Analisi e studio dei problemi e dei casi d'uso di VOIP (SIP) over UMTS

Sommario

Introduzione 2
Stato dell'arte 4
Voice over IP (VoIP) 4
Session Initiation Protocol (SIP) 4
Asterisk PBX 6
RTP - Realtime Transport Protocol 6
RTCP - Realtime Transport Control Procol 7
Pacchetti RTP 8
Pacchetti RTCP 10
SR - Sender Report 11
Applicazione: RealTimeLogger 13
Modulo TcpDumpLogger 14
Modulo di Logging e Gestione database SQLite 17
Parametri registrati 18
Interfaccia grafica 19
Test effettuati 20
Conclusioni e Sviluppi futuri 23

Introduzione

La crescente diffusione e l'abbassamento dei costi delle connessioni internet per dispositivi mobili hanno portato allo sviluppo di applicazioni multimediali capaci di fornire i servizi di audio in modalità real-time anche per dispositivi mobili.

Questo complesso servizio multimediale si è sviluppato soprattutto grazie all'introduzione ed alla ricerca, a partire dalla metà degli anni '90, della tecnologia Voice over IP (VoIP).

La rete VoIP permette non solo di effettuare conversazioni telefoniche su reti a commutazione di pacchetto mediante il protocollo IP, ma anche una facile convergenza e cooperazione con dati di tutt'altro tipo come possono essere flussi video, messaggi di testo e così via. Questa soluzione porta un notevole risparmio di risorse ed una maggiore scalabilità ma, contemporaneamente, richiede anche la ricerca intensiva di nuovi standard per i servizi multimediali e di soluzioni legate alla qualità del servizio. Ad esempio, le attuali applicazioni nel mondo reale della telefonia VoIP si trovano a dover affrontare problematiche legate a problemi di latenza (sostanzialmente si deve ridurre il tempo di transito e di elaborazione dei dati durante le conversazioni) e di integrità dei dati (prevenire perdite e danneggiamenti delle informazioni contenute nei pacchetti per mantenere la corretta coerenza temporale dei pacchetti stessi).

La tecnologia VoIP richiede dunque un protocollo di comunicazione per il trasporto "affidabile" dei dati (pacchetti voce su IP) che, nella grande maggioranza delle implementazioni, è il protocollo RTP (Real-time Transport Protocol); inoltre consente ampi margini di interoperabilità, mediante opportune piattaforme di transcodifica, fra protocolli di signaling come ad esempio SIP (Session Initiation Protocol) ed ambienti fra loro completamente diversi.

Come è facilmente intuibile, la presenza di molteplici soluzioni poneva diverse annose questioni come: l'interoperabilità, la trasparenza delle applicazioni rispetto alle piattaforme proprietarie su cui erano state sviluppate, la sicurezza e la qualità del servizio fruibile in uno scenario di mercato fin troppo frammentato. Per superare questi ostacoli si iniziò un forte studio sulla standardizzazione del servizio di conferencing da parte di diversi enti quali: ITU (International Telecommunication Union), ETSI (European Telecommunications Standards Istitute), IETF (Internet Enginerig Task Force) e 3GPP (3rd Generation Partnership Project).

Questo lavoro di tesi si è concentrato sull'analisi dei problemi legati all'utilizzo di applicazioni voip in ambiente Android, un sistema operativo sviluppato dall'OHA (Open Headset Alliance) a partire dal novembre 2007, completamente opensource, giunto alla release 4.0.3. Android risulta essere una piattaforma ancora in fase di forte evoluzione e sviluppo ma, ad oggi, è già in grado di fornire il framework e gli strumenti necessari (tra cui un versatile emulatore) per lo sviluppo e la ricerca di molteplici tipi di applicazioni e servizi multimediali, attraverso l'uso delle API (Application Program Interface) standard di Java ed il linguaggio XML (eXtensible Markup Language). La possibilità di utilizzare una piattaforma aperta, consente di poter sviluppare un'applicazione

trasparente rispetto alla piattaforma stessa, aiutando notevolmente il lavoro di convergenza, integrazione e sviluppo comunitario.

Si sono quindi dapprima appresi il funzionamento, le potenzialità e le problematiche legate all'ambiente Android per poi sviluppare un client di logging che fosse in grado di monitorare le interfacce di rete e tutte le informazioni rese disponibili dalla piattaforma Android, quali: potenza segnale, id cella, posizione gps.. etc.

Si sono poi affrontati problematiche relative all'integrazione di componenti come il noto strumento di sniffing TCPDUMP, testando l'applicativo su dispositivi reali differenti affrontando diverse casistiche di utilizzo.

Stato dell'arte

Negli ultimi anni, di pari passo con l'evoluzione delle tecnologie VoIP, si è manifestato un crescente interesse al servizio di comunicazione mobile basata su internet che ha condotto, grazie ad un notevole sforzo in termini di ricerca, alla progettazione ed alla realizzazione di diverse soluzioni per dispositivi mobili. La diffusione di dispositivi mobili con grandi capacità multimediali ha permesso inoltre che queste tecnologia approdassero con facilità su piattaforme mobili. Grazie all'elevata capacità elaborativa di dispositivi di ultima generazione quali Apple iPhone, dispositivi con sistema Android, Blackberry e SymbianOS, le software houses non hanno tardato a rilasciare applicativi che implementassero la tecnologia voip e ogni suo derivato, permettendo lo scambio di contenuti multimediali che estendessero i classici concetti di telefonata o SMS (Short Message Service).

Nello specifico per dispositivi Android troviamo diverse alternative opensource e non, che danno la possibilità all'utente finale di inviare/ricevere audio verso un contatto registrato allo stesso servizio. Tra queste troviamo i noti applicativi disponibili sulla maggior parte dei dispositivi mobili, quali: Fring, Nimbuzz, Skype, tutti closed source e utilizzanti protocolli proprietari, tra gli applicativi opensource troviamo SipDroid usante protocolli SIP per il segnaling e RTP per il trasporto dati, oltre al client integrato che è stato introdotto a partire dalla release 2.3 del sistema Android. E' proprio su quest'ultimo che si è focalizzato lo studio iniziale di questo lavoro.

Voice over IP (VoIP)

Voice over IP, acronimo VoIP, è una tecnologia che consente di effettuare una conversazione telefonica sfruttando una connessione internet, o un'altra rete che utilizza il protocollo IP. Più specificatamente con VoIP si intende l'insieme dei protocolli di comunicazione che permettono tale conversazione. Il vantaggio principale di questa tecnologia è quello di sfruttare l'allocazione dinamica del protocollo IP, dando luogo quindi ad una commutazione di pacchetto e risolvendo il problema di assegnare obbligatoriamente parte della banda disponibile per ogni telefonata effettuata. Sulla rete vengono instradati pacchetti di dati contenenti informazioni vocali, codificati in formato digitale, e ciò solo quando è necessario (solo quando uno degli utenti collegati sta parlando). Oggi grazie ad alcuni provider è possibile effettuare telefonate anche verso la rete telefonica tradizionale Public Switched Telephone Network (PSTN), rendendo questa tecnologia un punto di forza per sviluppi di applicazioni future.

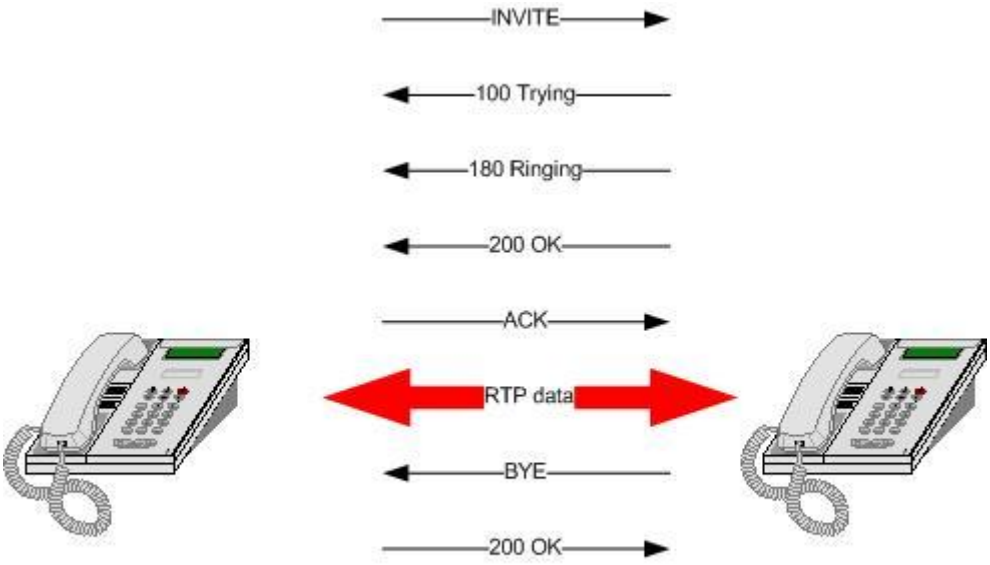
Session Initiation Protocol (SIP)

Il protocollo SIP (Session Initiation Protocol) è un protocollo basato su IP standardizzato dall'IETF nella RFC 3261 e impiegato principalmente per applicazioni di telefonia su IP o VoIP.

SIP gestisce in modo generale una sessione di comunicazione tra due o più entità, ovvero fornisce meccanismi per instaurare, modificare e terminare (rilasciare) una sessione. Attraverso il protocollo SIP possono essere trasferiti dati di diverso tipo (audio, video, messaggistica testuale, ecc). Inoltre, SIP favorisce un'architettura modulare e scalabile, ovvero capace di crescere con il numero degli

utilizzatori del servizio. Queste potenzialità hanno fatto sì che il SIP sia, oggi, il protocollo VoIP più diffuso nel mercato residenziale e business, sorpassando di molto altri protocolli quali H.323 ed MGCP.

Il modello usato per la sintassi del protocollo SIP è text-based, derivato dall' HTTP. Per instaurare una sessione, avviene un three-way handshaking (concettualmente simile a quello che avviene con il protocollo TCP) per la negoziazione dei codec da utilizzare per la trasmissione dei dati multimediali, utilizzando il protocollo SDP (Session Description Protocol).



A SIP call session between 2 phones – without SIP PROXY

Inoltre è impiegabile sia in contesti client-server sia in contesti peer to peer, è facilmente estendibile, programmabile ed è indipendente dal protocollo di trasporto.

Una sessione SIP è composta da transazioni, una transazione è un insieme di messaggi, identificabile da un transaction-ID, un identificativo che ne specifica la sorgente, la destinazione e il numero di sequenza.

Questo tipo di protocollo oltre che nelle ambito della telefonia su IP si sta però sempre più diffondendo anche negli applicativi di conferencing garantendo prestazioni elevate.

Grazie alle specifiche definite dall' IETF ed alle molteplici librerie preesistenti in rete è possibile inoltre portare questo protocollo su dispositivi mobili senza particolari difficoltà.

Il protocollo SIP è affiancato solitamente dal protocollo SDP che si occupa della descrizione dei media che saranno utilizzati nel trasporto dei dati multimediali, questo protocollo nasce come componente di SAP (Session Announcement Protocol) ma tutt'oggi è utilizzato in congiunzione ad altri protocolli quali: RTP (Real-time Transport Protocol), RTSP (Real-time Streaming Protocol) e lo stesso SIP.

Unendo poi SIP e SDP al protocollo di trasporto RTP si riesce ad ottenere una trasmissione con compensazione del jitter e rilevazione di pacchetti fuori sequenza, comuni durante le trasmissioni su una rete IP.

Storicamente progettato per funzionare su multicast, il protocollo RTP è principalmente utilizzato sulla base del protocollo di trasmissione UDP (User Datagram Protocol), un protocollo di tipo connectionless, che non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi, quindi di minore affidabilità ma in compenso molto rapido ed efficiente per le applicazioni time-sensitive, quali VoIP e streaming audio/video.

Asterisk PBX

Asterisk è il centralino SIP utilizzato come server nelle comunicazioni voip prese in esame da questo lavoro. Asterisk è una implementazione libera di un software PBX (Private Branch eXchange) che permette di ottenere le stesse funzioni offerte da altri sistemi proprietari con una spesa decisamente inferiore ed una maggiore flessibilità. La prima stesura di Asterisk venne realizzata da Mark Spencer, un ingegnere informatico statunitense che, attorno all'anno 2000, fondò una società che sviluppa schede d'interfaccia FXS ed FXO. Spencer sviluppò Asterisk per favorire la diffusione delle interfacce Digium e, permettendone la libera distribuzione, solleticò l'interesse di moltissimi utilizzatori professionali ed appassionati, diffondendone largamente così la conoscenza. Asterisk è oggi un punto di riferimento nel settore. La completezza dei suoi contenuti e la sua affidabilità lo rendono una piattaforma ideale per una vasta gamma di applicazioni: è utilizzato come elemento portante per realizzare centralini in grado cioè di utilizzare sia le linee telefoniche tradizionali sia i canali IP, applicazioni per la gestione di Call Center ed altro ancora.

RTP - Realtime Transport Protocol

RTP è l'acronimo per *Realtime Transport Protocol* (protocollo di trasporto in tempo reale) e assolve a tutti i compiti necessari per instaurare una comunicazione real time.

Abbiamo detto che RTP offre un servizio di trasporto in tempo reale appoggiandosi al protocollo UDP quindi dovrà sopperire a certe mancanze di UDP, in particolar modo:

• Fornire informazioni sul payload type (il formato dei dati inviati, ovvero la codifica effettuata sull'informazione annidata al pacchetto che vedremo potrà variare da pacchetto a pacchetto)

• Il sequence number (identificatore di sequenza per poter ristabilire l'ordine dei pacchetti ed eventualmente valutare quanti pacchetti sono andati perduti o scartati)

• Il timestamping (posizione temporale in millisecondi dei dati contenuti nel pacchetto, utile a sincronizzare ad esempio audio e video e vedremo abbassare/eliminare il jitter)

• Identificare le sorgenti per sincronizzare diversi flussi (ad esempio in una conferenza ci può essere la sovrapposizione della voce di due persone che parlano contemporaneamente)

RTP è il protocollo di trasporto e non fornisce nessun meccanismo per garantire la consegna del pacchetto, l'arrivo in tempo, la qualità del servizio e l'ordine di ricezione; il tutto viene demandato agli strati sottostanti.

Esempio di utilizzo
Consideriamo l'utilizzo di RTP per il trasporto vocale. Supponiamo che la voce sia codificata in PCM a 64 kbit/sec (normale comunicazione telefonica) e che i dati vengano raccolti ogni 20 ms. 1 *sec*=64 *kbit*⇒1*msec*=64 *bit*⇒20*msec*=1280*bit*=160*byte*

Questo blocco di 20 ms (160 byte) viene incapsulato in un pacchetto RTP dove, tra le varie informazioni, verrà specificato il formato (PCM 64kbit/s) col quale sono stati campionati i dati, successivamente verrà passato al livello di trasporto per essere inviato. Il destinatario riceverà il pacchetto RTP tramite il livello di trasporto, vedrà il metodo di codifica dei dati (PCM), e passerà il tutto all'applicazione che riprodurrà l'informazione.

Profili RTP
I profili RTP permettono di definire parametri comuni in una comunicazione semplicemente indicando il profilo di riferimento. I profili Audio/Video standard sono definiti nell'RFC3551. E' possibile specificare un profilo proprietario semplicemente seguendo le istruzioni fornite nell'RFC3555.

RTCP - Realtime Transport Control Procol

RTCP e l'acronimo di *Realtime Transport Control Procol* ed e un protocollo di controllo che lavora parallelamente ad RTP e fornisce periodicamente informazioni sulla QoS e altre variabili della sessione, come ad esempio:

• Numero di pacchetti persi tra due host

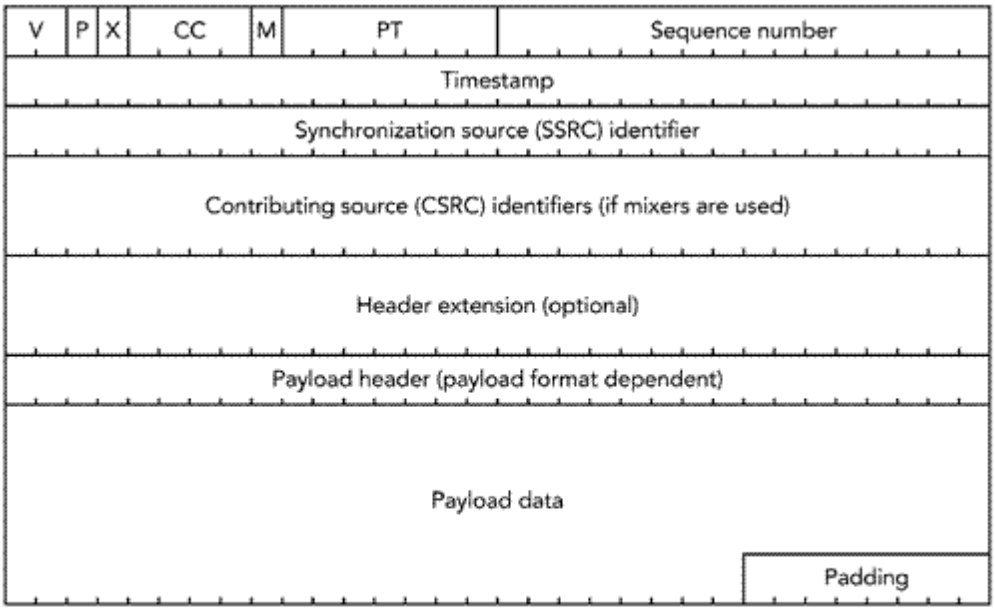
• Jitter medio rilevato (vedremo in seguito le tecniche per il trattamento del jitter)

• Informazioni sugli utenti attivi nella conferenza (canonical name CNAME, telefono, nome etc..)

• Desiderio di uno o più utenti di lasciare la sessione

RTP non necessita di RTCP per funzionare ma è consigliato utilizzare il protocollo di controllo per poter offrire un'esperienza migliore agli utilizzatori del nostro software e interfacciarsi con altri software che lo utilizzano.

Pacchetti RTP



V = version number
P = padding
X = extensions
CC = count of contributing sources
M = marker
PT = payload type

V – Version Number

Indica la versione del protocollo, ad oggi siamo alla versione 2 quando si superera la 3a versione sarà possibile specificarla nel campo opzionale "Header Extension".

Campo a 2 bit

P – Padding

Indica se il pacchetto è stato riempito con bit di padding o meno. L'ultimo ottetto del padding contiene il numero di ottetti da ignorare alla fine dei dati compreso se stesso. L'operazione di padding non è obbligatoria per tutti i pacchetti RTP. Viene spesso utilizzata quando si richiede la crittazione dei dati dove, un formato a lunghezza "stabile", e preferibile.

Campo a 1 bit

X – Extensions

Se impostato ad uno indica la presenza di una estensione dopo l'header statico.

Campo a 1 bit

CC – Count of contributing sources

Solitamente un pacchetto è inviato da una sola sorgente, nel caso in cui fossero presenti più sorgenti (ad esempio quando siamo in presenza di un mixer), CC indica il numero di sorgenti presenti all'interno del campo CSRC. Campo a 4 bit

M – Marker

Abilitando il bit M possiamo specificare al receiver che i dati contenuti nel pacchetto sono, ad esempio, un keyframe di un video oppure l'inizio di una sequenza audio con sottotitoli da reindirizzare o altre azioni scaturibili lato receiver dal nostro client.

Campo a 1 bit

PT – Payload Type

Identifica il formato dei dati inclusi nel pacchetto permettendo all'applicazione un corretto playout.
Campo a 7 bit

Sequence Number

Ogni pacchetto inviato dal sender contiene un sequence number incrementale. Non possono esistere due pacchetti con lo stesso sequence number. Questo campo permette al receiver di ripristinare il corretto ordine dei dati e individuare il numero di pacchetti persi (o scartati per discordanze temporali). Il valore iniziale del sequence number viene impostato casualmente e successivamente incrementato di 1 ad ogni invio. Una conseguenza importante della lunghezza del campo a 16 bit e che quando si raggiunge il massimo (2^16) si torna a 0.

Campo a 16 bit

Timestamp

Il timestamp viene registrato nell'istante di cattura del primo byte della fonte. Se ad esempio sappiamo che ogni pacchetto contiene 80 ms di campione, ogni pacchetto avrà un timestamp di distanza pari a 80ms. Questa informazione risulterà essenziale per un corretto playback del contenuto del pacchetto lato receiver e per il calcolo del jitter.

Campo a 32 bit

SSRC – Syncronization Source Identifier

Identificativo della sorgente che ha creato il contenuto del payload. Solitamente è un valore numerico generato a caso dalla sorgente quando inizializza la sessione ed è univoco all'interno della sessione stessa. Nel caso di collisione, in quanto non è da escludere che due host non generino lo stesso SSRC, vi è un algoritmo risolutivo.

Durante la sessione ad ogni SSRC viene associato il CNAME relativo all'host.

Campo a 32 bit

CSRC – Contributing source

Contiene una lista delle fonti presenti nel payload. Il numero di fonti è specificato nel campo CC (vedi sopra) e può essere al massimo di 15 elementi. Questo campo viene utilizzato dai mixer per inserire gli SSRC di ogni fonte presente nel payload.

Lista (da 0 a 15) di campi da 32 bit. Max 60 byte.

Payload Header

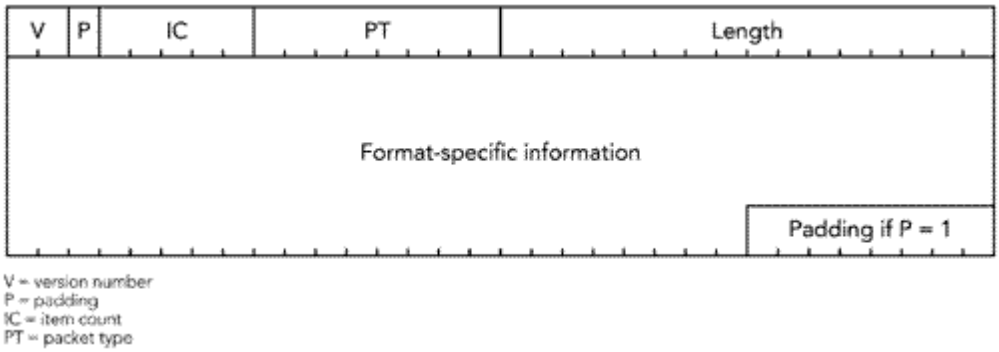
In alcuni casi è necessario specificare informazioni addizionali utili all'applicazione per effettuare una migliore interpretazione dei dati del pacchetto, tali informazioni possono essere specificate all'interno del campo.

Campo a 32 bit

Payload Data

L'effettiva informazione codificata da inviare.

Pacchetti RTCP



I pacchetti RTCP sono formati da una parte iniziale (header) identica per tutti e il resto del pacchetto viene differenziato in base alle 5 tipologie previste.

V – Version Number

Versione del pacchetto RTCP, attualmente siamo alla 2a versione.

Campo a 2 bit

P – Padding

Stesso significato del bit nel pacchetto RTP (vedi sopra). Diventa obbligatorio nel caso in cui non si raggiunga una lunghezza pari ad un multiplo di 32.

Campo a 1 bit

IC – Item Count

Indica il numero di report presenti all'interno di un pacchetto RTCP.

Campo a 5 bit

PT – Packet Type

Attualmente sono definiti 5 tipologie di pacchetti RTCP, in futuro tale numero potrà espandersi con altre tipologie di report.

Campo a 8 bit

Length

Identifica la lunghezza complessiva del pacchetto RTCP. Il suo valore rappresenta un multiplo di 32 bit in quanto ogni pacchetto RTCP è composto da multipli di 32 bit (vedi padding).

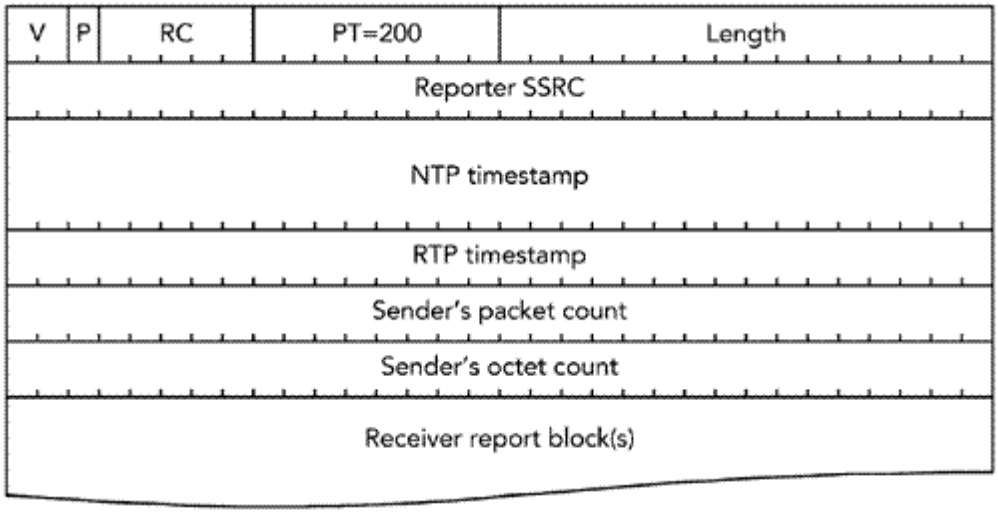
Campo a 16 bit

Il protocollo prevede 5 tipi di report: Receiver Report, Sender Report, Source Description, Bye, App. Ci concentreremo solo sul tipo di report usato nella nostra analisi, il Sender Report.

SR - Sender Report

I Sender Report vengono inviati dai sender poco dopo aver terminato l'invio di un pacchetto RTP e

contengono le seguenti informazioni:



Reporter SSRC

NTP Timestamp

Timestamp in formato NTP dell'istante di invio del pacchetto RTP.

Campo a 64 bit

RTP Timestamp

Corrisponde allo stesso istante di lettura dell'NTP Timestamp ma relativo alla macchina locale.

Campo a 32 bit

Sender's Packet Count

Numero complessivo di pacchetti RTP inviati dall'inizio della sessione. Nel caso in cui nascano collisioni sul nome tale valore viene resettato.

Campo a 32 bit

Sender's Octect Count

Numero complessivo di byte di dati (sono escluse intestazioni etc...) inviati dall'inizio della sessione.

Campo a 32 bit

Loss fraction
Indica il rapporto dei pacchetti persi su quelli attesi.

$$lossFraction = \frac{numero\ di\ pacchetti\ persi\ nell'intervallo}{numero\ di\ pacchetti\ attesi\ nell'intervallo} * 256$$

numero di pacchetti attesi nell'intervallo=seq_num ultimo pacchetto−seq_num primo pacchetto

Campo a 8 bit

Cumulative number of packet lost
Numero di pacchetti persi in tutta la sessione (non solo nell'intervallo attuale).
Campo a 24 bit

Extended highest sequence number received
Indica l'ultimo sequence number ricevuto calcolando anche i wrap around.
Campo a 32 bit

Interarrival jitter
Rappresenta il jitter medio rilevato dal destinatario per ogni pacchetto e si calcola nel seguente modo:
Ri=Timestamp dell'i-esimo pacchetto rilevato dal receiver
Si=Timestamp indicato dal mittente nel pacchetto RTP dell'i-esimo pacchetto

relative _ transit _ time=Ri−Si

Ora definiano la differenza tra due pacchetti:

$D(i\ ,\ j)=(\ Rj−Ri\ )−(\ Sj−Si\ )=(\ Rj−Sj\ )−(\ Ri−Si\ )$

e il valore del campo Interarrival Jitter:

$$J_i = J_{i-1} + \frac{(|D(i-1,i)| - J_{i-1})}{16}$$

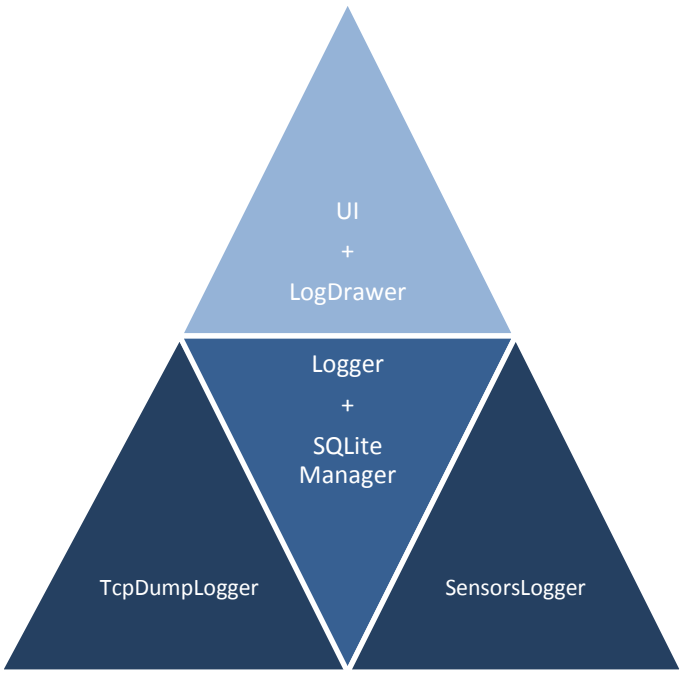
La divisione per 16 è un fattore correttivo per permettere una lenta variazione del valore.
Campo a 32 bit

LSR (Last Sender Report)
Se nessun Report SSRC SR è stato ricevuto il campo viene impostato a 0 (zero) altrimenti viene preso l'ultimo sender report ricevuto (quello con NTP timestamp maggiore).

Applicazione: RealTimeLogger

Al fine di testare l'utilizzo della comunicazione voip su rete UMTS/EDGE è stato necessario implementare un client di logging ex-novo vista l'assenza di soluzioni dedicata in ambiente Android.

E' stato anzitutto progettato un diagramma a blocchi che descrivesse le parti funzionali dell'applicazione da implementare:



Come si può notare l'applicazione è stata progettata a livelli, il livello più alto gestisce l'interfaccia grafica quindi nel livello immediatamente inferiore c'è il modulo che si occupa dell'interazione fra i logger e la gestione dei dati attraverso un database SQLite. Nel livello inferiore ritroviamo i vari logger dei sensori e il logger dell'applicativo tcpdump necessario per intercettare e quindi elaborare i dati della sessione sip/voip in corso.

Abbiamo utilizzato un approccio bottom-up per lo sviluppo partendo quindi dall'implementazione dei logger, quindi passando al livello intermedio del gestore del DB SQLite, infine concludendo con lo sviluppo dell'interfaccia grafica.

Modulo TcpDumpLogger

Il modulo TcpDumpLogger è stato implementato nella classe Java omonima attraverso l'ausilio combinato delle funzioni Android per l'esecuzione di binari nativi dell'ambiente Linux sottostante (in questo caso il binario tcpdump compilato per architettura arm) e l'utilizzo delle Regex (Regular Expression) per l'estrazione dei dati stessi.

Il modulo è stato implementato sviluppando due sottomoduli:

1. Logger RTP
2. Logger RTCP

Il logger RTP si occupa dell'analisi (attraverso tcpdump) dei pacchetti RTP in arrivo sulla porta scelta dal client sip in uso, in particolare al fine di rendere la procedura client-indipendente è stata sviluppata una procedura d'analisi euristica per cui il Logger inizialmente si mette in ascolto di tutti i pacchetti RTP ricevuti dal client, quindi identifica come probabile porta destinazione della comunicazione SIP quella ricevente il maggior rate di pacchetti.

Dopo aver stabilito la porta destinazione e quindi anche l'ip del server sorgente il modulo RTP inizia ad analizzare i pacchetti ricevuti al fine di calcolare il client packet jitter ed avvia il sottomodulo responsabile dell'analisi RTCP.

La configurazione standard di un server Asterisk (SIP/VOIP) prevede solitamente che il demone invii le statistiche della comunicazione attraverso il protocollo RTCP. La prassi vuole che le statistiche di un flusso dati RTP siano inviate sulla prima porta dispari successiva alla porta usata per la comunicazione RTP (che solitamente è pari). Quindi il sottomodulo RTCP si occupa di analizzare tali pacchetti destinati a tale porta ricavandone le informazioni utili alla stima della qualità della comunicazione voip in corso.

In particolare il sottomodulo RTP per analizzare i pacchetti RTP ricevuto avvia il binario `tcpdump` con le seguenti opzioni:

```
tcpdump -ni ANY -tt -l -s 0 -T rtp src SERVER_IP and src port SERVER_PORT
```

e stampa un output del tipo:

```
1329926382.383270 IP 151.77.167.250.13098 > 109.53.94.229.43800: udp/rtp 33 c3  53667
182080
```

Analizzando in dettaglio l'output si può notare che sono stampati molti campi di interesse ovviamente attraverso l'uso delle Regular Expression tali dati sono filtrati e sono selezionati i soli parametri necessari al calcolo del client jitter (operazione che deve svolgere appunto il sottomodulo RTP):

Timestamp **1329926382.383270**
Source Address **151.77.167.250**
Source Port **13098**
Destination Address **109.53.94.229**
Destination Port **43800**
Protocol **udp/rtp**
RTP Payload size (B) **33**
RTP Media type **c3**
RTP Header "Marker' field ("*"=y " "=n) *
RTP Sequence number **53667**
RTP Timestamp **182080**

Il sottomodulo RTP si limita a recuperare il sequence number ed il timestamp da ogni pacchetto RTP attraverso la seguente Regular Expression:

```
Pattern p=Pattern.compile("([0-9]+\\.[0-9]+) IP
\\b\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\b\\.[0-9]+ >
\\b\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\b\\.[0-9]+: udp/rtp [0-9]{2} [a-z][0-9]*
([0-9]+) ([0-9]+)");
```

Quindi il sottomodulo RTP una volta recuperato il timestamp si occupa del calcolo del client jitter (interarrival jitter) attraverso la formula nota:

```
diff=(currentRTime-lastRTime)-((currentSTime-lastSTime)/8); // divido per 8 per
ottenere i ms dato che prima erano 160 campioni a 8khz

interjitter=interjitter+((Math.abs(diff)-interjitter)/16.0F);
```

E quindi comunica il valore atteso al modulo superiore di Logging e Gestione DB SQLite:

```
Logger.log(time, "CLIENT_JITTER", interjitter);
```

Il sottomodulo RTCP per analizzare i pacchetti RTCP ricevuto avvia il binario tcpdump con le seguenti opzioni:

```
tcpdump -ni ANY -tt -l -s 0 -T rtcp src SERVER_IP and dst port CLIENT_PORT
```

e stampa un output del tipo:

```
1329993270.615020 IP 151.77.167.250.12323 > 2.195.1.33.33591:  sr @3538981837.65 161280
1008p 33264b 6l 16496s 477j @0.00+14.17 sdes 12
```

Analizzando in dettaglio l'output si può notare che sono stampati molti campi di interesse ovviamente attraverso l'uso delle Regular Expression tali dati sono filtrati e sono selezionati i soli parametri necessari al salvataggio delle seguenti statistiche: Packets_Sent, Packets_Lost, Bytes_Sent, Server_Jitter (operazione che deve svolgere appunto il sottomodulo RTP):

Timestamp **1329993270.615020**
Source Address **151.77.167.250**
Source Port **12323**
Destination Address **2.195.1.33**
Destination Port **33591**
Type of RTCP Packet ***sr***
NTP Timestamp reference **@3538981837.65**
Media timestamp reference **161280**
Number of packets sent **1008p**
Number of Bytes sent **33264b**
Cumulative number of packets lost **6l**
extended last seq number received **16496s**
Jitter **477j**
orig. ts from last rr from this src **@0.00**
time from recpt of last rr to xmit time **+14.17**
No of Bytes of Source Description (sdes) data in the report *sdes* ***12***

Il sottomodulo RTCP si limita a recuperare le statistiche di interesse da ogni pacchetto RTCP attraverso la seguente Regular Expression:

```
Pattern p=Pattern.compile("[0-9]+\\.[0-9]+ IP
\\b\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\b\\.[0-9]+ >
\\b\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\b\\.[0-9]+:  sr @([0-9\\.]+) [0-9]+ ([0-
9]+)p ([0-9]+)b ([0-9]+)l [0-9]+s ([0-9]+)j [0-9@\\.\\+]+ sdes [0-9]+");
```

Quindi il sottomodulo RTCP comunica le statistiche attese al modulo superiore di Logging e Gestione DB SQLite:

```
Logger.log(time, "SERVER_PKT_SENT", curr_totalpkts-last_totalpkts);
Logger.log(time, "SERVER_BYTES_SENT", curr_totalbytes-last_totalbytes);
Logger.log(time, "SERVER_PKT_LOST", curr_totalpktslost-last_totalpktslost);
Logger.log(time, "SERVER_JITTER", jitter);
```

Modulo di Logging e Gestione database SQLite

Il modulo di Logging consente di registrare in un database SQLite i dati ricevuti dai listener dei sensori e dal modulo di analisi dei pacchetti dalla rete. Il modulo mostra una interfaccia con metodi statici, quindi accessibili da qualsiasi parte del programma, in modo da rendere facile l'aggiunta di un nuovo valore.

```
public static void startLog(Context context)
synchronized static public void stopLog()
public static boolean isRecording()
```

I metodi sopra elencati servono per avviare o interrompere il logger, una chiamata di log quando il logger non è in fase di registrazione non avrà alcun effetto.

```
public static void log(long time, String what, float value)
public static void log(long time, String what, double value)
public static void log(long time, String what, int value)
public static void log(long time, String what, long value)
public static void log(long time, String what, String value)
```

Un singolo log è caratterizzato da un tempo (in millisecondi) dell'orario in cui si è verificato l'evento, un campo chiave (what) che indica a quale valore si riferisce il log e un campo valore. Il campo valore può avere tipi differenti tra cui: numeri interi, numeri in virgola mobile e stringhe. Le chiamate ai metodi log sono asincrone, in modo da evitare di bloccare la funzione che ha chiamato un metodo `log` prima di aver memorizzato il valore nel database.

I log ricevuti sono inseriti in una tabella SQLite di nome `log`. La tabella presenta una colonna primaria `time` (dove viene inserito l'orario dell'evento) ed un insieme di colonne secondarie di tipo what (dove viene inserito il valore dell'evento)

Time	LOG_STATUS	BATT_LVL	BATT_VLT	BATT_TEMP	TEL_STATE	GSM_CELL
1331160728232	START					
1331160730135		68	3775	240		
1331160730137					0	64202b5f0a2
1331160730138						

La prima volta che il database viene creato viene eseguito il seguente comando per la creazione della tabella log:

```
db.execSQL("CREATE TABLE log (time INTEGER PRIMARY KEY DESC);");
```

Inizialmente quindi la tabella log contiene solo la colonna time. Quando un nuovo log viene aggiunto, se la colonna corrispondente al parametro what non esiste viene modificata la tabella in modo da aggiungere tale colonna con il seguente comando:

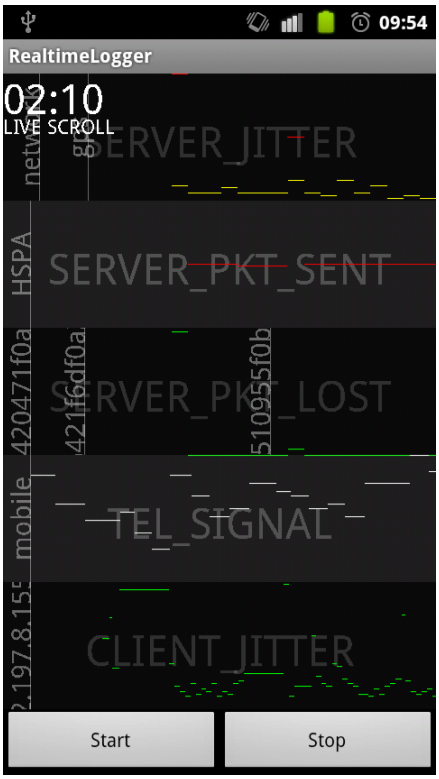
```
database.execSQL("ALTER TABLE log ADD "+what+' '+type+';');
```

Parametri registrati

I parametri memorizzati sono numerosi e di diversa natura, la tabella seguente riassume i log memorizzati e il loro significato

Tipo di evento (what)	Tipo valore	Esempio	Descizione
LOG_STATUS	TEXT	START	Evento di start o stop del logger
BATT_LVL	INTEGER	68	Livello batteria %
BATT_VLT	INTEGER	3775	Tensione batteria mV
BATT_TEMP	INTEGER	240	Temperatura batteria (decimi di °C)
BATT_PLG	INTEGER	2	Stato del connettore di ricarica
TEL_STATE	INTEGER	0	Stato della linea GSM
GSM_CELL	TEXT	64202b5f0a2	Id della cella GSM connessa
TEL_DATA_STATE	INTEGER	2	Stato della connessione dati
TEL_DATA_NET	TEXT	HSPA	Tipo di rete a cui si è connessi
TEL_SIGNAL	INTEGER	14	Segnale telefonico (-dB)
IP	TEXT	109.53.74.151	Ip
CONN	TEXT	Mobile	Tipo di connessione (mobile / wifi)
CLIENT_JITTER	REAL	2.20342343	Jitter client (ms)
SERVER_PKT_SENT	INTEGER	250	Pacchetti inviati dal server
SERVER_BYTES_SENT	INTEGER	8250	Byte inviati dal server
SERVER_PKT_LOST	INTEGER	0	Pacchetti persi dal server
SERVER_JITTER	REAL	3.32423423	Jitter server (ms)
LOC_LON	REAL	14.219977855682	Longitudine (decimale)
LOC_LAT	REAL	40.855450630186	Latitudine (decimale)
LOC_PROV	TEXT	gps	Sorgente posizione (network / gps)

Interfaccia grafica



L'interfaccia grafica consiste di due schermate principali, la prima per gestire lo start e lo stop del logger e la visualizzazione dei dati in tempo reale. La seconda schermata invece serve per esportare i dati memorizzati in diversi formati.

Una volta avviato il programma per iniziare una nuova sessione di registrazione è sufficiente premere il bottone start. A questo punto il programma avvia un servizio in background che inizia a registrare i log ricevuti dai sensori. Una volta che la registrazione è iniziata, il dispositivo può essere usato normalmente per eseguire altre operazioni, tra cui appunto una conversazione in voip (ma anche la registrazione della posizione gps o la durata della batteria).

Scorrendo con le dita lo schermo è possibile visualizzare grafici di sessioni precedenti, inoltre toccando con due dita sullo schermo (pinch-to-zoom) è possibile eseguire lo zoom in un punto preciso nel tempo.

La schermata è divisa in righe, ogni riga permette la visualizzazione di un grafico a linee e di un evento singolo (mostrato come una linea verticale). Per esempio il cambio di una cella gsm o il cambio dell'ip è un evento mentre il jitter o il livello del segnale telefonico è rappresentato come un grafico a linee. La descrizione del grafico viene indicato sullo sfondo mentre il tipo dell'evento non è descritto, ma è di facile interpretazione (per esempio se leggo HSPA riconosco subito che si tratta del tipo di connessione mobile)

Il tempo in alto a sinistra indica la finestra temporale di visualizzazione dei dati (nel caso dell'esemio mostrato in figura corrisponde a 2min e 10sec). Se la scritta LIVE SCROLL è verde indica che la finestra di visualizzazione si sposterà in modo da visualizzare sempre gli ultimi eventi memorizzati. Ogni grafico inoltre può avere un colore diverso,in modo da rendere più leggibile l'insieme dei dati.

La mappatura delle informazioni visualizzate sullo schermo è statica, non è quindi possibile scegliere quali grafici o quali eventi visualizzare.

Per interrompere la registrazione dei dati è sufficiente premere il pulsante Stop.

La seconda schermata (accessibile premendo il tasto Esporta dati dal menu) consente di estrarre i dati dal database e convertirli in formati utilizzabili da programmi esterni. Abbiamo implementato l'estrazione dei dati in un formato Matlab (*.m) per la visualizzazioni di grafici e in un formato Google earth (*.kml) per visualizzare la traccia del percorso effettuato.

Il programma cerca le sessioni create nel database e permette l'estrazione di alcune o tutte le sessioni.

I dati esportati sono memorizzati nella scheda di memorizzazione esterna disponibile in un elevato numero di dispositivi mobili Android nella cartella `realtimelogger`. I dati esportati per Matlab possone essere utili per eseguire calcoli statistici o di correlazione tra gli eventi memorizzati (come per esempio il confronto tra il livello del segnale cellulare e il jitter). I dati esportati in Google Earth possono invece essere utili per correlare i dati in base al percorso effettuato.

Test effettuati

Per verificare il corretto funzionamento dell'applicazione sviluppata abbiamo effettuato diversi test in varie condizioni. Il primo è stato quello di testare l'applicazione in situazioni ottimali con un buona ricezione di segnale HSDPA non in movimento, il risultato è stato quello di una buona comunicazione priva di ritardi e/o interruzioni, come mostrato nel seguente grafico. Dal grafico si nota infatti che il packet loss risulta praticamente nullo ed il ritardo lato client e server è minimo.

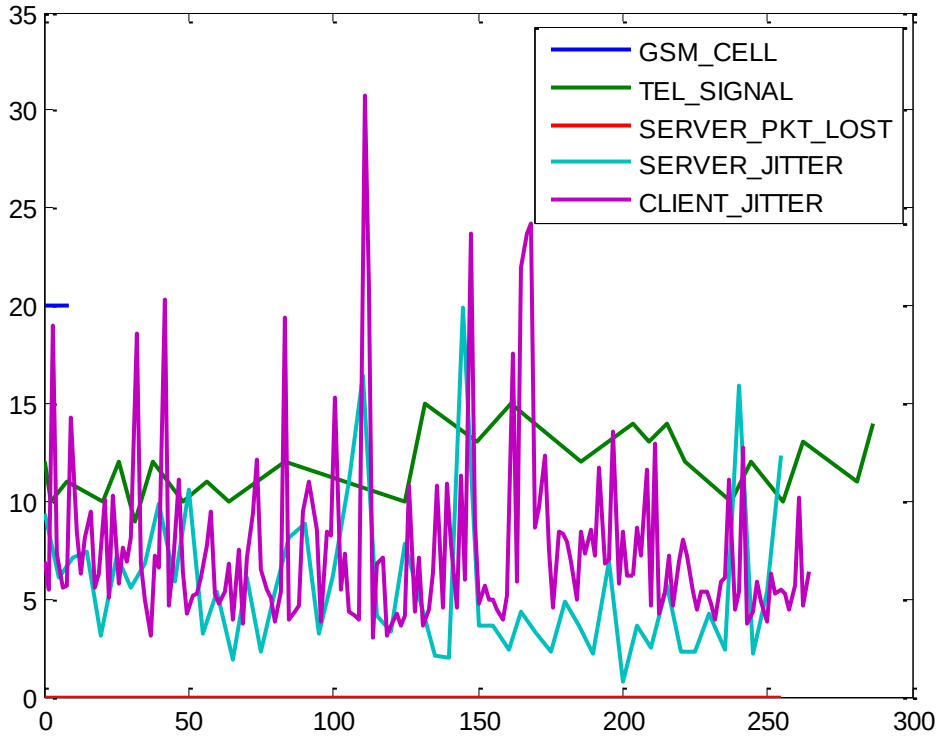


Figura 1 C/A5 Comunicazione non in movimento in modalità HSDPA senza pacchetti persi

Nel secondo test invece è stata analizzata la comunicazione in movimento (in auto 80km/h circa), come si può notare infatti il grafico seguente mostra come in presenza di un cambio cella ci sia un brusco aumento del packet loss ed un relativo incremento sia del client che del server jitter. La comunicazione è stata pressocché soddisfacente anche se non paragonabile ad una tradizionale comunicazione mobile, in alcuni casi infatti il numero di pacchetti persi è stato così elevato da causare una temporanea interruzione della comunicazione, ma non la effettiva terminazione. Per fini puramente dimostrativi è stato anche tracciato un breve grafico del percorso effettuato, grazie all'ausilio delle coordinate geografiche raccolte attraverso l'uso del ricevitore GPS integrato.

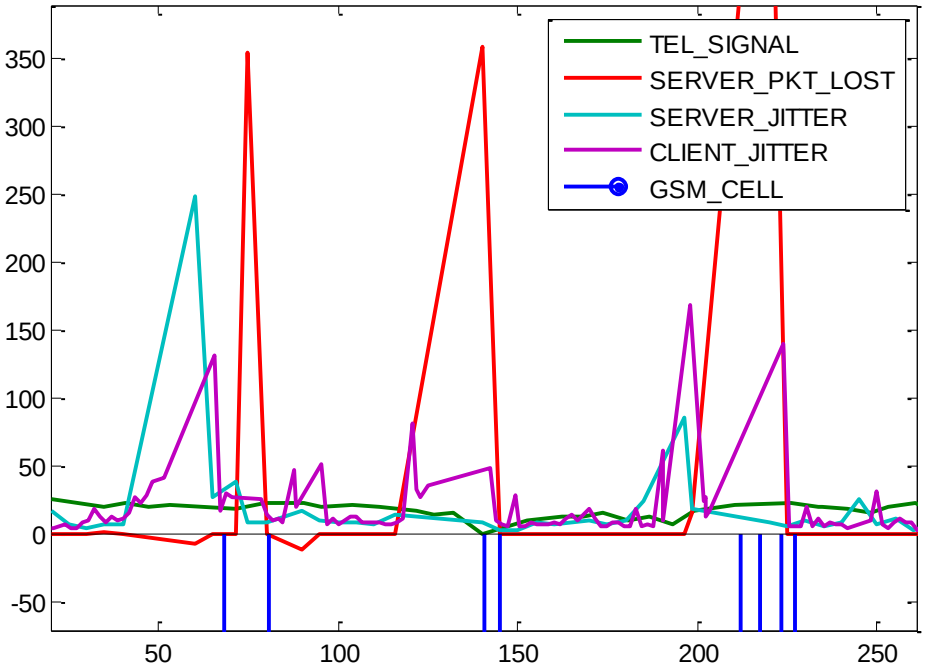
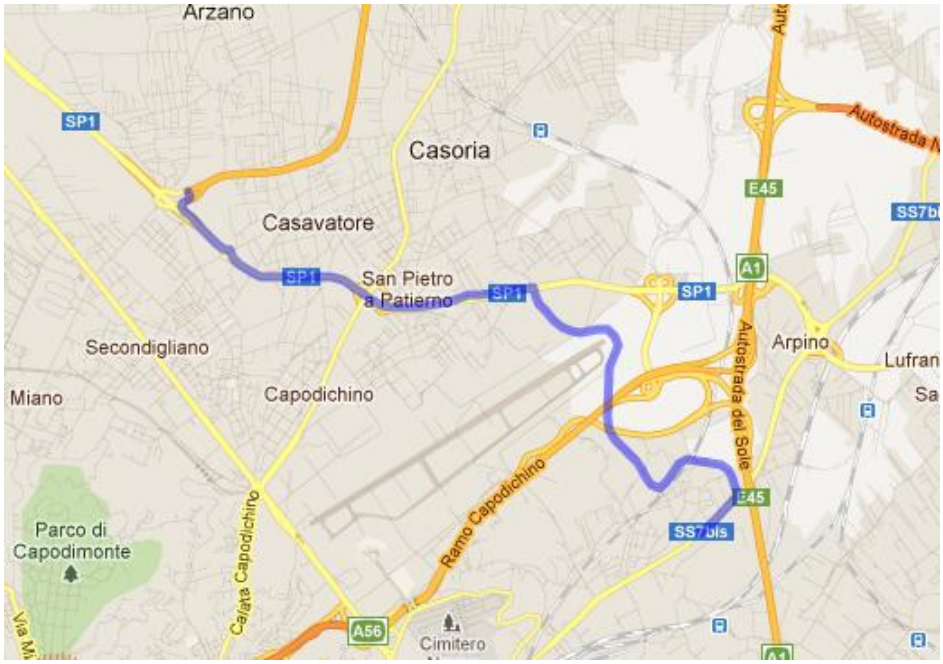


Figura 2 B/AA.50 Comunicazione in movimento con numerosi cambi cella (in blu). Modalità UMTS



L'ultimo test effettuato (in movimento) è stato quello di 'forzare' il telefono cellulare a funzionare in modalità EDGE in modo da valutare le prestazioni confrontandole con quelle effettuate in modalità UMTS. I risultati non sono stati soddisfacenti, come si può notare dal grafico sia il packet loss che il jitter lato server e lato client sono risultati superiori ai valori ottenuti in modalità UMTS. Tuttavia considerando la vasta copertura UMTS degli odierni operatori di telefonia mobile il passaggio in modalità EDGE resta un evento occasionale, del tutto tollerabile in una comunicazione del genere.

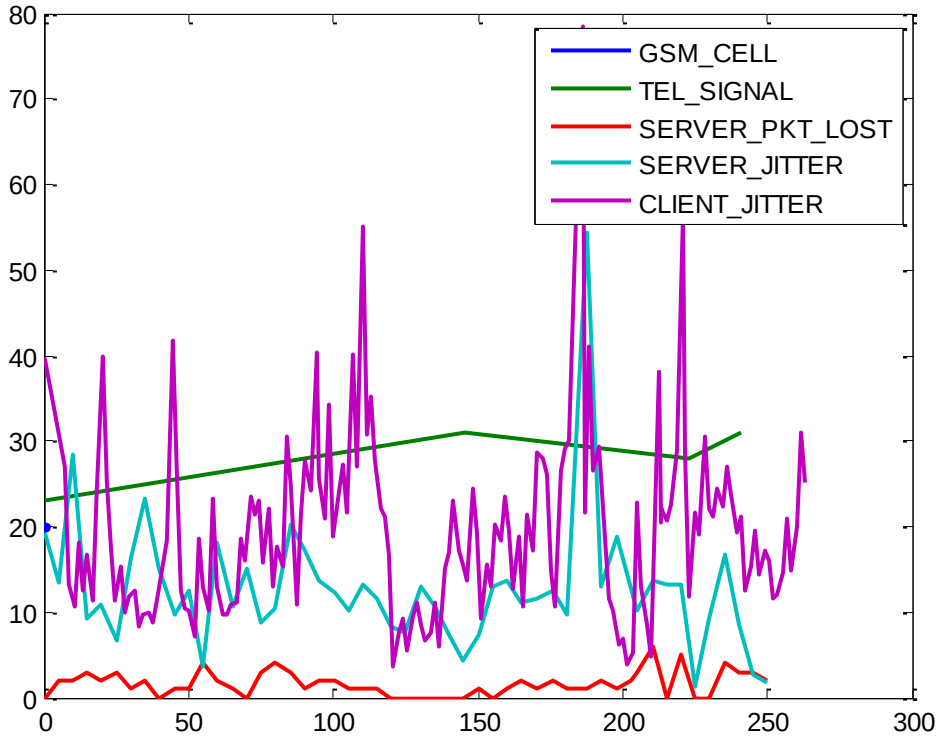


Figura 3 C/7.59 Comunicazione in movimento in modalità EDGE

Conclusioni e Sviluppi futuri

Pertanto si può concludere che l'uso del Voip su dispositivi mobili soprattutto in movimento ha una qualità audio non paragonabile alla tradizionale conversazione telefonica mobile. Questo come si è accennato è dovuto a vari fattori legati al cambio cella e alla potenza del segnale telefonico. Inoltre lato software non sono stati utilizzati meccanismi che si adattassero ai ritardi e al packet loss dovuto alla trasmissione audio realtime su connessione dati mobile.

Tuttavia in caso di condizioni 'statiche', in presenza di buon segnale UMTS la comunicazione è stata più che soddisfacente, garantendo ritardi ridotti e bassa percentuale di packet loss.

Guardando al futuro potrebbe essere interessante integrare nuovi sensori e effettuare l'analisi delle comunicazioni con protocolli non RTP (es. Skype).