ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di laurea in Informatica

MULTIPLEXING DI PROTOCOLLI SIP E RTP SU CANALI VIRTUALI MULTIPERCORSO: OSCURAMENTO PACCHETTI

Relatore:

Chiar.mo Prof.

VITTORIO GHINI

Presentata da:

LORENZO SAPORETTI

Seconda sessione

2010 / 2011

Indice

Capitolo 1

Introduzione

I dispositivi mobili per la comunicazione multimediale sono diventati negli ultimi anni una realtà quotidiana. Questi dispositivi come smartphone e tablet vengono dotati di interfacce wireless oltre che di normale connessione 3G tipica dei cellulari, riuscendo quindi ad avere una maggiore capacità di connessione rispetto ai normali computer portatili.

Tramite il “Voice over Internet Protocol”, tecnologia utilizzata per la comunicazione vocale nel web, si riesce utilizzando la sola connessione ad internet, a creare vere e proprie telefonate. È facile notare come questa tecnologia riesca, con un costo minore, ad eguagliare la normale comunicazione utilizzata con i cellulari; proprio per questo motivo la creazione di applicazioni che sfruttano il VoIP è una pratica sempre piu diffusa.

Uno dei protocolli più utilizzati nelle chiamate VoIP è sicuramente SIP (Sessione Initiation Protocol), implementato da diversi programmi che si occupano di comunicazione attraverso la rete. Questo protocollo permette, tra le altre cose, di instaurare una connessione con un altro utente senza sapere il suo IP, ma utilizzando il suo identificatore univoco. Senza questo protocollo bisognerebbe quindi essere in possesso dell'IP dell'utente con cui si vuole instaurare una connessione, informazione che in molti casi è variabile. Le caratteristiche di questo protocollo come anche quelle del VoIP verranno analizzate in dettaglio nel capitolo 2, che si occupa di mostrare lo scenario nel quale è

stato progettato il canale.

Questo progetto di tesi si basa sull'architettura ABPS, un insieme di regole e comportamenti per le chiamate VoIP, tramite la quale si vuole massimizzare la qualità di questa comunicazione e risolvere il problema dell'interruzione della comunicazione in caso di cambio di tipo di connessione da parte del dispositivo mobile, e quindi il cambio di rete e di IP.

In questa tesi, realizzata in collaborazione con Luca Trioschi, si vuole studiare e realizzare un canale virtuale di comunicazione che ha lo scopo di permettere lo scambio di qualunque tipo di dato utilizzando l'architettura ABPS, indipendentemente dal protocollo utilizzato per lo scambio di messaggi e dalla eventuale presenza di firewall o NAT che proteggono la connessione ad internet del dispositivo che ne fa uso, oltre a minimizzare il keepalive sulle porte mantenute aperte sulla rete globale.

Differentemente dal collega Luca Trioschi, questa tesi si vuole soffermare maggiormente sugli aspetti riguardanti l'oscuramente dei pacchetti che attraversano il canale virtuale creato. Gli aspetti progettuali riguardanti l'implementazione di questa tesi verranno trattati nel capitolo 5 mentre i risultati ottenuti implementando e studiando questo canale virtuale possono essere letti nel capitolo 7.

Capitolo 2

Scenario

2.1 I dispositivi e la rete

Questo progetto si sviluppa nel contesto delle applicazioni per VoIP che operano su dispositivi mobili dotati di una o più interfacce di comunicazione wireless e mira a fornire un protocollo che implementa un canale di trasmissione virtuale che opera sfruttando tutte queste interfacce.

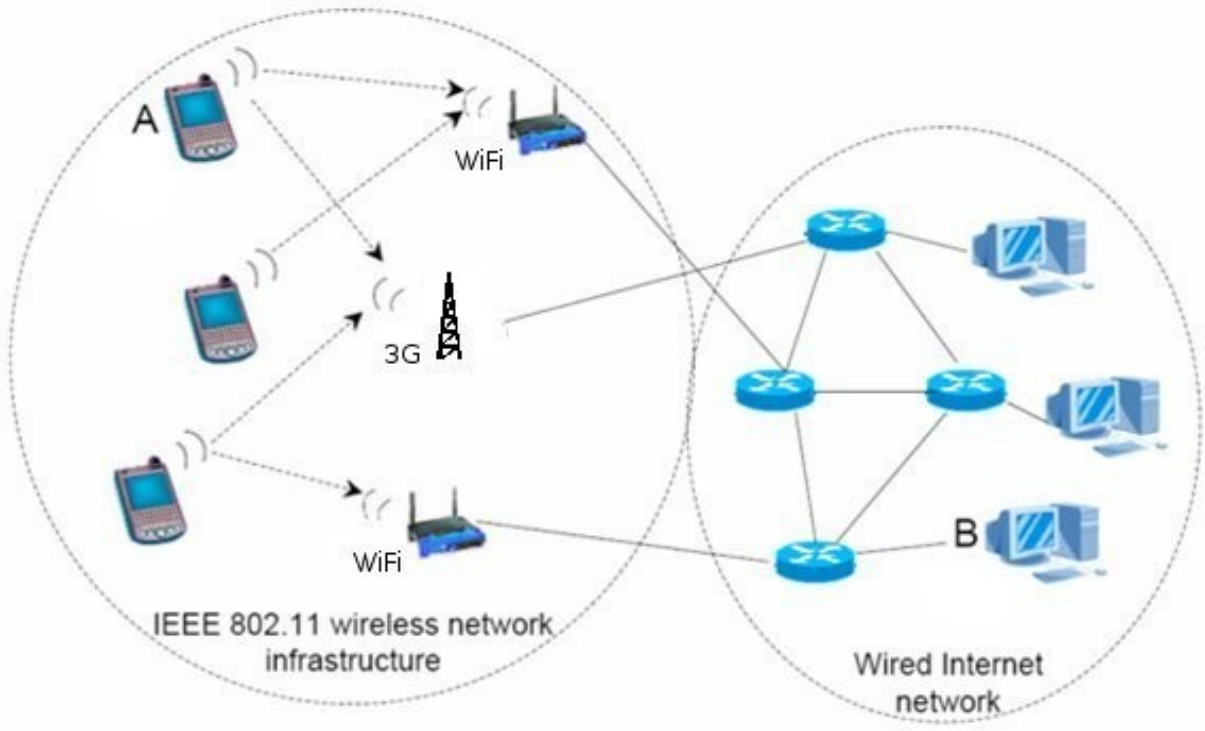

Figura 2.1

Nella figura 2.1 è mostrata una possibile situazione di due terminali VoIP connessi alla

rete.

Il client A è attivo su un dispositivo mobile dotato di più interfacce wireless che possono essere WiFi standard 802.11 a/b/g/n o anche 3G (ad esempio UMTS). Questo dispositivo mobile si suppone essere locato in un un'area urbana che quindi offre la copertura 3G e anche nelle vicinanze di un dispositivo WiFi al quale può connettersi e attraverso il quale può accedere ad internet.

Il client B invece si trova su un computer fisso dotato di una connessione internet che può essere diretta o mediata da un proxy o un firewall.

La possibilità di avere più interfacce wireless eterogenee sul nodo mobile è un vantaggio in quanto permette al dispositivo di rimanere continuamente connesso anche nel caso in cui una delle connessioni non sia più disponibile. Questa situazione è tutt'altro che remota in quanto l'utente di questi dispositivi, muovendosi, può uscire dall'area di copertura di una di queste connessioni. Grazie alla presenza di molteplici interfacce di rete è possibile cambiare access point e tipo di connessione senza compromettere la continuità del collegamento ad internet (handover). Questo comportamento viene definito nelle specifiche del 3GPP come “Voice Call Continuity” (VCC).

Un secondo vantaggio derivante dal numero di connessioni wireless disponibili è la possibilità di scegliere con quale interfaccia di rete rimanere connessi nel caso siano disponibili più access point. Il modello “Always Best Connected” (ABC) suggerisce infatti di selezionare la migliore interfaccia di rete tra quelle disponibili e di mantenere il collegamento alla rete tramite quella. Quando poi le prestazioni dell'interfaccia attualmente attiva cominciano a degradarsi, si procede alla ricerca di una nuova connessione con la quale mantenere il collegamento, sostituendola all'attuale. La selezione della migliore interfaccia di rete da utilizzare viene fatta seguendo non solo il criterio della qualità del segnale, ma basandosi anche su fattori come il costo del servizio, la copertura di rete, la velocità di trasmissione, la sicurezza e le preferenze

dell'utente.

L'architettura “Always Best Packet Switching” (ABPS) si basa su un modello come questo e ha l'obiettivo di massimizzare la qualità del servizio offerto all'utente del dispositivo mobile, sfruttando al meglio tutte le interfacce disponibili.

2.2 Il protocollo SIP

Il “Session Initiation Protocol” (SIP) è un protocollo studiato per creare, aggiornare e terminare sessioni di scambio di dati multimediali, come possono essere delle chiamate video e/o audio, tra due o più partecipanti. All'interno di questo protocollo ogni utente viene identificato tramite il suo nome utente e il dominio presso il quale è registrato, ad esempio se il nome utente è “luca.trioschi” e il dominio “ekiga.net”, l'identificatore dell'utente sarà “luca.trioschi@ekiga.net”. In pratica il protocollo SIP usa quasi lo stesso metodo di identificazione che viene usato nel mondo delle e-mail. Inutile sottolineare che il nome utente deve essere univoco all'interno del dominio presso il quale è registrato.

Nella figura 2.2 viene mostrata una schematizzazione di come avviene una chiamata tra due dispositivi che utilizzano il VoIP.

Come è stato detto i messaggi SIP vengono utilizzati solo per la gestione della chiamata, e non per l'invio dei dati multimediali. Nello schema sopra proposto si possono notare due possibili strade per arrivare dal nodo “caller” al nodo “callee”: una di queste, quella che passa attraverso i due proxy, è quella usata per i pacchetti SIP e l'altra, quella più breve, viene utilizzata per i pacchetti RTP e RTCP. Sono proprio i pacchetti RTP e RTCP quelli adibiti al trasporto dei messaggi voce e video e vengono trasmessi direttamente da un client all'altro senza passare dai proxy per far loro fare il percorso più breve possibile. I client, con l'utilizzo del protocollo SIP, sono in grado di

registrare un utente presso il server del dominio, fare e ricevere richieste per avviare comunicazioni multimediali, terminare e sospendere comunicazioni attive.

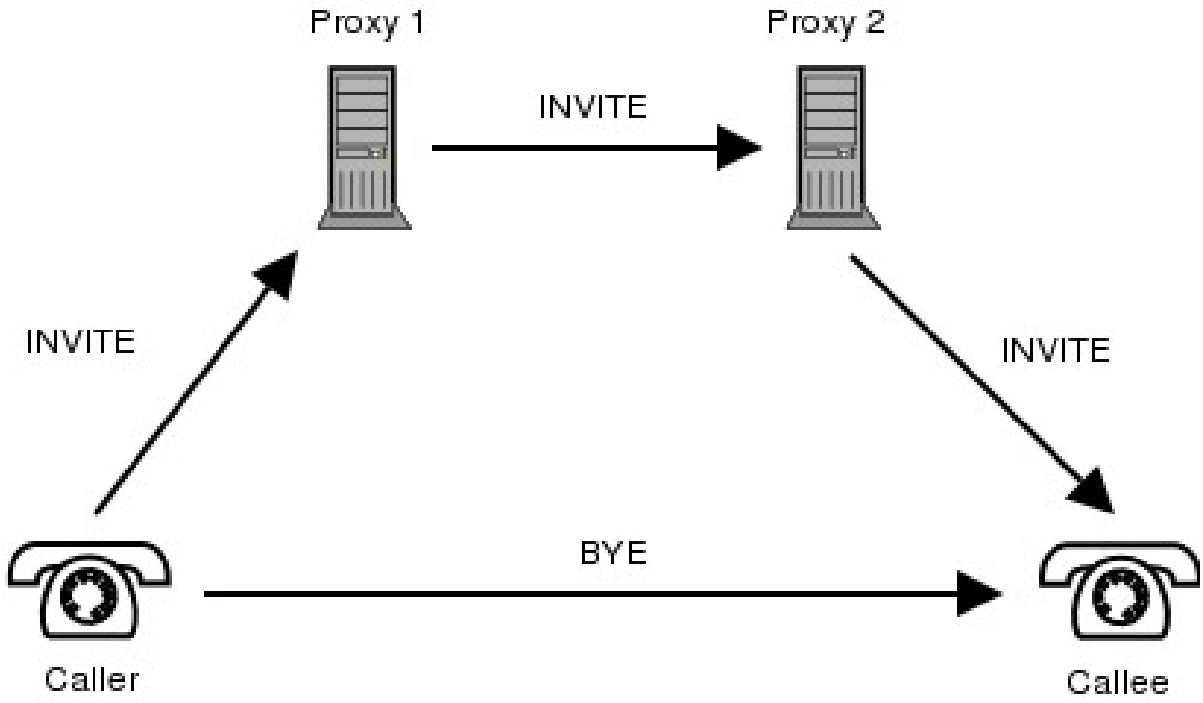

Figura 2.2

Il caso particolare della figura riportata in precedenza mostra un pacchetto di INVITE, che viene usato per inviare una richiesta ad avviare una comunicazione, che arriva a destinazione passando per i proxy e un flusso di dati diretto tra i due client per il successivo scambio di dati.

Per creare una sessione SIP di grandi dimensioni, come un servizio pubblico di telefonia, possono essere necessari molteplici componenti di svariata natura: dai server per l'accounting ai dispositivi per trasformare chiamate VoIP in chiamate compatibili con la rete telefonica, ai server per le conferenze. Nonostante l'eterogeneità di questi dispositivi si può notare che sono tutti composti dalla combinazione di componenti logici più semplici: SIP User Agent, Proxy Server, Redirect Server, Location Server e

Registrar Server.

SIP User Agent è il software in grado di generare messaggi SIP. spesso quando si parla di questo componenti si intende un client SIP software o un telefono VoIP. Questo componente è in grado sia di iniziare una comunicazione sia di riceverla.

Registrar Server è il software al quale vengono inviati i messaggi di registrazione di un utente dai SIP User Agent. Può trovarsi su un server dedicato o essere collocato su un proxy.

Proxy Server è un server intermedio che può rispondere alle richieste o inviarle ad un server, un client o un altro proxy. Un proxy server analizza i parametri di instradamento dei messaggi e nasconde la posizione dei destinatari dei messaggi, essendo essi identificati unicamente con la coppia nome utente e dominio.

Redirect Server è un software che reinstrada le richieste SIP consentendo al chiamante di contattare altri URI

Location Server è un database contente delle informazioni riguardanti gli utenti di un determinato dominio come il profilo o l'indirizzo IP al quale possono essere contattati.

2.3 La realizzazione della comunicazione

Per realizzare la comunicazione ci sono delle operazioni fondamentali che ogni client deve effettuare.

Ogni utente di client, rappresentati nella figura 2.2 con dei telefoni, per prima cosa deve autenticarsi presso il server centrale del servizio in modo tale da segnalare la sua disponibilità a ricevere comunicazioni. In questo modo il server rimane a conoscenza

di quali utenti sono disponibili e a che indirizzo possono essere contattati. Quando l'utente “caller” vuole effettuare la chiamata a “callee”, manda messaggi SIP al proprio server per sapere a quale indirizzo è possibile contattare l'altro utente, in più vengono inviati messaggi anche a “callee” per segnalare l'intenzione di “caller” di avviare una comunicazione. Una volta che “callee” ha accettato l'invito di “caller” ad avviare la comunicazione e che entrambi i client sanno come contattarsi l'un l'altro, lo scambio di messaggi multimediali può avere inizio.

La reale comunicazione avviene attraverso lo scambio diretto di messaggi seguendo i protocolli “Real-time Transport Protocol” (RTP) e “Real Time Control Protocol” (RTCP) da parte dei partecipanti alla comunicazione. Per ogni comunicazione viene aperta una coppia di canali, uno in ogni direzione, per ognuno dei due protocolli che devono essere seguiti.

Per quanto riguarda la comunicazione tra più di due utenti il discorso non cambia, semplicemente aumenta il numero di canali di comunicazione che devono essere tenuti aperti.

2.4 La seamless mobility

Le “Seamless Host Mobility Architecture” sono le architetture ideate per l'integrazione di reti eterogenee e si pongono come obiettivo l'identificare univocamente i “Multi-homed nodes” (MN), dare la possibilità a ogni MN di essere contattato da altri nodi con i quali ha già effettuato una connessione e tenere sotto controllo la qualità del servizio (QoS) dei canali in modo da mantenere la VCC. Queste architetture non trovano una posizione ben definita all'interno del classico stack ISO/OSI in quanto possono essere realizzate ad ogni livello della pila, dal livello Data-link a quello Applicazione.

Nelle comunicazioni basate sul protocollo IP, il compito di identificare univocamente un dispositivo e rappresentare una destinazione raggiungibile per i messaggi diretti a quello specifico dispositivo è proprio l'indirizzo IP. Purtroppo a causa della natura mobile dei dispositivi MN l'indirizzo IP associato agli stessi può non essere lo stesso dall'inizio alla fine della trasmissione dati, come si è detto quando parlato dei vantaggi dell'avere più interfacce di rete su un unico dispositivo. Quando questa situazione si verifica causa una interruzione nella comunicazione in quanto i dispositivi che stanno mandando dati a quello che ha cambiato IP, chiamati “Corrispondent Node” (CN), non possono più mandare correttamente pacchetti a destinazione, non prima di tornare a conoscenza dell'indirizzo IP corretto a cui spedire i messaggi. Questa interruzione è in contrasto con i principi di VCC e il compito delle “Mobile Management Architecture” (MMA) è proprio quello di trovare una soluzione a questa situazione. Nonostante ci siano più MMA, che differiscono tra di loro non solo per la soluzione adottata ma anche per lo strato ISO/OSI in quale sono implementate, tutte si basano sugli stessi due semplici principi:

1.definire un identificatore univoco per il MN indipendente dall'host e dalla configurazione della rete alla quale è agganciato

2.fornire un servizio di localizzazione sempre raggiungibile dai CN, in modo da mantenere un'associazione tra l'identificatore univoco di ogni MN e l'indirizzo reale al quale può essere contattato.

Il servizio di localizzazione è fornito da un Location Registry (LR) attivo su un server (il Location Server) con indirizzo IP pubblico e fisso e quindi raggiungibile da qualsiasi client. Con questo sistema è sufficiente che il CN sia a conoscenza dell'identificatore univoco del MN per poter richiedere al LR l'IP del MN per cominciare o continuare una comunicazione

2.5 L'architettura ABPS

Il sistema ABPS è una soluzione efficace e completa per il QoS e terminal mobility per il VoIP basato su SIP mediante reti wireless. Le due entità che la compongono sono:

SIP mobility entità realizzata a livello applicazione che gestisce le conseguenze di un handover a livello 3 della pila ISO/OSI (livello Rete). Questa entità è realizzata mediante un server dotato di indirizzo IP pubblico e fisso.

Vertical mobility entità realizzata ai livelli Data-link e Network per monitorare lo stato di ogni interfaccia di rete a utilizzabile dal dispositivo e selezionare l'interfaccia migliore con la quale mantenere il collegamento. Alcuni dei parametri utilizzati per valutare le varie interfacce di rete sono stati elencati precedentemente in questo capitolo. Questa applicazione viene eseguita sul client mobile del servizio.

Il server rappresenta l'ancora del terminale mobile: ogni comunicazione VoIP del client passa attraverso di esso e qualunque comunicazione di altri CN diretta al client terminale viene invece inviata al server. Questo perché i CN credono che il client si trovi effettivamente presso il server. In questo modo la mobilità del terminale client viene gestita interamente dal server a cui fa riferimento, incapsulando la complessità dell'operazione e nascondendola al resto della rete.

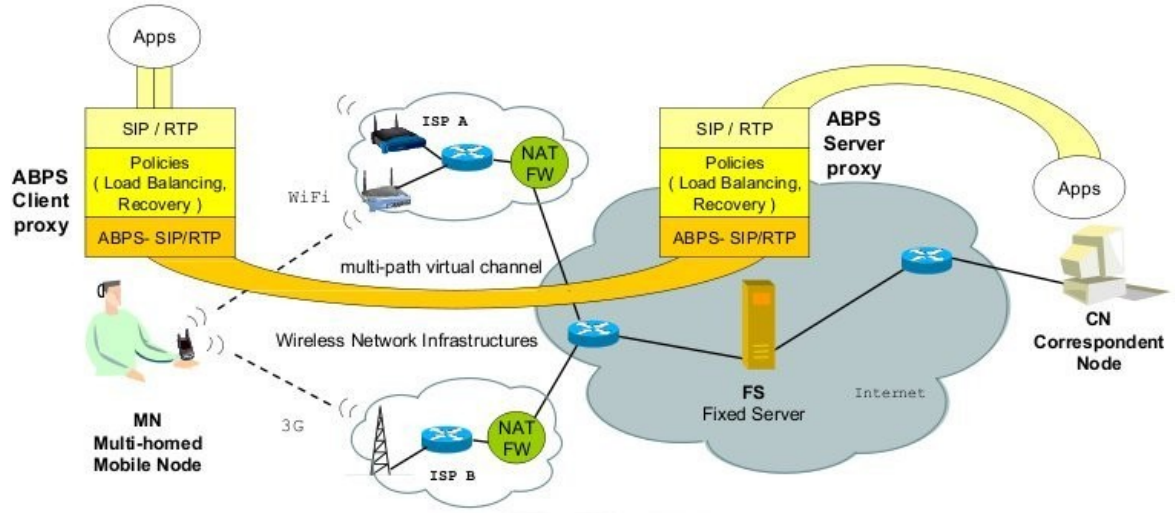

Figura 2.3

Nella figura 2.3 viene mostrato uno scenario di comunicazione tra due client VoIP comprendente anche il sistema ABPS posto tra il Fixed Server (FS) e il MN dotato di più interfacce di rete e più access point disponibili. Il CN manda messaggi al FS convinto di starli spedendo al MN, sarà poi compito del server reindirizzare questi messaggi verso il destinatario finale.

Capitolo 3

Obiettivo

L'obiettivo di questo lavoro è quello di contribuire a creare un canale virtuale tra il nodo mobile e il server di supporto, operante su una sola coppia di porte, nel quale è possibile trasmettere qualunque tipo di dato indipendentemente dal protocollo da utilizzare o dalla presenza o meno di un firewall o NAT, oltre a minimizzare il keep-alive sulle porte mantenute aperte sulla rete.

Grazie a questo canale è possibile anche apportare modifiche ai dati trasmessi senza modificare il comportamento dei programmi che lo utilizzano: ad esempio si può applicare un qualunque algoritmo di crittografia ai dati trasmessi per rendere più sicura la comunicazione, o anche inserire una firma digitale per l'autenticazione del mittente dei messaggi.

Altro grande vantaggio del canale virtuale creato con questo metodo è la sua indipendenza dall'applicazione che accompagna e supporta: il programma utilizzatore continua a funzionare come se non stesse utilizzando il canale in oggetto, sia che si tratti del client sia che si tratti del server. In pratica il canale funziona lasciando ignare della sua esistenza le applicazioni che accompagna: il software che realizza il canale interviene solo sui messaggi che sono stati mandati da/destinati ad applicazioni che sfruttano questa funzionalità facendo credere al programma originale che il canale virtuale non è stato utilizzato. Sia le applicazioni server sia quelle client traggono vantaggio da questa caratteristica: le prime perché possono continuare a rispondere contemporaneamente a chiamate sia da client che fanno uso del canale sia da client che non lo usano, le seconde perché possono abilitare e disabilitare questa funzionalità

anche runtime senza compromettere il funzionamento o la connessione.

Un ulteriore beneficio derivante dall'utilizzo di questo canale virtuale, osservabile utilizzando programmi progettati per funzionare solo su macchine con IP pubblico, è la possibilità del software di non essere più legato all'utilizzo di protocolli per il rilevamento della presenza e delle impostazioni del NAT, come ad esempio il protocollo STUN. Questo particolare protocollo è stato creato per permettere alle applicazioni in esecuzione su un computer di scoprire la presenza ed il tipo di NAT o firewall interposti tra la macchina e la rete pubblica e di conoscere come questi ultimi li stiano rendendo visibili all'esterno. Il problema di questo protocollo è la sua incapacità di funzionare in presenza di un NAT simmetrico (o bidirezionale), problema che la soluzione studiata in questa tesi non presenta.

Capitolo 4

Strumenti

Allo scopo di creare il canale virtuale di comunicazione, scopo del progetto e oggetto di questa tesi, sono stati utilizzati, studiati e modificati i seguenti programmi opensource:

- Pjsip 1.8.10
- Siproxd 0.8.0
- Libosip2 3.5.0

4.1 Pjsip 1.8.10

Pjsip è una piattaforma sopra la quale è possibile la realizzazione di applicazioni, embedded e non, per la comunicazione VoIP. Questa piattaforma fornisce una libreria di funzioni platform-indipendent per la creazione e l'invio di pacchetti SIP RTP e RTCP e per la gestione dei dati multimediali. I principali componenti compresi nel pacchetto pjsip sono:

pjlib	una libreria di funzioni multipiattaforma
pjlib-util	una libreria di funzioni aggiuntive a pjlib
pjnath	una libreria di supporto per NAT
pjmedia	una libreria perla comunicazione multimediale
pjmedia-codec	raccolta di codec multimediali per pjmedia

Compreso nel pacchetto di pjsip si trova anche pjsua: un client testuale di telefonia

mobile scritto sfruttando le librerie fornite da pjsip.

Pjsip è una piattaforma molto utilizzata: tra i programmi di telefonia mobile (e non) costruiti sfruttando la sua piattaforma e le librerie che fornisce troviamo MicroSIP (telefonia mobile per Windows), csipsimple (telefonia mobile per Android), pjsip-jni (interfaccia per SIP per programmi Java), Telephone (telefonia mobile per MAC), Artemisa, TransferHTTP, opensoftphone (telefonia mobile scritto in Java), Host Identity Protocol (HIP), VoiDroid (telefonia mobile per Android), SvSIP (pjsip per NintendoDS e iPod Touch), Sipek Phone (telefonia mobile e servizi di messaggistica istantanea), QjSimple (telefonia mobile), Puppy SIP (telefonia mobile), REMWAVE (telefonia mobile per MAC), Siphon (telefonia mobile per iPhone e iTouch) e molti altri ancora.

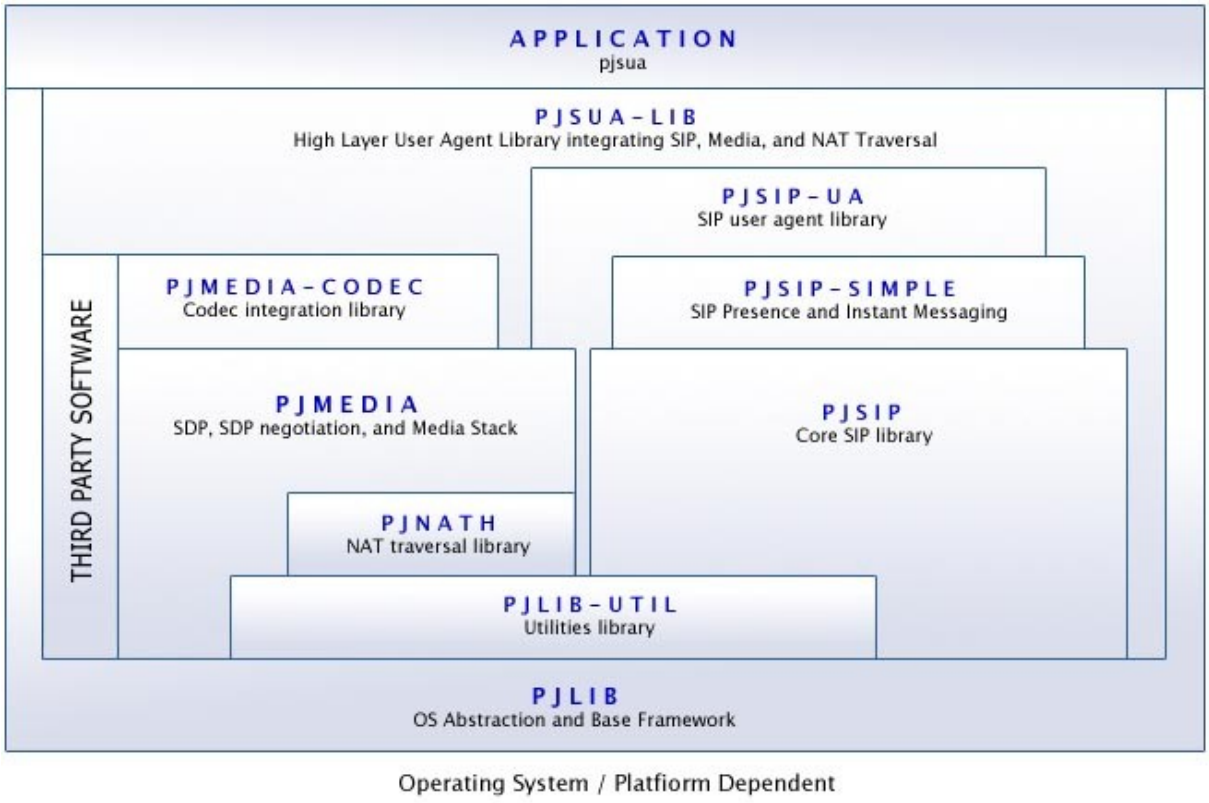

Figura 4.1

Nella figura 4.1 viene mostrata la struttura del progetto pjsip con tutti i suoi componenti.
Il programma è interamente scritto in C, con qualche inserto in C++. Tutte le informazioni riguardanti pjsip si possono trovare sul sito internet www.pjsip.org.

4.2 Siproxd 0.8.0

Siproxd è un proxy server che gestisce la registrazione di client SIP, riscrivendo il corpo dei pacchetti SIP per creare connessioni con un server centrale. Il software supporta client SIP e telefoni SIP-compatibili.
All'interno dell'architettura ABPS, questo specifico programma ricopre il ruolo di "SIP Mobility" installato sul Fixed Server. Quindi, come già detto, a questo programma verranno inviati tutti i messaggi inviati da e diretti a tutti i client che fanno riferimento ed esso.
Al contrario del programma precedentemente descritto, siproxd è disponibile solo per sistemi operativi Unix e derivati.
Anche questo programma è interamente scritto in C. Tutte le informazioni riguardanti siproxd sono reperibili all'indirizzo siproxd.sourceforge.net.

4.3 Libosip2 3.5.0

Libosip2 è una libreria di funzioni per la gestione dei pacchetti SIP disponibile per i sistemi operativi della famiglia Unix. Questa libreria ha l'obiettivo di fornire un set di istruzioni comuni a a tutti i "SIP Agents".
È stato deciso di inserire questo pacchetto tra il software studiato e modificato in quanto è stato effettivamente necessario apportare delle modifiche ad alcune delle

funzioni che mette a disposizione per risolvere un problema che si è presentato studiando il comportamento di siproxd.
Se non fosse stato per il malfunzionamento del programma usato come Proxy Server non sarebbe stato necessario modificare le funzioni implementate in questa libreria e quindi nemmeno citarla in questo capitolo: sarebbe stata sufficiente averla installata sulla macchina adibita alla funzione di Proxy Server.
Tutte le informazioni riguardanti libosip2 sono reperibili sul sito internet www.gnu.org/s/osip/ e www.gnu.org/software/osip/doc/html/.

4.4 Altri software

Anche lo sviluppo del progetto di tesi è stato eseguito interamente sfruttando software libero.
Lo sviluppo di pjsip è stato portato avanti su sistemi operativi della famiglia Linux e più precisamente: una Debian 6 (Squeeze) installata su una macchina a 64bit, una Debian 5 (Lenny) installata su una macchina a 32bit e una Ubuntu 11 installata su una macchina a 32bit. Come editor di testo sono stati utilizzati il famoso Eclipse e, all'occorrenza, il classico VIM. Per compilare i sorgenti prodotti si è ricorsi a “Gnu C Compiler” (GCC). Una volta portata a termine la parte di sviluppo, i test sul funzionamento sono stati eseguiti anche su un sistema operativo proprietario: Microsoft Windows 7 installato su un computer a 64bit, con l'ausilio di una versione enterprise di Visual Studio 2010.
Lo sviluppo e i test di siproxd invece sono stati fatti interamente su una macchina a 32bit con una versione di OpenSuse installata. Come editor è stato utilizzato unicamente VIM e come compilatore ancora GCC. Ovviamente, come da specifiche per il funzionamento del SIP Mobility, il computer sul quale è installato siproxd è

connesso ad internet con indirizzo IP pubblico e fisso ed è contattabile direttamente da ogni client.

Capitolo 5

Progettazione

5.1 Introduzione alla soluzione

Prima di parlare del metodo che è stato usato per creare il canale virtuale di comunicazione, verrà spiegato meglio come funziona la comunicazione tra dispositivi SIP secondo l'architettura ABPS.

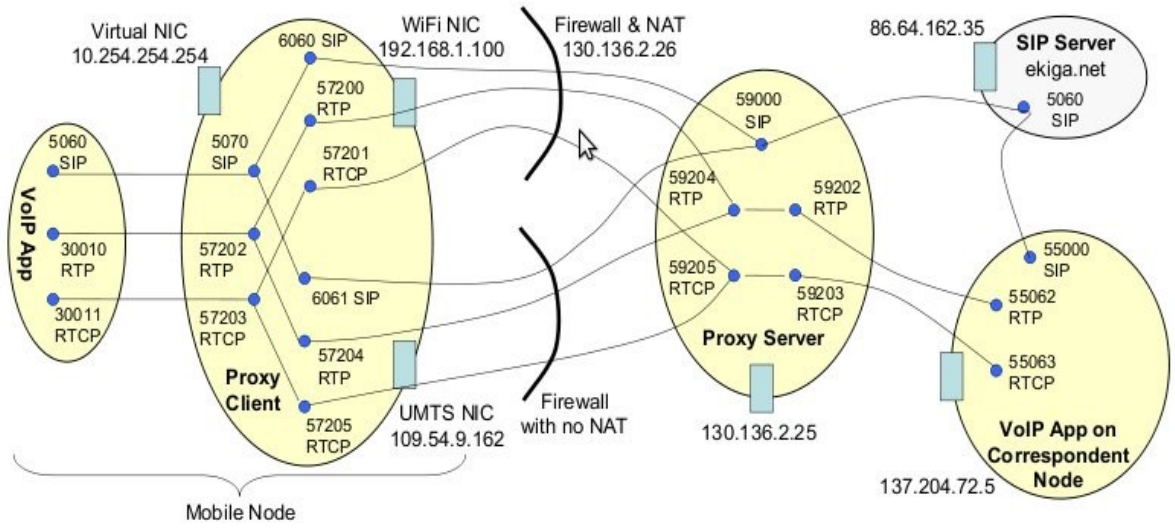

Figura 5.1

La figura 5.1 mostra la schematizzazione di una comunicazione tra due nodi client, utilizzati da due utenti del dominio “Ekiga”.

Il primo nodo, un nodo mobile che nella figura è posto sulla sinistra, è un MN dotato di un'interfaccia WiFi e una UMTS. Come si può osservare dall'immagine, dal lato WiFi il dispositivo è connesso con un IP privato ad una rete locale dotata di firewall e

NAT, mentre dal lato UMTS è connesso con un indirizzo IP pubblico presumibilmente non fisso direttamente ad internet. Anche quest'ultima connessione di rete è protetta da un firewall, ma essendo la connessione diretta non troviamo la presenza di NAT. Questo primo nodo sfrutta l'architettura ABPS, come si può notare dal fatto che tutti i pacchetti in uscita ed in entrata passano dal Proxy Server.

Il secondo nodo, individuabile nella figura nell'angolo in basso a destra, è un CN ed è dotato di una unica interfaccia di rete con la quale si connette direttamente a internet con un IP pubblico. Questo nodo non sfrutta i servizi del Proxy server, infatti i suoi messaggi SIP arrivano direttamente al server centrale del servizio: il SIP Server di ekiga.net. Ai fini della connessione è irrilevante se il CN è un nodo mobile o fisso, o il tipo di connessione di cui dispone.

Come prima cosa ogni utente deve registrarsi presso il server del dominio, mediante una serie di messaggi SIP, che utilizzano una porta standard. Mentre il CN manda direttamente i dati al server centrale, il MN li manda al proxy server che ne modifica il contenuto per poi spedirli al server di Ekiga. L'operazione di modifica del contenuto dei pacchetti SIP effettuata dal Proxy Server serve ad indurre il server di Ekiga a credere che l'utente che sta effettuando l'operazione di registrazione sia raggiungibile all'indirizzo del Proxy Server in quanto effettivamente locato presso di esso. Si noti che prima dell'inizio di una comunicazione l'unico canale aperto è quello SIP; gli altri due vengono aperti successivamente.

Nel momento in cui l'utente collegato sul MN vuole iniziare una comunicazione con quello connesso sul CN, invia il messaggio SIP di INVITE al Proxy Server a cui fa riferimento, quest'ultimo ne modifica nuovamente il corpo e lo rimanda a Ekiga che avverte il CN della possibilità di instaurare una connessione. Nel messaggio arrivato al CN sono presenti, oltre al nome dell'utente che vuole cominciare la comunicazione, anche l'IP al quale esso è raggiungibile, che poi è l'IP del Proxy Server. Se la richiesta di connessione viene accettata dal CN, avviene un'altra serie di scambi di messaggi

SIP tra il Proxy server e il CN per accordarsi su quali porte mandare i messaggi multimediali. Questo scambio di messaggi volti a scegliere le porte da utilizzare per la comunicazione avviene anche tra il Proxy Server e il MN. Una volta che le porte sono state definite, vengono creati altri canali di comunicazione RTP e RTCP sui quali comincia lo scambio di messaggi contenenti dati audio o video. Come si può osservare nella figura 5.1, sia il MN che il CN mandano questi messaggi al Proxy Server, MN perché è il suo punto di riferimento, CN perché convinto che l'utente con il quale sta comunicando sia raggiungibile direttamente a quell'indirizzo. È compito del Proxy Server indirizzare i messaggi verso il destinatario finale. Al termine della comunicazione i canali RTP e RTCP aperti per essa vengono chiusi e rimangono attivi solo i canali SIP.

Vediamo adesso come cambia la situazione dopo la creazione del canale virtuale oggetto di questa tesi.

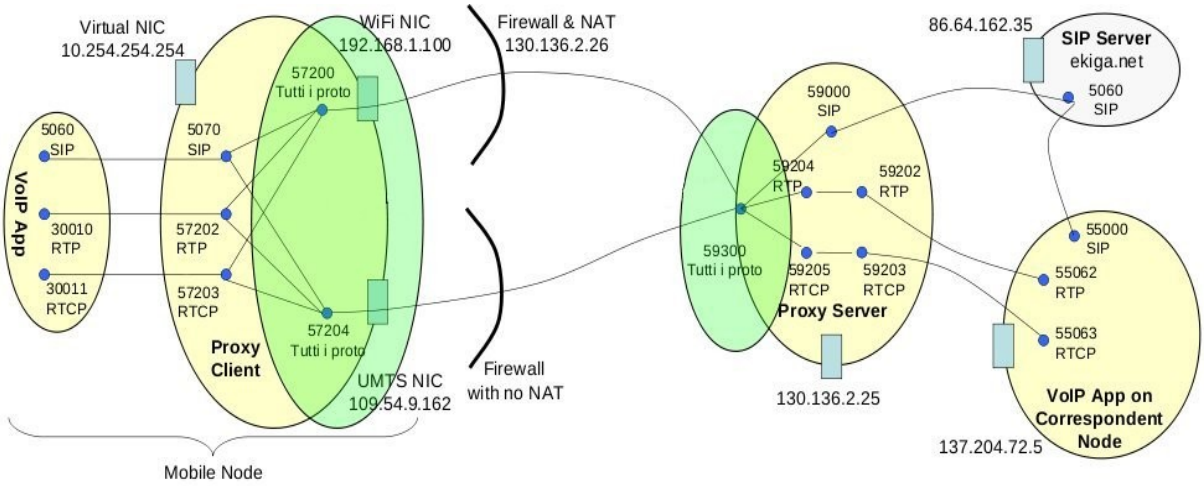

Figura 5.2

La figura 5.2 mostra la stessa situazione della figura 5.1, ma con l'aggiunta dei due oggetti verdi che rappresentano le modifiche atte a creare il canale virtuale. Si noti che,

a differenza di prima dove dal MN a Proxy Server per ogni protocollo era presente un canale di comunicazione, ora c'è un canale unico per ogni dispositivo di rete presente sul MN nel quale vengono mandati tutti i messaggi di tutti i protocolli da supportare. Il modo in cui avviene la comunicazione è lo stesso descritto in precedenza, ma con i pacchetti scambiati tra il Proxy Server e il MN fatti passare da un unico canale: il canale virtuale multipercorso.
Passiamo ora ad analizzare le modifiche e le aggiunte apportate per realizzare questo canale.

5.2 Il metodo utilizzato

Per giungere all'obiettivo è stato scelto di intraprendere una strada meno invasiva possibile per i programmi preesistenti. Come prima cosa sono stati creati altri due programmi, uno per il pjsip e uno per il siproxd, che hanno il compito di realizzare il canale di comunicazione. Questi programmi girano sulla stessa macchina dei software che accompagnano ma parallelamente ad essi e la loro presenza è nascosta: i programmi originali non si accorgono della loro esistenza né del lavoro che svolgono o del canale che creano.
In pratica, tramite procedure inserite nel codice sorgente dei programmi originali, per ogni messaggio che deve essere mandato da uno dei due programmi originali viene creato un nuovo pacchetto contenente dati utili all'identificazione del client e alla riuscita della comunicazione, oltre ovviamente al contenuto del messaggio originale. Questo nuovo pacchetto viene spedito al programma parallelo che si trova sulla stessa macchina il quale lo riceve e lo manda al suo corrispondente utilizzando il canale virtuale. L'altro programma da noi creato legge il messaggio mandatogli e lo consegna al destinatario originale alla porta sulla quale il programma originale si aspetta di

riceverlo. L'ultimo passo è ingannare il programma destinatario del messaggio, tramite un'altra procedura inserita nel codice sorgente del programma, per fargli credere che ha ricevuto il pacchetto direttamente dal programma mittente. Per poter realizzare questa soluzione è necessario che anche il programma parallelo al siproxd tenga traccia di tutti i client connessi. Per fare questo è necessario poter identificare univocamente ogni singolo client, per questo ad ogni programma parallelo a pjsip è stato assegnato un numero identificativo grazie al quale il programma parallelo a siproxd può distinguere i vari mittenti. Purtroppo questa soluzione è adottabile solo per i pacchetti che provengono da pjsip, infatti solo il programma parallelo al client è a conoscenza dell'identificatore. Per i messaggi provenienti da siproxd è stato utilizzato un altro metodo per il riconoscimento: ad ogni canale di comunicazione di ogni client è associato un numero identificativo univoco N, chiamato fake_sender_port, numero che viene fatto vedere a siproxd come la porta dalla quale il messaggio è stato spedito. In questo modo quando siproxd tenta di mandare un pacchetto di risposta sulla tessa porta, il programma parallelo è in grado di capire sia il destinatario sia il tipo di pacchetto. Si veda, per maggiori dettagli sulla necessità di tale identificativo univoco, la sezione 5.4 concernente i multiplexer.

D'ora in poi ci si riferirà a questi programmi paralleli con i nomi “Client Multiplexer” (CM) per il programma che accompagna il pjsip e “Server Multiplexer” (SM) per quello che accompagna il siproxd. Per le procedure inserite nel codice sorgente dei programmi originali verranno usati i nomi “Send Trickery” (ST) per le procedure che deviano verso CM o SM i pacchetti in uscita e “Receive Trickery” (RT) per quelle che modificano il mittente dei messaggi in ingresso, in modo da far credere al programma originale di aver ricevuto dati come se CM e SM non esistessero.

Questo metodo è stato ideato per giungere al risultato voluto facendo in modo che i programmi originali non modificassero il loro comportamento, anzi che non “si rendessero conto” dei cambiamenti apportati: i client continuano a registrarsi al server

usando l'indirizzo IP che vedono come loro (sia che si tratti di un indirizzo pubblico sia che si tratti di uno privato) e il server manda messaggi per il client a quell'indirizzo. Sono CM e SM che risolvono la situazione e mandano i pacchetti agli indirizzi giusti.
I principali vantaggi di questa soluzione sono due. Il primo è la possibilità di applicare facilmente questa soluzione a programmi anche molto diversi da quelli studiati per questa tesi. Il secondo è che l'impatto minimo che le modifiche hanno sul programma originale minimizza la probabilità che i cambiamenti eseguiti interferiscano con il funzionamento di software costruiti sfruttando questo programma. Questo secondo vantaggio è osservabile soprattutto per il pjsip.
Passiamo ora a descrivere nel dettaglio i componenti, sopra introdotti, che permettono il funzionamento della soluzione ideata.

5.3 Le procedure

Le procedure ST e RT sono l'unica modifica apportata ai sorgenti dei programmi originali, escludendo il pezzo di codice che crea i processi paralleli CM e SM.

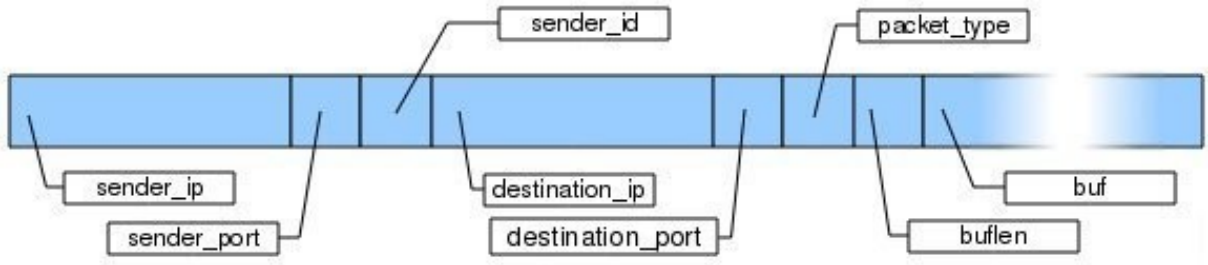

Figura 5.3

Le procedure ST hanno il compito di deviare il flusso di dati verso il programma parallelo e aggiungere al messaggio i dati necessari al corretto svolgimento delle

operazioni sopra descritte. La figura 5.3 mostra il nuovo pacchetto creato da queste procedure e che verrà poi inviato al posto del messaggio originale. I campi che compongono questo nuovo pacchetto sono:

- sender_ip – IP che il mittente del pacchetto vede come proprio
- sender_port – la porta dalla quale il mittente ha spedito il pacchetto
- sender_id – identificatore univoco del mittente (solo per il programma client)
- destination_ip – IP al quale il pacchetto doveva essere mandato
- destination_port – porta sulla quale il pacchetto doveva essere mandato
- packet_type – tipo di pacchetto
- buflen – dimensione del messaggio originale
- buf – il messaggio originale

I tipi di pacchetti gestiti sono 5: SIP, RTP voce, RTP video, RTCP voce e RCP video. I dati aggiuntivi che vengono mandati insieme al messaggio originale possono essere visti come un header per la comunicazione attraverso il canale virtuale inserito in testa al pacchetto. Si noti che attraverso il canale virtuale oggetto di questa tesi passano solamente pacchetti di questo tipo. Nel pacchetto era inizialmente previsto anche un campo che contenesse una sorta di firma, in modo da poter identificare il mittente del messaggio. Nella versione finale del progetto questo campo non è presente perché l'implementazione di questa funzionalità esula dagli obiettivi, anche se il progetto è stato pensato per dare la possibilità di inserire e inviare dati come questi.

All'interno del programma originale è possibile trovare le procedure ST subito prima della funziona che si occupa della spedizione del messaggio. All'interno di questa funzione, oltre al pacchetto, vengono modificati anche la destinazione del messaggio e la sua dimensione: la prima modifica serve per far arrivare il messaggio al programma parallelo, la seconda per far mandare tutto il messaggio, comprensivo dei dati aggiuntivi inseriti. Una volta modificati questi parametri viene lasciato il compito di

mandare il messaggio alla procedura originale che se ne è sempre occupata e subito dopo viene richiamata la seconda parte della ST che si occupa di rimettere tutto come prima dell'intervento: in questo modo pacchetto e destinazione risulteranno quelle originali in caso il programma, o un altro software che ne sfrutta le funzioni nel caso di pjsip, debba fare delle altre operazioni su di esso.
Il compito delle RT è invece quello di indurre i programmi originali a credere che il pacchetto sia arrivato direttamente dall'altro programma originale, senza l'intervento di CM e SM. È possibile trovare le RT immediatamente dopo alle procedure che leggono i dati ricevuti nei programmi originali. Il loro compito è quello di modificare il pacchetto togliendo i dati aggiunti dalla ST, la struttura dati contenente i dati del mittente e sostituendoli con quelli che trova nel pacchetto che ha ricevuto (i campi sender_ip e sender_port per la precisione) e la dimensione dei dati letti impostandola al valore presente nel campo buflen del pacchetto ricevuto.
Le procedure ST e RT che si trovano sul Proxy Server però devono tenere conto di una variabile in più rispetto a quelle che lavorano sui client: siccome il Proxy Server scambia messaggi non solo con i client registrati presso di lui, ma anche con altri client, con il server centrale del servizio e anche con altri Proxy Server, è necessario che queste funzioni intervengano solo su determinati pacchetti. Questa funzionalità viene gestita in collaborazione con il SM e verrà analizzata successivamente.
La caratteristica delle procedure ST e RT è quella di trovarsi al più basso livello possibile in modo da intercettare tutti i pacchetti in entrata e in uscita, comparendo il meno possibile all'interno del codice sorgente.

5.4 I multiplexer

I programmi CM e SM, che rimangono invisibili ai programmi originali pur essendo processi figli dei due originali e girando in parallelo sulla stessa macchina, sono i responsabili della creazione del canale virtuale e del corretto arrivo a destinazione dei pacchetti che vi viaggiano attraverso.

Ciascuno di questi due processi tiene aperte solo due porte per la ricezione dei dati: una per la comunicazione in locale con il programma che supporta e l'altra per le comunicazioni attraverso la rete con la sua controparte.

Il funzionamento del CM è molto semplice. Quando arriva qualcosa dal pjsip, quindi un pacchetto deviato dalla ST, legge il messaggio, finisce di riempire la struttura in cui è incapsulato il pacchetto originale con i pochi dati in suo possesso (come la porta mittente e l'ID del client), e lo inoltra al SM. CM conosce l'indirizzo e la porta ai quali contattare SM in quanto la porta è una costante nota e l'IP è quello originale di destinazione del pacchetto (il campo destination_ip). Quando invece arriva qualcosa dal SM, quindi dal canale virtuale, legge il dato e lo rimanda al pjsip. Anche in questo caso conosce come contattare il programma originale: l'IP è quello locale, mentre la porta è specificata nel pacchetto ricevuto nel campo destination_port.

Il funzionamento di SM è invece più complesso. Oltre a svolgere le attività del CM, deve anche mantenere traccia di tutti i client connessi attraverso il canale virtuale. Per poterlo fare il SM mantiene in memoria una lista di strutture contenenti i dati relativi ai client connessi.

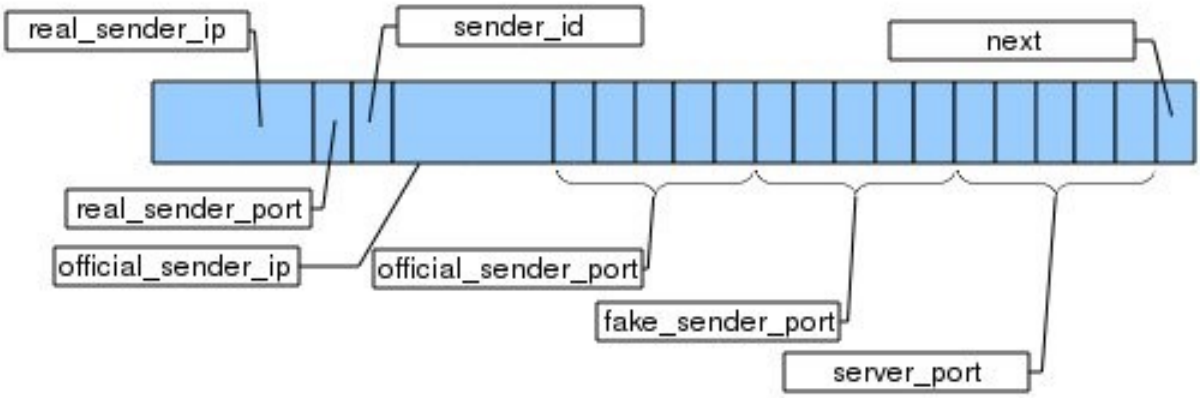

Figura 5.4

La figura 5.4 mostra la struttura dati utilizzata dal SM per memorizzare i dati dei client. I campi che compongono questa struttura sono:

- real_sender_ip – IP dal quale arrivano i messaggi
- real_sender_port – porta dalla quale arrivano i messaggi
- sender_id – identificatore univoco del client che ha mandato il pacchetto
- official_sender_ip – IP che il mittente vede come suo (privato o pubblico che sia, specificato nel pacchetto in arrivo come sender_ip)
- official_sender_port – porta dalla quale il mittente ha mandato il pacchetto (una per ogni tipo di pacchetto, specificato nel pacchetto in arrivo come sender_port)
- fake_sender_port – porta che il SM dovrà far vedere al siproxd come porta mittente (una per ogni tipo di pacchetto)
- server_port – porta sulla quale il server deve ricevere il pacchetto (una per ogni tipo di pacchetto, specificata nel pacchetto in arrivo come destination_port)
- next – puntatore al successivo elemento della lista

Ogni volta che SM riceve un pacchetto da un client, quindi attraverso il canale virtuale, esegue una determinata serie di azioni.

Per prima cosa aggiorna la lista dei client connessi: sulla base dell'identificatore

presente nel pacchetto ricevuto ricerca il corretto elemento della lista. Nel caso sia il primo messaggio ricevuto da un client, la lista non conterrà alcun elemento associato al client in questione; in questa eventualità viene creato e inserito in testa alla lista un nuovo elemento contenente tutte le informazioni disponibili, come l'IP al quale può essere contattato e l'indirizzo con il quale si sta registrando, oltre ovviamente al suo identificatore. Una volta trovato o inserito l'elemento corretto ne aggiorna i campi come la porta usata dal mittente e quella sulla quale il siproxd deve ricevere il messaggio per quello specifico tipo di pacchetto. Nel caso sia il primo pacchetto di un certo protocollo per quel client, SM gli assegna una numero univoco: la fake_sender_port. Questo numero univoco verrà mostrato al siproxd come la porta dalla quale il messaggio è stato inviato in modo che i successivi messaggi di quel tipo destinati a quel client vengano mandati su quella porta.

Successivamente SM modifica il messaggio sostituendo la porta usata dal mittente con quella fittizia assegnata a quel tipo di pacchetto.

Infine manda il messaggio modificato verso siproxd alla porta sulla quale si aspetta di riceverlo.

Le operazioni eseguite da SM quando riceve un messaggio proveniente da siproxd sono simili a quelle precedentemente descritte. Nuovamente come prima cosa cerca nella lista dei client connessi quello al quale è destinato il messaggio, però in questo caso usa come chiave di ricerca la porta sulla quale il pacchetto è diretto confrontandola con le fake_sender_port. Dopo aver identificato il client corretto modifica il contenuto del pacchetto sostituendo la porta di destinazione (fake) con la porta sulla quale il pjsip sta realmente in ascolto per ricevere i dati di quel tipo. Dopo che le modifiche al pacchetto sono state apportate inoltra il pacchetto al CM: anche stavolta la porta è una costante nota, mentre l'indirizzo IP è tra i dati salvati nella lista nel campo real_sender_ip.

Come già anticipato quando si è parlato delle procedure RT e ST, SM è in grado di

rispondere ad un altro tipo di richiesta. Mediante un canale di comunicazione locale è in grado di ricevere e mandare dati all'applicazione originale, più precisamente alla procedura ST. Quando questa procedura sta per intervenire su un pacchetto manda a SM, utilizzando questo canale locale, la coppia IP e porta del destinatario del pacchetto. Il multiplexer controlla nella lista che sta mantenendo in memoria se la coppia ricevuta appartiene o meno ad uno dei client di cui sta tenendo traccia e risponde, attraverso lo stesso canale, con un messaggio positivo o negativo. Grazie a questo messaggio la procedura inserita nel codice originale del siproxd è in grado di distinguere quali sono i pacchetti provenienti da o diretti a un client che fa uso del canale virtuale. Questa distinzione è necessaria per non andare ad interferire con i messaggi che non necessitano di modifiche perché diretti a indirizzi che possono appartenere a: client non facenti uso del canale virtuale, il server centrale del servizio od anche altri Proxy Server.

5.5 I dettagli della comunicazione

Si va ora ad analizzare nel dettaglio il percorso che viene intrapreso ogni volta che un messaggio deve essere fatto passare dal canale virtuale.

La figura 5.5 mostra lo schema di percorso dei pacchetti fatti passare attraverso il canale virtuale. Per semplicità il nodo mobile è stato rappresentato con un'unica interfaccia di rete wireless in quanto il comportamento del software è lo stesso per ogni NIC. Non sono stati riportati nemmeno eventuali firewall o NAT che possono frapporsi tra il CM e il SM in quanto il sistema è studiato per funzionare sia in presenza che in assenza di tali software.

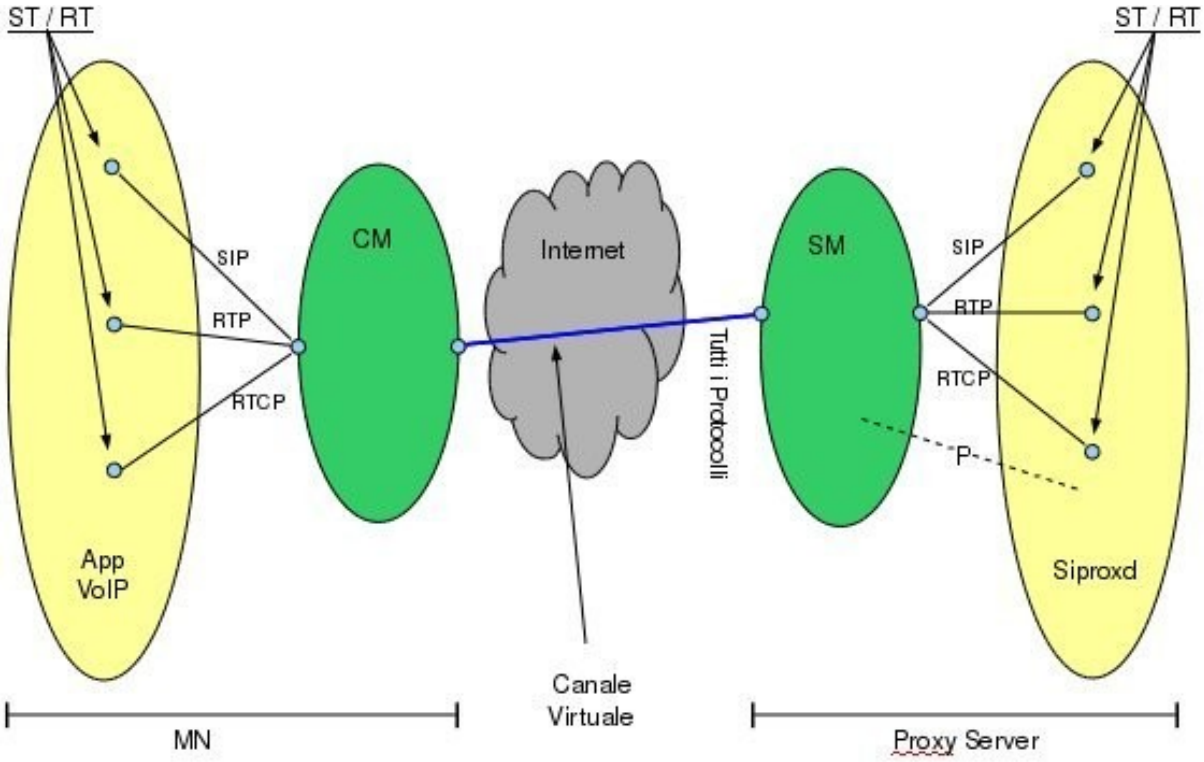

Figura 5.5

Dall'applicazione VoIP al Proxy Server

Per quanto riguarda i pacchetti generati dall'applicazione VoIP la prima modifica si verifica con l'esecuzione della procedura ST. Essa crea il nuovo pacchetto, il cui schema è mostrato nella figura 5.3, e lo riempie con i dati in suo possesso: l'IP della macchina sulla rete alla quale è connessa (lo prende dal file di configurazione dato come parametro all'avvio dell'applicazione), l'IP e la porta di destinazione presi dalla struttura contenente tali dati (pj_sockaddr), il tipo di pacchetto e la lunghezza del buffer (entrambi passati come parametri alla funzione). Dopo aver riempito questi campi, copia l'intero messaggio originale, che trova all'interno del buffer di spedizione passatogli anch'esso come parametro, all'interno del nuovo pacchetto, precisamente nel campo buf. Ora che il nuovo pacchetto è stato riempito con tutti i dati disponibili, il suo intero contenuto viene copiato all'interno del buffer dei messaggi in uscita, al posto

del messaggio originale, e la dimensione del pacchetto viene aggiornata (più precisamente viene incrementata di 50 Byte cioè la dimensione dei dati aggiunti al pacchetto originale o, volendo, il payload dell'header per la comunicazione attraverso il canale virtuale). Come ultima azione, questa procedura modifica il contenuto della struttura contenente il destinatario del messaggio, sostituendo IP e porta del Proxy server con quelli del CM. Al termine della ST riprende l'esecuzione delle normali istruzioni del programma di VoIP e la prima che viene eseguita è quella utilizzata per l'invio del messaggio. Questa istruzione, adibita alla spedizione del normale messaggio al Proxy server, manda invece il pacchetto preparato dalla ST al CM. Immediatamente dopo l'esecuzione di questa istruzione viene eseguita un'altra procedura che può essere vista come la seconda parte della ST: il suo compito è quello di ripristinare l'originale contenuto del buffer e della struttura dati contenente il destinatario del messaggio appena spedito.

Il messaggio spedito dall'applicazione VoIP arriva a destinazione al CM che, dopo aver ricevuto il messaggio, completa i due campi del nuovo pacchetto non riempiti dalla ST: inserisce quindi l'ID del client (che è una costante nota) e la porta dalla quale il messaggio è stato spedito (ricava questa informazione dalla struttura dati contenente i dettagli sul mittente del messaggio appena ricevuto). Ora che il pacchetto è completamente riempito con i dati necessari viene inviato al SM attraverso il canale virtuale che nella figura 5.5 è rappresentato dalla linea blu che passa attraverso internet.

Una volta che il pacchetto è stato ricevuto dal SM esso esegue la serie di operazioni precedentemente elencata che ora verrà mostrata più dettagliatamente. Inizialmente cerca nella lista dei client connessi l'elemento associato al mittente del messaggio appena ricevuto, usando come chiave di ricerca l'ID presente nel pacchetto. Nel caso la ricerca non andasse a buon fine viene creato un altro elemento per la lista, la cui struttura è mostrata nella figura 5.4. In questo nuovo elemento vengono inseriti gli

elementi costanti cioè l'identificatore del client e l'IP della macchina (entrambi specificati all'interno del pacchetto appena ricevuto). A questo punto il nuovo elemento viene inserito nella lista dei client connessi e si procede con le altre operazioni che vengono eseguite dopo aver trovato l'elemento corretto della lista. Queste operazioni riguardano l'aggiornamento dei campi variabili: l'IP e la porta ai quali il client può essere contattato (presi dalla struttura contenente i dettagli sul mittente del messaggio), la porta usata dal client per quel tipo di messaggi e quella sulla quale il pacchetto va inoltrato al Proxy Server (prese entrambe dal pacchetto ricevuto). Nel caso questo tipo di messaggio non abbia ancora una porta fittizia associata, gliene viene assegnata una: nel caso sia un messaggio SIP gli viene riservata la prima porta fittizia libera, altrimenti vengono assegnate due porte cioè la prima pari libera per i pacchetti RTP e quella immediatamente successiva per i pacchetti RTCP (in modo da rispettare una caratteristica del siproxd analizzata in seguito). Ora che la lista dei client connessi è stata correttamente aggiornata, è necessario apportare una modifica al pacchetto ricevuto: aggiornare il campo nel quale è memorizzata la porta dalla quale è stato inviato il messaggio dall'applicazione VoIP sostituendolo con la porta fittizia assegnata a quel tipo di pacchetto. A questo punto il messaggio è pronto per essere mandato al Proxy Server alla porta sulla quale si aspetta di riceverlo (indicata nel pacchetto).

La procedura del Proxy Server che riceve il pacchetto mandato da SM è la stessa che riceve tutti i pacchetti diretti al Proxy Server, relativamente allo stream corrente. Subito dopo l'istruzione che riceve i dati in ingresso si trova la procedura RT. Come prima cosa questa procedura controlla chi ha inviato il messaggio prendendo il dato dalla struttura dati contenente i dettagli del mittente; nel caso messaggio provenga da localhost (quindi da SM) la procedura esegue il suo compito, in caso contrario termina senza modificare niente siccome il pacchetto non è tra quelli che vengono gestiti dal sistema oggetto di tesi. Le operazioni che devono essere eseguite sui pacchetti provenienti dal SM sono volte a nascondere tutte le operazioni eseguite da ST, CM e

SM. Per prima cosa la procedura RT modifica la struttura contenente i dati del mittente del pacchetto, inserendo i dati del mittente che trova all'interno del pacchetto ricevuto. Si noti che l'IP è quello specificato dall'applicazione VoIP e quindi potrebbe essere un indirizzo privato e non raggiungibile da internet. Si noti anche che la porta è quella fittizia inserita da SM e non quella realmente usata dall'applicazione client. La successiva operazione eseguita da questa procedura è quella di copiare nel buffer di lettura il messaggio originale e di modificare la quantità di dati letti impostandola al valore corretto; sia il messaggio originale che la sua dimensione si possono trovare nel pacchetto ricevuto, rispettivamente nei campi buf e buflen. Dopo che questa procedura ha terminato il suo lavoro, la normale esecuzione di siproxd può continuare in quanto nel buffer di lettura c'è solo il messaggio originale e nella struttura contenente i dati del mittente ci l'IP specificato dall'applicazione VoIP e la porta fittizia assegnata dal SM necessaria all'identificazione del destinatario per i messaggi in uscita.

Dal Proxy Server all'applicazione VoIP

Si procede ora all'analisi nel dettaglio di ciò che succede quando un messaggio deve andare dal Proxy Server all'applicazione VoIP. Anche in questo caso la prima differenza rispetto al normale svolgimento della comunicazione è l'intervento della procedura ST. Al contrario della stessa procedura presente sul client questa ha un compito ulteriore: controllare che il destinatario del messaggio sia un client che sfrutta il canale virtuale. Questa operazione è necessaria perché i pacchetti inviati dal siproxd, al contrario di quelli inviati dal client, possono essere diretti anche a client non facenti uso del canale oggetto di tesi, al server centrale del servizio o ad altri Proxy Server. È necessario che la ST del server intervenga solamente su messaggi diretti a client che comunicano usando il canale oggetto di tesi. Per sapere se il pacchetto deve essere deviato, la procedura ST invia a SM un messaggio contenente IP e porta sui quali il messaggio sta per essere inviato (informazione presa dalla struttura dati contenente i

dettagli del destinatario). Si noti che l'IP e la porta che vengono passati al SM sono quelli visti da siproxd, quindi è possibile che l'IP sia quello con cui il client ha fatto la registrazione (pubblico o privato che sia) e che la porta sia quella fittizia inserita dal SM. Il messaggio contenente IP e porta mandato da ST a SM viene spedito attraverso un canale privato (un socketpair) che nella figura 5.5 è rappresentato dalla linea tratteggiata e dalla lettera P. Il SM legge il messaggio e controlla se la coppia IP e porta passatagli da ST appartiene ad uno dei client presenti nella lista di quelli connessi. SM manda a ST, attraverso lo stesso canale da cui è arrivata la coppia IP/porta, un messaggio positivo o negativo a seconda dell'esito della ricerca. ST riceve questa risposta e in caso sia negativa termina la propria esecuzione senza modificare né deviare il messaggio che verrà inoltrato normalmente al destinatario originale. Nel caso invece di una risposta positiva, la procedura incapsula il messaggio originale all'interno di un pacchetto identico a quello utilizzato dalla ST dell'applicazione VoIP e rappresentato in figura 5.3. Anche questa ST riempie i campi della struttura del nuovo pacchetto con i dati noti: l'IP del mittente che è una costante nota, l'IP e la porta del destinatario che possono essere ricavati dalla struttura contenente i dati (sockaddr), la dimensione del messaggio originale e il messaggio originale (questi ultimi due passati come parametri alla procedura). Anche questa ST riscrive la struttura contenente i dati del destinatario mettendoci IP e porta del SM, copia l'intero nuovo pacchetto nel buffer di invio dei messaggi e incrementa la dimensione dei dati da inviare (anche stavolta di 50 Byte). Nuovamente, dopo il termine della ST, si riprende la normale esecuzione del codice di siproxd e nuovamente viene eseguita la funzione che invia il messaggio presente nel buffer all'indirizzo specificato nell'apposita struttura. Nel caso la ST sia intervenuta il pacchetto viene quindi mandato a SM, in caso contrario il messaggio viene inviato al destinatario designato. Quest'ultimo caso non è stato riportato nella figura 5.5: il messaggio viene inviato attraverso internet senza sfruttare il canale virtuale e quindi non è un caso di interesse per questa tesi. Sul siproxd, al contrario di

come avviene sull'applicazione VoIP, non è necessaria la seconda parte della procedura appena descritta per ripristinare la situazione allo stato in cui era prima dell'intervento della ST. Nel caso il pacchetto sia stato deviato, è compito del SM mandarlo alla corretta destinazione. Dopo aver ricevuto il pacchetto, cerca nella lista dei client connessi il destinatario del messaggio confrontando l'IP con quello segnalato dal client nei pacchetti che invia e la porta con quelle fittizie associategli. Una volta trovato il giusto elemento della lista, completa il pacchetto ricevuto con i dati mancanti: il tipo di pacchetto (che ricava dalla ricerca del client grazie alla porta fittizia) e la porta sulla quale il pacchetto è stato inviato (dalla struttura contenente i dati del mittente). Manca ancora un'ultima cosa da fare e cioè modificare la porta su cui far arrivare il messaggio all'applicazione VoIP: si ricorda infatti che nel pacchetto ora c'è la porta fittizia utilizzata per identificare il corretto elemento della lista e questo valore va sostituito con la porta dalla quale i pacchetti di quel tipo sono stati realmente mandati dal client. Ora che il pacchetto contiene tutti i dati corretti (tranne l'identificatore del client che ovviamente non è necessario, a meno che non si voglia in futuro inserire nel CM un controllo sui pacchetti in arrivo) può essere inviato al destinatario attraverso il canale virtuale. IP e porta ai quali il destinatario può essere raggiunto sono entrambi memorizzati nel relativo elemento della lista dei client connessi nei campi real_sender_ip e real_sender_port. Dopo aver attraversato il canale virtuale, il pacchetto inviato da SM è pronto per essere letto da CM. Una volta ricevuto il messaggio, il CM non deve fare altro che rimandarlo all'applicazione VoIP alla porta specificata all'interno pacchetto nel campo destination_port. Anche nell'applicazione VoIP la procedura che legge il messaggio è quella che viene usata dal programma originale per ricevere i pacchetti, e anche qui immediatamente dopo la chiamata a questa procedura di lettura interviene la RT. Il suo compito è identico a quello della RT del siproxd, tranne per il fatto che non deve controllare la sorgente del messaggio in quanto il client riceve messaggi unicamente dal Proxy Server. Quindi la RT comincia

subito modificando la struttura dati contenente i dettagli del mittente inserendovi l'IP e la porta del siproxd prendendoli dal contenuto del pacchetto (i campi sender_ip e sender_port). Dopo questa modifica la funzione procede copiando nel buffer di ricezione il messaggio originale e correggendo il valore della quantità di dati letti (anche questi presi dal pacchetto ricevuto nei campi buf e buflen). Anche in questo caso le operazioni eseguite dalla RT sono volte a nascondere tutte le operazioni eseguite da ST, SM e CM.

Eseguendo queste operazioni tutti i pacchetti che vengono scambiati tra le applicazioni VoIP e il Proxy Server possono essere fatte passare attraverso lo stesso canale di comunicazione indipendentemente dal protocollo utilizzato e senza apportare pesanti modifiche ai programmi preesistenti, che peraltro rimangono all'oscuro di quanto succede allo scambio di messaggi che vedono ancora come diretto. Inoltre grazie al lavoro del SM è possibile realizzare lo scambio di pacchetti anche in presenza di firewall (anche simmetrici) o NAT senza dover ricorrere a protocolli come STUN.

5.6 Debug dell'esistente

Nonostante i programmi di partenza fossero in grado di effettuare una corretta comunicazione, non erano progettati per seguire l'architettura descritta nel capitolo 2. È stato quindi necessario apportare delle ulteriori modifiche al sorgente di questi programmi affinché rispecchiassero il comportamento descritto dalle specifiche ABPS.
Purtroppo anche una volta conformati all'architettura desiderata, i programmi presentavano della anomalie di funzionamento che avrebbero potuto compromettere il funzionamento della soluzione studiata in questa tesi.

Verranno quindi riportate le modifiche che si sono rese necessarie a consentire il corretto funzionamento del canale virtuale in accordo con le specifiche ABPS, oltre a una serie di malfunzionamenti del software che sono stati corretti.
Si noti che i malfunzionamenti sono specifici dei programmi che sono stati studiati durante questa tesi e che potrebbero non essere presenti qualora la realizzazione del canale virtuale venisse effettuata sfruttando altri software come client e Proxy Server.

Pjsip

Sul software usato come client è stato riscontrato un solo problema degno di nota. Purtroppo questo unico problema era decisamente rilevante ai fini della realizzazione del progetto di tesi.
Questo grave problema era che il comportamento di pjsip non era conforme alle specifiche ABPS, ma seguiva lo schema noto come “SIP Trapeziod”: solamente i messaggi SIP venivano inviati al proxy server, mentre lo scambio di pacchetti per la comunicazione multimediale avveniva direttamente con il CN. Ovviamente questo comportamento era da correggere in quanto tutto il progetto è stato studiato per funzionare in una situazione che segue l'ABPS. Fortunatamente questo problema si è rivelato isolato unicamente al pjsip, anche se siproxd ha riportato molti più malfunzionamenti di quanti ci si fosse aspettato. Per correggere questo malfunzionamento è stato sufficiente modificare il contenuto della struttura dati contenente l'indirizzo IP e la porta di destinazione per i pacchetti RTP e quella contenente gli stessi dati per i messaggi RTCP. Nella situazione di partenza in queste strutture venivano messi IP e porta ai quali era possibile contattare il CN con il quale si stava comunicando. Sostituendo questi valori con IP e porta della macchina su cui gira il siproxd la situazione voluta è stata raggiunta.

Libosip

Le modifiche apportare a questa libreria sono tutte volte alla correzione del comportamento della funzione “osip_via_param_del_byname(header,name)” utilizzata da siproxd per modificare il contenuto dei pacchetti SIP ricevuti dai client, per la precisione per rimuovere il parametro “rport:” prima di inserire i parametri “rport=x;” e “received=y.z.w.k”.

Siproxd

Come già anticipato, questo programma ha riportato diversi problemi che impedivano il corretto funzionamento del canale virtuale. La maggior parte di essi è stata risolta, ma per altri è stato possibile solamente applicare una soluzione provvisoria che permette la realizzazione del canale oggetto di tesi, ma non risolve definitivamente il problema. La maggior parte dei malfunzionamenti riguardavano la riscrittura dei campi dei pacchetti SIP ricevuti da effettuare prima di inoltrare gli stessi al server centrale del servizio. È stato a causa di questi problemi che si è reso necessario riscrivere alcune procedure fornite dalla libreria libosip2.

Passiamo ora ad analizzare i problemi riscontrati e le soluzioni apportate, cominciando dai problemi di riscrittura del corpo dei pacchetti SIP.

La funzione che ha il compito di inserire i parametri “rport=x;received=y.z.w.k” ai pacchetti SIP ricevuti contenenti il campo “rport;”, più precisamente la funzione “ip_add_received_param” che si trova in “sip_utils.c”, è stata modificata per rimuovere il vecchio parametro “rport” al campo “VIA” in aggiunta ai suoi normali compiti.

Anche nella funzione “ip_add_received_param” che deve preparare ed inoltrare al server centrale del servizio le richieste ricevute (tranne le “REGISTER”) è stato aggiunto un pezzo di codice che provvede all'inserimento dei parametri

“rport=x;received=y.z.w.k”. Questo nuovo codice interviene solo quando il pacchetto SIP è trasportato usando il protocollo UDP. Il programma originale provvedeva già a eseguire questa modifica al pacchetto se il protocollo utilizzato per inviarlo è il TCP.
La procedura, situata in “proxy.c” e chiamata “proxy_response”, che ha il compito di inoltrare al client corretto la risposta ricevuta dal server del servizio ad una richiesta (che non sia una “REGISTER”) è stata anch'essa modificata. La risposta gestita da questa funzione dovrebbe contenere i parametri “rport=x;received=y.z.w.k” inseriti al momento della trasmissione della richiesta, come descritto precedentemente. Questi parametri indicano il destinatario della risposta. Tale destinatario veniva letto dal campo “VIA” solo per i messaggi SIP trasmessi su TCP, mentre ora viene letto anche per i pacchetti trasmessi su UDP. Come si può facilmente intuire le ultime due modifiche descritte lavorano in simbiosi l'una con l'altra.
Un'ulteriore cambiamento che serve alla corretta modifica del corpo di un messaggio SIP è quella che si occupa di riscrivere il campo “a=rtcp” del pacchetto SDP da mandare all'altro end-system. È stato aggiunto il codice che si occupa del corretto inserimento della porta RTCP per il nuovo stream voce da creare. Questo codice interviene quando il Proxy Server riceve una richiesta “INVITE” o la risposta positiva ad una richiesta di questo tipo. Questa modifica è stata apportata alla funzione “proxy_rewrite_invitation_body” che è implementata nel file “siproxd.c”
Con questa modifica si conclude la serie di cambiamenti atti a correggere la errata riscrittura del corpo dei pacchetti da inoltrare. Il problema di siproxd descritto qui di seguito è un comportamento non corretto riguardante un'altra funzione del programma e al quale, purtroppo, non è stato possibile trovare una soluzione definitiva. Attualmente sono state inserite delle correzioni provvisorie che permettono il corretto funzionamento del progetto di questa tesi, ma queste correzioni non possono essere considerate la soluzione finale ai problemi qui descritti ed è necessario trovare e applicare una soluzione definitiva al malfunzionamento.

Il seguente cambiamento è stato inserito nella funzione “main” che si trova in “siproxd.c”. Quando da un client arriva una richiesta di “REGISTER” da inoltrare, il siproxd aggiunge al pacchetto un campo “VIA” contenente i parametri “rport=x;received=y.z.w.k” in modo che quando arriva la risposta a tale richiesta il programma può andarsi a leggere il destinatario della risposta, cioè il mittente della richiesta. Purtroppo il siproxd, dopo aver aggiunto correttamente il campo “VIA” e inoltrato il messaggio, va a leggere nel pacchetto di risposta il client finale a cui inoltrarla dal campo “CONTACT” invece che dal “VIA”. La soluzione provvisoria, e poco corretta, è stata riscrivere il campo “CONTACT” i dati contenuti nel campo “VIA” in modo che il programma legga i dati corretti, anche se dal campo sbagliato. La strada giusta sarebbe stata modificare il comportamento del siproxd in modo da fargli leggere IP e porta del destinatario dal campo “VIA”, ma questa modifica sarebbe stata troppo complicata da realizzare.

Come ultimo viene riportata una caratteristica di siproxd che non può essere definita un malfunzionamento, ma che è stata ritenuta degna di nota se non altro per evidenziare il fatto in previsione di sviluppi futuri. Questa caratteristica riguarda la memorizzazione della porta per lo stream RTCP, gestita all'interno del file “rtpproxy_relay”. Questo programma è impostato per usare come porta per i messaggi RTCP quella successiva a quella usata per lo stream RTP e non non accetta l'uso di una porta diversa. Per questo motivo la porta utilizzata per i messaggi RTCP non viene memorizzata in una struttura apposita, bensì viene ricavata partendo da quella per i pacchetti RTP. Non solo le porte utilizzate per la trasmissione di messaggi di questo tipo devono necessariamente essere consecutive, ma è anche necessario che il numero della porta per lo stream RTP deve essere pari. Questo comportamento non è di per sé un problema e durante lo svolgimento del progetto di tesi si è rimasti conformi a questa restrizione.

Capitolo 6

Note implementative

In questo capitolo vengono analizzate alcune scelte progettuali che sono state fatte durante la progettazione e la realizzazione del progetto di tesi.
Cominciamo analizzando il pacchetto creato dalle procedure ST e la cui struttura è mostrato nella figura 5.3.

- sender_ip (16 Byte) → array di caratteri da 16 elementi: sufficiente per conenere un indirizzo IP (che può avere al massimo 15 caratteri) più il carattere terminatore di stringa. In tutte le strutture dati introdotte in questo progetto i campi creati per contenere indirizzi IP sono uguali a questo
- sender_port (4 Byte) → intero a 32 bit senza segno
- destination_ip (16 Byte) → array di caratteri da 16 elementi
- destination_port (4 Byte) → intero a 32 bit senza segno
- packet_type (4 Byte) → intero a 32 bit senza segno
- buflen (4 Byte) → intero a 32 bit senza segno
- buf (2000 Byte) → array di caratteri da 2000 elementi: grande quanto i buffer di lettura e di scrittura che usa pjsip

La dimensione dei dati aggiunti al messaggio originale è quindi di 50 Byte. Questo header viene aggiunto a tutti i pacchetti inviati dall'applicazione VoIP, anche ai ping da 2 Byte inviati da pjsip a siproxd. La dimensione finale di un ping che deve passare dal canale virtuale aumenta quindi del 2500%, aumento che potrebbe sembrare eccessivo, ma considerando che questi ping vengono mandati al ritmo di uno ogni 10 secondi, il

consumo di banda si può considerare sostenibile. Analizzando gli altri messaggi, quindi i messaggi SIP per la gestione della connessione e delle chiamate e i messaggi RTP e RTCP contenenti dati multimediali, troviamo poi che la dimensione media di questi pacchetti si aggira intorno ai 700 Byte. A questi pacchetti, che vengono mandati a ritmo decisamente più alto rispetto ai ping, specialmente gli RTP, l'aggiunta di un header di 50 Byte non comporta un aumento eccessivo di uso delle risorse in quanto la dimensione del messaggio finale aumenta di meno del 10%. Si noti anche che questa struttura dati è stata creata per essere memorizzata con allineamento al Byte, in modo che occupi la stessa quantità di memoria su tutti sistemi indipendentemente dal tipo di processore o dalle politiche di allocazione della memoria. Questa è una precauzione necessaria in quanto questa è la struttura dati che deve essere spedita dal MN al Proxy Server e che le due macchine potrebbero essere differenti sia dal punto di vista dell'architettura hardware, sia dal punto di vista del sistema operativo attivo.
Si passa ora ad analizzare la struttura dati usata per creare la lista dei client connessi e mostrata in figura 5.4.

- real_sender_ip (16 Byte) → array di caratteri da 16 elementi
- real_sender_port (4 Byte) → intero a 32 bit senza segno
- sender_id (4 Byte) → intero a 32 bit senza segno
- official_sender_ip (16 Byte) → array di caratteri da 16 elementi
- official_sender_port (20 Byte) → array di interi a 32 bit senza segno da 5 elementi
- fake_sender_port (20 Byte) → array di interi a 32 bit senza segno da 5 elementi
- server_port (20 Byte) → array di interi a 32 bit senza segno da 5 elementi
- next (4~8 Byte) → indirizzo di memoria

Si noti che anche questa struttura dati è stata creata per essere memorizzata con allineamento al Byte, in modo che occupi la minor quantità di memoria possibile. Questo perché si vuole mantenere il programma SM il più leggero possibile in modo

da minimizzare l'impatto che esso ha sulle prestazioni del Proxy Server.
Il pacchetto che viene invece mandato dalla procedura ST presente sul siproxd attraverso il socketpair ha una dimensione decisamente più ridotta:

- ip (16 Byte) → array di caratteri da 16 elementi
- port(4 Byte) → intero a 32 bit senza segno

Nonostante sia sufficiente la porta per scoprire se un client è presente nella lista mantenuta i memoria dal SM, si è scelto di inviare anche l'IP in previsione di uno sviluppo futuro che tenga conto della possibilità di chiamate audio/video in conferenza. Si parlerà più approfonditamente di questa possibilità nel capitolo 8.
All'interno della libreria pjsip sono stati definiti nuovi tipi e nuove chiamate per gestire la creazione di programmi che utilizzando il protocollo SIP, permettendo l'astrazione da hardware e sistema operativo e creando cosi' una piattaforma indipendente. Il Client Multiplexer e le procedure ST e RT che operano a lato client, essendo state sviluppate utilizzando le funzioni e i tipi forniti da pjsip, godono dell'indipendenza fornita da questa stessa piattaforma, riuscendo in questo modo a funzionare su diverse tipologie di hardware e software.
Per rendere funzionante il progetto di tesi è necessario abilitare alcune opzioni nel file di configurazione di pjsua:

bound-addr serve per specificare l'IP che la macchina ha sulla rete locale, o comunque l'IP che verrà visto dal programma (pjsua potrebbe fare riferimento ad una interfaccia virtuale che maschera quelle reali in modo che, in caso di cambio di interfaccia dovuto all'applicazione delle regole di ABPS, il programma in esecuzione continui a funzionare indipendentemente dal tipo di connessione presente). Questo parametro è necessario in quanto la funzione ST del client prende l'IP da inserire nel nuovo pacchetto dal valore specificato in

questa opzione.

outbound serve per indicare l'IP della macchina usata come Proxy Server e la porta sulla quale inviarle i pacchetti SIP. A quest'opzione va aggiunto anche il parametro ";lr" che fa in modo che i pacchetti di risposta del Proxy Server vengano inviati all'IP e sulla porta specificati all'interno dei pacchetti SIP provenienti dal client. L'opzione "outbound" permette di deviare il flusso dei pacchetti verso il Proxy Server, aderendo così alle specifiche dell'architettura ABPS. Il parametro ";lr" è necessario affinché il Proxy Server tenti di inviare i pacchetti destinati ad applicazioni VoIP che fanno uso del canale virtuale, verso un IP ed una porta riconoscibili dal SM come appartenenti ad un client registrato.

In pratica le opzioni qui descritte fanno in modo che la ST possa completare con il corretto indirizzo IP il pacchetto da mandare al CM, che l'applicazione VoIP funzioni seguendo le regole dell'architettura ABPS e che il SM sia in grado di mantenere in memoria e riconoscere i client connessi al Proxy Server mediante il canale virtuale.

Capitolo 7

Valutazione

Per valutare in modo esaustivo il progetto di tesi bisogna analizzarlo da diversi aspetti che vanno dalla qualità della trasmissione all'utilizzo di risorse per realizzarla.
Per testare il funzionamento di questo progetto di tesi, sono state effettuate diverse prove di connessione e comunicazione tra dispositivi che presentano differenti configurazioni di rete e architetture hardware. Nello specifico i test sono stati compiuti sia su macchine aventi indirizzi IP pubblici, sia su elaboratori connessi ad una rete privata con accesso ad internet filtrato da firewall e protetto da NAT. Oltre a questo le differenze dei dispositivi utilizzati per i test si estendono anche all'architettura del processore (32 e 64 bit) e al sistema operativo. I sistemi operativi con i quali è stata testata la connessione sono quelli elencati nel capitolo 4.4. Le prove di comunicazione sono state eseguite anche tra software che fanno riferimento al Proxy Server e applicazioni che comunicano direttamente con il server centrale del servizio.
I test effettuati nelle varie situazioni di connessione, architettura e sistema operativo hanno riportato esiti analoghi. Riguardo alla qualità del segnale trasmesso abbiamo potuto verificare che la quantità di pacchetti che giungono a destinazione si aggira sempre attorno al 95% di quelli inviati tant'è vero che anche ascoltando l'audio ricevuto si può apprezzare una buona qualità. Si passa ora ad analizzare la quantità di risorse richieste per il funzionamento dei programmi utilizzati. Per quanto riguarda l'applicazione VoIP (pjsua) si è potuto constatare che il consumo di CPU non è mai superiore al 10% e la quantità di memoria occupata è compresa tra 1.5 e 2.3 MByte.

Analizzando siproxd è emerso che il l'utilizzo del processore non supera il 5% e la memoria richiesta per l'esecuzione è inferiore a 1 MByte. Come ultimo metro di valutazione si propone il tempo che un messaggio impiega per andare da un client ad un altro. Dai test effettuati è emerso che tra l'invio di un messaggio da parte di un MN e la ricezione dello stesso da parte del CN passano in media 125 millisecondi. I tre test, che sono stati eseguiti dopo aver introdotto il canale virtuale, hanno riportato risultati estremamente vicini agli stessi test effettuati durante una comunicazione realizzata senza l'utilizzo del canale oggetto di tesi. Tutti i valori qui riportati sono stati osservati durante l'esecuzione di una chiamata tra due applicazioni VoIP. Ovviamente nel caso di più chiamate contemporanee effettuate da più applicazioni il carico di lavoro del Proxy Server, e probabilmente anche il tempo impiegato da ogni pacchetto per giungere a destinazione, aumenterà in proporzione.

Un ulteriore parametro con il quale sarebbe stato interessante valutare la qualità del risultato ottenuto è la scalabilità del software realizzato e quella del software che ne fa uso. Purtroppo per mancanza di strumenti non è stato possibile valutare il progetto sotto questo punto di vista.

Si analizzano infine i limiti del progetto in questione. Come primo punto viene segnalata la non gestione delle chiamate in conferenza. Alla situazione attuale non è infatti possibile che un unico stream audio/video venga inoltrato a più di un destinatario, a meno che tale funzionalità non sia interamente gestita dal Proxy Server: se per ogni pacchetto audio o video in ingresso il software del Proxy Server invia un messaggio ad ogni partecipante alla comunicazione di gruppo (tranne ovviamente il mittente del pacchetto in ingresso) allora il sistema è in grado di sostenere la chiamata in conferenza in quanto ogni partecipante viene gestito singolarmente. Anche nel caso in cui questa funzionalità sia interamente gestita dall'applicazione VoIP il sistema è in grado di sostenerla: se per ogni pacchetto multimediale creato l'applicazione ne invia uno ad ognuno dei client che lo devono ricevere, allora i Multiplexer creati per gestire

il canale virtuale permettono la chiamata in conferenza, perché anche in questo caso ogni partecipante viene gestito singolarmente. In pratica se ogni partecipante viene visto come facente parte di una diversa comunicazione (anche se a tutti vengono mandati gli stessi dati) una chiamata in conferenza è possibile anche allo stato attuale delle cose.

Altro punto da analizzare è la possibilità che un utente può avere di tenere attive più chiamate contemporaneamente. Questo caso è parzialmente gestibile dal sistema attuale e unicamente se solamente una delle chiamate sia attiva contemporaneamente. La comunicazione è gestita parzialmente perché non ci sono problemi per lo stream in uscita in quanto il sistema, alla fine di tutti i passaggi, consegna il messaggio al destinatario originale designato dall'applicazione VoIP che lo ha generato. Per quanto riguarda lo stream in ingresso la questione è più delicata in quanto è il SM che conosce le reali porte utilizzate dal client. Si consideri la seguente situazione: un utente sta avendo due chiamate contemporaneamente e ha avuto uno scambio di messaggi con uno dei due interlocutori. A questo punto l'utente riattiva l'altra comunicazione, ma non vengono inviati dati al secondo interlocutore. I messaggi provenienti dal secondo interlocutore verranno mandati all'IP corretto, ma sulle porte utilizzate per la comunicazione con il primo interlocutore in quanto non sono ancora stati mandati dati dalle porte usate per la connessione corrente e quindi il SM non ha potuto aggiornare le porte utilizzate. Una soluzione provvisoria a questa situazione è l'invio di una specie di ping da tutte le porte ogni volta che si cambia la chiamata attiva: in questo modo il SM può tenere aggiornate le porte correntemente utilizzare dall'applicazione VoIP e rimandare i pacchetti alla corretta destinazione. Essendo quella proposta una soluzione poco elegante oltre che provvisoria, si rimanda l'ideazione e la progettazione di una gestione migliore delle chiamate parallele ad uno sviluppo futuro.

Si noti che i comportamenti del software nel caso di chiamate in conferenza e nel caso

di più chiamate contemporanee dello stesso utente non sono state testate e che le precedenti affermazioni sono basate su ragionamenti fatti analizzando il codice esistente, senza alcun riscontro pratico. Si sottolinea inoltre che la gestione di questi due casi esula dagli obiettivi del progetto di tesi.

Capitolo 8

Conclusioni e sviluppi futuri

Come già detto nel capitolo 3, l'obiettivo di questo lavoro era quello di contribuire a creare un canale virtuale tra il nodo mobile e il server di supporto, operante su una sola coppia di porte, nel quale fosse possibile trasmettere qualunque tipo di dato indipendentemente dal protocollo da utilizzare o dalla presenza o meno di un firewall o NAT, oltre a minimizzare il keep-alive sulle porte mantenute aperte sulla rete. Questo obiettivo è stato raggiunto: il canale opera sull'unica coppia di porte tenute aperte sulla rete minimizzando così il keep-alive, funziona in presenza di firewall o NAT ed è indipendente dal protocollo utilizzato per la trasmissione dei dati attraverso di esso. I vantaggi descritti nel terzo capitolo di questa tesi sono stati riscontrati nel funzionamento dei programmi modificati per sfruttare questo canale virtuale, tranne la possibilità di attivare o disattivare a runtime l'utilizzo del canale. Questa feature non è stata riscontrata non perché non realizzabile, ma in quanto non sono stati fatti test sulla attivabilità/disattivabilità. Si ricorda inoltre che il funzionamento del progetto è stato testato solamente all'interno del caso di chiamata singola da parte di ogni utente.

Sulla base del progetto realizzato sono possibili molteplici sviluppi realizzabili in futuro. Qui di seguito ne vengono proposti alcuni.

La gestione dell' IPv6: il sistema e i pacchetti sono stati studiati per la gestione di indirizzi di rete IPv4. Con l'avvento degli indirizzi di rete IPv6 si renderà necessario implementarne la gestione.

Gestione dinamica degli identificatori del client: nella soluzione proposta in questa tesi è necessario che ad ogni client venga assegnato un identificatore univoco mediante il

quale il SM è in grado di distinguerlo. Il problema di questa soluzione è che nel caso due client abbiano, per errore ovviamente, due identificatori uguali, nessuno dei due sarà in grado di stabilire una connessione funzionante finché l'altro non cessa l'invio di pacchetto al SM. È quindi consigliabile implementare un assegnamento dinamico degli identificatori, interamente gestito da SM e CM, in base magari all'IP segnalato dal client in fase di registrazione e l'IP dal quale SM vede arrivare i pacchetti (official_client_ip e real_client_ip).

Attivazione e disattivazione runtime del canale virtuale: definire un comando o un'opzione che permette la non esecuzione delle procedure ST e RT sul client, in modo da disabilitare l'utilizzo del canale virtuale, continuando a inviare e ricevere messaggi nel modo originale.

Controllo sui pacchetti in ingresso da parte di CM: con il completamento del campo sender_id dei messaggi in uscita da parte del SM (che inserirà l'identificatore del client a cui il pacchetto sta per essere mandato) potrebbe essere possibile inserire un controllo sui pacchetti ricevuti dal CM che inoltrerebbe all'applicazione VoIP solamente quelli contenenti il corretto identificatore.

Sostituzione del Proxy Server: considerata la ristretta quantità di modifiche al software originale del Proxy Server necessarie per permettere l'utilizzo del canale virtuale, si consiglia di sostituire siproxd con un altro programma che svolga lo stesso compito, ma che presenti meno anomalie di funzionamento. Uno dei candidati potrebbe essere "Kamailio" (www.kamailio.org/w/).

Gestione delle chiamate in conferenza e di più chiamate contemporanee da parte di uno stesso utente: come già spiegato nei precedenti capitoli, la gestione di questi tipi di chiamate non faceva parte degli obiettivi di questo progetto e non è quindi stata implementata. Nel capitolo sette si è tentato di prevedere il comportamento del software prodotto al presentarsi di queste situazioni, ma è necessaria una gestione mirata di questi due servizi. In particolare è necessaria una soluzione per le chiamate

in conferenza che tenga conto delle varie possibili gestioni del servizio da parte del software del Proxy Server e dell'applicazione VoIP.

Firmare e criptare i messaggi: come già accennato è possibile far passare qualsiasi tipo di dato attraverso il canale virtuale. Uno sviluppo interessante potrebbe essere quello di far inserire dai due multiplexer un sistema di crittografia: il multiplexer che invia il pacchetto cifra il messaggio, mentre quello che lo riceve lo decodifica. In questo modo i pacchetti che vengono mandati attraverso la rete non sono in chiaro, ma le applicazioni che fanno uso del canale virtuale non devono essere modificate per supportare la crittografia. Anche un meccanismo per l'inserimento di una sorta di firma digitale (in modo da poter identificare il mittente dei pacchetti) potrebbe essere realizzato all'interno della comunicazione tra i multiplexer in modo indipendente dall'applicazione originale.

Bibliografia e Sitografia

•**Always Best Packet Switching: the mobile VoIP case study** – Vittorio Ghini, Giorgia Lodi, Fabio Panzieri - http://www.cs.unibo.it/~ghini/accepted/Ghini_JCM17.pdf

•**Libosip2 documentation** - http://www.gnu.org/s/osip/

•**Pjsip online manual** and **pjsip online documentation** – comunità di pjsip - http://www.pjsip.org/docs/latest/pjlib/docs/html/index.htm

•**Siproxd online documentation** and **siproxd forum** – comunità di siproxd http://siproxd.sourceforge.net/index.php?op=odoc - http://sourceforge.net/projects/siproxd/forums

•**The "Always Best Packet Switching" architecture for SIP-based mobile multimedia services** – Vittorio Ghini, Stefano Ferretti, Fabio Panzieri - http://www.cs.unibo.it/~ghini/didattica/sistdistrib/ABPS_VoIP_JSS.pdf