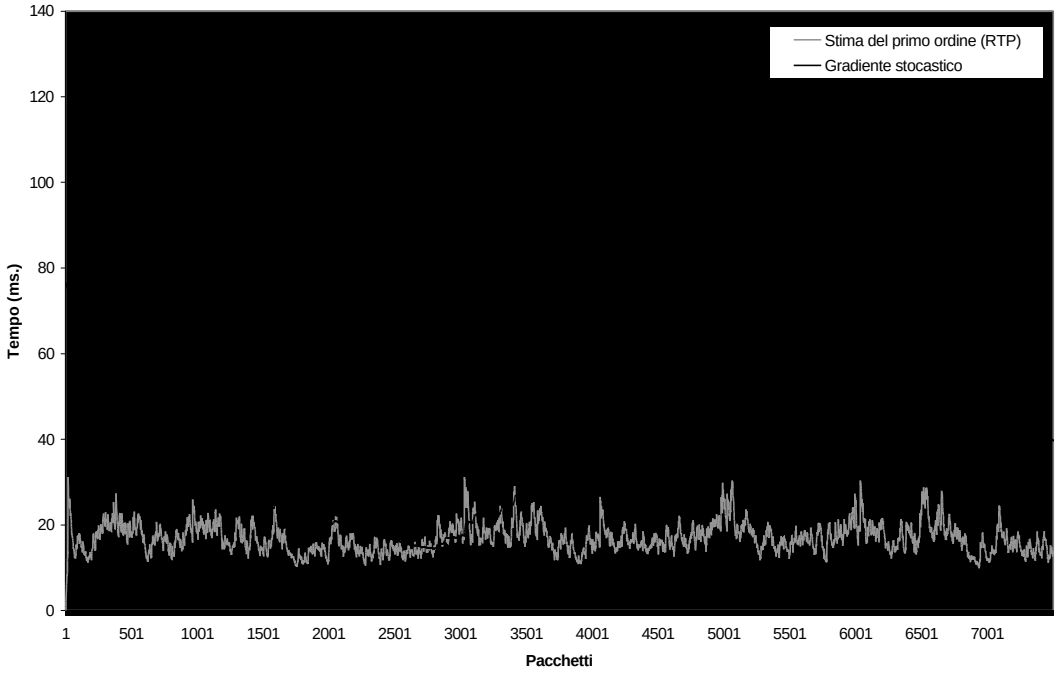

Figura 4-2: Confronto sulle diverse misurazione del jitter.

Nel paragrafo 4.5 verrà presentato un nuovo metodo per il calcolo del jitter, che si propone di rispondere in modo più preciso, alle esigenze di un meccanismo di controllo del bit rate, che sono: stabilità e reattività ai cambiamenti della rete.

4.4 Misurazione delle perdite

Il secondo importante parametro, che viene utilizzato all'interno di un sistema per il controllo adattivo del bit rate, è la percentuale di perdite. Abbiamo già osservato in precedenza, l'esistenza di due differenti tipologie di perdite. La prima riguarda i pacchetti che vengono scartati dai router, nel momento in cui si trovano in uno stato di congestione massima e, di conseguenza, non hanno più spazio all'interno delle code d'attesa. La seconda, riguarda invece i pacchetti che vengono regolarmente ricevuti, ma sono scartati dal meccanismo che regola il playout, poiché la ricezione è

avvenuta oltre il tempo di riproduzione (*deadline*). Per entrambe queste classi di perdite, è possibile effettuare delle misurazioni precise.

Per valutare le perdite riconducibili ai router, viene in aiuto nuovamente RTP, all'interno del quale, è stato implementato un algoritmo che permette di calcolare la percentuale di perdite istantanee. L'algoritmo fa riferimento al numero di pacchetti aspettati e a quelli realmente ricevuti, per effettuare un computo della percentuale di perdite. Il numero dei pacchetti aspettati, può essere calcolato dal ricevente, tramite la differenza tra, il più alto numero di sequenza ricevuto e il primo numero di sequenza ricevuto, nell'intervallo di monitoraggio. Questo metodo, valuta quindi come persi, i pacchetti che non vengono ricevuti nell'intervallo di tempo, durante il quale viene effettuato il monitoraggio, ma che potrebbero arrivare successivamente. Come si evince dalla figura 4-3, questo può portare a nostro avviso, ad avere valori di perdite non totalmente attendibili, per ciò che riguarda lo stato della rete. Infatti, un pacchetto che è arrivato leggermente in ritardo (*m-1* in figura), rispetto all'intervallo fissato, viene considerato perso. E' tuttavia possibile che lo stesso pacchetto, pervenga al ricevente nell'intervallo successivo, con la possibilità di compensare una perdita effettiva in quel periodo di monitoraggio.

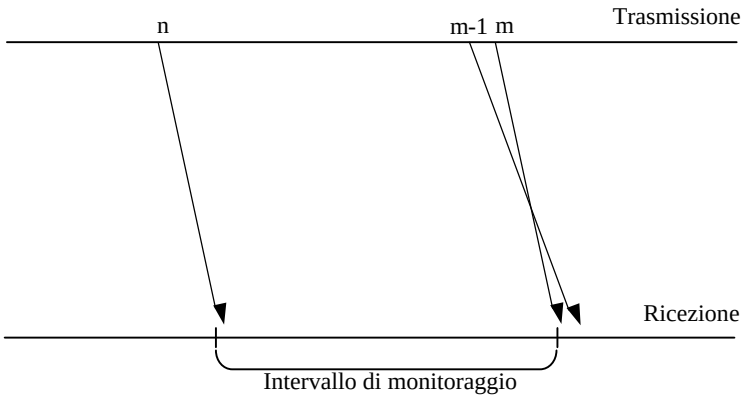

Figura 4-3: Monitoraggio delle perdite con la tecnica utilizzata in RTP.

Un altro fattore da tenere in considerazione, riguarda il fatto che i router hanno ormai a disposizione capacità di memoria molto ampie, dove

memorizzare i pacchetti ricevuti, nell'attesa di essere instradati. Di conseguenza, la probabilità che i pacchetti siano scartati è bassa, mentre è più probabile, che arrivino a destinazione con ritardi più alti, ma sempre all'interno dell'intervallo di monitoraggio di appartenenza, non venendo così considerati persi.

La seconda tipologia di perdite, stima come persi, i pacchetti che vengono scartati dal meccanismo di playout. In questo caso, la percentuale di perdite all'interno di un intervallo di monitoraggio, è data dal numero di pacchetti scartati, poiché arrivati in ritardo, in funzione di tutti quelli arrivati in quell'intervallo.

Entrambi i metodi considerano persi, i pacchetti che non arrivano entro un periodo di tempo prefissato. Possiamo comunque rilevare che, con il primo metodo si considerano persi, pacchetti che potrebbero invece essere ricevuti e utilizzati, con il secondo metodo invece, si considerano persi i pacchetti ricevuti ma che non è più possibile utilizzare.

All'interno della nostra applicazione, sono stati implementati tutti e due i metodi di rilevamento delle perdite, ma si è preferito utilizzare il secondo, per i motivi sopra descritti.

4.5 Il nostro meccanismo

Dopo aver analizzato, le problematiche relative alla valutazione del jitter e delle perdite, procediamo illustrando il meccanismo per il controllo del bit rate, sviluppato all'interno della nostra applicazione audio.

Il meccanismo utilizza, per calcolare il data rate, sia l'informazione relativa alle perdite, sia quella relativa alla varianza dei ritardi. Ciò permette, di avere una misura dello stato di congestione permanente (tramite le perdite) e anche una misura della congestione transiente (tramite il jitter).

Iniziamo illustrando la formula da noi sviluppata per la stima del jitter:

$$J = M_{Local}^{N} - D_B \qquad [1]$$

M_{Local}^{N} è la media dei relative transit time degli *N* pacchetti pervenuti nell'intervallo tra due Receiver Report.

D_B è il più piccolo relative transit time, rilevato dall'inizio della trasmissione, fino all'istante in cui si invia il rapporto.

Calcolato in questo modo, il jitter *J*, possiede le caratteristiche idonee che permettono di considerarlo, un parametro di valutazione della congestione. E' infatti, sufficientemente rapido nelle variazioni, possiede un *range* e un unità di misura (millisecondi) opportuni, ed infine, mantiene memoria della storia passata, ma è significativo solo nell'intervallo di tempo limitato cui si riferisce.

La quantità *J*, espressa in millisecondi, ci informa di quanto è risultata più lenta in media la connessione, rispetto alla velocità massima fino a quel momento ottenuta. Questo significa, che all'inizio di una nuova comunicazione, il meccanismo presume di essere in uno stato di bassa congestione, ma non appena l'applicazione ricevente, comincia a monitorare i ritardi relativi, aggiornando il migliore, si otterrà un riferimento sempre più preciso.

Mediamente, se i ritardi sono molto più elevati del valore minimo, significa che il tragitto da sorgente a destinazione, risulta maggiormente difficoltoso, per cui abbiamo una misura diretta, dello stato di traffico del tratto di rete attraversato dai pacchetti.

Ora che abbiamo stabilito una nuova misura del jitter e abbiamo definito un modo per misurare le perdite (tramite il meccanismo di playout), illustriamo il meccanismo di controllo, che permette attraverso questi due parametri, di definire, ad ogni istante, il data rate più opportuno a seconda dello stato della connessione.

Il meccanismo di controllo, può essere suddiviso in due fasi distinte e consecutive: la prima fase, utilizza il parametro delle perdite per definire un limite superiore al data rate, che chiamiamo *max_rate*, mentre la seconda, fa uso del jitter, per calcolare l'effettivo data rate di emissione, in base al valore di max_rate. Analizziamo ora le due fasi in modo approfondito.

Si è detto in precedenza, che le perdite devono dare una indicazione sulla congestione permanente della rete. Questa considerazione, ci ha portato a ritenere non sufficientemente preciso, il parametro delle perdite istantanee (ossia rilevate nell'ultimo intervallo di report), come indicatore di questa situazione.

In realtà l'insufficienza permanente di ampiezza di banda, non è data, a nostro avviso, da un eventuale picco nelle perdite istantanee, ma dalla continuità nel processo di perdita. Nella valutazione di questo parametro, è necessario quindi, considerare un fattore di memoria, che effettui una media pesata, dei valori istantanei delle perdite con la storia passata, attraverso l'utilizzo di un parametro di stima.

Ci proponiamo quindi, di non effettuare regolazioni in corrispondenza della singola misurazione di perdite, ma su una media che tenga conto dell'andamento globale delle stesse. Vogliamo così rilevare la congestione permanente, ovvero uno stato in cui la banda disponibile, non supporta in maniera persistente la trasmissione ad un determinato livello di data rate. In una situazione di questo tipo, ci sembra opportuno agire, limitando il data rate, in modo da restringere l'occupazione di banda da parte della nostra applicazione.

Vediamo ora l'algoritmo, che permette di dare una stima del limite superiore del data rate. L'algoritmo è simile a quello adottato dall'INRIA in Free Phone, al quale sono state apportate modifiche significative, che come vedremo, ne migliorano la funzionalità. Il limite superiore alla quantità di byte da spedire, ottenuto in questa fase, verrà poi utilizzato nella fase successiva, per misurare il data rate effettivo di trasmissione.

$$M_{Loss}^{n} = a * M_{Loss}^{n-1} + (1-a) * R_{Loss}$$

$$if\ \left(M_{Loss}^{n} > HWM\right)\quad then\ decrease(max\_rate)$$

$$else\ if\ \left(M_{Loss}^{n} < LWM\right)\quad then\ increase(max\_rate)$$

M_{Loss}^{n} è la stima della media delle perdite, pesata con memoria *a*,

calcolata al report *n*.

R_{Loss} è la percentuale di perdite istantanee, relative all'utlimo receiver report.

HWM e *LWM* sono rispettivamente, il limite superiore e inferiore della finestra di controllo.

Max_rate è il massimo data rate stabilito, considerando solo la percentuale di pacchetti persi.

Ad ogni report ricevuto dal receiver, il sender utilizza l'informazione sulle perdite istantanee, per aggiornare la media pesata. Dopo di che il valore aggiornato della media, è usato per un eventuale cambiamento del max_rate, qualora lo stato della rete si sia modificato dall'ultimo report ricevuto.

Vediamo ora la seconda fase, che effettua una stima della quantità effettiva di byte di trasmissione. Tale stima, è ottenibile attraverso l'utilizzo del jitter, al quale viene delegata, la rilevazione della congestione transiente, ovvero lo stato della rete nell'intervallo tra un report e il successivo. La definizione formale, che porta a determinare il numero di byte di trasmissione è la seguente:

$$out\_rate = max\_rate - (R_{Jitter} * b) \qquad [2]$$

R_{Loss} è la stima del jitter calcolata come in [1].

ß è un fattore di conversione, che permette di trasformare il jitter, che è espresso in millisecondi, in una quantità espressa in byte.

Out_rate è l'effettivo data rate di output, che tiene conto sia della congestione permanente, sia di quella transiente.

Dopo aver illustrato il meccanismo di controllo del bit rate, passiamo ad analizzare i parametri utilizzati, giustificandone la scelta.

L'obiettivo che ci siamo prefissi, nella scelta dei parametri, è stato quello di far si che il meccanismo fosse il più equo possibile, adattandosi allo stato

della rete nel modo migliore, senza esagerare nell'utilizzo della banda disponibile. Abbiamo quindi effettuato una taratura, che tenesse conto della qualità dell'audio, ma che fosse anche coerente con ciò che è stato appena detto. Nel capitolo successivo, verranno portati a conferma delle scelte effettuate, una serie di dimostrazioni di tipo sperimentale, mentre qui di seguito, analizziamo le scelte fatte dal punto di vista teorico.

La prima fase del processo di "registrazione" dei parametri, è relativa al controllo della congestione permanente, quindi occorre determinare *a*, HWM e LWM.

L'analisi del processo di perdite, ci ha permesso di individuare un andamento contraddittorio nelle informazioni che il receiver invia al sender, attraverso i report, per quanto concerne le perdite istantanee. Questa alternanza tra valori di perdita bassi e alti, è dovuto all'alta variabilità dello stato della rete, che ad ogni istante può mutare la condizione in cui si trova. In figura 4-4, mostriamo l'andamento della percentuale di perdite istantanee, misurate attraverso il meccanismo di playout.

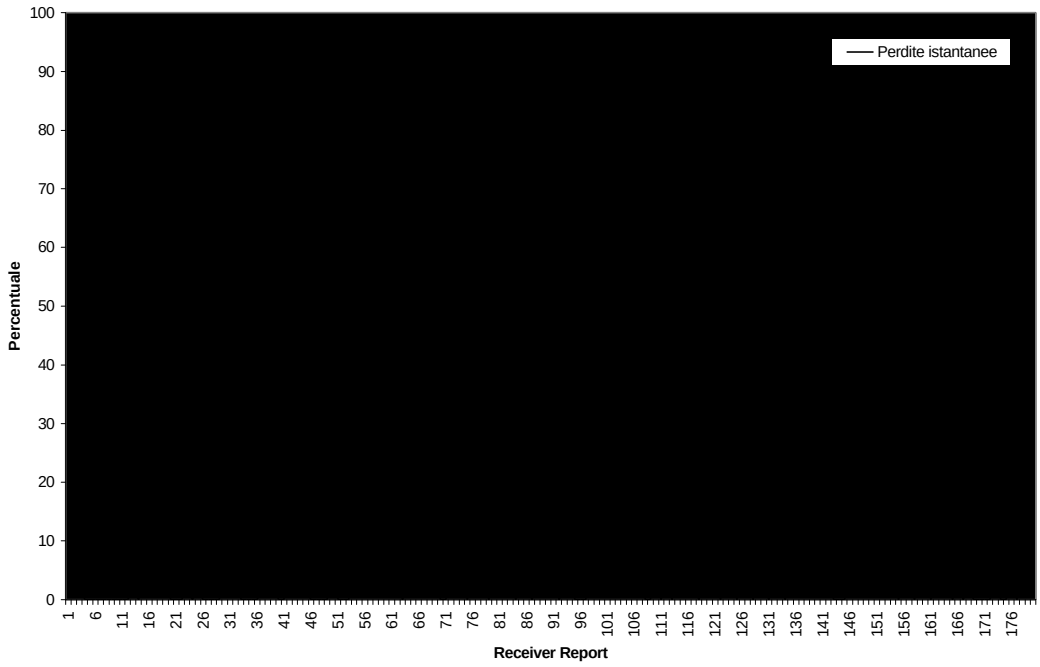

Figura 4-4: Andamento delle perdite rilevate con il meccanismo di controllo del playout.

Per riuscire ad avere quindi, un'interpretazione più precisa dello stato della rete, si è deciso di introdurre un processo di media pesata, per quanto riguarda le perdite.

La media pesata, permette di smorzare le oscillazioni, rendendo meno sensibile il meccanismo, ad eventuali report contraddittori e a picchi di perdite isolati, lasciando invariato il caso in cui, ci sia una tendenza ben precisa verso una percentuale di perdite fisse.

Si riesce così a mantenere, a nostro avviso, una buona sensibilità alla congestione permanente, mantenendo il limite superiore del data rate, entro valori che la rete può sopportare, smorzando nello stesso tempo, la rilevazione di quella transiente (che si ottiene anche attraverso le perdite), per evitare di abbassare troppo questo limite. Tale compito è lasciato al jitter, il quale, provvede a limare ulteriormente il data rate, qualora ve ne sia bisogno.

Il parametro che permette di effettuare una media pesata tra la storia e i valori istantanei delle perdite, che abbiamo definito come a, è stato valutato in 0,3. Questo ci permette di smorzare adeguatamente le oscillazioni della rete, mantenendo ugualmente una appropriata velocità di aggiornamento, per quanto riguarda la congestione permanente.

Passiamo ora ad analizzare i parametri HWM e LWM. Il compito di queste due soglie, è quello di regolare l'intervento del meccanismo sul max_rate. Si può vedere, analizzando l'algoritmo proposto in precedenza, che, nel caso in cui la media pesata della percentuale di perdite, si attesti su valori compresi tra questi due estremi, il limite superiore al data rate, rimane invariato, mentre qualora si esca da questo intervallo, il meccanismo agisce opportunamente sul max_rate. Si è cercato di stabilire due valori, che permettessero al meccanismo di intervenire, né troppo raramente, né troppo frequentemente.

In lavori precedenti, sono stati individuati rispettivamente, in 15% e 5% i valori per HWM e LWM [BCG 95]. Dopo attente valutazioni, abbiamo ritenuto un valore appropriato, quello assegnato a LWM, ma non siamo stati d'accordo, nell'assegnare una soglia così elevata a HWM. Abbiamo già

detto in precedenza, che la percentuale di perdite, per non rendere impossibile la conversazione, non deve superare il 9-10%, di conseguenza, non ci sembra opportuno, che il meccanismo si attivi, abbassando il data rate, solo se la media delle perdite supera il 15%, valore per il quale la qualità dell'audio risulta irrimediabilmente compromessa. E' stato individuato, perciò, un nuovo valore, per la soglia superiore, uguale al 10%. Questo livello di soglia, permette al meccanismo di intervenire rapidamente, qualora la media pesata della percentuale delle perdite superi questo limite.

Il meccanismo interviene sul max_rate, attraverso due funzioni, *increase* e *decrease*. La prima permette di aumentare max_rate, nel caso in cui la media pesata, scenda sotto la soglia LWM. La seconda, viceversa, riduce max_rate, qualora la media superi la soglia HWM. Per definire queste due funzioni, abbiamo effettuato diverse prove, cercando di raggiungere l'obiettivo di ottenere rapidità di intervento, ma anche accuratezza, ottenendo le seguenti due funzioni:

Increase (max_rate): *max_rate* + 10 %
Decrease (max_rate): *max_rate* - 15 %

Si è preferito avere una maggiore velocità durante la diminuzione, per adattarsi tempestivamente al peggioramento della connessione e viceversa, una maggior cautela, nel ripristino della situazione, per evitare di ricongestionare la rete, agendo ad ogni modo, con una discreta velocità.

Vedremo nel capitolo successivo tutti i risultati sperimentali che hanno portato a queste scelte.

E' importante notare come, in entrambi i casi, il data rate cambi gradualmente, con il notevole vantaggio, di rendere meno evidente il passaggio da un tipo di codifica ad un altro. Utilizzando un codec a data rate variabile, si ha un maggior grado di libertà, rispetto all'utilizzo di codec fissi. Riportiamo nella tabella 4-1, i vari livelli raggiungibili, in una ipotetica discesa dal massimo al minimo data rate possibile.

Si nota come in questo processo di discesa, rimangano definiti 11 codec virtuali diversi e come il passaggio dal primo al secondo implichi la perdita di quasi 10 Kbit/sec mentre, il passaggio dal decimo all'undicesimo, ne coinvolga solo due. Questo ci permette di essere più incisivi, dove la qualità dell'audio ne risente meno e, viceversa più cauti, dove la diminuzione del data rate, anche di pochi Kbit, può inficiare la qualità dell'audio.

Passo	*Data Rate kbit/sec*	*Gap*
1	64	-
2	54,4	9,6
3	46,2	8,2
4	39,3	6,9
5	33,4	5,9
6	28,4	5
7	24	4,4
8	20,4	3,6
9	17,4	3
10	14,7	2,7
11	13,2	1,5

Tabella 4-1: Sequenza dei data rate ottenibili in caso di peggioramento continuo della rete.

Per concludere l'analisi del meccanismo di controllo del data rate, resta da esaminare il fattore di conversione *ß*. Abbiamo visto come il parametro jitter influisca direttamente sul data rate di trasmissione. Il valore che viene inviato dal receiver al sender tramite i report è, come abbiamo detto, uno scostamento medio, dei tempi di relative transit time dei pacchetti pervenuti alla destinazione, nell'intervallo di monitoraggio, dal miglior delay relativo rilevato.

Dato che valori più alti, indicano uno scostamento maggiore dal miglior valore, non è errato considerare quest'informazione, come rilevatore dello stato di congestione ed effettuare tramite esso, un opportuno adattamento del rate di spedizione. E poiché, il valore riassume la condizione più recente della rete, può essere considerato come parametro di congestione transiente.

Dalla formula [2] si osserva come il parametro jitter, determini una ulteriore diminuzione nel data rate rilevato dalla congestione permanente. Il parametro *ß* consente di convertire il jitter espresso in millisecondi, in un numero di bit da sottrarre al max_rate. Tale valore, viene ricalcolato ad ogni report ricevuto, in funzione del max_rate, attraverso la seguente funzione:

$$ß = (max\_rate - 13.2) / 500$$

Il parametro *ß* mappa il *range* [0..500] ms., nei data rate compresi tra max_rate e il minimo posto a 13,2 Kbit/sec.

Le motivazioni che ci hanno portato a definire un intervallo di questo tipo sono sostanzialmente due: la prima consiste nel fatto che la nostra misurazione del jitter è relativa, e non assoluta, in quanto misura lo scostamento da una misura che rappresenta la condizione ideale di non congestione. La seconda, si basa sul fatto che in una conversazione audio i ritardi end-to-end, non devono superare i *600 ms.* per mantenere l'interattività. Sotto questa ipotesi, abbiamo quindi considerato che, per valori di jitter maggiori o uguali a 500 millisecondi, qualunque sia il max_rate, è opportuno collocare il data rate di output al minimo (13,2 Kbit/sec).

Il funzionamento della funzione di conversione e del controllo combinato, vengono illustrate nel grafico di figura 4-5, che riassume tre possibili situazioni diverse per il max_rate. Dal grafico si può valutare, come il parametro *ß* esegua un *mapping* corretto del jitter, espresso in millisecondi, in bit. Inoltre è possibile valutare il meccanismo di controllo, nella sua globalità

Se il max_rate determinato dalle perdite è basso, allora il jitter agisce in misura più lieve e precisa. Al contrario se il max_rate è elevato, l'azione del jitter è più veloce e decisa. In tutte e tre le situazioni, si può notare come vi sia una convergenza al data rate più basso, qualora il jitter raggiunga o superi la soglia dei 500 millisecondi.

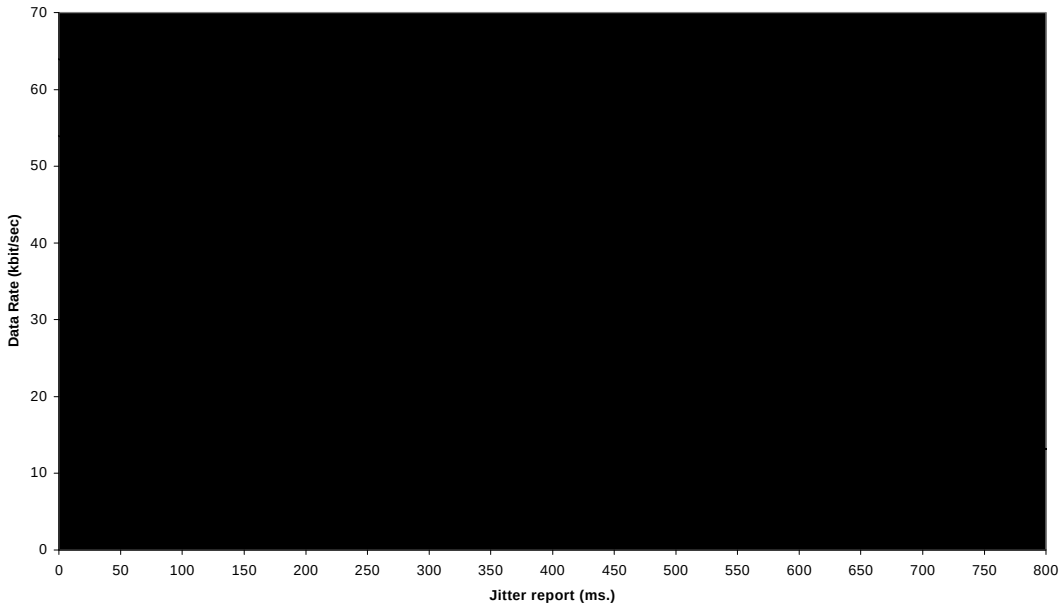

Figura 4-5: Comportamento del fattore *ß* in tre situazioni diverse.

4.6 Implementazione del meccanismo

Descriviamo brevemente, come è stato implementato il meccanismo di controllo, inserito all'interno della nostra applicazione. La gestione del controllo del bit rate è affidata a colui che sta trasmettendo l'audio, mentre il receiver ha il compito di fornire i report, periodicamente, con le informazioni opportune.

Il receiver, deve quindi mantenere aggiornate le informazioni sul jitter e sulle perdite, per ogni pacchetto che riceve. Nel momento in cui deve essere inviato un report al sender, viene chiamata la funzione *receiver_report()*, che provvede a calcolare la percentuale istantanea delle perdite e il jitter, per quell'intervallo. Queste informazioni, sono poi inserite in un pacchetti di tipo *RREPORT* e spedite al sender.

Il sender, nel momento in cui riceve un pacchetto di report, attiva la funzione *sender_report()*, la quale esegue il vero e proprio meccanismo di controllo del bit rate, utilizzando le informazioni contenute nel pacchetto.

La percentuale delle perdite istantanee, viene utilizzata per aggiornare la media pesata delle perdite, dopo di che, si effettua l'aggiornamento del max_rate. L'ultimo passo, consiste nell'utilizzare il jitter per adattare il max_rate allo stato corrente della rete, ottenendo così, il nuovo data rate di output, che verrà utilizzato per la trasmissione dei pacchetti successivi.

4.7 Riassunto

In questo capitolo abbiamo analizzato, approfonditamente il meccanismo per il controllo adattivo del bit rate. In principio, abbiamo illustrato, alcuni aspetti teorici importanti. In seguito dopo aver valutato, le scelte adottate in altri tool audio real-time, ci siamo soffermati sulla valutazione del jitter e delle perdite. Infine, abbiamo illustrato il meccanismo da noi sviluppato, analizzandone gli aspetti teorici ed implementativi.

Nel prossimo capitolo, valuteremo, attraverso una approfondita sperimentazione, le scelte effettuate nello sviluppo del meccanismo e l'efficienza dello stesso, mettendolo anche a confronto, con il sistema adottato in Free Phone.

Capitolo 5

Sperimentazione

Nel presente capitolo, saranno illustrati i risultati delle prove effettuate, con l'intento di dimostrare, a livello sperimentale, la bontà delle scelte e, soprattutto, valutare l'efficienza del meccanismo sviluppato.

5.1 Basi sperimentali

I test realizzati, avevano come obiettivo sia la valutazione delle scelte effettuate, durante lo sviluppo del nostro meccanismo di controllo, sia quello di giudicare l'efficienza del sistema, mettendolo a confronto con il meccanismo adottato in Free Phone. Per questo motivo, non è stato possibile effettuare sperimentazioni comparative in ambiente reale, su Internet, a causa della variabilità della rete. Il fatto di dover effettuare test di confronto, per valutare l'efficienza del meccanismo, richiedeva la medesima situazione per più prove successive; abbiamo quindi indirizzato i nostri test, su una base simulativa.

Sono stati predisposti, due scenari diversi, sui quali effettuare le nostre prove. Il primo scenario, permetteva di simulare ritardi di rete reali, il secondo permetteva di testare il meccanismo su banda limitata. Descriviamo, brevemente, i due scenari, per avere un'idea più precisa, di come sono state effettuate le sperimentazioni.

5.1.1 Ritardi di rete reali

Questa serie di test simulativi, ci ha permesso di valutare il meccanismo, sulla base di ritardi di rete reali. Per avere la possibilità di effettuare prove di questo genere, ci siamo serviti di tracce di ritardi, ottenute attraverso il monitoraggio del tratto di rete, che collega il Dipartimento di Scienze dell'Informazione di Cesena, con il C.E.R.N di Ginevra. Per la misurazione

dei ritardi effettivi di rete, senza la necessità di dover effettuare una sincronizzazione, tra l'orologio del trasmettitore e quello del ricevitore, è stata utilizzata la medesima macchina (situata al CSR di Cesena), per la spedizione e la ricezione dei pacchetti, mentre la seconda macchina (situata al CERN), aveva la funzione di *echo-server*. In questo modo, l'istante di trasmissione e quello di ricezione, sono stati misurati sulla stessa macchina, quindi la differenza tra i due, rispecchiava ritardi di rete reali. La situazione è illustrata in figura 5-1.

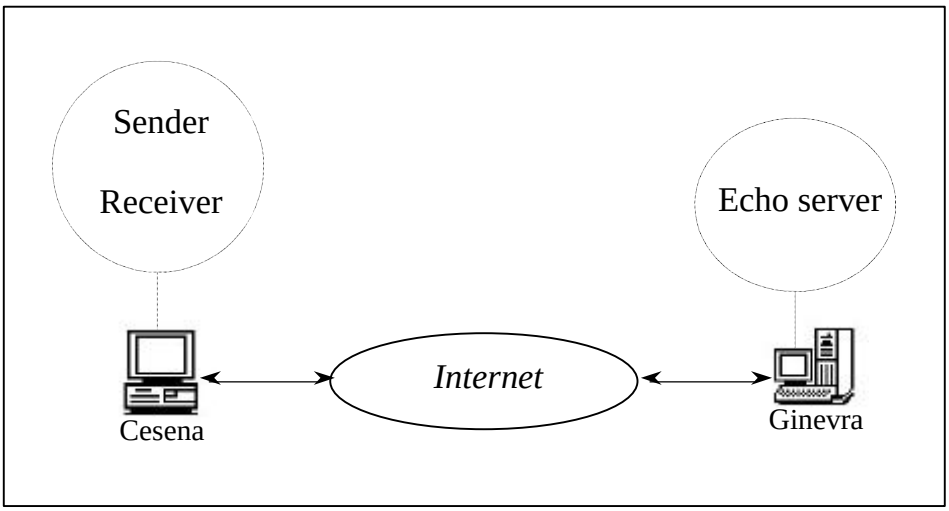

Figura 5-1: Architettura per la misurazione dei ritardi su Internet.

I ritardi ottenuti, sono definiti *round trip time* (RTT), e come si evince dalla figura, sono misurati su un percorso di andata e ritorno. E' possibile considerare con buona approssimazione, che i ritardi *end-to-end* siano la metà dei round trip time, poiché il tragitto dei pacchetti sarebbe dimezzato. Vediamo ora come è stata effettuata la simulazione dei ritardi di rete durante le fasi di sperimentazione.

Per effettuare i test sono state utilizzate due macchine, un Pentium a 100 Mhz con 32 MB di RAM e un Pentium II 266 Mhz con 128 MB di RAM. Le due macchine sono state collegate tramite un cavo di rete Ethernet a 10 Mbit/sec. Su entrambe le macchine, è stata "lanciata" la nostra applicazione audio, inoltre sulla macchina più veloce, venivano mandati in esecuzione

due processi sviluppati da noi, che simulavano i ritardi di rete, utilizzando le tracce ricavate precedentemente.

Ogni pacchetto che doveva essere spedito da una macchina all'altra, prima di raggiungere il destinatario, veniva filtrato da uno dei due intermediari, che provvedeva a ritardarlo di una quantità precisa, ricavata dalla traccia di ritardi. In figura 5-2 è illustrata l'interazione tra i vari processi coinvolti nei test.

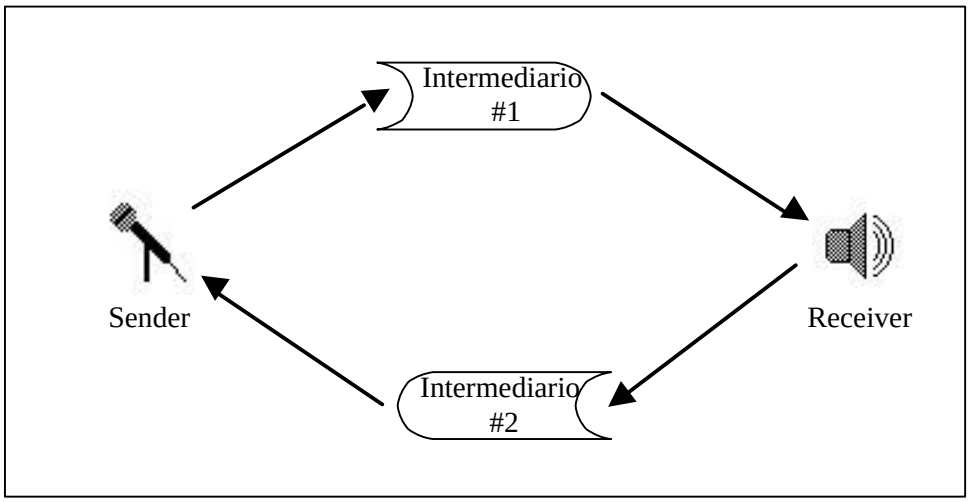

Figura 5-2: Interazione tra i processi utilizzati nei test effettuati tramite simulazione della rete.

Questo tipo di prove, ci hanno permesso di effettuare un confronto completo, tra il nostro meccanismo di controllo del bit rate, e quello adottato da Free Phone. Le sperimentazioni, effettuate in questa situazione, non hanno permesso di avere una risposta dalla rete, nel momento in cui si agiva sul data rate, in quanto i ritardi erano fissati a priori e non potevano variare, al variare della quantità di byte spediti. L'obiettivo di queste prove è stato quello di valutare l'efficienza del nostro metodo, attraverso una serie di confronti con il metodo utilizzato in Free Phone, dal punto di vista della velocità di adattamento alla variazione dei ritardi e della precisione negli interventi.

5.1.2 Larghezza di banda limitata

I test effettuati con lo scenario precedente, come abbiamo visto, non ci hanno fornito informazioni sulla risposta dalla rete, nel momento in cui il meccanismo di controllo della bit rate, andava ad agire sul data rate. Per avere un riscontro di questo tipo, che ci permettesse di apprezzare l'efficacia del meccanismo, è stato necessario eseguire delle prove su una connessione con una larghezza di banda limitata. Prove di questo tipo ci hanno permesso di valutare sia l'efficienza del meccanismo nel risolvere situazioni di congestione, sia l'efficienza nello sfruttare la massima quantità di banda disponibile, evitando ovviamente, di congestionare la rete. Non è stato possibile eseguire questo tipo di test su Internet, in quanto, non vi era la possibilità di appurare la quantità di banda utilizzata dall'applicazione, e di conseguenza, poter valutare l'efficienza del sistema nello sfruttamento della banda stessa. Su una connessione a banda limitata, al contrario, ad ogni istante si è a conoscenza della quantità di banda disponibile, e di quella realmente utilizzata dall'applicazione.

Le due macchine utilizzate precedentemente, sono state in questo caso connesse tra loro tramite un collegamento seriale. Ovviamente è stato necessario utilizzare un cavo di tipo *null-modem* (piedini 2 e 3 invertiti) per realizzare il collegamento fisico. Il sistema operativo Linux integra nel proprio *kernel* il supporto per il collegamento seriale *point-to-point*. La gestione di questo tipo di connessione viene fatta attraverso un demone di sistema chiamato *pppd*. Lo schema realizzato è raffigurato sotto.

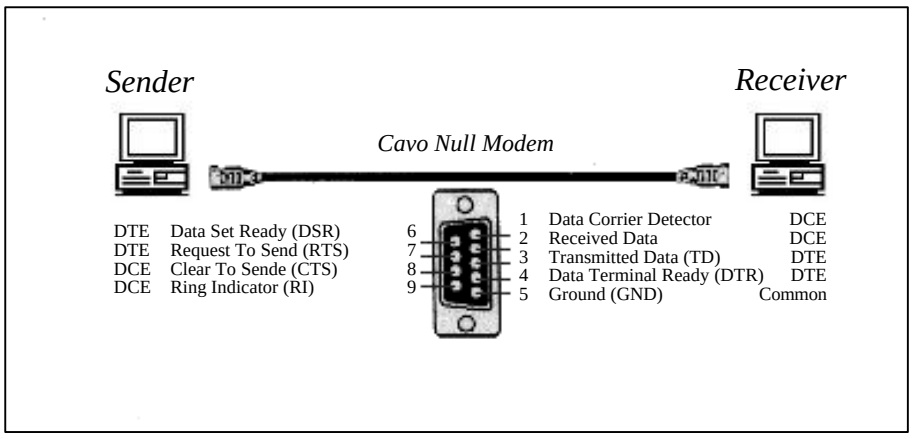

Figura 5-3: Collegamento di due PC tramite cavo seriale.

Realizzata la connessione fisica, il controllo della banda passante è stato effettuato tramite il comando *setserial*, che permette di settare la velocità della porta seriale a sottomultipli di 115200 bps. Come si può notare la larghezza di banda nominale, è più che sufficiente a supportare il data rate più elevato utilizzato dalla nostra applicazione (64 Kbit/sec).

Per simulare stati di congestione della rete abbiamo settato la velocità massima della connessione a 57,6 Kbit/sec in alcune prove e a 38,4 Kbit/sec, in altre. In questo modo la banda disponibile, non permetteva all'applicazione di trasmettere al data rate più elevato, ed era di conseguenza costretta a adattarsi alla banda disponibile.

Attraverso prove di questo tipo, siamo riusciti ad analizzare l'efficienza della meccanismo, con differenti valori per i parametri che ne regolano il funzionamento, analizzando la risposta fornita dalla rete, in termini di riduzione nei ritardi e nella perdita di pacchetti. Siamo così riusciti ad effettuare una regolazione di tali parametri, con l'obiettivo di ottimizzare il funzionamento del sistema di controllo del data rate.

Dopo aver illustrato i due scenari, utilizzati per effettuate le prove, nei paragrafi successivi, vengono forniti i risultati ottenuti durante i test. Prima si analizzano i risultati che hanno portato alla definizione dei parametri che regolano il funzionamento del meccanismo. In seguito analizziamo il confronto tra il nostro sistema e Free Phone.

5.2 Scelta dei parametri di controllo

In questo paragrafo, cerchiamo di giustificare tutte le scelte effettuate, nella definizione dei valori assegnati ai parametri, che regolano il meccanismo di controllo. I parametri che sono stati sottoposti ad uno studio accurato sono: HWM e LWM che regolano l'intervento del meccanismo e le percentuali di incremento e riduzione del data rate.

Nell'analisi che ha determinato la scelta dei valori, si è cercato di trovare un compromesso, tra la velocità di reazione ai cambiamenti della rete e la

precisione negli interventi, onde evitare di diminuire troppo, o troppo poco, il data rate. Dei quattro parametri presi in esame, due sono stati definiti a priori, sulla base di considerazioni teoriche, che ci hanno permesso di poter scegliere valori opportuni, senza dover effettuare prove sperimentali.

I due parametri definiti, senza l'ausilio di prove sperimentali sono: la soglia inferiore di intervento LWM e la percentuale di diminuzione del data rate. LWM è stato fissato al 5%, valore che a nostro avviso, indica un buon compromesso tra velocità di adattamento, e cautela nell'intervento. Abbiamo ritenuto, infatti, che, qualora la media pesata del tasso di perdita si mantenga sopra questa soglia, non vi sia possibilità di aumentare il data rate, in quanto il numero di pacchetti persi, si rivela acusticamente importante. Nel momento in cui la media scende sotto questa soglia, si può invece procedere con un aumento graduale della quantità di byte da utilizzare per la codifica.

Il secondo parametro che abbiamo fissato a priori, è stata la percentuale di diminuzione del max_rate. Per la scelta di un valore opportuno, ci siamo basati sulla seguente considerazione: qualora lo stato di congestione della rete peggiori, è necessario diminuire il data rate molto velocemente, per tentare di porre rimedio alla situazione. Il meccanismo di controllo, deve in tal caso avere la possibilità di diminuire drasticamente il data rate, in breve tempo. Basandoci su questa ipotesi, abbiamo scelto una percentuale di diminuzione del 15%. Questo valore permette di passare dal data rate più elevato (64 Kbit/sec) a quello più basso (13,2 Kbit/sec) in dieci passaggi e, cosa più importante, nell'arco di solamente quattro interventi, la quantità di byte utilizzati per la spedizione, viene pressoché dimezzata. Non è stata scelta una percentuale più elevata, per evitare che ad ogni passaggio da una codifica migliore, ad una più scarsa, la qualità dell'audio, ne risentisse in misura troppo elevata.

Rimangono ora da analizzare, i due ultimi parametri, che regolano il meccanismo di controllo, che sono: la soglia superiore d'intervento HWM e la percentuale di incremento del data rate. Per questi due parametri sono

stati effettuati diversi test, durante i quali abbiamo provato diverse combinazioni di valori.

Gli esperimenti sono stati ripetuti sotto condizioni differenti di larghezza di banda. In entrambi i casi, sono state effettuate prove, nelle quali il comportamento del meccanismo aveva un comportamento più aggressivo e prove, dove invece si aveva una tendenza più conservativa, nell'adattamento alla banda disponibile.

Per il parametro HWM, sono stati utilizzati due valori di prova, sui quali effettuare i test sperimentali. Abbiamo utilizzato una soglia pari al 15% come in Free Phone e una pari al 10%, valore più coerente con le valutazioni teoriche, che indicano come limite imprescindibile per la qualità dell'audio, una percentuale di perdita non superiore al 10%.

Per quanto riguarda la percentuale di incremento del data rate, i valori adottati, durante i test, sono stati 5, 10 e 15 per cento. Abbiamo utilizzato questo range di valori, per avere la possibilità di valutare il meccanismo, in situazioni di comportamento più o meno aggressive.

Analizziamo ora i risultati ottenuti, separatamente, per le due diverse situazioni di larghezza di banda, non prima tuttavia, di avere precisato, per completezza, che in entrambi i casi la quantità di banda definita, è utilizzata per la spedizione di pacchetti completi. Ogni pacchetto contiene un overhead aggiuntivo, dovuto agli header UDP, PPP e alle informazioni di controllo da noi aggiunte, che restringe ulteriormente di circa 8 Kbit/sec, la quantità di banda per i dati audio veri e propri. Ne consegue che per entrambe le situazioni la banda effettiva per i dati, ammontava a 30 Kbit/sec e 50 Kbit/sec circa.

5.2.1 Test con larghezza di banda a 38,4 Kbit/sec

In questa situazione, la banda disponibile per l'applicazione era notevolmente ristretta, di conseguenza, minime oscillazioni nel data rate determinavano una congestione della connessione, con conseguente perdita di pacchetti.

Sulla base dei risultati ottenuti, riportati in tabella 5-1, si deduce che una soglia del 15% per il parametro HWM, risulta troppo elevata, con conseguente ritardo nell'attivazione del meccanismo, per cercare di diminuire il data rate. In una situazione di questo tipo, dove la banda disponibile è molto scarsa e, dove di conseguenza, la minima oscillazione può portare ad un aumento delle perdite, è assolutamente fondamentale un intervento tempestivo del meccanismo, cosa che non accade con una soglia di intervento troppo elevata.

	HWM = 10%	*HWM = 15%*
Media Data rate	24,52 Kbit/s	26,76 Kbit/s
Media % perdite	7,04	9,56

Tabella 5-1: Risultati ottenuti dai test su HWM con banda a 38,4 Kbit/sec.

Dalla tabella possiamo valutare, come utilizzando una soglia pari al 10%, si ottiene una ragguardevole diminuzione della percentuale media di perdite, mentre la media del data rate, ne risente in misura minima.

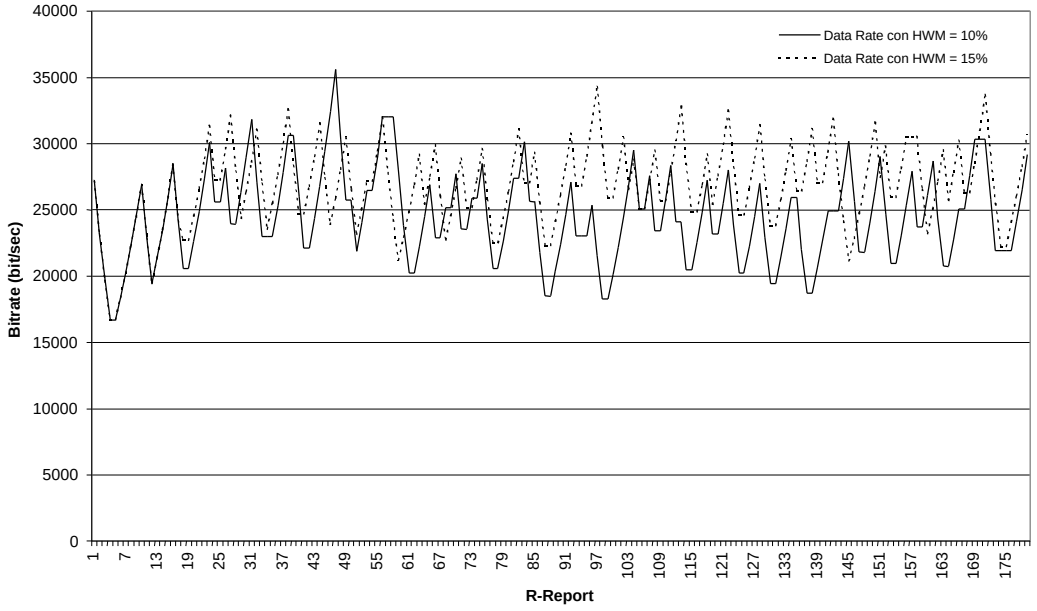

Figura 5-4: Confronto sulla capacità di adattamento all'ampiezza di banda limitata variando il parametro HWM, su una connessione a 38,4 Kbit/sec.

Attraverso il grafico di figura 5-4 dove viene illustrato il data rate effettivo (ossia la quantità di bit utilizzati per la codifica), si può notare che con HWM pari al 15%, il data rate supera la soglia dei 30 Kbit/sec, che dovrebbero essere il limite massimo per la codifica delle informazioni audio, un numero di volte maggiore rispetto al caso in cui, HWM rimane fissato al 10%. Ciò è dovuto al fatto che con una soglia di intervento più bassa, il meccanismo si attiva prima e di conseguenza, si evita al data rate di andare oltre la soglia limite troppe volte.

Per quanto riguarda il parametro di incremento del data rate, anche in questo caso, abbiamo rilevato che in situazioni di banda molto limitata, è consigliabile mantenere un comportamento prudente nell'aumento del data rate. Utilizzando ad esempio una percentuale di incremento pari a 15, si corre il rischio di incrementare il data rate troppo in fretta, con la conseguenza di avere troppe oscillazioni tra periodi durante i quali la connessione non è soggetta a congestione e periodi invece, di alta congestione.

Una percentuale del 5%, potrebbe essere considerata in questa situazione di banda, un ottimo valore, in quanto permette di mantenere bassa la percentuale di perdite e non vi è molta differenza per quanto riguarda la media del data rate, rispetto alle due scelte restanti. Vedremo tuttavia, nei test con una banda più ampia, che un incremento troppo lento, a lungo andare, porta ad un utilizzo insufficiente della banda disponibile. Questo fatto è più evidente nelle situazioni dove la larghezza di banda è maggiore. Se ne deduce che il valore che permette di avere un ottimo compromesso tra velocità di reazione e cautela nell'intervento, equivale ad una percentuale del 10%. Nella tabella successiva, è possibile valutare i risultati ottenuti durante le prove, che riassumono quanto detto in precedenza.

	Incremento 5%	*Incremento 10%*	*Incremento 15%*
Media Data rate	22,86 Kbit/sec	23,11 Kbit/sec	25,63 Kbit/sec
Media % perdite	5,11	7,32	10,25

Tabella 5-2: Risultati ottenuti dai test sulla percentuale di incremento con banda a 38,4 Kbit/sec.

Per completezza mostriamo anche il grafico (figura 5-5), dove viene messo a confronto l'andamento del data rate, con le tre diverse percentuali di incremento.

Dal grafico si può valutare come utilizzando una percentuale di incremento pari al 15%, il data rate raggiunge picchi più elevati, e se da un lato, ciò permette di sfruttare meglio la banda, dall'altro la rete entra in uno stato di congestione più velocemente, con il conseguente aumento nella percentuale di pacchetti persi e nei ritardi. Poiché la differenza tra le medie dei data rate non è elevata, è quindi consigliabile mantenere un approccio più cautelativo nella scelta del parametro di incremento.

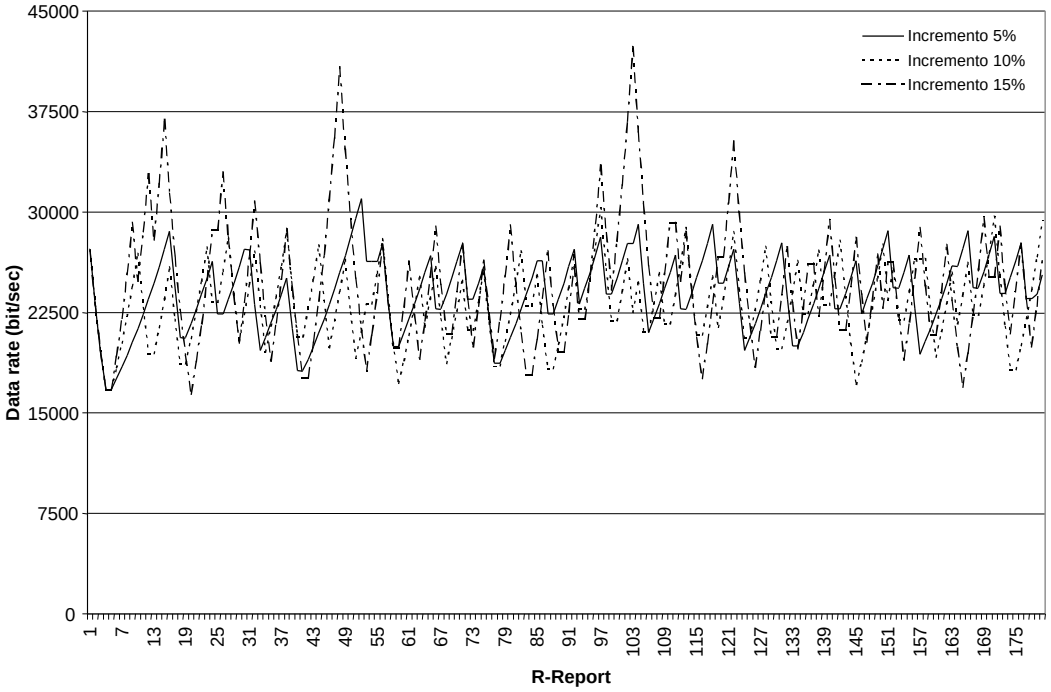

Figura 5-5: Confronto sulla capacità di adattamento all'ampiezza di banda limitata variando il parametro di incremento, su una connessione a 38,4 Kbit/sec.

5.2.2 Test con larghezza di banda a 57,6 Kbit/sec

Le prove effettuate su una connessione con una maggiore disponibilità di banda, ci hanno permesso di valutare il comportamento del meccanismo, in una situazione dove il data rate aveva un più ampio spazio di movimento e, di conseguenza, variando i parametri di controllo, le differenze in termini di

data rate erano più evidenti. In questa situazione abbiamo cercato di valutare, oltre all'efficacia negli interventi per limitare la congestione, anche la capacità del metodo di sfruttare opportunamente la banda a disposizione.

Le valutazioni sulla scelta del valore da assegnare alla soglia superiore di intervento HWM, sono le medesime di quelle fatte nel caso precedente. Come si può valutare dal grafico seguente, una soglia di intervento pari al 15%, tende ad attivare il meccanismo in ritardo, permettendo al data rate di raggiungere più spesso valori troppo elevati (oltre la soglia dei 50 Kbit/sec) e di conseguenza, facendo aumentare lo stato di congestione della connessione. Al contrario un valore uguale al 10%, permette al meccanismo di intervenire più rapidamente, nel momento in cui la situazione va peggiorando.

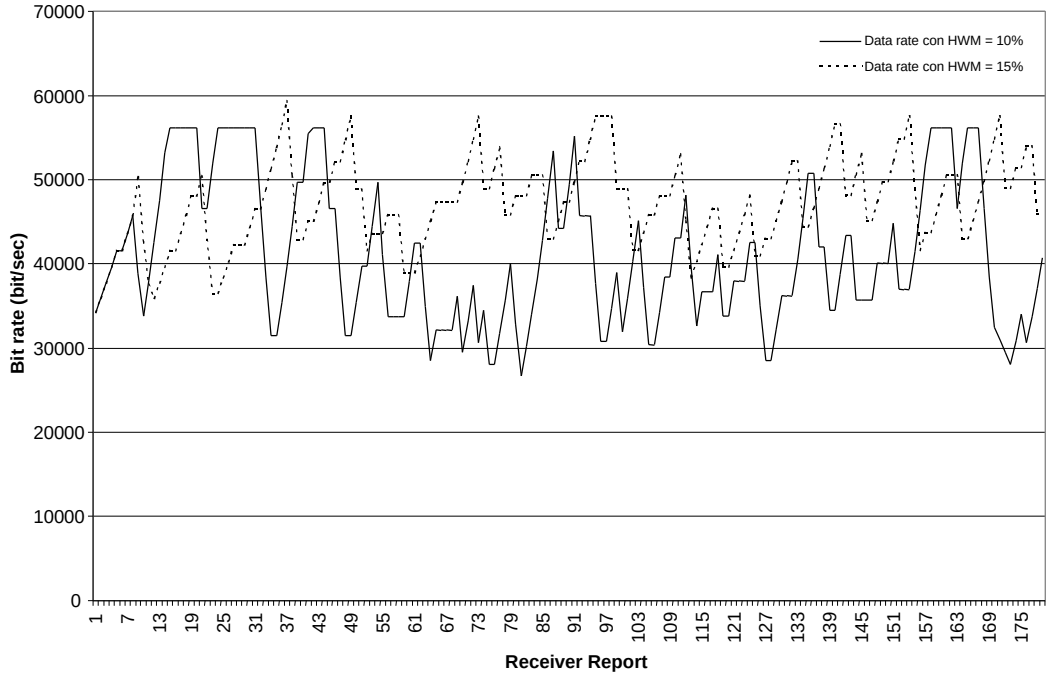

Figura 5-6: Confronto sulla capacità di adattamento all'ampiezza di banda limitata variando il parametro HWM, su una connessione a 57,6 Kbit/sec.

E' giusto rilevare, che utilizzando una soglia di intervento pari al 15% abbiamo un maggiore utilizzo di banda (vedi tabella 5-3), ma a nostro

avviso è più importante cercare di ridurre al minimo la quantità di perdite, sfruttando magari qualche kbyte in meno, in fase di codifica dell'audio.

	HWM = 10%	*HWM = 15%*
Media Data rate	41,66 Kbit/sec	47,05 Kbit/sec
Media % perdite	5,02	6,91

Tabella 5-3: Risultati ottenuti dai test su HWM con banda a 57,6 Kbit/sec.

Per finire analizziamo anche in questa situazione di banda la scelta del parametro di incremento. Con una banda più ampia, l'analisi del valore opportuno deve tenere conto, anche dell'utilizzo della banda. Si deve quindi cercare un compromesso tra sfruttamento della connessione e limitazione delle perdite e dei ritardi. La tabella 5-4 riassume la media dei data rate e le percentuali di perdita con i tre valori di incremento.

	Incremento 5%	*Incremento 10%*	*Incremento 15%*
Media Data rate	38,86 Kbit/sec	42,11 Kbit/sec	46,63 Kbit/sec
Media % perdite	4,61	6,02	9,47

Tabella 5-4: Risultati ottenuti dai test sulla percentuale di incremento con banda a 57,6 Kbit/sec.

Maggiore è la percentuale di incremento, più alto è lo sfruttamento di banda. Una quantità di byte più elevata in fase di trasmissione comporta tuttavia, una percentuale maggiore di perdite. E' quindi consigliabile mantenere un data rate più limitato, con il conseguente vantaggio, di diminuire le perdite e i ritardi di circa il 3% (utilizzando una valore di incremento del 10%).

Un incremento del 5%, non è stato considerato adatto, poiché comporta uno scarso utilizzo della banda (circa 12 Kbit/sec in meno, sul limite di 50 Kbit/sec), sebbene la percentuale di perdite sia estremamente bassa.

Analizzando il grafico relativo alla tabella (figura 5-7), notiamo i numerosi picchi che superano abbondantemente i 50 Kbit/sec, ottenuti con un incremento del 15%. In queste situazioni la rete si congestiona rapidamente con la conseguenza, che durante la trasmissione, si avranno periodi dove le percentuali perdite e i ritardi, saranno estremamente elevate. Al contrario con un incremento del 5%, il data rate raramente supera la soglia dei 50 Kbit/sec, questo è positivo in termini di carico della connessione (si avranno meno perdite e ritardi più contenuti), ma è limitativo dal punto di vista dello sfruttamento della banda.

Si può quindi individuare nel valore 10, la percentuale di incremento più appropriata, ottenendo un buon compromesso tra cautela nell'aumento del data rate e utilizzo in modo efficiente della banda che si ha a disposizione in ogni istante.

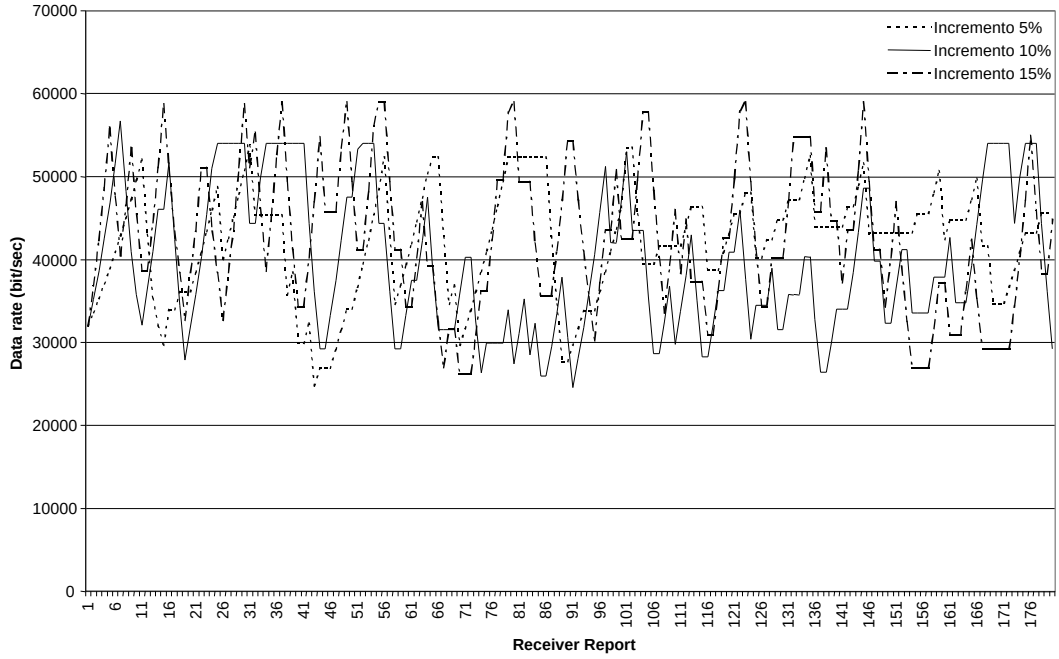

Figura 5-7: Confronto sulla capacità di adattamento all'ampiezza di banda limitata variando il parametro di incremento, su una connessione a 57,6 Kbit/sec.

Abbiamo illustrato in questo paragrafo i risultati che ci hanno portato alla scelta dei parametri, a nostro avviso più corretti, per il meccanismo di controllo del bit rate sviluppato all'interno dell'applicazione BoAT. Ci

sembra che i valori scelti, siano coerenti con gli obiettivi che ci eravamo prefissi quando abbiamo iniziato lo sviluppo del meccanismo: reattività negli interventi e accuratezza nella definizione del data rate.

Nel prossimo paragrafo, valuteremo l'efficacia del meccanismo, mettendolo a confronto, con quello adottato dall'INRIA in Free Phone.

5.3 Misurazione delle prestazioni fornite dal meccanismo

Dopo avere dimostrato sperimentalmente, le scelte effettuate per la definizione dei parametri che regolano il meccanismo, in questo paragrafo, procediamo ad illustrare i test effettuati per la valutazione del sistema di controllo del data rate nel suo insieme. I risultati sono messi a confronto con quelli ottenuti utilizzando il meccanismo di controllo adottato in Free Phone.

Suddividiamo i test effettuati, in tre gruppi: quelli riguardanti il jitter, quelli inerenti alle perdite ed infine quelli completi nei quali si può valutare il meccanismo nel suo insieme.

5.3.1 Test di valutazione della funzione di controllo del jitter

Valutiamo di seguito la qualità del funzione di stima del jitter, attraverso l'analisi dei risultati ottenuti durante i test sulla funzione. Nel grafico di figura 5-8, è rappresentata una traccia di 9000 valori di relative transit time (poiché i tempi di partenza e di arrivo sono stati presi sulla stessa macchina, coincidono con i round trip time), equivalenti a 6 minuti di monitoraggio (9000 x 40 ms. = 6 min.). Viene inoltre presentata la media dei ritardi all'interno di un intervallo di report e il minor ritardo rilevato. Dal grafico si può dedurre come sia giustificata la scelta di considerare all'interno del meccanismo di controllo del bit rate, la variazione dei ritardi, per regolare il data rate. Un meccanismo basato solamente sulla perdita di pacchetti, non sarebbe in grado di effettuare regolazioni, qualora le perdite fossero irrilevanti, ma siano presenti ritardi simili a quelli illustrati nel grafico. Infatti, la mancanza di perdite non implica necessariamente che la rete sia

decongestionata, mentre la variabilità nei ritardi può indicare la difficoltà che i router attraversano, durante un periodo di trasmissione dei pacchetti.

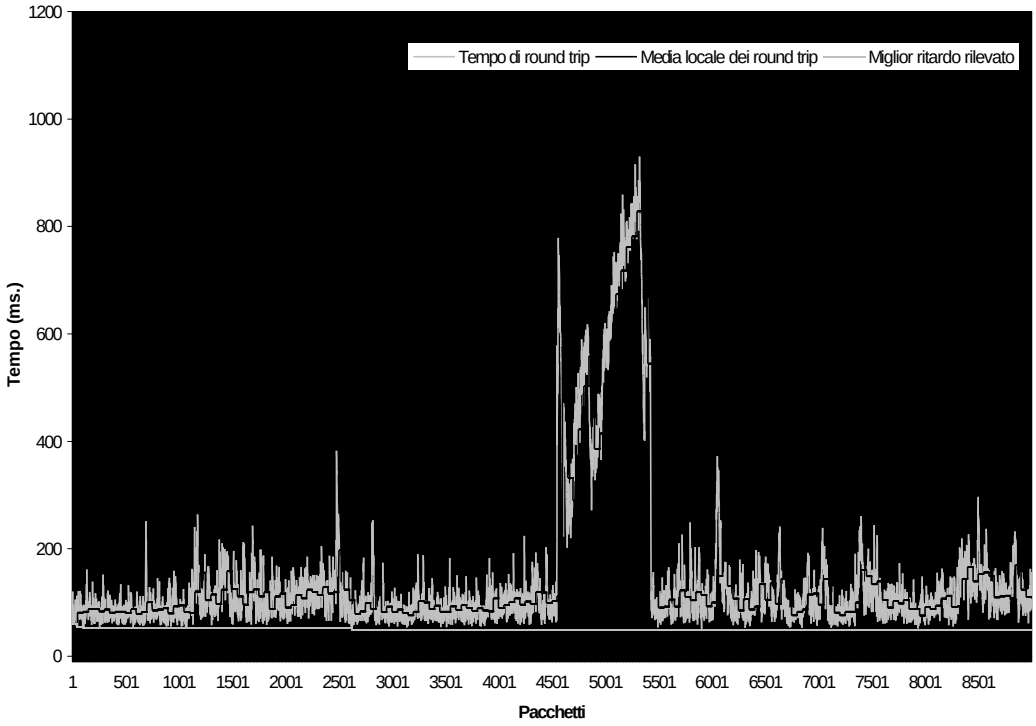

Figura 5-8: Monitoraggio dei relative transit time.

L’andamento della media locale, come si può vedere dal grafico, è leggermente spostato verso destra, rispetto all’andamento dei round trip time, poiché riassume l’andamento dei ritardi che si sono avuti negli ultimi due secondi.

Il calcolo del jitter da noi proposto, come scostamento della media locale dal miglior ritardo, è in grado di rappresentare, con una misura espressa in millisecondi, il livello di congestione transiente presente sulla rete.

Nel grafico successivo mostriamo l’andamento del data rate (in funzione dei jitter inviati ogni due secondi tramite i Receiver Report), riferito alla traccia dei ritardi del grafico di figura 5-8.

E’ evidente come il data rate abbia un andamento simmetrico (ma rovesciato), rispetto a quello dei ritardi. Maggiore è lo scostamento tra il

minimo ritardo e la media nell'intervallo, minore sarà il data rate. E' importante notare come, mediamente, il data rate sia elevato (51,23 Kbit/sec), cosa che indica una buona qualità dell'audio trasmesso.

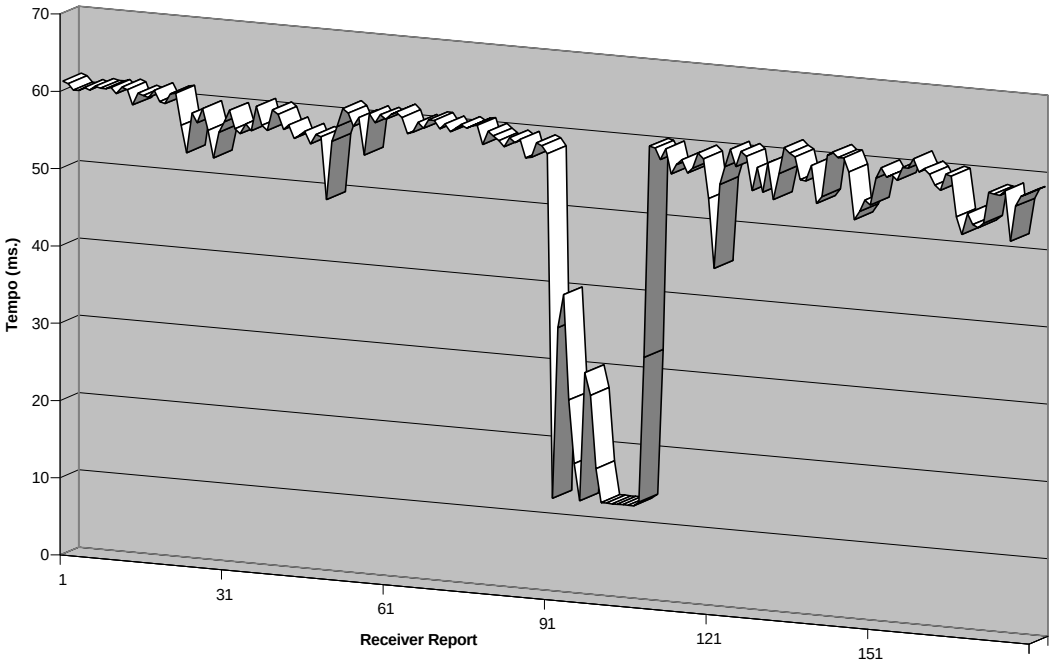

Figura 5-9: Aggiustamento automatico del bit rate in base al jitter.

L'incidenza del jitter sulla qualità dell'audio spedita, è maggiore nei casi in cui i ritardi aumentano e come conseguenza si ha una diminuzione del data rate. Confrontando il grafico del data rate con quello precedente dei ritardi, si può vedere come il punto più basso nella qualità dell'audio (13,2 Kbit/sec), coincida con un periodo di alta congestione, nel quale i ritardi sfiorano quasi il secondo. I receiver report racchiusi nell'intervallo compreso dal 94 al 111, riescono perfettamente a fornire informazioni precise sullo stato della rete, grazie alla frequenza con la quale vengono spediti (ogni 2 secondi), che permette di aggiornare con velocità lo stato della rete. Infatti, la diminuzione del data rate è immediata, così come il successivo aumento, nell'istante in cui la situazione ritorna alla normalità. Nel resto della traccia il data rate si mantiene sempre sopra il 25 Kbit/sec, evitando quindi di influire troppo pesantemente sulla quantità di bit da usare per la trasmissione. Rispetto ad una trasmissione senza controllo del data

rate (o con controllo basato unicamente sulle perdite), vi è un risparmio di circa il 20% nell'occupazione di banda, senza che ciò influisca sulla qualità della conversazione.

5.3.2 Test di valutazione sul metodo di misurazione delle perdite

Analizziamo ora, la parte del meccanismo basata sulle perdite, effettuando un confronto con la tecnica adottata in Free Phone. Ricordiamo che in Free Phone, per la valutazione del data rate, incide solo la percentuale di perdite istantanee ottenute ad ogni report, mentre nel nostro metodo, si utilizza sia la media pesata della percentuale di perdite, sia la stima del jitter, per regolare opportunamente il data rate.

Il grafico seguente riassume l'aggiustamento del data rate effettuato da entrambi i meccanismi durante una simulazione di sei minuti di trasmissione. I valori rappresentati in questo grafico, ricavati con il nostro meccanismo, non sono definitivi per la codifica, ma rappresentano solamente il limite superiore alla quantità di byte che debbono essere usati (max_rate), ai quali andrà eventualmente sottratto il jitter.

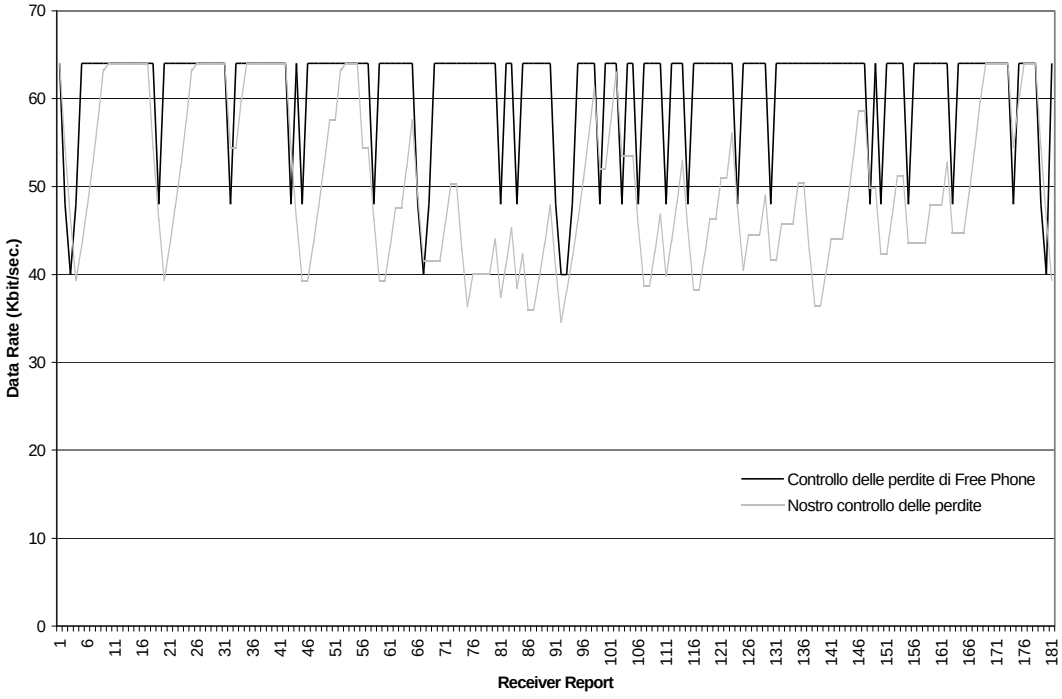

Figura 5-10: Il nostro meccanismo a confronto con quello di Free Phone, considerando solo le perdite per la regolazione del bit rate.

Dal grafico si evince come i due meccanismo abbiano un comportamento sensibilmente differente nella definizione del data rate, sulla base delle informazioni ricevute attraverso i Receiver Report. Si evidenza innanzi tutto, come il nostro sistema abbia un andamento più regolare, rispetto a quello di adottato dall'INRIA. Ciò è dovuto al fatto, che noi consideriamo l'informazione sulle perdite, per dare una valutazione dello stato di congestione permanente della rete, valutando quindi gli aumenti o le diminuzioni di congestione nella sua globalità. Ne consegue, che l'andamento del data rate non evidenzia salti improvvisi, ma aumenti e diminuzioni graduali. Al contrario in Free Phone, utilizzando le perdite istantanee e quindi valutando la congestione transiente, il meccanismo segue ogni singola variazione dello stato di congestione, ed ha quindi un comportamento più brusco. Queste continue oscillazioni del data rate, si traducono in variazioni continue del codec utilizzato per la compressione dei dati, con il conseguente rischio che ne risenta la qualità dell'audio. Ricordiamo che Free Phone adotta otto codec fissi, e molto spesso il passaggio da un tipo di codifica all'altro, si ripercuote sull'audio.

Altro punto che a nostro avviso fa ritenere il meccanismo adottato nell'applicazione da noi sviluppata, più appropriato, sta nella maggiore reattività ai peggioramenti nello stato della rete. Infatti, come si deduce dal grafico, nel momento in cui la rete peggiora, il nostro meccanismo, sebbene adotti una media pesata per valutare la congestione, si attiva in genere prima di Free Phone. Ciò è dovuto al fatto che la soglia oltre la quale si interviene sul data rate, nel nostro caso è stata fissata al 10%, mentre nel meccanismo dell'INRIA è stata posta al 15%. Questa maggiore velocità di intervento, si traduce in una diminuzione più marcata del data rate.

E' interessante notare ciò che succede nell'intorno del report 136. Free Phone, in questo intervallo, mantiene il bit rate al valore più elevato (64 Kbit/sec), mentre il nostro meccanismo, dopo aver aumentato il data rate fino a 50 Kbit/sec circa, riduce progressivamente il bit rate portandolo fino a 30 Kbit/sec, prima di ricominciare a salire. Il motivo di questa discrepanza tra i due metodi, sta nella differente soglia di intervento adottata. Infatti, come abbiamo già detto in precedenza, Free Phone

interviene nel momento in cui le perdite raggiungono il 15%. In questa situazione in cui la percentuale di perdite si attesta sul 13% circa, Free Phone continua ad utilizzare il suo codec migliore, mentre il nostro sistema, avendo una soglia di intervento più bassa (10%), reagisce e provvede ad abbassare il bit rate. Consideriamo più corretto il comportamento del nostro meccanismo, in quanto, in situazioni di perdite superiori al 10 per cento, il rate di spedizione dovrebbe essere opportunamente abbassato, per cercare di compensare la situazione non favorevole.

Anche nelle fasi in cui la rete migliora, e quindi si procede ad aumentare il rate di spedizione, il nostro meccanismo ha un comportamento mirato principalmente ad evitare di ricongestionare nuovamente la rete, di conseguenza il data rate si riporta ai valori più alti con maggiore cautela, rispetto a Free Phone.

In generale possiamo riassumere che la componente del meccanismo che agisce sul data rate in base alle informazioni sulle perdite, per ciò che riguarda il nostro sistema, tende a considerare maggiormente lo stato globale della rete, mantenendo un andamento più regolare, con lo scopo di evitare continue variazioni che a nostro avviso, rendono la conversazione più sgradevole.

Concludiamo mostrando una tabella riferita la grafico precedente, nella quale è riassunta la media del data rate ottenuto con i due metodi, a parità di perdite (nell'esperimento la media si attesta sul 6,5%), dalla quale si rileva, che il sistema da noi sviluppato, tende a limitare maggiormente l'utilizzo di banda, senza che ciò influisca, se non in minima parte, sulla qualità dell'audio.

	Free Phone	*BoAT*
Media bit rate	61,04 kbit/sec	50,31 kbit/sec

Tabella 5-5: Media del data rate ottenuto con entrambi i metodi, con perdite fisse (media 6,5%).

5.3.3 Valutazione complessiva del nostro meccanismo

Dopo avere analizzato l'andamento del data rate in funzione del jitter e delle perdite separatamente, non rimane che valutare il comportamento complessivo del nostro sistema. Nel grafico successivo mostriamo l'andamento del data rate regolato dal meccanismo di controllo nel suo insieme. Anche in questo caso per valutarne meglio il comportamento, si effettua un confronto con il meccanismo utilizzato in Free Phone.

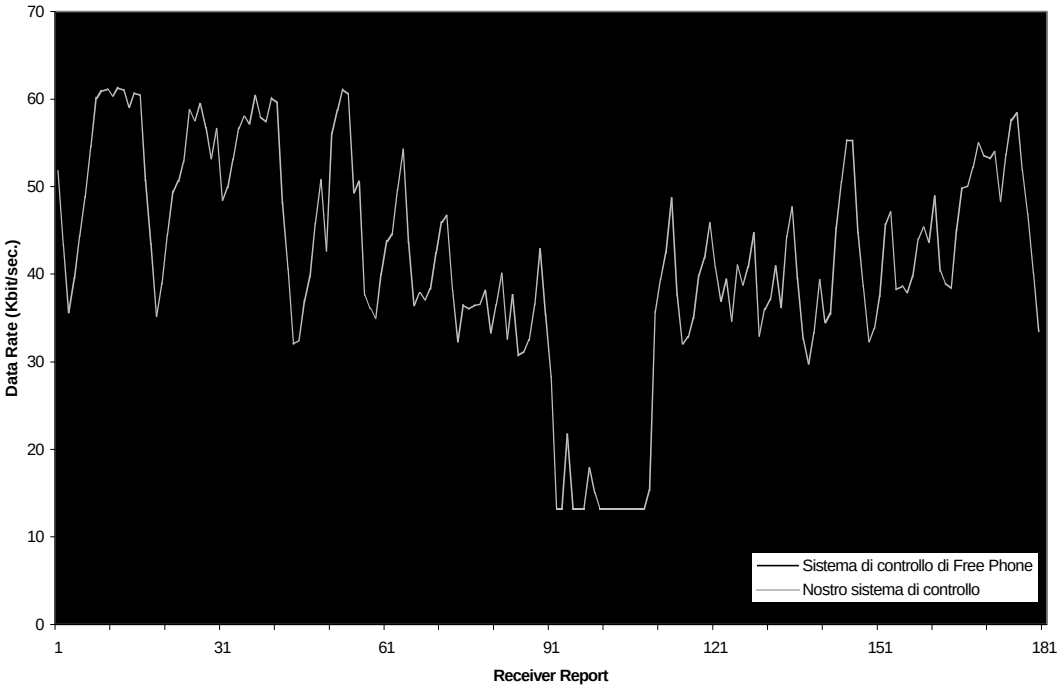

Figura 5-11: Il nostro meccanismo di controllo a confronto con quello di Free Phone, considerando sia le perdite, sia il jitter, per la regolazione del bit rate.

Dal grafico si evidenzia come il data rate stimato con il nostro meccanismo non raggiunga mai il rate di spedizione più elevato, sebbene in alcuni casi ci si avvicini. Il motivo è chiaramente l'incidenza sia delle perdite, sia del jitter, sul calcolo del data rate. Ad ogni modo, nonostante questa doppia incidenza, il data rate si mantiene in media sui 45 Kbit/sec, valore che non degrada minimamente la qualità sonora. Si può rilevare quindi, che il nostro sistema di controllo adattivo del bit rate, ha un utilizzo più oculato della banda disponibile. Questo comportamento fa si che in situazioni di carico elevato della rete, il nostro controllo permette di

mantenere le perdite e i ritardi su livelli più accettabili, grazie all'utilizzo di una quantità di byte minore per la trasmissione dei pacchetti.

Possiamo invece rilevare, come il nostro meccanismo si adatti meglio alla situazione della rete, attraverso l'utilizzo combinato di informazioni relative alle perdite e alla varianza dei ritardi. Infatti, possiamo vedere come il meccanismo adottato in Free Phone si mantenga per gran parte del tempo sui 64 Kbit/sec. Ciò significa che le perdite rimangono sotto il 15% e quindi, il meccanismo adottato nell'applicazione dell'INRIA non interviene. Il fatto che le perdite si mantengano sotto la soglia di intervento stabilita da Free Phone, non significa tuttavia, che la rete non sia in una situazione di carico pesante. E' in queste situazioni che interviene il jitter, il quale permette di interpretare meglio lo stato della rete e di limitare opportunamente il data rate di spedizione per cercare di ripristinare la situazione, mentre Free Phone si mantiene fisso sui 64 Kbit/sec.

Un ulteriore punto a favore del nostro sistema, è la maggiore velocità di reazione, nei momenti in cui la rete peggiora e il bit rate deve essere abbassato. Come si può valutare dal grafico, in tutte le situazioni in cui si deve diminuire il bit rate, il nostro meccanismo anticipa sempre Free Phone nell'effettuare, tali diminuzioni. Ciò indica, a nostro avviso, una migliore lettura e interpretazione dello stato della rete, rispetto a Free Phone, il quale tende sempre a reagire con maggiore lentezza.

Notiamo infine, che nell'intervallo di report, intorno al 90-esimo, la rete appare altamente congestionata (perdite intorno al 12% e jitter sui 700 ms.) e, come si può vedere, il nostro meccanismo agisce più velocemente portando il data rate al minimo, mentre Free Phone non scende sotto i 40 Kbit/sec. Ciò denota a nostro avviso, una maggiore efficienza del meccanismo da noi sviluppato, nella definizione del data rate più opportuno, a seconda delle condizioni della rete.

Nel momento in cui la situazione tende a migliorare, il nostro meccanismo si riporta gradualmente, su valori più elevati, mantenendo sempre il giusto compromesso tra velocità e accuratezza.

Concludiamo il confronto tra BoAT e Free Phone, mostrando una tabella riepilogativa, la quale riassume la media del bit rate, relativa al grafico precedente. Anche in questo caso la media delle perdite era fissata al 6,5% circa. Dalla tabella si rileva come il meccanismo sviluppato in BoAT, sfrutti meno pesantemente la banda, mantenendosi su valori di bit rate più bassi. Da ciò ne deriva che, se da un lato la qualità dell'audio spedito risulta più scarsa, a causa della minore quantità di byte utilizzati per la codifica, dall'altro si ha una diminuzione rilevante nella percentuale di perdite e ritardi, e questo porta ad un miglioramento complessivo della qualità della conversazione.

	Free Phone	*BoAT*
Media bit rate	61,04 kbit/sec	47,73 kbit/sec

Tabella 5-6: Media del data rate ottenuto con entrambi i metodi, a parità di perdite (6,5%).

Per la valutazione complessiva del nostro sistema di controllo adattivo del bit rate, integrato all'interno dell'applicazione BoAT, vogliamo evidenziare, per concludere, alcuni risultati ottenuti attraverso test effettuati su banda limitata a 57,6 kbit/sec.

Il primo risultato, mostra i benefici che è possibile ottenere adottando il nostro meccanismo di controllo, rispetto al caso in cui non si faccia uso di alcun sistema di regolazione del bit rate.

I due grafici seguenti, riassumono l'andamento delle perdite (nella medesima situazione) senza l'utilizzo di un sistema di controllo del bit rate (figura 5-12), e con l'ausilio del nostro meccanismo di controllo (figura 5-13), su banda limitata. Dal primo grafico si evidenzia come trasmettendo con un bit rate costante (generalmente il rate massimo di trasmissione per un audio tool è di 64 Kbit/sec), qualora la connessione non permetta di trasmettere a questa velocità, a causa del carico a cui è sottoposta, le perdite tendono ad essere sempre molto elevate. La media della percentuale di perdite si attesta, infatti, sul 20% circa. Da questa situazione, si rileva che

un’applicazione audio sprovvista di un meccanismo di controllo del bit rate, qualora trasmetta ad una velocità superiore rispetto a quella sostenibile dalla rete sottostante, si trova ad avere ritardi e soprattutto perdite estremamente elevati, tali da compromettere irrimediabilmente la qualità della conversazione.

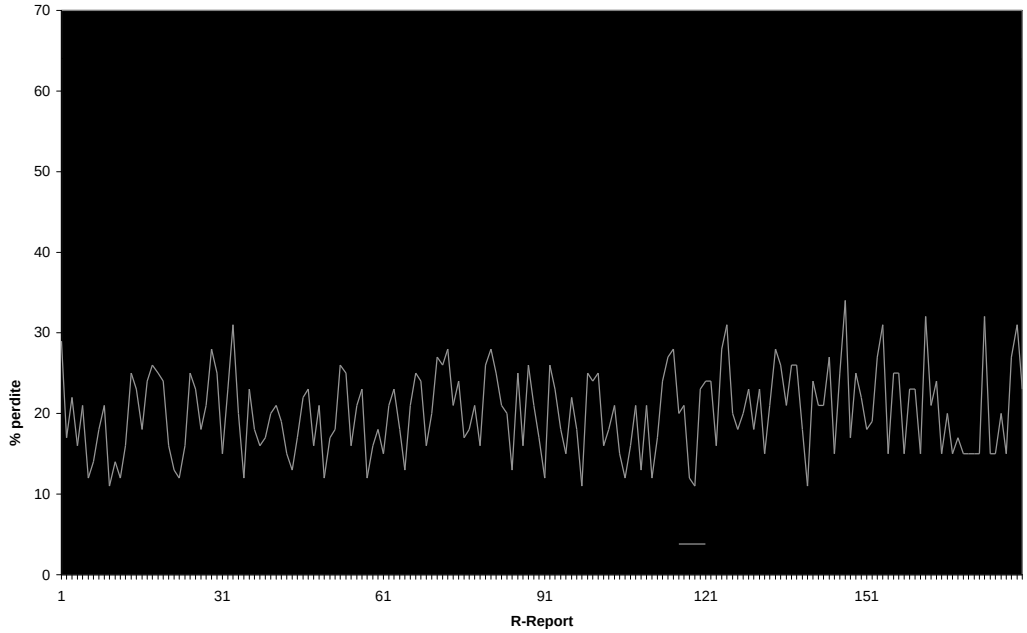

Figura 5-12: Andamento delle perdite, senza sistema di controllo del bit rate.

Al contrario, i benefici che un sistema di controllo del bit rate apporta, in termini di diminuzione delle perdite, attraverso un controllo accurato sulla quantità di byte utilizzati in fase di trasmissione sono rilevanti, come si può osservare dal secondo grafico.

In questo caso la media delle perdite si attesta sul 5,5%, con una notevole diminuzione rispetto al caso precedente. E’ giusto puntualizzare che una tale diminuzione nella percentuale delle perdite, la si ottiene solamente limitando il rate di trasmissione; infatti, in questa seconda situazione, la media del bit rate è di circa 45 Kbit/sec. Ciò significa che la qualità della voce recepita dall’ascoltatore, sarà leggermente più scarsa. Tuttavia questa minore qualità è compensata, dal maggior numero di pacchetti audio ricevuti.

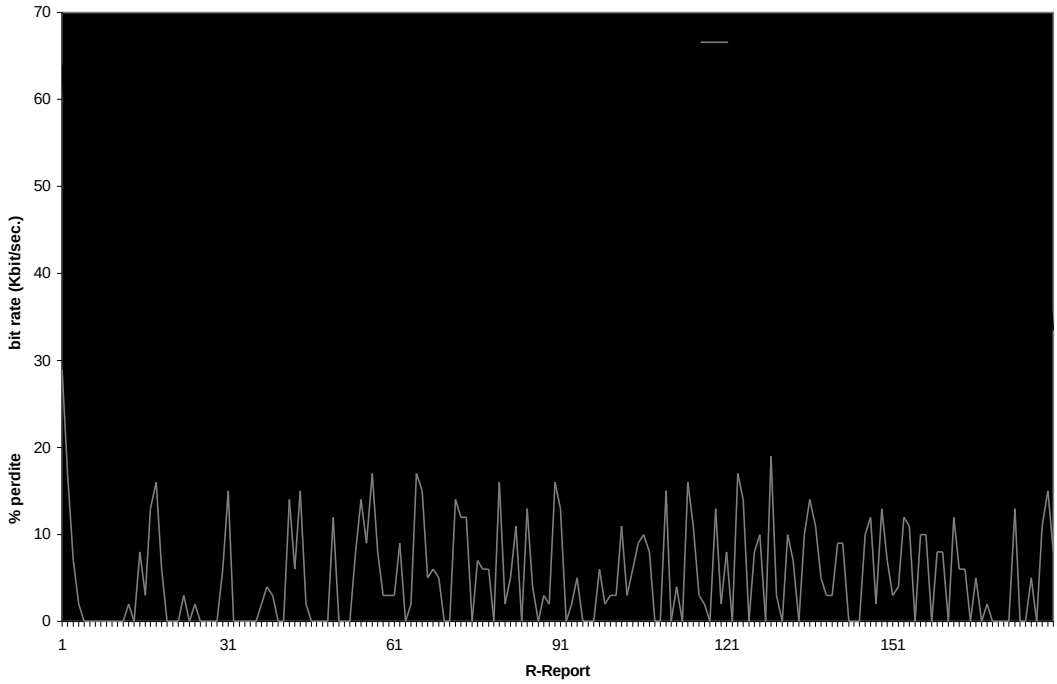

Figura 5-13: Andamento delle perdite, con l'utilizzo del sistema di controllo del bit rate, sviluppato in BoAT.

Il secondo risultato che, a nostro avviso, è interessante mostrare, riguarda l'andamento delle perdite, relative al grafico mostrato nel primo capitolo (figura 1-2). Il grafico evidenzia la sensibile diminuzione delle perdite, utilizzando il controllo del bit rate sviluppato in BoAT, rispetto a Free Phone.

Dalla figura 5-14, si rileva un andamento delle perdite più contenuto in BoAT, ed anche come i periodi con picchi di perdite siano sensibilmente più diradati, rispetto a Free Phone. Questo indica un controllo migliore dello stato della rete, che impedisce all'applicazione di spedire a data rate troppo elevati, nei momenti in cui il carico del traffico è notevole.

Si può quindi, ritenere raggiunto l'obiettivo che ci eravamo prefissi inizialmente, di sviluppare un sistema di controllo che fosse efficiente nell'adattarsi ai mutamenti della rete (soprattutto nei casi di peggioramento), ma anche preciso, nella definizione del data rate di trasmissione.

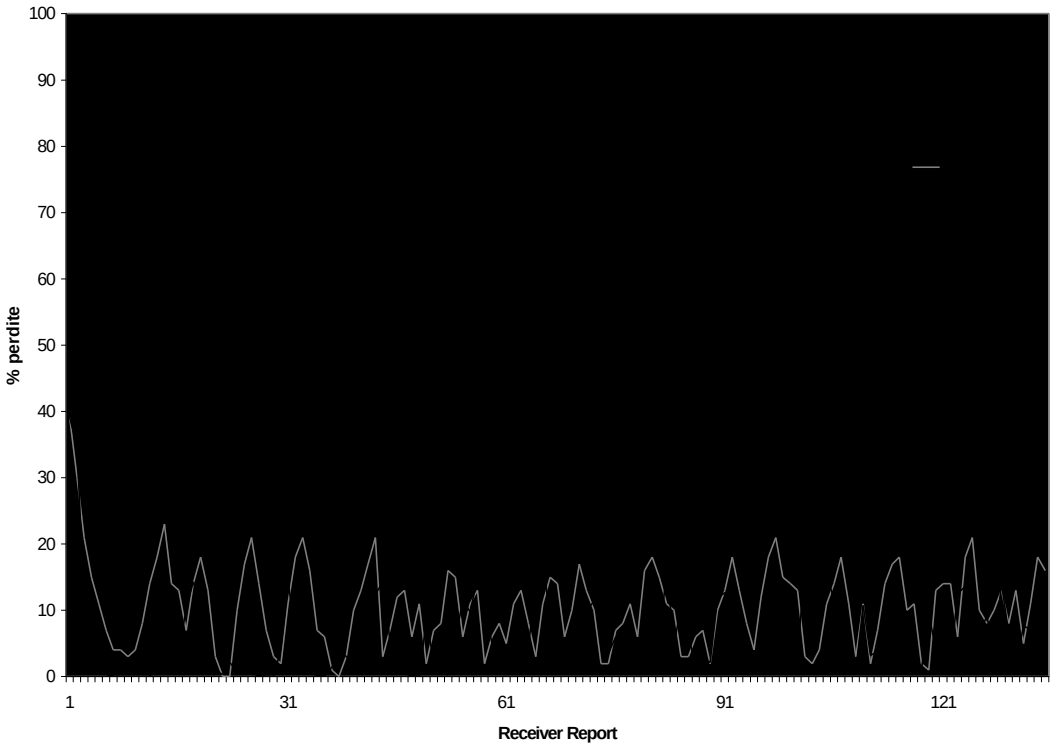

Figura 5-14: Confronto tra BoAT e Free Phone, sull'andamento delle perdite.

Mostriamo anche la tabella che riassume, per questo esperimento, la media del data rate e delle perdite, ottenute con i due meccanismi di controllo, che sono stati messi a confronto. E' importante notare che riducendo il bit rate di trasmissione, si ottiene una diminuzione rilevante nella percentuale delle perdite, di conseguenza, la minore qualità nell'audio spedito, è compensata dal maggior numero di pacchetti ricevuti.

	BoAT	*Free Phone*
Media % perdite	6,10%	10,56%
Media Data rate	42,72 kbit/sec	48,89 kbit/sec

Tabella 5-7: Riassunto delle prestazioni.

Possiamo concludere osservando che il meccanismo sviluppato, tende a cercare sempre un giusto compromesso tra qualità dell'audio (miglior bit rate di trasmissione) e risorse disponibili. I risultati, infatti, dimostrano che

il compromesso raggiunto, tra percentuale di perdite e bit rate, garantisce sempre buone prestazioni, limitando adeguatamente il data rate, con importanti benefici per quanto riguarda la qualità complessiva della conversazione.

5.4 Riassunto

In questo capitolo si è cercato di valutare la funzionalità del meccanismo per il controllo adattivo del bit rate, da noi sviluppato, attraverso approfonditi studi sperimentali. I risultati ottenuti, sono stati spesso confrontati con quelli ricavati utilizzando il meccanismo adottato in Free Phone, con l'obiettivo di verificare la bontà delle scelte effettuate.

Capitolo 6

Conclusioni

Durante questo lavoro di tesi, abbiamo sviluppato una completa applicazione software per supportare voce su Internet. Lo sviluppo del progetto si è articolato in più fasi distinte. E' stato inizialmente svolto un lavoro di ricerca e documentazione, per avere una conoscenza approfondita delle problematiche relative all'implementazione di un audio tool real-time.

La fase successiva è stata incentrata sulla progettazione e sviluppo dell'applicazione. L'implementazione è stata portata avanti a passi successivi e ad ogni passo, le funzionalità del programma venivano ampliate e ottimizzate. Inizialmente è stata costruita una semplice applicazione per la trasmissione di un flusso audio tra due estremità, cercando di ottimizzare tutti i passaggi, a partire dall'acquisizione dell'audio, fino alla riproduzione.

Terminato lo sviluppo dello scheletro dell'applicazione, è stato implementato il meccanismo di controllo del playout come descritto in [RGPSB 98]. Questa fase ha avuto un'evoluzione graduale; il meccanismo inizialmente implementato seguendo fedelmente le indicazioni teoriche, è stato via via migliorato, introducendo nuovi accorgimenti per ovviare ad alcuni problemi, emersi durante i primi esperimenti (vedi [Bal 99]).

Il passo successivo è stato quello della progettazione e sviluppo del meccanismo di controllo del bit rate. Inizialmente è stato inserito all'interno del tool il codec a data rate variabile, sviluppato in [Nal 98]. Successivamente è stato progettato e sviluppato il sistema di controllo, con lo scopo di porre rimedio alle scadenti qualità di servizio offerte dal best effert service di Internet. E' stato integrato anche un meccanismo di Forward Error Correction, per cercare di diminuire ulteriormente l'incidenza delle perdite sulla qualità dell'audio.

Particolare attenzione è stata posta nella creazione di sistemi di controllo completamente automatici: riteniamo infatti che un sistema manuale sia di difficile utilizzo da parte degli utilizzatori.

L'ultimo passo per quanto riguarda la fase di progettazione e sviluppo, è stata quella di fornire l'applicazione di una semplice interfaccia utente, sviluppata in ambiente X-Window, per rendere più agevole la gestione di alcuni controlli fondamentali di un'applicazione audio, come ad esempio la regolazione dei volumi di entrata e di uscita. Complessivamente il lavoro svolto, ha portato alla scrittura di circa 5500 linee di codice in linguaggio C.

L'ultima fase della tesi è stata quella relativa alla sperimentazione, durante la quale abbiamo studiato il comportamento dell'applicazione, sia nella sua globalità, sia soffermandoci sulle componenti più importanti del tool: il meccanismo di controllo del playout e il sistema di controllo adattivo del bit rate. Sono state effettuate anche prove in ambiente reale durante le quali l'applicazione ha avuto un buon comportamento. La fase di sperimentazione si è protratta per circa un mese e mezzo e durante questo periodo sono stati sviluppati alcuni programmi aggiuntivi, utili gli esperimenti effettuati.

Brevemente, per ciò che riguarda il meccanismo di controllo del bit rate, il lavoro più importante è stato quello di sperimentazione, che ci ha permesso di ottimizzare il sistema di controllo, attraverso una lunga serie di esperimenti.

Tutte le fasi inerenti lo svolgimento della tesi, hanno contribuito al raggiungimento dell'obiettivo che ci si era prefissi: lo sviluppo di un'applicazione per la trasmissione di audio in tempo reale su Internet, attraverso l'utilizzo di alcuni meccanismi innovativi, che ne migliorassero le prestazioni.

6.1 Sviluppi futuri

Durante lo sviluppo dell'applicazione, non sono stati presi in considerazione alcuni aspetti, per mancanza di tempo, che avrebbero

aumentato l'efficienza complessiva del tool. Illustriamo brevemente alcuni miglioramenti che potrebbero essere introdotti in futuro.

Per quanto riguarda problematiche relative all'audio, sarebbe opportuno implementare alcune tecniche per la soppressione di eventuali rumori di fondo e, più in generale, per migliorare la qualità dell'audio trasmesso. Inoltre i dispositivi di input (microfoni e *head-phone*), hanno comportamenti differenti tra loro, dovuti a diversi fattori, come ad esempio distanza dalla bocca, filtraggio dei rumori, ecc. L'applicazione dovrebbe quindi adattarsi automaticamente e in modo opportuno a queste differenze, adottando un meccanismo definito *automatic gain control*.

Il meccanismo di controllo del playout, acquisterebbe ulteriore efficienza, utilizzando un silence detector più efficace, il quale sia in grado di riconoscere, e separare, i periodi di silenzio da quelli di parlato, che sono presenti all'interno di una conversazione. Nella progettazione dell'applicazione, abbiamo preferito utilizzare un rilevatore del silenzio già pronto, e con discrete funzionalità, poiché sviluppare un nuovo silence detector adatto ai nostri scopi, avrebbe richiesto uno studio lungo e approfondito, e ciò andava oltre gli obiettivi di questa tesi.

Per ciò che riguarda il sistema di controllo del bit rate, una migliore integrazione tra il sistema e il meccanismo di Forward Error Correction, porterebbe ulteriori benefici all'intelligibilità della conversazione.

Infine, l'utilizzo del protocollo RTP per la trasmissione delle informazioni, produrrebbe ulteriori benefici per l'applicazione.

A prescindere da questi ulteriori miglioramenti, il comportamento dell'applicazione BoAT è, a nostro avviso, pienamente soddisfacente.