ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Triennale in Informatica

Monitoraggio della QoS in interfacce wireless per VoIP

Tesi di Laurea in Architettura degli Elaboratori

Candidato
Chiara Simoni

Relatore
Dott. Vittorio Ghini

Anno Accademico 2009-2010
Sessione III

Parole chiave: QoS, VoIP, wireless, monitoraggio, multihoming

INDICE

CAPITOLO 1
INTRODUZIONE

1.1 Presentazione del problema

Obiettivo di questo lavoro è stato progettare e sviluppare un software in grado di monitorare la qualità della connessione di un dispositivo ad una rete wireless.
Si prevede che sul dispositivo sia presente un'applicazione VoIP (Voice over Internet Protocol), per il cui buon funzionamento è necessario il rispetto di determinati requisiti di prestazione della connessione.
Nel presente elaborato sono perciò stati analizzati tutti gli elementi che influenzano la performance di una sessione VoIP seguendo le indicazione degli standard ITU-T per le telecomunicazioni, ed è stato creato un software che valuta la connessione di rete in base a questi parametri di QoS.

Lo scenario considerato è quello di un nodo mobile, connesso al suo Access Point, che a sua volta è collegato al gateway di default (spesso i due dispositivi coincidono), il quale instrada i pacchetti verso la rete esterna. Il nodo mobile è infine connesso ad un proxy, che ne gestisce la comunicazione con il nodo corrispondente durante la sessione VoIP.

Un dispositivo può avere diverse interfacce di connessione di rete wireless (ad es. WI-FI e UMTS); quando si collega ad un punto di accesso wireless esso usa una di queste interfacce, ma trattandosi un device mobile può, nel cambiare la sua posizione nel tempo, non rientrare più nella copertura dello stesso Access Point.
Il software descritto in questo documento si occuperà perciò di effettuare un controllo continuo sulla attuale connessione dell'interfaccia scelta, generando dei valori di valutazione indicativi della qualità della stessa, per permettere ad un ulteriore programma che analizzi queste stime, di decidere se annullare la connessione sulla corrente interfaccia per usarne un'altra più performante.

Durante l'esecuzione del programma, vengono analizzate in modo alternato la connessione con il default gateway, con il proxy e col correspondent node.
Nell'introdurre questo studio si intende sottolineare che la variabilità della qualità della connessione è più alta in corrispondenza del tratto di collegamento del dispositivo con l' Access Point e con il default gateway, poiché il collegamento in questione è di tipo wireless e il suo utilizzo previsto è mobile. Il lato che riguarda la rete esterna più lontana invece consiste solitamente in collegamenti wired fissi, dunque è più stabile nel tempo.
Per questo motivo il software progettato si occupa, nella maggior parte dell'esecuzione, di effettuare scansioni sul collegamento col default gateway e con meno frequenza su proxy e correspondent node.

Il programma implementato supporta entrambe le versioni IPv4 e Ipv6.

Questo programma è stato sviluppato utilizzando il già esistente protocollo di livello network ICMP, che definisce le regole per lo scambio di messaggi ICMP ECHO REQUEST/REPLY tra dispositivi diversi.

Il parametro decisivo per le prestazioni del VoIP è il one-way delay, perciò ad ogni scambio di questi messaggi ICMP esso viene calcolato come la metà del Round Trip Time (RTT). Risulta chiaro che la latenza ottenuta come metà del RTT non è un indicatore preciso, in quanto si tratta di un valore di media che non esprime il vero ritardo del singolo pacchetto; nonostante ciò questa è l'unica soluzione possibile in una applicazione eseguibile da un solo end-point della comunicazione, che in questo caso è il dispositivo mobile collegato alla rete tramite un'interfaccia wireless.

1.2 Contenuto della tesi

Qui di seguito saranno descritti gli argomenti trattati nei capitoli di questo documento.
Il capitolo 2 introduce lo scenario in cui si inserisce questo lavoro, trattando argomenti quali Quality of Service, Mobile IP e requisiti prestazionali del VoIP.
Il capitolo 3 definisce gli obiettivi del lavoro di ricerca e implementazione del programma.
Il capitolo 4 descrive la effettiva implementazione del programma sviluppato, quindi il suo funzionamento in generale e le scelte progettuali riguardo ai parametri di performance considerati per dare la stima sulla attuale connessione, infine viene fornita una descrizione più dettagliata del codice sorgente.
Il capitolo 5 elenca tutti i test eseguiti in laboratorio sul programma.
Il capitolo 6 esprime alcune conclusioni sullo studio svolto e offre uno sguardo ai possibili sviluppi futuri del presente lavoro.

CAPITOLO 2
SCENARIO

2.1 QoS

Si parla di QoS (acronimo di “Quality of Service”) come di qualità del servizio fornito, e in questo caso il servizio di cui si discute è la telefonia su reti a commutazione di pacchetto, conosciuto come VoIP, in contrapposizione con la tradizionale telefonia su reti a commutazione di circuito.
Il termine QoS è usato per indicare i parametri usati per caratterizzare la qualità del servizio offerto dalla rete (ad esempio perdita di pacchetti e ritardo), o gli strumenti per ottenere una qualità di servizio desiderata.

2.2 Wireless e Mobile IP

Il termine wireless indica una comunicazione tra dispositivi elettronici che non fa uso di cavi. Si tratta di un termine generico che comprende varie tecnologie di comunicazione, le più importanti sono:

- Wi-Fi
- UMTS
- GSM
- GPRS

L'architettura di base delle reti wireless è la seguente:

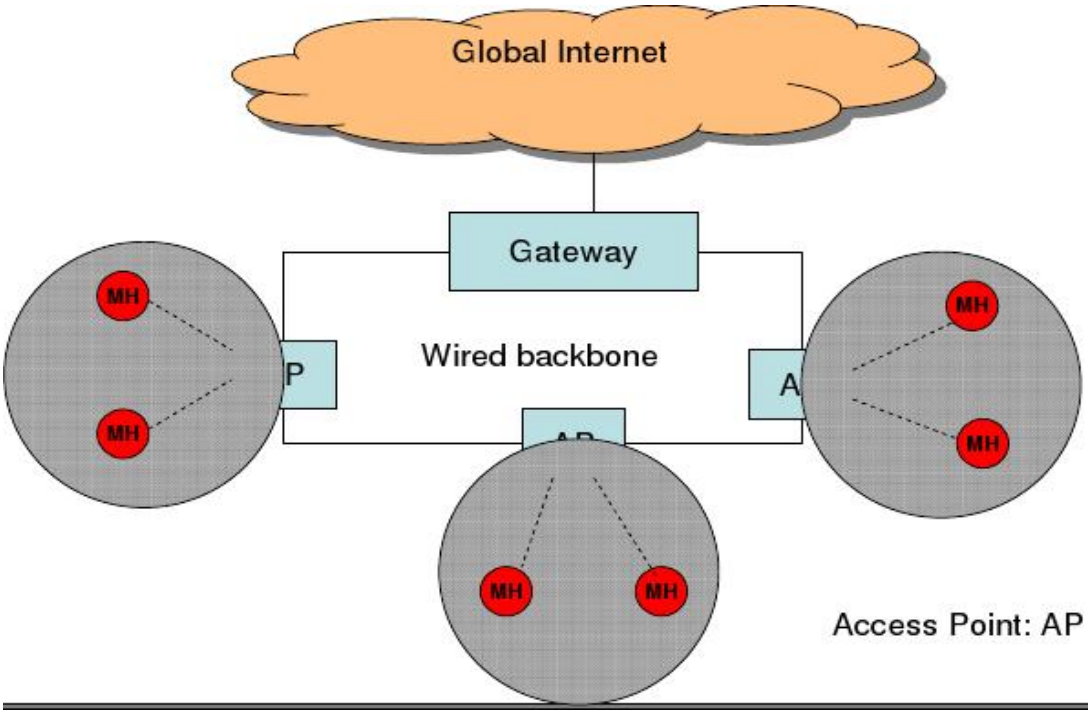


Figura 2.1: architettura di base delle reti wireless

Network	Velocità	Range	Spettro	Applicazioni	
802.11b	11Mbps	10-20m	Libero	LAN	
802.11e			Libero	LAN	
HIPERLAN	23,5 Mbps	50-100m	Libero	LAN, MAN	
Bluetooth	1 Mbps	10-100m	Libero	PAN	
GSM	9,6 Kbps	35 km	Licenziato	Outdoor	
GPRS	28,8 Kbps	35 km	Licenziato	Outdoor	
Edge	200 Kbps	35 km	Licenziato	Outdoor	
3G	144/384 kbps	10-35 Km	Licenziato	In/Out door	
HSDPA	14 Mbps	10-35 Km	Licenziato	In/Out door	
TETRA	7,2 Kbps	35-50 km	Licenziato	In/Out door	

Figura 2.2: reti wireless a confronto [Parr2009]

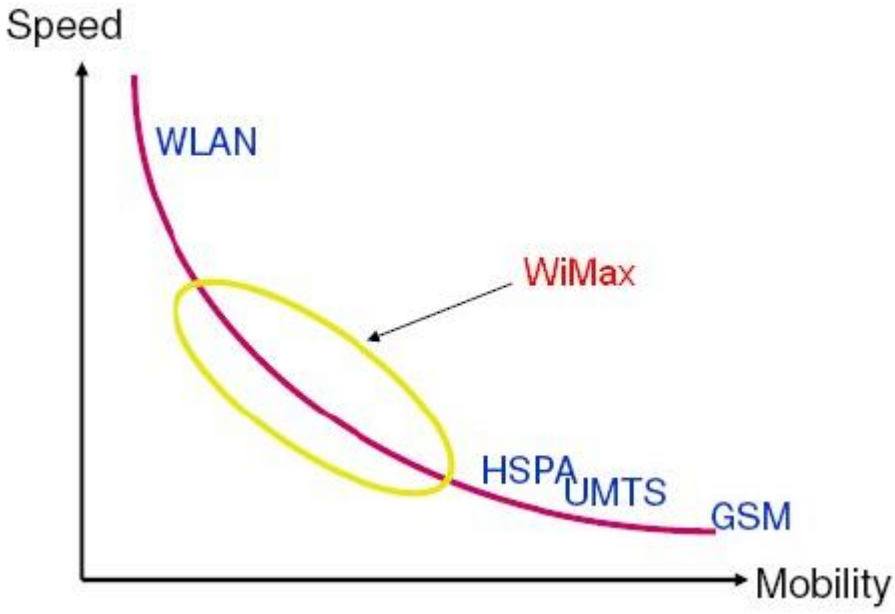


Figura 2.3: velocità delle reti wireless in relazione alla mobilità [Parr2009]

L'uso della tecnologia wireless sta aumentando di anno in anno, partendo dai 100 milioni di terminali mobili utilizzati nel 1997 a 1 miliardo nel 2009 [Parr2009]:

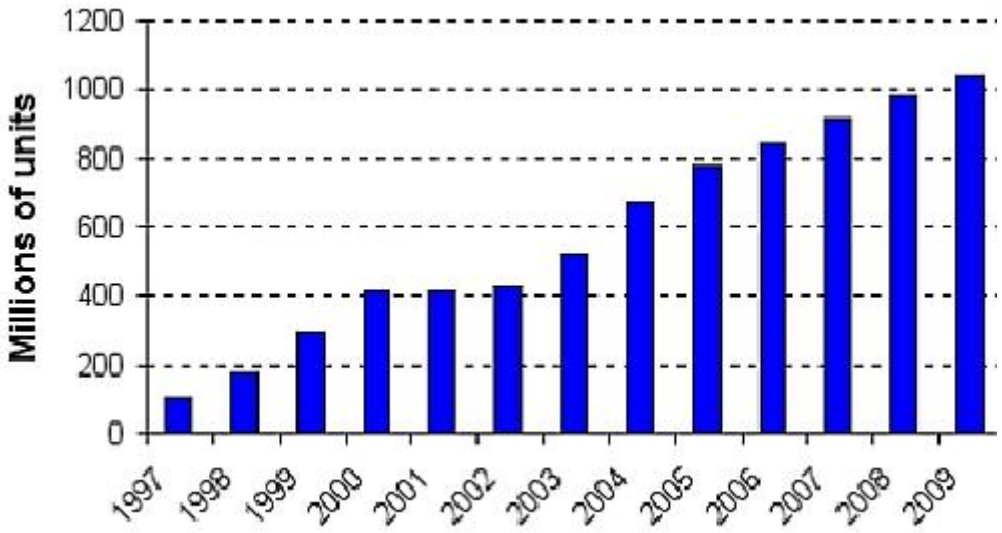


Figura 2.4: crescita del numero di terminali mobili

Nel 2002 l'ente IETF ha standardizzato il protocollo Mobile IP [IETF3344] che permette ai device mobili di muoversi da una rete ad un'altra mantenendo un indirizzo IP fisso.

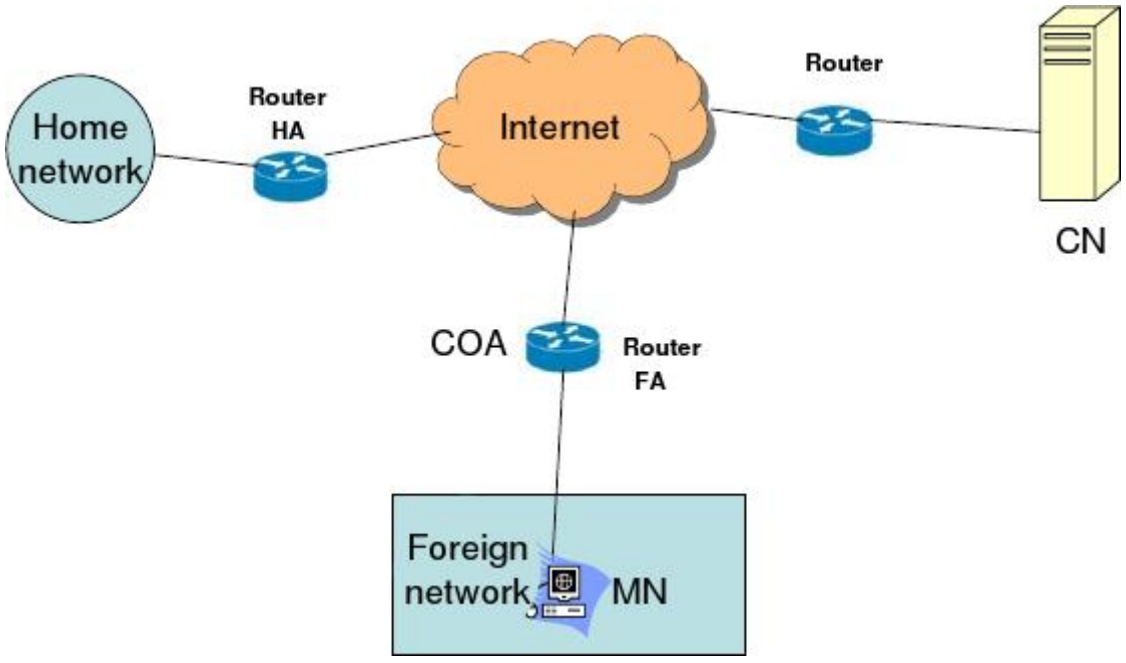


Figura 2.5: architettura del Mobile IP

in questa configurazione gli attori sono:

Mobile Node (MN) = host che cambia il punto di connessione da una rete o sotto-rete ad un'altra poiché cambia la sua posizione nel tempo. Un nodo mobile può cambiare la sua posizione senza cambiare il proprio indirizzo IP e può continuare a comunicare con altri nodi della rete Internet in qualsiasi posizione utilizzando il suo indirizzo IP.

Home Agent (HA) = router di una home network responsabile di inoltrare i datagrammi IP al nodo mobile, e che mantiene l' informazione della posizione corrente del il nodo di telefonia mobile.

Foreign agent (FA) = router in una foreign network che provvede al routing al MN registrato in FA. FA inoltra i pacchetti ricevuti da HA .

Correspondent node (CN) = un nodo in Internet che sta correntemente comunicando con un MN.

Care of address (COA) = tunnel usato per inviare pacchetti mentre un MN è fuori dalla sua rete domestica .

Un nodo mobile può avere due indirizzi: un indirizzo home permanente e un "*care-of-address*" (CoA), che è associato alla rete che il nodo mobile sta navigando.

L'implementazione del Mobile IP comprende due entità:

- un "*home agent*", che contiene le informazioni sul nodo mobile.
- *Un "foreign agent"*, che salva le informazioni sul nodo mobile che sta nella sua rete. I foreign agent sono a conoscenza del care-of-address, usato dal Mobile IP.

Un nodo che vuole comunicare col nodo mobile usa l'indirizzo home permanente del nodo mobile come indirizzo di destinazione dei pacchetti.
L' home agent redireziona quei pacchetti verso il foreign agent attraverso un tunnel IP incapsulando il datagramma con un nuovo header IP usando il care of address del nodo mobile.

Quando invece il nodo mobile vuole spedire pacchetti, esso li manda direttamente all'altro nodo con cui vuole comunicare tramite il foreign agent, senza mandare i pacchetti all' home agent, e usando il suo indirizzo home permanente come indirizzo sorgente dei pacchetti.
In questo modo si ottiene un routing triangolare.

2.3 VoIP e QoS

Ricercatori di varie università ed enti privati hanno condotto numerosi studi sulla qualità delle chiamate sia in fonia classica che con VoIP per cercare di stabilire dei parametri oggettivi e automatici di rilevazione della qualità, nonostante si tratti di un ambito relativamente soggettivo.
Successivamente a questi studi sono stati stabiliti degli standard sulla valutazione delle chiamate e sui requisiti di performance delle connessioni di rete per VoIP.
L'ente che ha prodotto queste recommendations è la ITU-T (International Telecommunication Union - Telecommunication Standardization Bureau);
per l'elenco delle recommendations prodotte sull'argomento fare riferimento all'appendice A.

2.3.1 Studi sulla QoS – da test soggettivi al modello generale

Lo studio della qualità di una chiamata parte con la definizione della scala MOS (Mean Opinion Score) data dallo standard ITU-T P.800 [ITUP.800] come un modello di riferimento per la stima della chiarezza di una trasmissione audio.
Essa è un indicatore della qualità dell'audio percepito dall'ascoltatore in una chiamata.
Questa scala di valori viene usata come metro di valutazione in test soggettivi di ascolto: in un tipico test di ascolto, i soggetti odono delle registrazioni processate attraverso 50 condizioni diverse di una rete, e votano su una semplice scala di opinioni di 5 punti, la cui media su tutti i soggetti, per ogni condizione, è chiamata MOS.
Il valore MOS è espresso con un numero da 1 a 5, in cui 1 è il massimo punteggio negativo, e 5 è il massimo punteggio positivo.

Di seguito la lista dei valori con il relativo giudizio:

MOS	QUALITY
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

Tabella 2.3.1.1: Mean Opinion Score (MOS)

Questi punteggi vengono dati in base alla chiarezza del segnale ascoltato durante una

chiamata e quindi in relazione inversa ai disturbi della comunicazione che si avvertono nella trasmissione.

Essendo però questo valore di qualità dell'ascolto relativamente soggettivo, il passo successivo per l'automatizzazione della valutazione dell'audio è dato dallo standard ITU-T G.107 [ITUG.107].
Esso definisce un modello parametrico computazionale, chiamato E-model, che intende stimare la qualità di una rete e quindi gli effetti combinati delle variazioni dei numerosi parametri di trasmissione che influiscono sulla qualità della conversazione telefonica.

L' E-model ha come output finale il “Rating Factor” R, si tratta di un intero tra 0 e 100, dove il valore 0 rappresenta una qualità estremamente bassa mentre il numero 100 una qualità molto alta dell'audio.
Il valore finale R è dato da una relazione additiva, cioè è calcolato come somma di vari fattori che rappresentano i vari disturbi che possono manifestarsi durante la trasmissione dell'audio:

$$R = Ro - Is - Id - Ie + A$$

Espressione 2.3.1.1: E-model

dove:

Ro = rapporto segnale/rumore
Is = combinazione di tutti i disturbi che avvengono più o meno simultaneamente col segnale vocale
Id = disturbi causati dal ritardo di arrivo dei pacchetti
Ie = disturbi causati dall'uso di codec con basso bit rate
A = elementi che compensano i fattori di disturbo

L' E-model è un modello oggettivo di valutazione della qualità di una trasmissione audio, ed è utile in quanto introduce una corrispondenza con il modello MOS di valutazione soggettiva dell'ascolto e con le categorie di qualità dell'audio definite dalla recommendation ITU-T G.109 [ITUG.109], la quale traduce i valori di R in livelli di soddisfazione dell'utente.

R (intervals)	MOS (intervals)	USER SATISFACTION LEVELS
90 - 100	4,34 – 4,5	Users very satisfied
80 - 90	4,03 – 4,34	Users satisfied
70 - 80	3,6 – 4,03	Some users dissatisfied
60 - 70	3,10 – 3,60	Many users dissatisfied
50 - 60	2,58 – 3,10	Nearly all users dissatisfied

Tabella 2.3.1.2: relazione tra i valori di R, di soddisfazione dell'utente e scala MOS

In questo modo si è in grado di valutare automaticamente la qualità della trasmissione audio, riconducendo le stime oggettive a valori soggettivi.

Una volta stabilita la scala di valutazione della generica trasmissione audio, si può passare ad analizzare la trasmissione audio su Internet, cioè il Voice Over IP, ed è quindi necessario fissare i requisiti di performance che essa deve rispettare.
Per conoscere questi parametri ci si può riferire allo standard ITU-T Y.1541 [ITUY.1541].
In questo documento si descrivono gli obiettivi di performance per servizi basati su IP, e per fare ciò i vari servizi sono stati suddivisi in 8 classi di QoS: l'appartenenza di un servizio ad una classe dipende da requisiti di interattività più o meno alta degli stessi e gli specifici valori di performance variano a seconda della classe di cui i servizi fanno parte.

Ecco la lista delle prime sei classi di QoS:

QOS CLASS	APPLICATIONS
0	Real-Time, jitter sensitive, altamente interattivo (VoIP)
1	Real-Time, jitter sensitive, interattivo
2	Transaction data, altamente interattivo
3	Transaction data, interattivo
4	Low loss only, short transactions, bulk data, video streaming
5	Tradizionali applicazioni delle reti IP di default

Tabella 2.3.1.3: prime 6 classi di QoS

I più importanti parametri di QOS del VoIP emersi da numerosi studi ([Walk2002], [Cam2001]) sono:

- One-way delay (latenza)
  Si tratta del tempo necessario a trasmettere tutti i bit del pacchetto dalla sorgente alla destinazione.
  E' una funzione della distanza fisica, del numero di apparecchiature attive e passive attraversate lungo il percorso e del carico istantaneo di rete.

- Burst packet loss (perdita di pacchetti in blocco)
  Gruppi di pacchetti consecutivi spediti, ma non ricevuti alla destinazione, oppure ricevuti in errore.

- One-way IP packet delay variation (jitter)
  Per un pacchetto all'interno di un flusso che va dal punto di misura MP1 ad MP2, è la differenza tra il one-way delay del primo pacchetto selezionato ed il one-way delay del secondo.

- Codec
  E' l'hardware o il software utilizzato per convertire il segnale analogico della voce in digitale e viceversa.

Dopo aver definito le classi di QoS, la recommendation Y.1541 fissa i requisiti di performance per ciascuna classe.

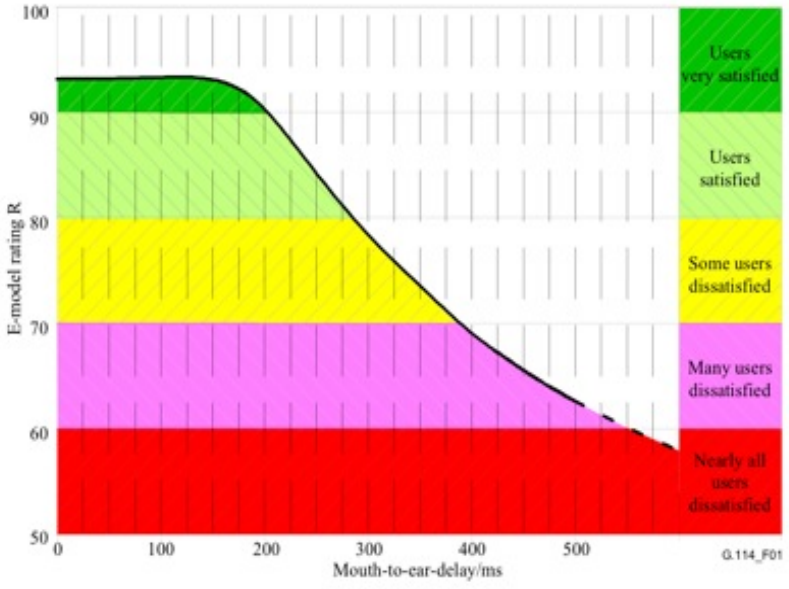
Di seguito sono mostrati quelli della classe 0 e 1:

NETWORK PERFORMANCE PARAMETERS	QOS CLASS 0 (VoIP)	QOS CLASS 1
MAXIMUM ONE-WAY DELAY	100 ms	400 ms
MAXIMUM JITTER	50 ms	50 ms
MAXIMUM PACKET LOSS	0,001%	0,10%
MAXIMUM PACKET ERROR	0,0001%	0,01%

Tabella 2.3.1.4: obiettivi di network performance

Il parametro più importante per la stima delle prestazioni del VoIP è il one-way delay, e una ulteriore recommendation, la ITU-T G.114 [ITUG.114] si è occupata di analizzare questo valore, in relazione al VoIP.
Secondo questo studio si raccomanda di non superare il one-way delay di 400 ms per le trasmissioni in generale, mentre per applicazioni interattive come il VoIP il limite di ritardo massimo scende a 150 ms [Org2010].
Riguardo all'uso dello E-model descritto in precedenza per "*speech applications*", gli effetti del ritardo possono essere controllati nel seguente grafo, in cui le ascisse rappresentano il mouth-to-ear delay in ms, e le ordinate mostrano le variazioni del fattore R dello E-model. Sono inoltre mostrate le categorie di qualità dell'audio della recommendation ITU-T G.109, che descrivono i livelli di soddisfazione dell'utente.

Grafico 2.3.1.6: determinazione degli effetti del ritardo assoluto tramite E-Model



CAPITOLO 3
OBIETTIVI

In seguito allo studio teorico delle prestazioni delle reti wireless rispetto al VoIP, si è progettato e sviluppato un software in grado di valutare le prestazioni di una connessione wireless secondo i parametri di qualità richiesti dal VoIP.

Il protagonista di questa trattazione è il Mobile Node che è collegato ad Internet via wireless e che eseguirà il programma qui implementato.
Esso comunica con un altro nodo (il correspondent node, il quale può essere fisso o meno) tramite VoIP.

Di seguito è rappresentato il sistema VoIP [IEEE2003]:

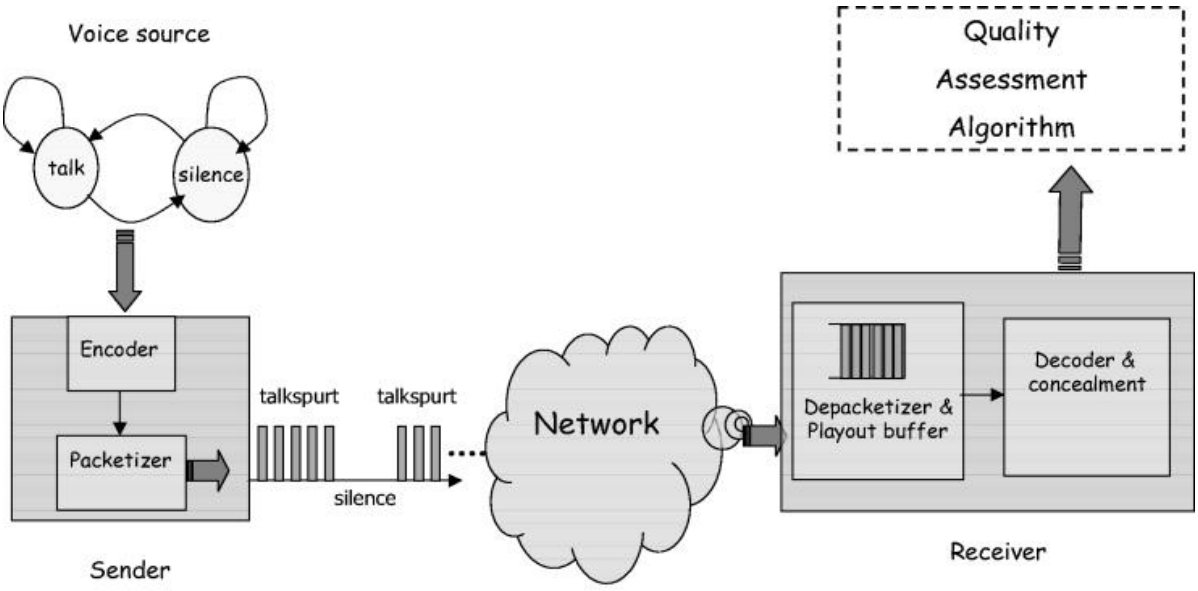


Figura 3.1: VoIP system

Questa invece è la configurazione della rete:

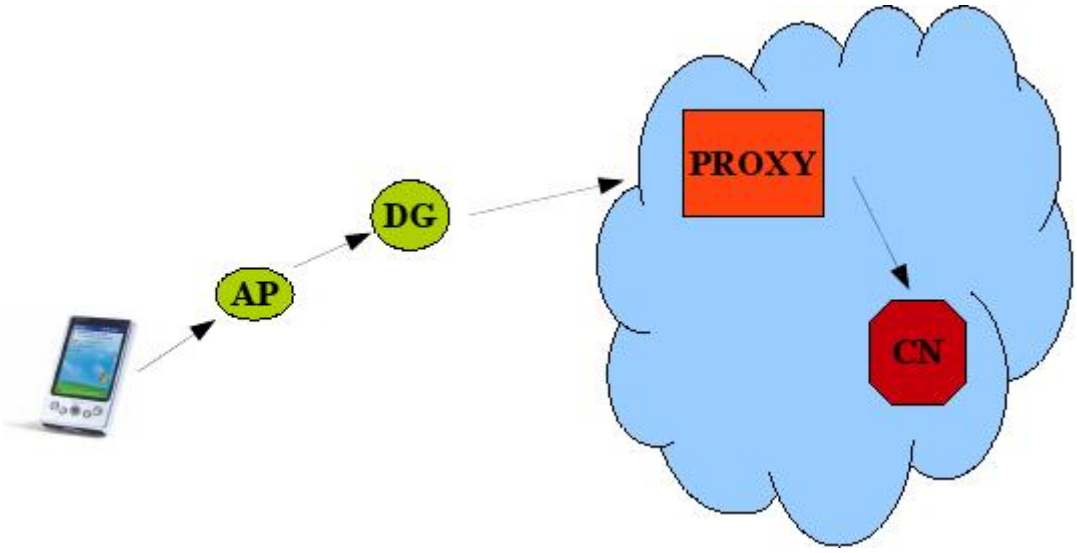


Figura 3.2: configurazione della connessione per VoIP

dove:
AP = Access Point
DG = Default Gateway
CN = Correspondent Node

Il nodo mobile è collegato al suo Access Point, che a sua volta si collega al gateway di default (spesso i due dispositivi coincidono) per collegarsi ad Internet.
Il nodo mobile è a sua volta connesso ad un proxy, che ne gestisce la comunicazione col il correspondent node durante la sessione VoIP.

Un dispositivo può avere diverse interfacce di connessione di rete wireless (ad es. WI-FI e UMTS); quando si collega ad un access point wireless esso usa una delle sue interfacce, ma trattandosi un device mobile può, nel cambiare la sua posizione nel tempo, non rientrare più nella copertura dello stesso access point.

Obiettivo del progetto è stato quello di creare un programma in grado di dare una stima della attuale connessione di rete wireless in corso, per permettere ad un ulteriore programma che analizzi queste valutazioni generate, di decidere se annullare la connessione sulla attuale interfaccia per usarne un'altra più performante.

Per la valutazione delle connessioni wireless all'interno del programma descritto dal presente documento ci si è basati sul modello E-Model citato nella sezione 2.3.1.

CAPITOLO 4
PROGETTAZIONE DEL SOFTWARE

4.1 Funzionamento del programma *QoSmonitor*

Il programma discusso in questo documento si chiama "***QoSmonitor***".
Il suo codice sorgente è contenuto in 2 file:

1. QoSmonitor.c - contiene il main() e tutte le funzioni necessarie al funzionamento del programma.
2. QoSmonitor.h – contiene la definizione dei parametri per la valutazione e la memorizzazione delle stime sulla connessione.

Per descrivere il funzionamento del programma è necessario puntualizzare due aspetti importanti che riguardano le scelte progettuali:

1)
Facendo riferimento alla configurazione di rete descritta nel precedente capitolo (figura 3.1), si intende sottolineare che **la variabilità della qualità della connessione è più alta in corrispondenza del tratto di collegamento dal dispositivo all' Access Point ed eventualmente al default gateway, poiché il collegamento in questione è di tipo wireless e il suo utilizzo previsto è mobile.** Il lato che riguarda la rete esterna più lontana invece consiste solitamente in collegamenti wired fissi, dunque è più stabile nel tempo.
Per questo motivo il software progettato si occupa, nella maggior parte dell'esecuzione, di effettuare scansioni sul collegamento col default gateway, e più raramente di valutare la connessione con il proxy e il correspondent node.

2)
I parametri di valutazione della connessione considerati nel progetto sono:

- one-way delay – deve essere inferiore ai 150 ms
- burst packet loss – deve essere inferiore a 5 pacchetti persi in blocco

Non sono invece stati considerati il jitter e il tipo di codec.

Il jitter è stato tralasciato poiché meno influente rispetto al più determinate one-way delay. Attualmente inoltre i sistemi utilizzano già un meccanismo per risolvere questo problema: si chiama "*jitter buffer*", un buffer in cui vengono fatti attendere i pacchetti audio in arrivo prima di essere riprodotti, per poter livellare i diversi ritardi di arrivo dei pacchetti e far percepire all'utente una costante emissione dell'audio.
Il programma descritto in questo documento si trova a lavorare con i pacchetti prima che il jitter buffer li riceva, quindi si immagina che le differenze di costanza percepite dal nostro

software nell'arrivo dei pacchetti siano poi attenuate del successivo passaggio nel jitter buffer.

Il funzionamento del programma si può descrivere principalmente come un ciclo infinito di scansione della rete e generazione dei valori di stima della qualità della connessione.
A seconda che si voglia fare un controllo più o meno serrato della rete, la frequenza di scansione varia dai 500 ms ai 10 secondi.
L'esecuzione inizia sempre con la scansione del default gateway e successivamente, se sono stati passati come argomenti, del proxy e del correspondent node.
Si passa dal default gateway al proxy solo dopo 20 secondi, e se la connessione col proxy è valutata positivamente, dopo la spedizione di 3 pacchetti, si procede alla connessione e alla scansione del correspondent node con l'invio di soli altri 3 pacchetti.
Che la scansione vada a buon fine o meno, si torna subito al test più prolungato del default gateway, che, come spiegato in precedenza, rappresenta il punto di trasmissione più fragile e soggetto a variazioni.

La scansione degli host consiste nell'invio di particolari pacchetti di richiesta al nodo target e nell'attesa della risposta da parte di esso.
Se la risposta arriva entro un timeout impostato a 300 ms, viene effettuato il calcolo del tempo impiegato tra l'invio della richiesta e l'arrivo della risposta, e generato il valore di qualità.
Se il timeout scade prima che si riceva una risposta, allora si attiva un meccanismo di spedizione immediata di un altro pacchetto e successiva attesa della risposta; se anche questa non arriva entro i 300 ms viene effettuata una nuova trasmissione in attesa della risposta e così via.
Per evitare di spedire in loop pacchetti ogni 300 ms, dopo 10 pacchetti non confermati, poiché è prevedibile che la connessione sia ormai inutilizzabile, il meccanismo di spedizione immediata si arresta e il programma continua a spedire pacchetti con la normale frequenza di spedizione. Se però l' host riceve un pacchetto di risposta, il programma torna a funzionare regolarmente, con la sopra citata gestione dei pacchetti persi.

Il punteggio della qualità della connessione generato dal programma dopo aver perso un pacchetto è:

- (punteggio di qualità precedente – 10), per il default gateway;
  Equivale a far scendere la valutazione di un gradino della scala dei punteggi.

- (punteggio di qualità precedente – 20), per proxy e correspondent node;
  Equivale a far scendere la valutazione di due gradini della scala dei punteggi.
  Poiché si inviano solo 3 pacchetti, con questo metodo, alla eventuale perdita di tutti e 3 i pacchetti inviati, si raggiunge velocemente un punteggio che esprime una valutazione molto negativa.

Nota: tutti i valori numerici citati nel paragrafo sono stati inseriti in un file di intestazione QoSmonitor.h, di modo da rendere velocemente modificabili tutti i parametri (scala di valutazione, frequenza di scansione, numero e tipo degli host scansionabili) senza dover apportare modifiche al codice sorgente del programma.

4.2 Calcolo del valore di qualità della connessione

Il contenitore della valutazione sul collegamento consiste in una coppia qualità-target, in cui la qualità rappresenta il punteggio di valutazione della connessione con il target, che può essere il default gateway, il proxy o il correspondent node.

Il valore della qualità consiste in un intero tra 0 e 100, che corrisponde al “Rating Factor” R definito dallo E-model standardizzato dalla Recommendation ITU-T G.107 precedentemente descritta.
Il valore 0 rappresenta il valore di peggior qualità, mentre 100 definisce una qualità ottima. Ogni valore oggettivo di qualità, quindi di R, corrisponde ad una valutazione soggettiva da parte di un ipotetico utente nei termini della scala MOS citata in precedenza.

Il valore viene calcolato principalmente in base al one-way delay verificato dal programma dopo ogni coppia ECHO REQUEST/REPLY.
Se è stata registrata la perdita di pacchetti questo numero è a sua volta decrementato in maniera tale da generare un intero che appartenga ad un livello più basso della scala rispetto al valore precedente.

Per fornire il punteggio di valutazione della rete si vuole tener conto della storia precedente della connessione in analisi, per evitare bruschi cambiamenti nella valutazione tra una ricezione all'altra che darebbero informazioni poco significative della reale condizione della rete.
Per questo si è deciso di usare il meccanismo detto “Markov Modulate Poisson Processes” (MMPP) per generale il punteggio di qualità ad ogni scansione, il numero di valori precedenti sempre considerati dall'algoritmo, oltre al punteggio attuale, è pari a 2.

Riguardo al grafico *2.3.1.*6, sono stati decisi dei cambiamenti nel fornire la scala di punteggi di qualità in base ai millisecondi di ritardo registrati, con lo scopo di dare un contrasto più netto i vari intervalli di ritardo e per dare una misurazione più severa dei ritardi.

La seguente tabella descrive la corrispondenza tra intervalli di punteggi di qualità (SCORE), intervalli di ritardo (ONE-WAY DELAY) e punteggi della scala MOS scelti in questo progetto:

ONE-WAY DELAY	**SCORES**
0 - 50 ms	100
50 - 100 ms	85
100 - 150 ms	80
150 - 200 ms	75
200 - 400 ms	65
400 ms in poi	50

Tabella 4.2.1: scala di valutazioni di QoS adottata nel progetto

SCORE LEVELS (intervals)	***MOS VALUES***	**USER SATISFACTION LEVELS**
80 - 100	*GOOD*	Users (very) satisfied
70 - 80	*FAIR*	Some users dissatisfied
60 – 70	*FAIR*	Many users dissatisfied
50 – 60	*POOR*	Nearly all users dissatisfied
0 - 50	*BAD*	All users dissatisfied

Tabella 4.2.2: soddisfazione utente e range di punteggi

4.3 Protocollo ICMP

Poiché il programma sviluppato è eseguito su un solo dispositivo, dei due che stanno comunicando tramite VoIP, non è possibile stabilire un nuovo meccanismo di passaggio delle informazioni circa le prestazioni, perciò si è dovuto usare quelli già esistenti e implementati.
Per ottenere le informazioni di performance si è quindi usato il protocollo a livello di rete ICMP (sottostante al livello di trasporto a cui appartiene il protocollo UDP usato dal VoIP), di cui tutti i dispositivi di rete hanno una implementazione.

Il protocollo ICMP è il protocollo usato nel ping classico; da questo è stata creata una implementazione del programma ping adattandola alle nostre esigenze.
Il programma ping si compone di 2 fasi ripetute potenzialmente all'infinito: una richiesta e la risposta che la conferma.
Un host effettua la richiesta, chiamata ICMP ECHO REQUEST, indirizzandola ad un destinatario; il destinatario, una volta ricevuto il pacchetto contenente la richiesta, lo rispedisce indietro al mittente specificando che si tratta di una risposta, detta ICMP ECHO REPLY.
Il contenuto del messaggio spedito dal sorgente non verrà modificato dal destinatario, ma rispedito indietro tale e quale, come detta il protocollo ICMP.

Nel caso del nostro progetto, il sorgente è il device su cui è lanciato il programma, chiamato local, e la destinazione è l'altro host coinvolto nella comunicazione, detto target.

Poiché non abbiamo alcun controllo sulla destinazione, se non tramite l'invio di un pacchetto ECHO ICMP REQUEST, è possibile calcolare il one-way delay solo usando il Round Trip Time (RTT), cioè memorizzando nel messaggio da inviare una informazione temporale che poi verrà riletta al momento in cui il pacchetto tornerà al sorgente (poiché rispedito dal target) e confrontata con l'orario di arrivo; dividendo per due il valore ottenuto si ottiene infine la latenza.

Risulta chiaro che la latenza ottenuta come metà del RTT non è un indicatore preciso, in quanto si tratta di un valore di media che non esprime il vero ritardo del singolo pacchetto; nonostante ciò questa è l'unica soluzione possibile in una applicazione eseguibile da un solo end-point della comunicazione, che in questo caso è il dispositivo mobile collegato alla rete tramite un'interfaccia wireless.

4.4 Input

Il programma implementato deve essere lanciato da un utente con diritti di root (l'utente dotato di massimi privilegi sulla macchina, cioè l'amministratore di sistema, detto anche *superuser*).

Esso analizza il tipo di connessione con più di un target, diversamente dal programma *ping* che ne scansiona solo uno alla volta.
I target sono passati a riga di comando al programma, che ottiene diversi dati in input specificati da particolari opzioni.

- sorgente:
  -I interfaceName - nome dell'interfaccia su cui si è connessi
  -l localIP - indirizzo IP dell'interfaccia su cui si è connessi

- Target destinazione:
  -g defaultGatewayIP (obbligatorio) - indirizzo IP del default gateway
  -D proxyName (opzionale) - nome del proxy
  -p proxyIP (opzionale) - indirizzo IP del proxy
  -c correspondentNodeIP (opzionale) - indirizzo IP del correspondent node con cui si sta comunicando tramite VoIP
  -v version (opzionale) - versione 4 o 6 del protocollo IP (default: 4)
  -r (opzionale) - frequenza di scansione della rete rilassata (default: urgente)

Almeno uno dei due parametri interfaceName o localIP è obbligatorio, poiché definisce quale sia l'interfaccia di connessione da analizzare.
Almeno uno dei parametri di destinazione deve essere presente. Poiché defaultGatewayIP è obbligatorio, esso è sufficiente per non generare un errore; in aggiunta si posso inserire il proxy e/o il correspondent node.
Le possibili combinazioni di input di destinazione sono perciò:

1. defaultGatewayIP
2. defaultGatewayIP proxyIP
3. defaultGatewayIP correspondentNodeIP
4. defaultGatewayIP proxyIP correspondentNodeIP

Nel caso 1 il programma valuterà in un ciclo infinito la connessione al default gateway.
Nei casi 2-3 il programma valuterà prima la connessione al default gateway, e, dopo un determinato periodo di tempo, passerà a valutare il proxy o il correspondent node, per poi tornare a valutare il default gateway.
Nel caso 4 invece si passa dal default gateway al proxy, poi al correspondent node, poi si

torna al default gateway.
Si sottolinea che solo nel caso in cui la connessione ha qualità positiva si passa a valutare la connessione col correspondent node, altrimenti si continua a comunicare col default gateway o il proxy.

4.5 Output

Il software progettato non restituisce alcun output in senso stretto:
si tratta infatti di un ciclo infinito in cui vengono spedite richieste di tipo ICMP ECHO REQUEST e ricevute risposte ICMP ECHO REPLY.

Per fornire una valutazione sulla connessione è dichiarata una variabile globale che viene modificata di volta in volta dal programma e il cui valore può essere osservato da una eventuale successiva applicazione che analizzi le stime fatte dal presente programma.
La variabile globale che contiene la valutazione consiste in realtà di un array di due interi: "*int quality[2]*".
il primo intero quality[0] contiene la valutazione per la connessione con il target indicato dal secondo intero quality[1].
Il target può essere default gateway, il proxy o il correspondent node, a seconda dell' host su cui si sta eseguendo il "*ping*" per darne una valutazione.
Quality[1] viene modificato quando si passa al prossimo host, quindi il relativo identificatore viene inserito in Quality[1] perché possa servire come informazione aggiuntiva al programma che verificherà le stime sulla qualità date dal programma qui progettato.

Oltre che essere chiamato da un altro programma, questo software può essere lanciato da terminale da un utente, il quale può verificare tramite delle stampe a video la qualità della connessione col target.
Quando poi si ferma il programma tramite la combinazione di tasti CTRL-C/Z (segnali SIGINT e SIGTSTP), vengono mostrate le statistiche sulla qualità della connessione in corso, cioè il valore di quality[0] e quality[1], il numero di pacchetti spediti e ricevuti, la percentuale di pacchetti persi, la grandezza dei dati nei pacchetti e infine il più piccolo ritardo misurato, il più grande e la loro media.

4.6 Implementazione

- Al lancio del programma vengono prima di tutto controllati i parametri ad esso passati, tramite la funzione getopt(), e per ogni parametro calcolato vengono popolate le variabili relative all'indirizzo IP dell'interfaccia locale, del proxy e del correspondent node.
  Per risalire all'indirizzo IP dell'interfaccia dato il suo nome viene chiamata la funzione ioctl() con l'argomento SIOCGIFADDR che dà informazioni su tutte le interfacce abilitate sulla macchina.
  Per risalire invece all'indirizzo IP del proxy dato il suo nome di dominio si usa la funzione gethostbyname().

- Se il controllo dell'input va a buon fine viene chiamata la funzione ThreadStart ()che lancia il thread joinable della funzione RunMonitor().

- Procedura RunMonitor():

1. crea tramite la funzione socket() un socket di tipo RAW, di livello rete, per il protocollo ICMP.

2. chiama la funzione bind() che collega al socket appena creato l'indirizzo IP dell'interfaccia locale data in input. Questo servirà poi a leggere solo i dati che arrivano sul socket che hanno come destinazione l'indirizzo IP dell'interfaccia locale usata, essendoci potenzialmente sulla macchina più interfacce connesse.

3. imposta la modalità urgente o rilassata di scansione della connessione a seconda che sia stata o meno inserita l'opzione -r al lancio del programma.
   La scansione urgente si lancia quando si sta usando il servizio VoIP e vi è la necessità di avere le prestazioni adatte per poter comunicare.
   La scansione in modalità rilassata invece si usa quando il dispositivo non sta accedendo al VoIP, ma si vuole comunque testare ogni tanto le prestazioni della rete.
   L'impostazione dell'urgenza o meno ha come effetto l'impostazione della frequenza di scansione della rete a 0.5 secondi (500 ms) per quella urgente, e a 10 secondi per quella rilassata.

   Infine il default gateway viene impostato come target per la valutazione.

4. A questo punto vengono definiti 3 cicli infiniti annidati:

   4.1 Il primo ciclo più esterno effettua la connessione con il target; ci possono essere 3 target: default gateway, proxy o correspondent node, che vengono scelti in base al valore presente nella variabile target.

Dopo aver ottenuto l'indirizzo IP del target, viene chiamata la funzione connect(). Anche se si sta usando il protocollo senza connessione ICMP, la funzione connect() può essere comunque chiamata per usare successivamente la funzioni read() che necessita di un socket connesso che contenga l'indirizzo IP dell' host da cui si intende ricevere i pacchetti.
L'utilità dell'uso di un socket connesso nel protocollo di livello network ICMP non connesso è data in previsione che il software che andrà a chiamare il nostro programma lancerà più istanze del programma, una per ogni interfaccia presente nel dispositivo, e con diversi indirizzi IP di Access Point come destinazioni.
Se usassimo un socket non connesso, la funzione read() riceverebbe tutti i pacchetti ICMP ECHO REPLY in arrivo per il dispositivo, anche quelli provenienti da host da cui la specifica istanza del programma non intendeva ricevere; usando invece la funzione connect la read leggerà solo i pacchetti destinati al processo relativo che arrivano quindi dal mittente corretto.
A differenza di ciò che fa il protocollo di livello superiore a ICMP che è TCP, qui la distinzione viene fatta solo in base all'indirizzo IP da cui si vuole ricevere e non al numero di porta su cui il processo è in attesa, poiché a livello rete il concetto di porta non è implementato.

Come ultima operazione infine viene impostato il massimo timeout di attesa dei pacchetti ICMP ECHO REPLY, a seconda che il target scelto sia il default gateway (RTT massimo = 100 ms) o proxy e correspondent node (RTT massimo = 300 ms).

4.2 Il secondo ciclo annidato contiene le operazioni di cambio del target, di spedizione del messaggio ICMP ECHO REQUEST e di impostazione del timeout per la funzione select() contenuta nell'ultimo ciclo più interno.
Dopo un certo periodo piuttosto lungo, il programma effettua lo switch del target, secondo questo ordine:
default gateway → proxy → correspondent node.
È sempre obbligatorio impostare il default gateway, se però uno degli altri due indirizzi non è presente, si procede a valutare l'altro; nel caso in cui sia presente solo il default gateway, il programma continua indefinitamente a valutarne il collegamento.
Una volta cambiato il target, nell'esecuzione viene inserito un break che termina l'esecuzione del ciclo e la riporta al primo ciclo più esterno, in cui avverrà la nuova connessione con il nuovo target, e da lì si ripartirà con l'esecuzione del ciclo per la valutazione della nuova connessione.

4.3 Il terzo ed ultimo ciclo effettua il vero e proprio controllo di qualità della connessione.
La funzione fondamentale è la select(): ad essa viene passato inizialmente il valore di timeout di ricezione dei pacchetti impostato nel precedente ciclo più esterno dopo la

loro spedizione.
Questo timeout serve a valutare la perdita di pacchetti, cioè pacchetti non ricevuti. Il timeout viene fissato entro un valore in ms di discreta qualità della connessione. Se il pacchetto non arriva entro quel limite, si può considerare il pacchetto come perso, e se arriverà successivamente verrà scartato.
Si distinguono 3 motivi per cui il timeout può scadere:
(a) (replyExpected == 1) && (replyArrived == 0)
(b) (replyExpected == 1) && (replyArrived == 1)
(c) (replyExpected == 0)

Le variabili booleane replyExpected e replyArrived sono indicatori di attesa e arrivo dei pacchetti:
- quando un pacchetto viene spedito dalla SendEchoRequests() replyExpected è impostato a 1 e replyArrived a 0, poiché si attende una risposta che però non è ancora arrivata.
- quando arriva una risposta ReadEchoReply() setta replyArrived a 1.

(a) Se il timeout di ricezione della risposta scade e non è ancora arrivato alcun pacchetto (è probabile che le prestazioni stiano quindi degradando), il programma esegue una istruzione break che forza di uscire dal ciclo e tornare al secondo ciclo più esterno dove verrà rieseguita la spedizione di un altro pacchetto, così da testare subito la rete.
Queste operazioni vengono svolte solo se il numero di pacchetti consecutivi persi è inferiore al doppio del massimo numero di pacchetti non confermati (10), altrimenti si rallenta la rispedizione veloce, dato che probabilmente non c'è possibilità che le prestazioni della connessione tornino a migliorare e si continua a rispedire con la frequenza normale di invio, senza più seguire la scadenza del timeout per la ricezione.

(b) Se invece prima che scada il timeout arriva una risposta, l'allarme viene modificato impostandolo al tempo di attesa che rimane prima di poter spedire un nuovo pacchetto (definito in precedenza come la frequenza che dipende dal valore di urgenza o meno dato a riga di comando).

In questo caso la variabile replyExpected assume il valore 0, per poter rendere vera l'ultima condizione *(c)* che permette di uscire dal ciclo ed effettuare una nuova spedizione di un pacchetto, reimpostare il timeout per la ricezione della risposta e rientrare nella select().

- Procedura SendEchoRequests()
  Si occupa di spedire a target per mezzo della chiamata alla funzione di libreria sendto() ogni messaggio ICMP ECHO REQUEST costruendolo in tutte le sue parti. Il messaggio ICMP viene poi incapsulato nel pacchetto IP.

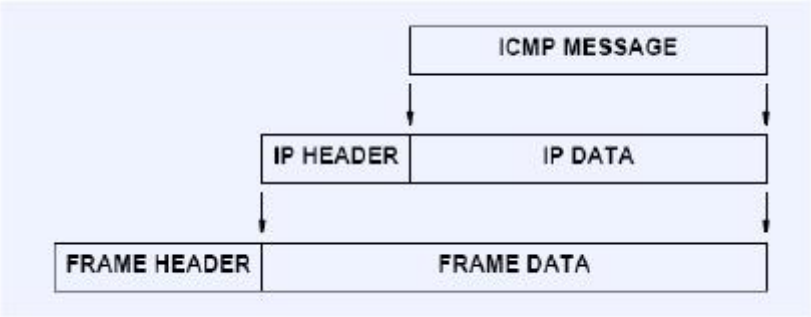


Figura 4.6.1: incapsulamento di ICMP in IP

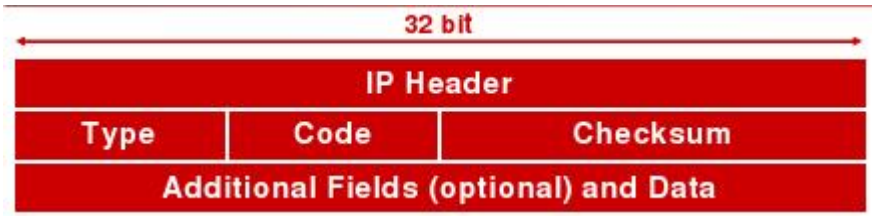


Figura 4.6.2: campi dell' header ICMPv4

Valore	Tipo di Messaggio
0	Echo Reply
3	Destination Unreachable
4	Source Quence
5	Redirect
8	Echo Request
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Information Request (obsolete)
16	Information Reply (obsolete)
17	Address Mask Request
18	Address Mask Reply

Le specifiche sono nelle RFC 792, 950, 1256

Figura 4.6.3: possibili valori del campo Type dell' header ICMPv4

L' header ICMP è grande 8 byte, e viene riempito dal nostro programma in questo modo:

```
ICMPmessage ->cmp_type = ECHO_REQUEST;
ICMPmessage->icmp_code = 0;
ICMPmessage->icmp_cksum = getChecksum();
ICMPmessage->icmp_id = processId;
ICMPmessage->icmp_seq = numeroSequenzaPacchetto;
```

E' stato scelto di creare un messaggio ICMP che, oltre agli 8 byte dell' header, contenesse 92 byte di dati, per un totale di 100 byte, che corrispondono a circa 30 ms di audio codificati con codec HE-AAC (definito nello standard MPEG-4 e accettato dal consorzio 3GPP) da 13 Kbps.
Il messaggio viene spedito e vengono impostate le variabili replyExpected e replyArrived rispettivamente a 1 e 0.
Per il default gateway la variabile replyArrived prende il valore 0 solo se il numero di sequenza dei pacchetti confermati è maggiore di 3; questo è utile a non far valutare di primi 3 pacchetti che non sono indicativi della qualità del servizio. Per il proxy e il correspondent node, visto che vengono spediti solo 3 pacchetti, la loro qualità viene invece valutata.

- Procedura ReadEchoReply()
  Ha il compito di leggere i messaggi arrivati dal target e, se si tratta di messaggi di tipo ICMP ECHO REPLY, di calcolare il valore da assegnare a quality[0], chiamando la funzione CalculateQuality().
  Essa inoltre imposta a 1 il booleano replyArrived per confermare che il pacchetto atteso è arrivato.

- Procedura CalculateQuality()
  Questa è la procedura che effettua la valutazione delle prestazioni e la scrittura del punteggio in quality[0].
  Essa riceve come input il valore del Round Trip Time (RTT) del pacchetto ricevuto calcolato in precedenza da ReadEchoReply(), e il numero di sequenza del pacchetto, che servirà per capire se scartare dei pacchetti con numeri di sequenza inferiori rispetto a quelli attesi.

  La generazione del punteggio di qualità è data dai seguenti passaggi:

a) calcolo del one-way delay dal RTT

b) generazione del punteggio temporaneo per la connessione in base alla tabella 4.2.1

c) se il pacchetto che arriva è quello che ci si aspettava (cioè non si è avuta alcuna perdita di pacchetti), bisogna generare il valore di qualità finale da quello attuale dato dalla tabella di riferimento.
Per fare cioè si utilizza il metodo “Markov Modulate Poisson Processes” (MMPP), il cui livello è stato deciso pari a 3.
Questo metodo è utile per dare una valutazione della condizione della connettività della rete in modo che venga considerata la storia delle valutazioni precedenti; in questo caso si considerano gli ultimi 2 valori precedenti.
L'implementazione del metodo MMPP viene fatta mantenendo gli ultimi tre valori di qualità in un array di interi lastQualities[3] e ogni volta il valore di qualità viene calcolato sommando i 3 valori dividendo la somma per 3.
Il valore in quality[0] viene inserito solo se diverso dal precedente, in modo che, se un programma si mette in attesa di un nuovo valore della variabile, non debba essere svegliato se il valore generato è sempre lo stesso.

d) Solo per il default gateway se il numero di sequenza del pacchetto è inferiore a 3 il punteggio dato è quello del livello migliore (100), poiché i primi pacchetti inviati non sono considerati nella valutazione della connessione.

- procedura terminateMonitor()
  La sua chiamata è causata dalla pressione dei tasti CTRL-C/Z per fermare il programma da terminale.
  Questa procedura restituisce tramite printf() delle statistiche sulla qualità della connessione, cioè il valore di quality[0] e quality[1], il numero di pacchetti spediti e ricevuti, la percentuale di pacchetti persi, la grandezza dei dati nei pacchetti e infine il più piccolo ritardo misurato, il più grande e la loro media.

- int normalizeTimeval(struct timeval *t)
  Normalizza i valori presenti nella struttura timeval passata.

- struct timeval timeDifference(struct timeval after, struct timeval before)
  Esegue la differenza tra due strutture timeval passate per ricavare il tempo trascorso da before ad after.

- uint16_t getChecksum(uint16_t *msg, int length)
  Genera il valore di checksum da inserire nel messaggio ICMP.

CAPITOLO 5
VALUTAZIONE

Per verificare il programma sviluppato sono stati eseguiti in laboratorio test sperimentali di vario tipo, dal test statico a quello in movimento, e su vari tipi di connessione wireless, come la Wi-Fi e quella UMTS.

Per la connessione Wi-Fi, il programma è stato testato passandogli i seguenti parametri di indirizzamento IP:
interfaccia wireless locale: 192.168.1.102
default gateway: 192.168.1.1
proxy: 94.143.220.197
correspondent node: 143.106.60.119
Il proxy corrisponde al server del dominio "*www.debian.fr*", localizzato in Francia, mentre il correspondent node rappresenta un server Debian che si trova in Brasile,
La scelta di questi server è motivata dall'esigenza di simulare il reale uso del programma analizzato: il proxy infatti si dovrebbe trovare più lontano rispetto al gateway ma più vicino rispetto al correspondent node.

I test operati sono di 3 tipi:

1. in prossimità dell' Access Point
2. in movimento, dentro l'area di copertura dell' AP con presenza di ostacoli
3. in movimento, uscendo dall'area di copertura dell' AP

Infine il programma è stato eseguito sempre in modalità urgente, dunque la scansione dell'interfaccia è avvenuta sempre ogni 500 ms.

5.1 Test statico in prossimità dell' Access Point – interfaccia Wi-Fi

L'esecuzione ha inizio con la scansione della connessione con il default gateway.
Il one-way delay registrato è di 1 ms perciò il valore generato dal programma, che rappresenta la qualità della connessione, è 100.
Dopo 20 secondi si passa alla connessione e successiva valutazione del proxy.
Il one-way delay misurato è di 10/11 ms, di conseguenza la stima della qualità generata è sempre 100.
Dopo l'invio di 3 pacchetti ICMP ECHO REQUEST al proxy, il programma si collega al correspondent node e continua l'invio di altri 3 pacchetti di richiesta.
La qualità in questo caso scende di molto, infatti nessuno dei 3 pacchetti inviati viene confermato entro il timeout di 300 ms (one-way delay massimo 150 ms) ed essi sono dunque persi. La qualità si abbassa quindi a 80, poi a 60, infine a 40 che indica una pessima

qualità di connessione.
Successivamente il programma torna a valutare il gateway di default, poi il proxy, il correspondent node, e così via. Non si sono però mostrati comportamenti differenti da quelli descritti.

Di seguito è mostrato il grafico che rappresenta la sessione di test sopra riportata:

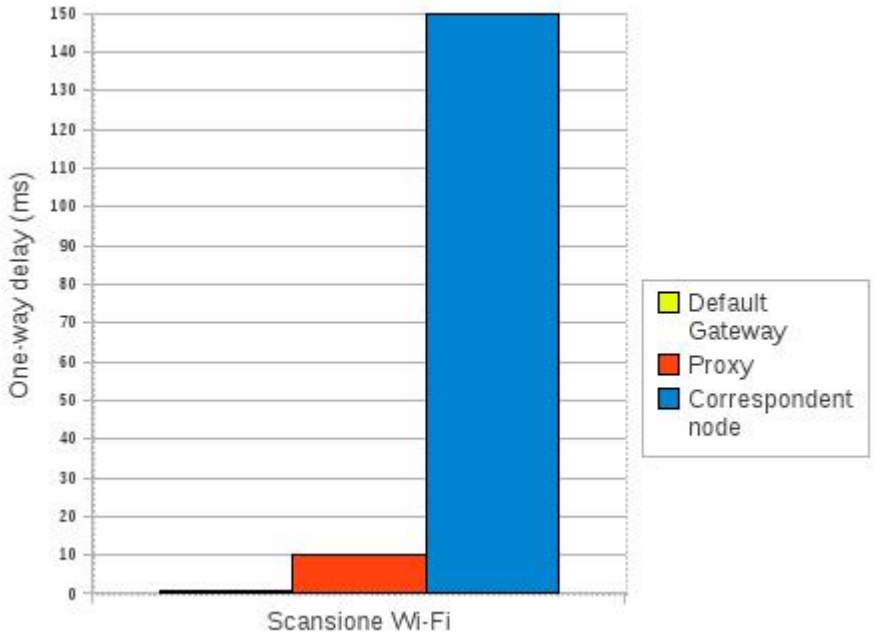


Figura 5.1.1: risultati del test statico su interfaccia Wi-Fi (one-way delay)

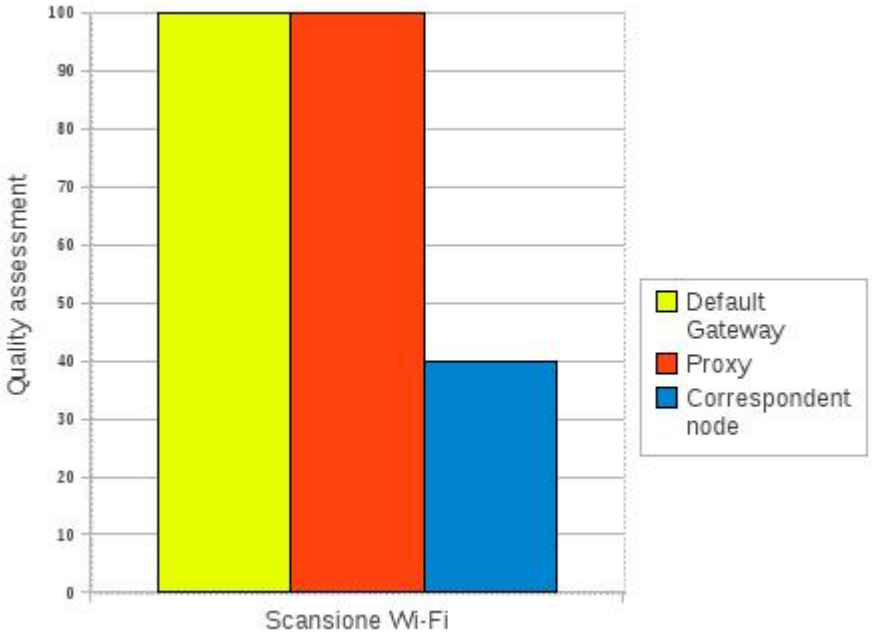


Figura 5.1.2: risultati del test statico su interfaccia Wi-Fi (quality)

5.2 Test in movimento con presenza di ostacoli nel raggio di azione dell' Access Point – interfaccia Wi-Fi

Il successivo test è stato effettuato spostando il dispositivo di molto lontano rispetto all' AP, per analizzare eventuali variazioni nei tempi di delay.
In effetti già dal monitoraggio del default gateway (Scansione Wi-Fi 1) si può notare un aumento, seppure minimo, del ritardo registrato. Il delay calcolato per il gateway è dapprima 1 ms ma poi aumenta a 4 ms, fino ad arrivare a 13 ms, pur mantenendo il valore di qualità a 100.
Il proxy analizzato successivamente genera un ritardo di 21 ms con qualità costante a 100, mentre nel correspondent node si registra una perdita di tutti e 3 i pacchetti speditigli, perciò la stima della qualità scende a 40.
Tornando a scandire (Scansione Wi-Fi 2) il gateway a questo punto si può notare un aumento del ritardo, che arriva a 20 ms, fino ad avere 43 pacchetti persi consecutivamente e una qualità pari a 0, poiché ci siamo posizionati dietro ad un ostacolo che scherma le frequenze Wi-Fi.
Con la scansione in modalità frequente il programma segnala il crollo delle prestazioni entro l'invio di 5 pacchetti (con una sequenza di valori di qualità di media pari a 90 – 80 – 70 – 60 – 50) dove già dal terzo pacchetto la valutazione passa da un livello positivo ad uno negativo.
Nella ripresa della connessione invece il passaggio da un valore pessimo (0) ad uno buono (>=80) avviene entro la conferma di 6 pacchetti (con una sequenza di valori di qualità di media pari a 33 – 44 – 59 – 68 – 76 – 81).

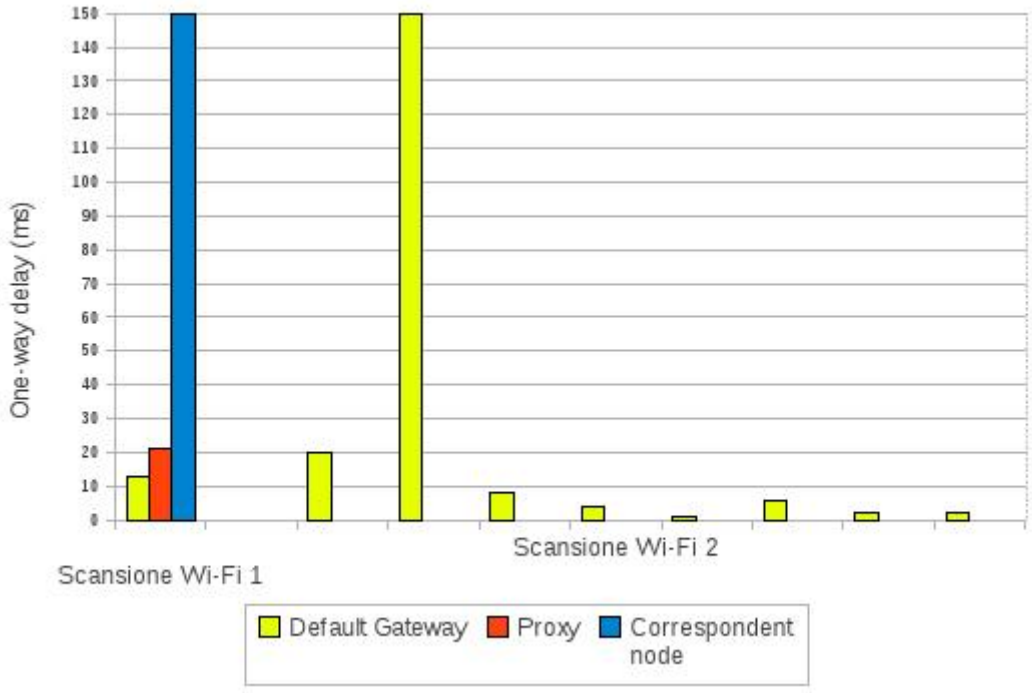


Figura 5.2.1: risultati del test dinamico 1 su interfaccia Wi-Fi (one-way delay)

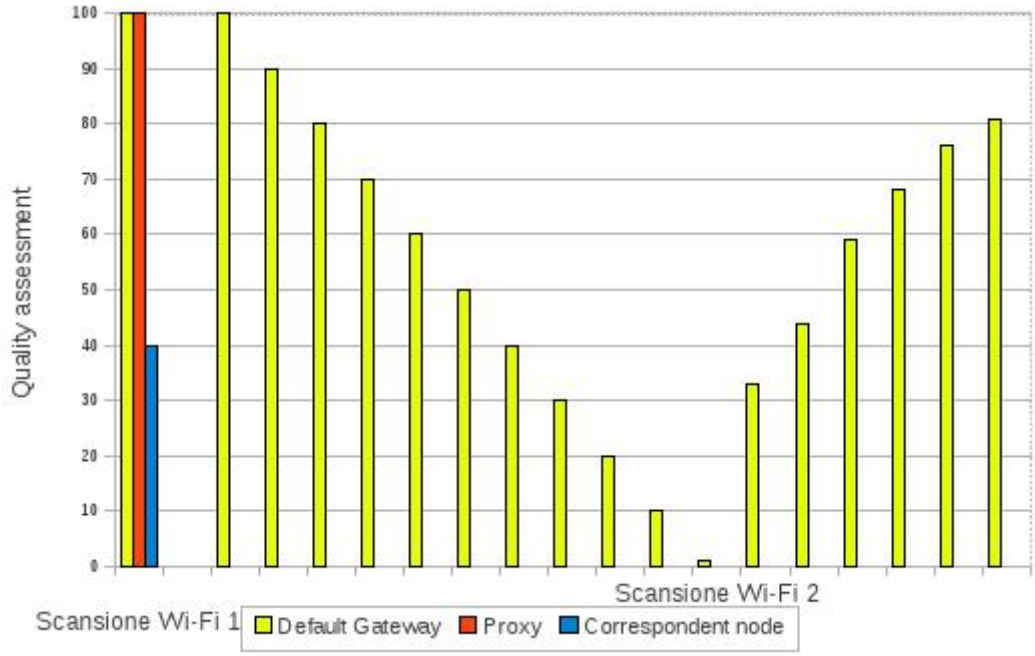


Figura 5.2.2: risultati del test dinamico 1 su interfaccia Wi-Fi (quality)

5.3 Test in movimento con uscita dal raggio di azione dell' Access Point – interfaccia Wi-Fi

In questo esperimento si è provato ad uscire per un po' completamente fuori dalla copertura dell' AP.
Perciò il ritardo misurato (Scansione Wi-Fi 1) è salito sempre più dai 16 ms, ai 50 ms, ai 100 ms, ai 170 ms, fino a iniziare a perdere velocemente pacchetti di risposta, dapprima 10 pacchetti di seguito, poi tutti quanti.
Riavvicinandoci al gateway (Scansione Wi-Fi 2) e rientrando quindi nel raggio di copertura dello stesso, si è visto che i pacchetti tornavano ad essere confermati e il nostro programma generava delle stime di qualità buone, anche se teoricamente ormai il socket avrebbe dovuto essere inutilizzabile.
Invece, poiché il nostro programma effettua periodicamente delle riconnessioni, in corrispondenza dello switch del target, è stato possibile ripartire con il monitoraggio senza riavviare l'esecuzione del programma.

Il seguente grafico mostra l'andamento di ritardi e qualità:

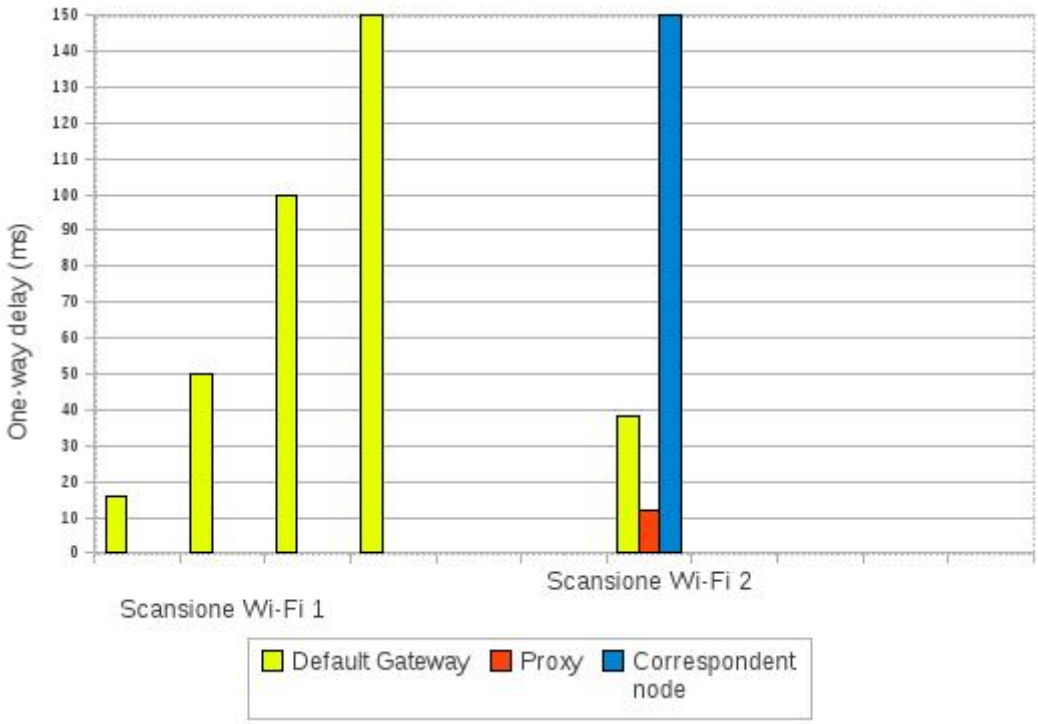


Figura 5.3.1: risultati del test dinamico 2 su interfaccia Wi-Fi (one-way delay)

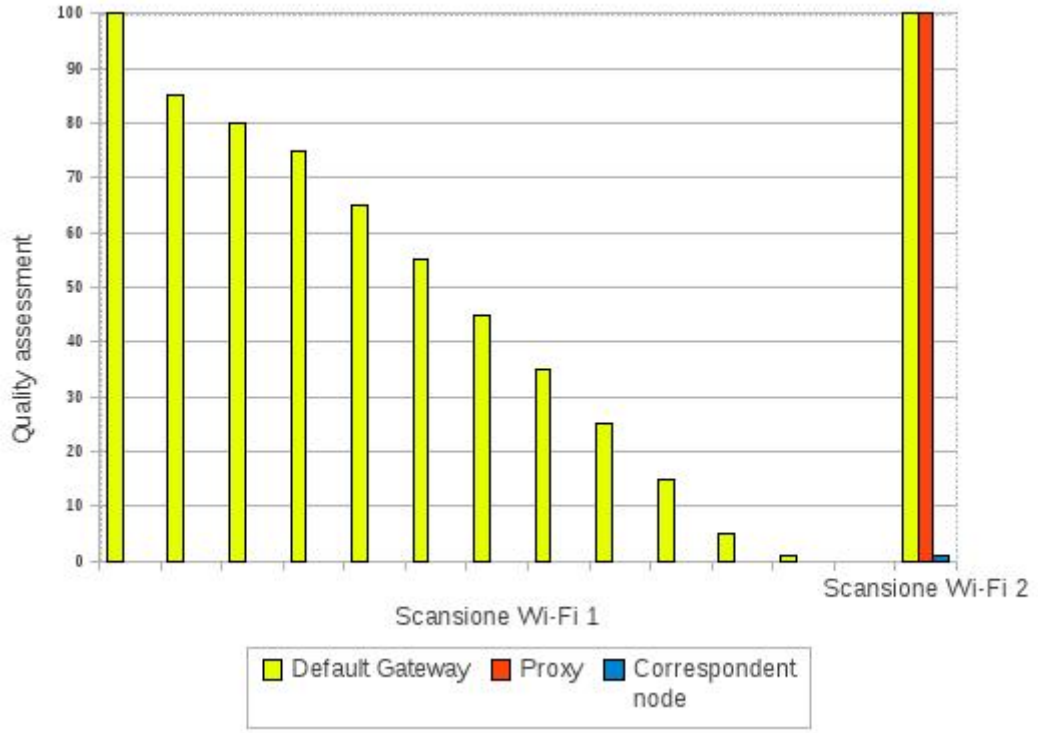


Figura 5.3.2: risultati del test dinamico 2 su interfaccia Wi-Fi (quality)

5.4 Test statico in prossimità dell' Access Point – interfaccia UMTS

Dopo aver analizzato il funzionamento del nostro programma nella valutazione delle performance di un'interfaccia Wi-Fi, abbiamo effettuato delle prove su un altro tipo di interfaccia: quella UMTS.
Il test presentato è stato effettuato mantenendo fisso il dispositivo di ricezione accanto all' AP.

I dati di indirizzamento IP passati al programma sono:
interfaccia UMTS locale: 109.54.13.8
default gateway: 94.143.220.197

Fin dall'inizio del test si alternano circa 20 pacchetti persi in blocco (“burst loss”) a conferme di singoli pacchetti che arrivano generando un one-way delay di 128/150 ms.
La qualità fornita è quindi sempre 0, tranne all'arrivo della singola conferma, in cui si sposta in media a 27.

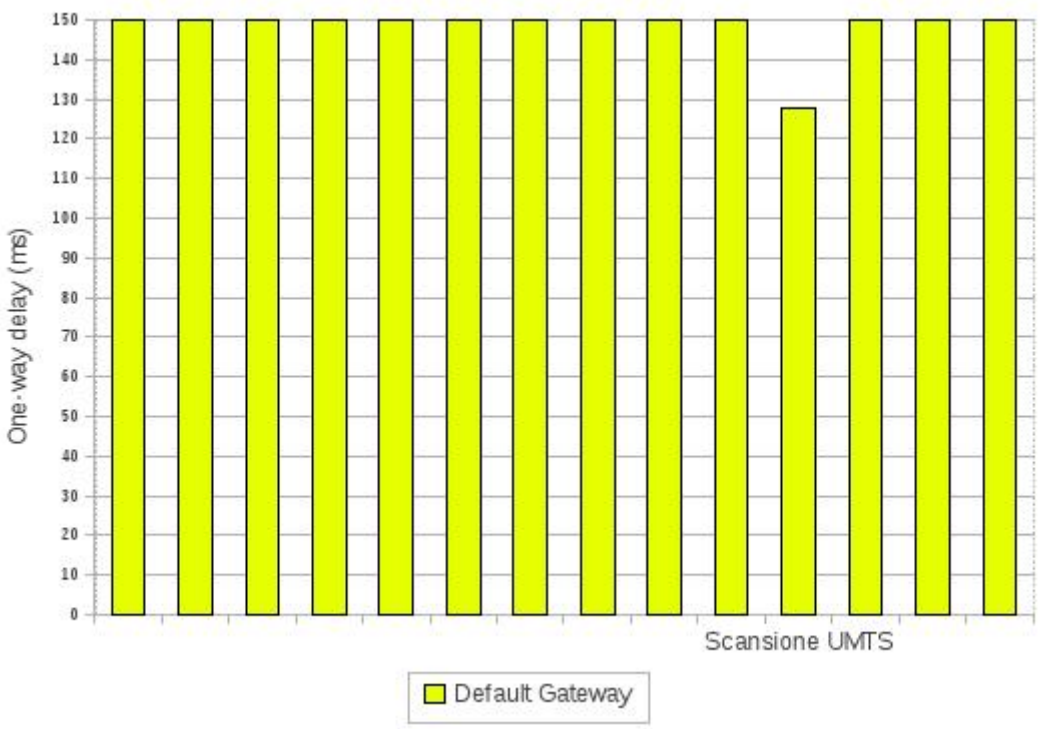


Figura 5.4.1: risultati del test statico su interfaccia UMTS (one-way delay)

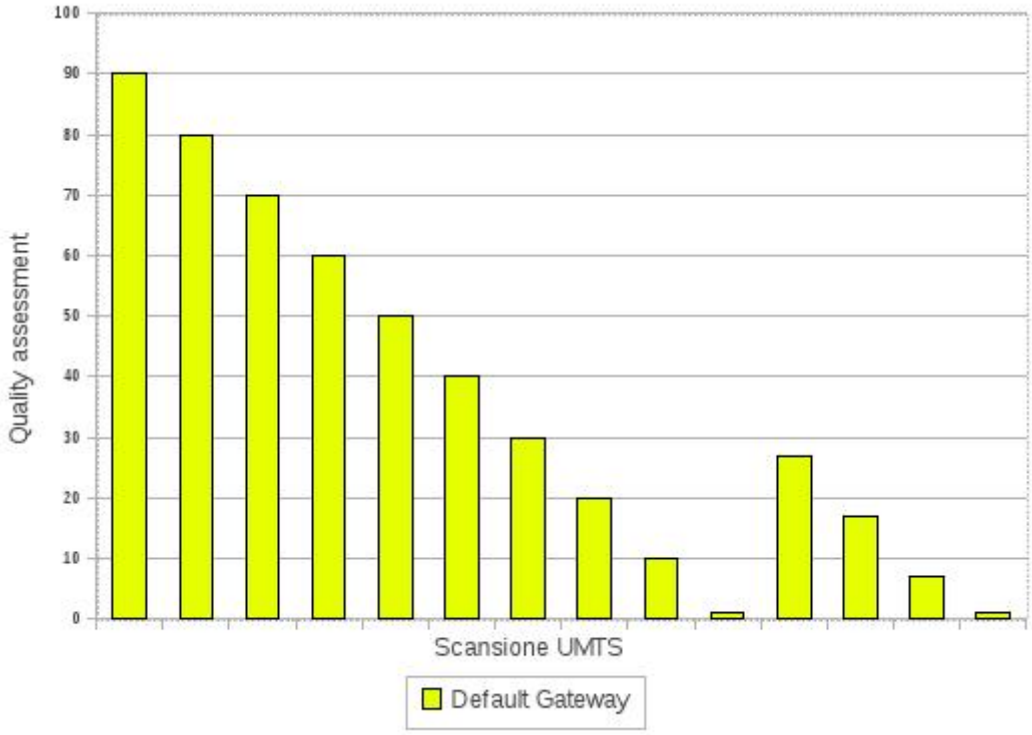


Figura 5.4.2: risultati del test statico su interfaccia UMTS (quality)

Da questi risultati si può trarre una riflessione finale sulle tecnologie wireless messe a confronto, ed in particolare su quella UMTS.
Essa ha un RTT molto elevato, perciò non riesce a supportare il VoIP.
In realtà questa consiste in una limitazione non tecnologica, ma imposta del gestore di telefonia che fornisce il servizio (nel nostro test è TIM), il quale probabilmente associa una priorità più bassa ai pacchetti dati per VoIP in modo da scoraggiarne l'utilizzo e la ricerca in questo campo.

CAPITOLO 6
CONCLUSIONE E SVILUPPI FUTURI

Una riflessione finale può essere svolta sulle tecnologie wireless utilizzate per i nostri test: mentre le interfacce Wi-Fi riescono a fornire performance adatte ad applicazioni interattive, quella UMTS ha un RTT troppo elevato, perciò non riesce a supportare il VoIP.
Questa però è una limitazione non tecnologica, ma imposta del gestore di telefonia che fornisce il servizio (nel nostro test è TIM), il quale probabilmente associa una priorità più bassa ai pacchetti dati per VoIP in modo da scoraggiarne l'utilizzo e la ricerca in questo campo.

Nel progettare il nostro programma abbiamo sottinteso che i dispositivi rispondono alle richieste ICMP ECHO REQUEST (ping).
Poiché però alcuni gestori non abilitano la risposta al ping per motivi di sicurezza, la nostra verifica di efficienza della connessione potrebbe essere inficiata dalle scelte dei gestori, perciò si prevede il seguente miglioramento.
Poiché che il proxy per la comunicazione VoIP sarà sviluppato e installato da noi, e successivamente configurato per accettare ping, una volta connessi al default gateway, se questo non risponde, invece che evitare di continuare la scansione al tratto successivo dando per scontato che la connessione è caduta, si potrebbe comunque proseguire il monitoraggio verso il proxy. In questo modo, che se il gateway non risponde al ping per motivi di sicurezza, noi riusciamo ad avere una stima reale della connessione per l'utilizzo con il VoIP.
In fase finale dell'implementazione questo miglioramento è stato apportato.

Un altro dei miglioramenti apportabili può essere quello di effettuare un test su AP e un altro sul default gateway una volta appurato che i 2 dispositivi sono separati, e non testare solo il gateway come è stato scelto nel presente programma, sottintendendo che i 2 dispositivi corrispondano.

Per quanto riguarda l'uso dell' IPv6 in questa implementazione, il test non è stato possibile, perciò si invita chi interessato a testarne il funzionamento successivamente, in quanto allo stato attuale la nuova versione del protocollo IP non è ancora utilizzata a regime.

Se qualcuno in futuro vorrà modificare e personalizzare, seguendo altri standard di valutazione, la scala dei punteggi o la frequenza di scansione, il metodo MMPP, o anche il numero e il tipo degli host scansionabili, potrà facilmente accedere al file “*QoSmonitor.h*” e provare diverse configurazioni semplicemente modificando delle costanti globali.

Nel futuro si potrebbe inoltre implementare il software qui sviluppato in modo che funzioni su altre architetture, come Android, Windows Mobile, Symbian e iPhone.

CAPITOLO 7
BIBLIOGRAFIA

[BMP1] Birke R., Mellia M., Petracca M., "*Understanding VoIP from Backbone Measurements*".

[Cam2001] Campanella M., "*QoS*", 2001, http://www2.garr.it/ws4_pdf/Qos.pdf .

[CEALSB2004] Carvalho L., Mota E., Aguiar R., Lima A. F., de Souza J. N., Barreto A., "*An E-Model Implementation for Speech Quality Evaluation in VoIP Systems*", 2004.

[Cer2005] Cerroni W., "*Il protocollo ICMP* ", 2005, http://www-tlc.deis.unibo.it/Didattica/CorsiCE/RetiLB/Slides/06-ICMP.pdf.

[Cla1] Clark A. D., "*Modeling the Effects of Burst Packet Loss and Recency on Subjective Voice Quality*".

[ColRos1] Cole R. G., Rosenbluth J. H., "*Voice over IP Performance Monitoring*".

[ComSte1993] Comer D. E., Stevens D. L., "*Internetworking with TCP/IP, Vol. III*", 1993.

[DinGou2003] Ding L., Goubran R. A., *"Speech Quality Prediction in VoIP Using the Extended E-Model"*, 2003.

[GBR1] Gai S., Baldi M., Risso F., "*ICMP*", http://netgroup.polito.it/teaching/reticalc/icmp.pdf.

[GJS2003] Gozdecki J., Jajszczyk A., Stankiewicz R., "*Quality of Service Terminology in* IP Networks", 2003.

[IEEE2003] "*IEEE/ACM Transactions on Networking, Vol. XI, No. 5*", 2003.

[IETF3344] IETF Recommendation 3344, "*IP Mobility Support for IPv4*", agosto 2002.

[MTK2003] Markopoulou A. P., Tobagi F. A., Karam M. J., "*Assessing the Quality of Voice Communications Over Internet Backbones* ", 2003.

[MelMos2006] Meltzer S., Moser G., "*MPEG-4 HE-AAC v2 audio coding for today's digital media world. EBU technical review. European Broadcasting Union", 2006.*

[NarMur2003] Narbutt M., Murphy L., *"Improving Voice over IP Subjective Call Quality"*, 2003.

[Org2010] Voip-Info.org, "*QoS*", 2010, http://www.voip-info.org/wiki/view/QoS.

[Parr2009] Parrucci M., "QoS in Wireless Network", 2009, http://www.sti.uniurb.it/parrucci/Materiale%20Didattico/Qos%20in%20Wireless%20Network_9.pdf

[Rix2004] Rix A. W., *"Perceptual Speech Quality Assessment – A Review"*, 2004.

[Ros2010] Rossi F. G., "*QoS in reti a commutazione di pacchetto*", 2010, http://www-3.unipv.it/retical/didattica/aa2010-11/retitelpv/lezioni/cap08.pdf.

[SFR2007] Stevens W. R., Fenner B., Rudoff A. M., "UNIX Network Programming", 2007.

[SunIfe1] Sun L., Ifeachor E., "*New Models for Perceived Voice Quality Prediction and their Applications in Playout Buffer Optimization for VoIP Networks*".

[UniSyr1] Syracuse University , "*Lecture Notes for Internet Security , ICMP: Internet Control Message Protocol*".

[Tak2004] Takahashi A., *"Opinion Model for Estimating Conversational Quality of VoIP", 2004.*

[ZanCig2004] Zanolli M., Lo Cigno R., "Gestione di QoS nelle Reti Wireless IEEE 802.11", 2004.

[Walk2002] Walker J.Q., "*Assessing VoIP Call Quality Using the E-model*", 2002.

[WKS1] Wenyu J., Koguchi K., Schulzrinne H., "*QoS Evaluation of VoIP End-points*".

APPENDICE A: ITU-T RECOMMENDATIONS

[ITUG.107] ITU-T Recommendation G.107, "The E-model, a computational model for use in transmission planning", maggio 2000.

[ITUG.108] ITU-T Recommendation G.108, "Application of the E-model: A planning guide", settembre 1999.

[ITUG.109] ITU-T Recommendation G.109, "Definition of categories of speech transmission quality", settembre 1999.

[ITUG.113] ITU-T Recommendation G.113, "Transmission impairments", febbraio 1996.

[ITUG.114] ITU-T Recommendation G.114, "One-way transmission time ", maggio 2003.

[ITUP.800] ITU-T Recommendation P.800, "Methods for subjective determination of transmission quality", agosto 1996.

[ITUY.1541] ITU-T Recommendation Y.1541, "Network performance objectives for IP-based services", febbraio 2006.

APPENDICE B: SERIES OF ITU-T RECOMMENDATIONS

- Series A Organization of the work of ITU-T
- Series B Means of expression: definitions, symbols, classification
- Series C General telecommunication statistics
- Series D General tariff principles
- Series E Overall network operation, telephone service, service operation and human factors
- Series F Non-telephone telecommunication services
- Series G Transmission systems and media, digital systems and networks
- Series H Audiovisual and multimedia systems
- Series I Integrated services digital network
- Series J Transmission of television, sound programme and other multimedia signals
- Series K Protection against interference
- Series L Construction, installation and protection of cables and other elements of outside plant
- Series M TMN and network maintenance: international transmission systems, telephone circuits,
- telegraphy, facsimile and leased circuits
- Series N Maintenance: international sound programme and television transmission circuits
- Series O Specifications of measuring equipment
- Series P Telephone transmission quality, telephone installations, local line networks
- Series Q Switching and signalling
- Series R Telegraph transmission
- Series S Telegraph services terminal equipment
- Series T Terminals for telematic services
- Series U Telegraph switching
- Series V Data communication over the telephone network
- Series X Data networks and open system communications
- Series Y Global information infrastructure and Internet protocol aspects
- Series Z Languages and general software aspects for telecommunication systems

APPENDICE C: CODICE SORGENTE DEL PROGRAMMA

C1 – File “QoSmonitor.h”

```
1.  #include <pthread.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <string.h>
5.  #include <errno.h>
6.  #include <sys/ioctl.h>
7.  #include <net/if.h>
8.  #include <signal.h>
9.  #include <netdb.h>
10. #include <netinet/in.h>
11. #include <netinet/ip_icmp.h>
12. #include <netinet/icmp6.h>
13. #include <unistd.h>
14. #include <sys/types.h>
15. #include <sys/time.h>
16. #include <sys/socket.h>
17. #include <arpa/inet.h>
18. #include <sys/select.h>
19. #include <sys/signal.h>
20.
21.
22.
23. /* CONSTANTS DEFINITIONS */
24. #define MIN_NUM_ARGS        2+1
25. #define INT_SIZE            4
26. #define MAX_INT                 2147438647
27.
28. #define ECHO_REQUEST        8
29. #define ECHO_REPLY          0
30. #define ICMP_HEADER_LENGTH  8
31. #define ICMP_DATA_LENGTH    92
32. #define SND_MSG_SIZE        (ICMP_HEADER_LENGTH +
    ICMP_DATA_LENGTH)
33. #define MIN_ICMP_MSG_SIZE   16
34. #define RCV_BUFFER_LENGTH   1500
35. #define IOCTL_BUF_LENGTH1024
36.
```

```
37.  #define QUALITY_SCORE              0
38.  #define QUALITY_TARGET             1
39.
40.  #define URGENT_FREQUENCE           500000          /* in microseconds, 0.5
     seconds, 500 ms */
41.  #define RELAXED_FREQUENCE          10000000    /* in microseconds, 10
     seconds, 10000 ms */
42.  #define TIMES_TO_WAIT_FOR_SWITCH_URG   20000000 /
     URGENT_FREQUENCE        /* 20 seconds = 40 x 500 = 20000 ms */
43.  #define TIMES_TO_WAIT_FOR_SWITCH_REL   60000000 /
     RELAXED_FREQUENCE       /* 60 seconds = 6 x 10000 = 60000 ms */
44.  #define SEC_IN_USEC                1000000L
45.  #define SEC_IN_MSEC                1000
46.  #define USEC_IN_MSEC               1000
47.
48.  #define IGNORE_FIRST_PACKETS       3
49.
50.  /* one-way-delays */
51.  #define BEST_DELAY                 50      /* BEST_SCORE */
52.  #define GOOD_DELAY                 100     /* GOOD_SCORE */
53.  #define ALARM_DELAY                150     /* ALARM_SCORE */
54.  #define BAD_DELAY                  200     /* BAD_SCORE */
55.  #define WORST_DELAY                400     /* POOR_SCORE,
     WORST_SCORE */
56.
57.  /* scores - MOS, ms */
58.  #define BEST_SCORE                 100     /* 0-50 ms */
59.  #define GOOD_SCORE                 85      /* 50-100 ms */
60.  #define ALARM_SCORE                80      /* 100-150 ms */
61.  #define BAD_SCORE                  75      /* 150-200 ms */
62.  #define POOR_SCORE                 65      /* 200-400 */
63.  #define WORST_SCORE                50      /* 400+ */
64.
65.  #define MIN_SCORE                  0
66.  #define MAX_SCORE                  100
67.
68.  #define    ONE_LEVEL_SCORES        10
69.
70.  #define MAX_PACKET_LOSS            5
71.  #define MMP_STATES                 3
72.
73.  /* targets */
74.  #define    DEFAULT_GATEWAY         0
75.  #define    PROXY                   1
```

```
76.  #define      CORRESP_NODE        2
77.
78.  /* for statistics */
79.  #define      MIN_DELAY                0
80.  #define      MAX_DELAY                1
81.
82.  /* number of packets to be sent to proxy and current node */
83.  #define MAX_PACKETS_PRCN      3
84.
85.  /* IP version */
86.  #define IPV4                          4
87.  #define IPV6                          6
88.
89.
90.
91.  /* FUNCTIONS PROTOTYPES */
92.  void ThreadStart();
93.
94.  void *RunMonitor();
95.  void SendEchoRequests();
96.  void ReadEchoReply(char *receivedMsg, int totalMsgLength);
97.  void CalculateQuality(int rtt, int seq);
98.
99.  int normalizeTimeval(struct timeval *t);
100. struct timeval timeDifference(struct timeval after, struct timeval before);
101. uint16_t getChecksum(uint16_t *msg, int length);
102.
103. void terminateMonitor();
```

C2 – File “QoSmonitor.c”

```
1.  #include "QoSmonitor.h"
2.
3.  struct sockaddr_in localIPaddress, dgIPaddress, proxyIPaddress, cnIPaddress,
    targetIPaddress;
4.  int defaultGateway, proxy, correspondentNode;
5.  int lastQualities[MMP_STATES];
6.  int quality[2];                    //array for quality: 0->quality, 1->target
7.  int indexSeq;
8.  int indexMod;
9.  int urgent;
10. int sd;
```

```
11.  int pid;
12.  int pktSeqNumSent;
13.  int replyExpected, replyArrived;
14.  int lostPackets;
15.  int delayStatistics[2]; //array for statistics: 0->minDelay, 1->maxDelay
16.  int totalSentPackets;
17.  int totalRcvPackets;
18.  int version;
19.
20.
21.  void usage(void)
22.  {
23.    puts("Usage: ./QoSmonitor.exe \
24.    -I interfaceName -l localIP -g defaultGatewayIP(req.) -D proxyName(opt.) -p
     proxyIP(opt.) -c correspondentNodeIP(opt.) -v version(opt.) -r(opt.)\n");
25.  }
26.
27.
28.  int main(int argc, char *argv[])
29.  {
30.          int key, src, dst, iosd, ret, i;
31.          struct hostent *p;
32.          char *strProxyIPaddress;
33.       char iobuf[IOCTL_BUF_LENGTH];
34.       struct ifconf ifcfg;
35.       struct ifreq *ifrq = NULL;
36.
37.
38.  /* CHECK INT TYPE
     ********************************************************************** */
39.
40.          if((sizeof(int)!=sizeof(long int)) || (sizeof(int)!= INT_SIZE))
41.          {
42.                  printf("Dimension of int and/or long int different from %d\n",
     INT_SIZE); exit(1);
43.          }
44.
45.
46.  /* CHECK ARGUMENTS
     ********************************************************************** */
47.
48.          puts("\n-------------------");
49.
50.          if(argc == 1)
```

```
51.         {
52.                 printf("Args: insert at least %d parametres\n", MIN_NUM_ARGS-1);
    usage(); exit(1);
53.         }
54.
55.         memset(&localIPaddress, 0, sizeof(localIPaddress));
56.         memset(&dgIPaddress, 0, sizeof(dgIPaddress));
57.         memset(&proxyIPaddress, 0, sizeof(proxyIPaddress));
58.         memset(&cnIPaddress, 0, sizeof(cnIPaddress));
59.         memset(&targetIPaddress, 0, sizeof(targetIPaddress));
60.         src = dst = 0;
61.         urgent = 1;
62.         defaultGateway = proxy = correspondentNode = 0;
63.         version = IPV4;
64.
65.         while((key = getopt(argc, argv, "I:l:g:D:p:c:v:r")) != EOF)
66.         {
67.                 switch(key)
68.                 {
69.                         case 'I':
70.                                 localIPaddress.sin_family = AF_INET;
71.                                 iosd = socket(PF_INET, SOCK_STREAM,  0);
72.                         if(iosd < 0)
73.                                 {
74.                         perror("socket ioctl failed");          exit(-1);
75.                         }
76.
77.                                 ifcfg.ifc_len = sizeof(iobuf);
78.                         ifcfg.ifc_buf = iobuf;
79.
80.                                 ret = ioctl(iosd, SIOCGIFCONF, &ifcfg);
81.                         if (ret < 0)
82.                                 {
83.                         perror("ioctl SIOCGIFCONF failed");     exit(-1);
84.                                 }
85.
86.                                 ifrq = ifcfg.ifc_req;
87.                         for(i=0; i < (ifcfg.ifc_len / sizeof(struct ifreq)); i++, ifrq++)
88.                         {
89.                         if(!strcmp(ifrq->ifr_name, optarg))
90.                                         {
91.                                                 ret = ioctl(iosd, SIOCGIFADDR, ifrq);
92.                                                 if(ret < 0)
93.                                                 {
```

```
94.                                     perror("ioctl SIOCGIFADDR failed");    exit(-1);
95.                                     }
96.                                     localIPaddress.sin_addr = ((struct
    sockaddr_in *)&(ifrq->ifr_addr))->sin_addr;
97.                                     printf("local: %s\n",
    inet_ntoa(localIPaddress.sin_addr));
98.                                     }
99.                             }
100.                            src = 1;
101.                            break;
102.                    case 'l':
103.                            localIPaddress.sin_family = AF_INET;
104.                            inet_aton(optarg, &localIPaddress.sin_addr);
105.                            printf("local: %s\n", inet_ntoa(localIPaddress.sin_addr));
106.                            src = 1;
107.                            break;
108.                    case 'g':
109.                            dgIPaddress.sin_family = AF_INET;
110.                            inet_aton(optarg, &dgIPaddress.sin_addr);
111.                            printf("default gateway: %s\n",
    inet_ntoa(dgIPaddress.sin_addr));
112.                            defaultGateway = 1;
113.                            break;
114.                    case 'D':
115.                            proxyIPaddress.sin_family = AF_INET;
116.                            p = gethostbyname(optarg);
117.                            strProxyIPaddress = inet_ntoa(*(struct in_addr *)p-
    >h_addr_list[0]);
118.                            inet_aton(strProxyIPaddress, &proxyIPaddress.sin_addr);
119.                            printf("proxy: %s\n",
    inet_ntoa(proxyIPaddress.sin_addr));
120.                            proxy = 1;
121.                            break;
122.                    case 'p':                               proxyIPaddress.sin_family =
    AF_INET;
123.                            inet_aton(optarg, &proxyIPaddress.sin_addr);
124.                            printf("proxy: %s\n",
    inet_ntoa(proxyIPaddress.sin_addr));
125.                            proxy = 1;
126.                            break;
127.                    case 'c':                               cnIPaddress.sin_family =
    AF_INET;
128.                            inet_aton(optarg, &cnIPaddress.sin_addr);
129.                            printf("correspondent node: %s\n",
```

```
    inet_ntoa(cnIPaddress.sin_addr));
130.                                correspondentNode = 1;
131.                                break;
132.                        case 'v':
133.                                if(atoi(optarg) == IPV6)
134.                                        version = IPV6;
135.                                break;
136.                        case 'r':
137.                                urgent = 0;
138.                                break;
139.                        default:
140.                                printf("getopt: must insert at least %d parametres\n",
    MIN_NUM_ARGS-1); usage(); exit(1);
141.                }
142.        }
143.
144.        puts("-------------------\n");
145.        if(!src)
146.        {
147.                printf("Args: insert option -I or -l for a source\n"); usage(); exit(1);
148.        }
149.        else if(!defaultGateway)
150.        {
151.                printf("Args: insert -g defaultGatewayIPaddress\n"); usage(); exit(1);
152.        }
153.
154.        pid = getpid();
155.        printf("pid = %d\n", pid);
156.        printf("version: IPv%d\n", version);
157.
158.        fflush(stdout);
159.
160.        ThreadStart();
161.
162.        return 1;
163. }
164.
165.
166. /* START MONITORING
    ********************************************************** */
167. void ThreadStart()
168. {
169.        int ret;
170.        pthread_t t;
```

```
171.
172.        ret = pthread_create(&t, NULL, RunMonitor, NULL);
173.        if(ret)
174.        {
175.                printf("pthread_create() failed: error = %d\n",ret); exit(-1);
176.        }
177.        else
178.                printf("created thread\n");
179.
180.        ret = pthread_join(t , NULL);
181.        if(ret != 0)
182.        {
183.                printf("pthread_join() failed: error = %d\n", ret); exit(-1);
184.        }
185.        printf("joined thread\n");
186.
187.
188.        printf("check end\n");
189.        fflush(stdout);
190.
191.        pthread_exit(NULL);
192. }
193.
194.
195. void *RunMonitor()
196. {
197.        fd_set rset, roriginal;
198.        int ret, maxfd;
199.        ssize_t nbytes;
200.        char receivedMsg[RCV_BUFFER_LENGTH];
201.        struct timeval alarm, tmp;
202.        int sendFreqSec, sendFreqUsec;
203.        int maxRcvFreqSec, maxRcvFreqUsec, remainingFreqSec,
    remainingFreqUsec;
204.        int timesToWaitForSwitch;
205.        float frequenceUsec;
206.        int target;
207.        int frequenceMsec, maxRcvFreqMsec;
208.
209.
210.        sd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
211.        if(sd < 0)
212.        {
213.        perror("socket raw failed"); exit(1);
```

```
214.        }
215.        printf("socket raw created\n");
216.        fflush(stdout);
217.
218.
219.        ret = bind(sd, (struct sockaddr*)&localIPaddress, sizeof(localIPaddress));
220.        if(ret < 0)
221.        {
222.        printf("bind() failed with IP %s\n", inet_ntoa(localIPaddress.sin_addr));
223.        perror("bind()"); exit(1);
224.        }
225.        printf("bind() on IP %s\n", inet_ntoa(localIPaddress.sin_addr));
226.
227.
228.        if(urgent)
229.        {
230.                frequenceUsec = (float)URGENT_FREQUENCE;
231.                timesToWaitForSwitch =
    (int)TIMES_TO_WAIT_FOR_SWITCH_URG;
232.        }
233.        else
234.        {
235.                frequenceUsec = (float)RELAXED_FREQUENCE;
236.                timesToWaitForSwitch =
    (int)TIMES_TO_WAIT_FOR_SWITCH_REL;
237.        }
238.        printf("- ping frequence: %.3f s\n- TimesToWaitForSwitch: %d\n",
239.             frequenceUsec/SEC_IN_USEC, timesToWaitForSwitch);
240.
241.        tmp.tv_sec = 0;
242.        tmp.tv_usec = frequenceUsec;
243.        normalizeTimeval(&tmp);
244.        sendFreqSec = tmp.tv_sec;
245.        sendFreqUsec = tmp.tv_usec;
246.        printf("sendFreqSec = %d, sendFreqUsec = %d\n\n", sendFreqSec,
    sendFreqUsec);
247.        frequenceMsec = (sendFreqSec*SEC_IN_MSEC)+
    (sendFreqUsec/USEC_IN_MSEC);       // in milliseconds
248.
249.        target = DEFAULT_GATEWAY;
250.
251.        signal(SIGINT, terminateMonitor);
252.        signal(SIGTSTP, terminateMonitor);
253.
```

```
254.        memset(delayStatistics, 0, sizeof(delayStatistics));
255.        delayStatistics[MIN_DELAY] = 1000000000;
256.        delayStatistics[MAX_DELAY] = -1;
257.        totalSentPackets = totalRcvPackets = 0;
258.
259.
260.        while(1)
261.        {
262.                /* VARIABLES INITIALIZATION
    ********************************************** */
263.                memset(lastQualities, -1, sizeof(lastQualities));
264.                memset(quality, 0, sizeof(quality));
265.                pktSeqNumSent = 0;
266.                indexSeq = indexMod = 0;
267.                replyExpected = replyArrived = 1;
268.                lostPackets = 0;
269.
270.                //set targetIPaddress
271.                switch(target)
272.                {
273.                        case DEFAULT_GATEWAY:
274.                                targetIPaddress.sin_family          =
    dgIPaddress.sin_family;
275.                                targetIPaddress.sin_addr    = dgIPaddress.sin_addr;
276.                                puts("-------------------------- SWITCH DEFAULT
    GATEWAY ----------------------------------------");
277.                                break;
278.                        case PROXY:
279.                                targetIPaddress.sin_family          =
    proxyIPaddress.sin_family;
280.                                targetIPaddress.sin_addr    = proxyIPaddress.sin_addr;
281.                                puts("-------------------------- SWITCH PROXY
    ------------------------------------------------");
282.                                break;
283.                        case CORRESP_NODE:
284.                                targetIPaddress.sin_family          =
    cnIPaddress.sin_family;
285.                                targetIPaddress.sin_addr    = cnIPaddress.sin_addr;
286.                                puts("-------------------------- SWITCH
    CORRESPONDENT NODE ---------------------------------------");
287.                                break;
288.                }
289.                quality[QUALITY_SCORE] = BEST_SCORE;
290.                quality[QUALITY_TARGET] = target;
```

```
291.
292.
293.                printf("totalSentPackets: %d\n", totalSentPackets);
294.                printf("totalRcvPackets: %d\n", totalRcvPackets);
295.
296.
297.                do
298.                {
299.                        ret = connect(sd, (struct sockaddr*)&targetIPaddress,
    sizeof(targetIPaddress));
300.                }while((ret<0) && (errno == EINTR));
301.                if(ret < 0)
302.                {
303.                        printf("connect() failed with IP %s\n",
    inet_ntoa(targetIPaddress.sin_addr));
304.                        perror("connect()"); exit(1);
305.                }
306.                printf("connect() with IP %s \n", inet_ntoa(targetIPaddress.sin_addr));
307.                fflush(stdout);
308.
309.
310.                tmp.tv_sec = 0;
311.                if(target == DEFAULT_GATEWAY)
312.                        tmp.tv_usec = (BEST_DELAY*USEC_IN_MSEC) * 2;
313.                else
314.                        tmp.tv_usec = (ALARM_DELAY*USEC_IN_MSEC) * 2;
315.                normalizeTimeval(&tmp);
316.                maxRcvFreqSec = tmp.tv_sec;
317.                maxRcvFreqUsec = tmp.tv_usec;
318.                printf("maxRcvFreqSec = %d, maxRcvFreqUsec = %d\n",
    maxRcvFreqSec, maxRcvFreqUsec);
319.                maxRcvFreqMsec = (maxRcvFreqSec*SEC_IN_MSEC)+
    (maxRcvFreqUsec/USEC_IN_MSEC); // in milliseconds
320.
321.                if(frequenceMsec < maxRcvFreqMsec)
322.                {
323.                        printf("SendFrequence must be greater or equal to
    maxRcvFrequence (>= %d)\n", (ALARM_DELAY*USEC_IN_MSEC) * 2);
    exit(1);
324.                }
325.
326.                tmp.tv_sec = 0;
327.                if(target == DEFAULT_GATEWAY)
328.                        tmp.tv_usec = frequenceUsec -
```

```
((BEST_DELAY*USEC_IN_MSEC) * 2);
329.                else
330.                        tmp.tv_usec = frequenceUsec -
((ALARM_DELAY*USEC_IN_MSEC) * 2);
331.                normalizeTimeval(&tmp);
332.                remainingFreqSec = tmp.tv_sec;
333.                remainingFreqUsec = tmp.tv_usec;
334.                printf("remainingFreqSec = %d, remainingFreqUsec = %d\n",
remainingFreqSec, remainingFreqUsec);
335.
336.
337.                FD_ZERO(&roriginal);
338.                FD_SET(sd, &roriginal);
339.                maxfd = sd + 1;
340.
341.
342.                while(1)
343.                {
344.                        //if there are commandline arguments proxy or mobileNode,
switch targetNode
345.                        if((quality[QUALITY_TARGET] == DEFAULT_GATEWAY)
&& (proxy || correspondentNode) &&
346.                          (pktSeqNumSent >= timesToWaitForSwitch) )
347.                        {
348.                                if(proxy)
349.                                        target = PROXY;
350.                                else if(correspondentNode)
351.                                        target = CORRESP_NODE;
352.
353.                                break;
354.                        }
355.                        else if((quality[QUALITY_TARGET] == PROXY) &&
(pktSeqNumSent >= MAX_PACKETS_PRCN))
356.                        {
357.                                if((correspondentNode) &&
(quality[QUALITY_SCORE] >= ALARM_SCORE))
358.                                        target = CORRESP_NODE;
359.                                else
360.                                        target = DEFAULT_GATEWAY;
361.
362.                                break;
363.                        }
364.                        //back to default gateway
365.                        else if((quality[QUALITY_TARGET] == CORRESP_NODE)
```

```
&& (pktSeqNumSent >= MAX_PACKETS_PRCN))
366.                    {
367.                            target = DEFAULT_GATEWAY;
368.                            break;
369.                    }
370.
371.
372.                    SendEchoRequests();
373.
374.                    //set timeout for sending echo request
375.                    alarm.tv_sec = maxRcvFreqSec;
376.                    alarm.tv_usec = maxRcvFreqUsec;
377.
378.
379.                    while(1)
380.                    {
381.                            rset = roriginal;
382.                            ret = select(maxfd, &rset, NULL, NULL, &alarm);
383.
384.
385.                            if(ret < 0)
386.                            {
387.                                    if(errno == EINTR) continue;
388.                                    else
389.                                    {
390.                                            perror("select() failed"); exit(1);
391.                                    }
392.                            }
393.
394.                            // packet loss check
395.                            //end of rcv timeout and replys are not yet arrived
396.                            else if((ret == 0) && (replyExpected == 1) &&
(replyArrived == 0))
397.                            {
398.                                    int tmpQ;
399.                                    lostPackets++;
400.
401.                                    if((quality[QUALITY_TARGET] == PROXY) ||
(quality[QUALITY_TARGET] == CORRESP_NODE))
402.                                            tmpQ = quality[QUALITY_SCORE] -
(2*ONE_LEVEL_SCORES);
403.                                    else
404.                                            tmpQ = quality[QUALITY_SCORE] -
ONE_LEVEL_SCORES;
```

```
405.                                    if(tmpQ < MIN_SCORE)
406.                                            tmpQ = MIN_SCORE;
407.                                    quality[QUALITY_SCORE] = tmpQ;
408.                                    lastQualities[indexMod] =
    quality[QUALITY_SCORE];
409.                                    indexSeq++;
410.                                    indexMod = indexSeq % MMP_STATES;
411.
412.                                    printf("------ # %d LOSTS PACKETS\n",
    lostPackets);
413.                                    printf("------ quality = %d\n\n",
    quality[QUALITY_SCORE]);
414.                                    if(lostPackets >= (MAX_PACKET_LOSS*2))
415.                                    {
416.                                            alarm.tv_sec = remainingFreqSec;
417.                                            alarm.tv_usec = remainingFreqUsec;
418.                                            replyExpected = 0;
419.                                            continue;
420.                                    }
421.                                    else
422.                                            break;
423.                            }
424.                            //end of rcv timeout and there are NO replys expected
425.                            else if((ret == 0) && (replyExpected == 1) &&
    (replyArrived == 1))
426.                            {
427.                                    alarm.tv_sec = remainingFreqSec;
428.                                    alarm.tv_usec = remainingFreqUsec;
429.                                    replyExpected = 0;
430.                                    continue;
431.                            }
432.                            //end of send timeout and there are no packet sent
433.                            else if((ret == 0) && (replyExpected == 0))
434.                            {
435.                                    break;
436.                            }
437.
438.
439.                            if(FD_ISSET(sd, &rset)) //new echo reply received
440.                            {
441.                                    memset(receivedMsg, 0, sizeof(receivedMsg));
442.                                    nbytes = read(sd, &receivedMsg,
    RCV_BUFFER_LENGTH);
443.                                    if(nbytes < 0)
```

```
444.                                    {
445.                                            if(errno == EINTR)
446.                                                    continue;
447.                                            else
448.                                            {
449.                                                    perror("read() failed"); exit(1);
450.                                            }
451.                                    }
452.
453.                                    ReadEchoReply(receivedMsg, nbytes);
454.                            }
455.                    }
456.            }
457.    }
458. }
459.
460.
461. /* SEND ECHO REQUESTS
     ********************************************************** */
462. void SendEchoRequests()
463. {
464.     char sendICMPmsg[SND_MSG_SIZE];
465.     int ret;
466.
467.     pktSeqNumSent++;
468.     if(pktSeqNumSent == MAX_INT)
469.             pktSeqNumSent = 1;
470.
471.     //IPv4
472.     if(version == IPV4)
473.     {
474.             struct icmp *ICMP4message;
475.             ICMP4message = (struct icmp*)sendICMPmsg;
476.             ICMP4message->icmp_type = ECHO_REQUEST;
477.             ICMP4message->icmp_code = 0;
478.             ICMP4message->icmp_cksum = 0;
479.             ICMP4message->icmp_id = pid;
480.             ICMP4message->icmp_seq = pktSeqNumSent;
481.             memset(ICMP4message->icmp_data, ' ', ICMP_DATA_LENGTH);
482.             gettimeofday((struct timeval*)ICMP4message->icmp_data, NULL);
483.             ICMP4message->icmp_cksum =
     getChecksum((u_short*)ICMP4message, SND_MSG_SIZE);
484.     }
485.     //IPv6
```

```
486.        else
487.        {
488.                struct icmp6_hdr *ICMP6message;
489.                ICMP6message = (struct icmp6_hdr*)sendICMPmsg;
490.                ICMP6message->icmp6_type = ICMP6_ECHO_REQUEST;
491.                ICMP6message->icmp6_code = 0;
492.                ICMP6message->icmp6_cksum = 0;
493.
494.                ICMP6message->icmp6_id = pid;
495.                ICMP6message->icmp6_seq = pktSeqNumSent;
496.
497.                memset((char *)(ICMP6message + 1), ' ', ICMP_DATA_LENGTH);
498.                gettimeofday((struct timeval*)(ICMP6message + 1), NULL);
499.
500.                ICMP6message->icmp6_cksum =
    getChecksum((u_short*)ICMP6message, SND_MSG_SIZE);
501.        }
502.
503.
504.        ret = sendto(sd, sendICMPmsg, SND_MSG_SIZE, MSG_NOSIGNAL, (struct
    sockaddr*)&targetIPaddress,
505.                  sizeof(targetIPaddress));
506.        if((ret < 0) && (errno == EPIPE))
507.        {
508.                perror("sendto() target host not responding");    exit(1);
509.        }
510.        if(ret < 0)
511.        {
512.                perror("sendto() failed");    exit(1);
513.        }
514.        printf("sent echo request to %s, pktSeqNumSent = %d, ICMPmsgLength =
    %d\n\n",
515.              inet_ntoa(targetIPaddress.sin_addr), pktSeqNumSent, SND_MSG_SIZE);
516.        fflush(stdout);
517.
518.
519.        replyExpected = 1;
520.        if((quality[QUALITY_TARGET] == DEFAULT_GATEWAY) &&
    (pktSeqNumSent <= IGNORE_FIRST_PACKETS))
521.                replyArrived = 1;
522.        else
523.                replyArrived = 0;
524.
525.        totalSentPackets++;
```

```
526.
527.        return;
528.  }
529.
530.
531.  /* RECEIVE ECHO REPLIES
      ********************************************************** */
532.  void ReadEchoReply(char *receivedMsg, int totalMsgLength)
533.  {
534.        struct timeval *sentTimePtr;
535.        struct timeval sentTime, spentTime;
536.        struct timeval now;
537.        int sequence, ok = 0, ICMPmsgLength = 0, rtt;
538.
539.        gettimeofday(&now, NULL);
540.
541.        //IPv4
542.        if(version == IPV4)
543.        {
544.              struct ip *IP4header;
545.              struct icmp *ICMP4header;
546.              int IP4headerLength;
547.
548.              IP4header = (struct ip*)receivedMsg;
549.              if(IP4header->ip_p != IPPROTO_ICMP)
550.              {
551.                    printf("not accepted protocol\n");
552.                    return;
553.              }
554.              IP4headerLength = IP4header->ip_hl << 2;
555.
556.              ICMP4header = (struct icmp*)(receivedMsg + IP4headerLength);
557.              if(ICMP4header->icmp_id != pid)
558.              {
559.                    printf("wrong pid\n");
560.                    return;
561.              }
562.
563.              ICMPmsgLength = totalMsgLength - IP4headerLength;
564.              if(ICMPmsgLength < MIN_ICMP_MSG_SIZE)
565.              {
566.                    printf("malformed packet\n");
567.                    return;
568.              }
```

```
569.
570.                 if(ICMP4header->icmp_type == ECHO_REPLY)
571.                 {
572.                         sentTimePtr = (struct timeval*)ICMP4header->icmp_data;
573.                         sequence = ICMP4header->icmp_seq;
574.                         ok = 1;
575.                 }
576.         }
577.         //IPv6
578.         else
579.         {
580.                 int ICMPmsgLength;
581.                 struct icmp6_hdr *ICMP6header;
582.                 int IP6headerLength = 40;
583.
584.                 ICMP6header = (struct icmp6_hdr *)(receivedMsg +
    IP6headerLength);
585.                 if(ICMP6header->icmp6_id != pid)
586.                 {
587.                         printf("wrong pid\n");
588.                         return;
589.                 }
590.
591.                 ICMPmsgLength = totalMsgLength - IP6headerLength;
592.                 if(ICMP6header->icmp6_type == ICMP6_ECHO_REPLY)
593.                 {
594.                         sentTimePtr = (struct timeval*)(ICMP6header + 1);
595.                         sequence = ICMP6header->icmp6_seq;
596.                         ok = 1;
597.                 }
598.         }
599.
600.         if(ok)
601.         {
602.                 sentTime = (struct timeval)(*sentTimePtr);
603.                 spentTime = timeDifference(now, sentTime);
                                        // elapsed time
604.                 rtt = (spentTime.tv_sec*SEC_IN_MSEC)+
    (spentTime.tv_usec/USEC_IN_MSEC);          // in milliseconds
605.                 printf("received echo reply from %s to %d VS %d, ICMPmsgLength =
    %d, rtt=%d ms\n",
606.                         inet_ntoa(targetIPaddress.sin_addr), sequence, pktSeqNumSent,
    ICMPmsgLength, rtt);
607.
```

```
608.                CalculateQuality(rtt, sequence);
609.
610.                if(sequence >= pktSeqNumSent)
611.                {
612.                        replyArrived = 1;
613.                        lostPackets = 0;
614.                }
615.        }
616.  }
617.
618.
619.  /* GENERATE QUALITY SCORE
     ********************************************************** */
620.  void CalculateQuality(int rtt, int seq)
621.  {
622.        int oneWayDelay, sum=0, i;
623.        int tmpQuality, resultQuality;
624.        float tmp;
625.        int round;
626.
627.        if(seq < pktSeqNumSent) // packet with seqNum less than expected are
     discarded
628.        {
629.                printf("discarded\n"); return;
630.        }
631.        else
632.        {
633.                // calculate one-way delay from RTT ceiling
634.                tmp = round = 0;
635.                tmp = rtt % 2;
636.                if(tmp)
637.                        round = 1;
638.                oneWayDelay = (rtt/2) + round;
639.                tmp = round = 0;
640.
641.                if(oneWayDelay < delayStatistics[MIN_DELAY])
642.                        delayStatistics[MIN_DELAY] = oneWayDelay;
643.                if(oneWayDelay > delayStatistics[MAX_DELAY])
644.                        delayStatistics[MAX_DELAY] = oneWayDelay;
645.        }
646.
647.
648.        if((quality[QUALITY_TARGET] == DEFAULT_GATEWAY) && (seq <=
     IGNORE_FIRST_PACKETS))
```

```
649.                resultQuality = BEST_SCORE;
650.        else
651.        {
652.                // calculate quality value depending on one-way delay
653.                if(oneWayDelay <= BEST_DELAY)
654.                        tmpQuality = BEST_SCORE;
655.                else if(oneWayDelay <= GOOD_DELAY)
656.                        tmpQuality = GOOD_SCORE;
657.                else if(oneWayDelay <= ALARM_DELAY)
658.                        tmpQuality = ALARM_SCORE;
659.                else if(oneWayDelay <= BAD_DELAY)
660.                        tmpQuality = BAD_SCORE;
661.                else if(oneWayDelay <= WORST_DELAY)
662.                        tmpQuality = POOR_SCORE;
663.                else if(oneWayDelay > WORST_DELAY)
664.                        tmpQuality = WORST_SCORE;
665.
666.
667.                // MMPP, calculate resultQuality
668.                lastQualities[indexMod] = tmpQuality;
669.                for(i = 0; i<MMP_STATES; i++)
670.                {
671.                        if(lastQualities[i] != -1)
672.                                sum += lastQualities[i];
673.                        else
674.                                break;
675.                }
676.
677.                tmp = (float)sum / i;
678.                resultQuality = sum / i;
679.                if((tmp-resultQuality)>=0.5)
680.                        round = 1;
681.                resultQuality += round;
682.
683.                lastQualities[indexMod] = resultQuality;
684.                indexSeq++;
685.                indexMod = indexSeq % MMP_STATES;
686.
687.                // limits for resultQuality
688.                if(resultQuality > MAX_SCORE)
689.                        resultQuality = MAX_SCORE;
690.                else if(resultQuality < MIN_SCORE)
691.                        resultQuality = MIN_SCORE;
692.        }
```

```
693.
694.        // print quality if different from preceding quality value
695.        if(quality[QUALITY_SCORE] != resultQuality)
696.                quality[QUALITY_SCORE] = resultQuality;
697.
698.        totalRcvPackets++;
699.
700.        printf("------ oneWayDelay = %d ms, quality = %d\n\n", oneWayDelay,
    quality[QUALITY_SCORE]);
701.
702.        return;
703. }
704.
705.
706.
707. /* FUNCTIONS MANIPULATING STRUCT TIMEVAL
    ************************************** */
708.
709. int normalizeTimeval(struct timeval *t)
710. {
711.        if(t->tv_usec >= SEC_IN_USEC)
712.        {
713.                t->tv_sec  += ( t->tv_usec / SEC_IN_USEC);
714.                t->tv_usec =  ( t->tv_usec % SEC_IN_USEC);
715.        }
716.        return(1);
717. }
718.
719. struct timeval timeDifference(struct timeval after,struct timeval before)
720. {
721.        struct timeval diff;
722.        diff.tv_sec = 0;
723.        diff.tv_usec = 0;
724.
725.        normalizeTimeval(&after);
726.        normalizeTimeval(&before);
727.
728.        diff.tv_sec = after.tv_sec - before.tv_sec;
729.        if(diff.tv_sec < 0)
730.        {
731.                diff.tv_sec = 0;
732.                diff.tv_usec = 0;
733.        }
734.        else
```

```
735.        {
736.                diff.tv_usec = after.tv_usec - before.tv_usec;
737.                if(diff.tv_usec < 0)
738.                {
739.                        if (diff.tv_sec > 0)
740.                        {
741.                                /* Devo scalare di uno i secondi e sottrarli ai micro
    secondi
742.                                ossia aggiungo 1000000 all'ultima espressione */
743.                                diff.tv_sec =diff.tv_sec - 1;
744.                                diff.tv_usec += SEC_IN_USEC;
745.                        }
746.                        else
747.                                diff.tv_usec = 0;
748.                }
749.        }
750.
751.        return(diff);
752. }
753.
754.
755. /* CHECKSUM FUNCTION FOR ICMP
    ************************************************** */
756. uint16_t getChecksum(uint16_t *msg, int length)
757. {
758.        int nleft = length;
759.        uint32_t sum = 0;
760.        uint16_t *w = msg;
761.        uint16_t answer = 0;
762.
763.        while(nleft > 1)
764.        {
765.                sum += *w++;
766.                nleft -= 2;
767.        }
768.
769.        if(nleft == 1)
770.        {
771.                *(unsigned char*)(&answer) = *(unsigned char*)w;
772.                sum += answer;
773.        }
774.
775.        sum = (sum >> 16) + (sum & 0xffff);
776.        sum += (sum >> 16);
```

```
777.        answer = ~sum;
778.        return(answer);
779.  }
780.
781.
782.  /* SIGINT or SIGTSTP when pressing CTRL-C/Z
      ************************************ */
783.  void terminateMonitor()
784.  {
785.        char *strTarget;
786.        int percLostPkts;
787.
788.        switch(quality[QUALITY_TARGET])
789.        {
790.              case DEFAULT_GATEWAY:
791.                    strTarget = "Default Gateway";
792.                    break;
793.              case PROXY:
794.                    strTarget = "Proxy";
795.                    break;
796.              case CORRESP_NODE:
797.                    strTarget = "Correspondent Node";
798.                    break;
799.        }
800.
801.        percLostPkts = ((totalSentPackets-totalRcvPackets) *100)/totalSentPackets;
802.
803.        printf("\nStatistics for last target %s:\n \
804.        last quality = %d\n \
805.        packets sent = %d\n \
806.        packets received = %d\n \
807.        packet loss = %d%%\n \
808.        size of data = %d bytes\n",
809.             strTarget,
810.             quality[QUALITY_SCORE],
811.             totalSentPackets,
812.             totalRcvPackets,
813.             percLostPkts,
814.             SND_MSG_SIZE);
815.
816.        if(totalRcvPackets  != 0)
817.        {
818.              printf("\
819.        minimum one-way delay = %d ms\n \
```

```
820.        maximum one-way delay = %d ms\n \
821.        average one-way delay = %.2f ms\n",
822.                    delayStatistics[MIN_DELAY],
823.                    delayStatistics[MAX_DELAY],
824.                    (float)(delayStatistics[MIN_DELAY]
    +delayStatistics[MAX_DELAY])/2);
825.        }
826.
827.        fflush(stdout);
828.
829.        exit(1);
830.  }
```